EDUDEMOS

3-in-1 Demonstrator:

Solar Energy, Wind Energy, and Water Collection in a Single Educational Project

EDUDEMOS

The “EduDemoS” project is funded by the European Union. Views and opinionsexpressed are however those of the author(s) only and do not necessarily reflect thoseof the European Union or the European Education and Culture Executive Agency (EACEA). Neither the European Union nor EACEA can be held responsible for them.

This first section will contain all the necessary information to understand the project, its funding, objectives, and participants. The sections of this first block of the guide will be:

1. INTRODUCTION TO EDUDEMOS

1. 1. What is EduDemos?

EduDemos is an educational project that develops sustainable demonstrators based on renewable energies such as water, sun, and air. These demonstrators are interactive and replicable tools used to teach young people and teachers about sustainability and digital skills, integrating hands-on learning with environmental awareness.

1. 2. Objectives

The main objective of the EduDemos project is to raise awareness among new generations, both young people and teachers, about sustainability through the use of demonstrators based on renewable energies such as sun, water, and air. Additionally, it aims to enhance digital skills in the educational field by providing practical and replicable technological tools. Another of its purposes is to foster a collaborative network among educators to exchange knowledge and practical experiences, promoting the adoption of sustainable practices and contributing to a greener future in Europe.

1. 3. Methodology

The methodology of the EduDemos project is structured around the practical implementation of sustainable demonstrators that use renewable energy sources such as water, sun, and air. These demonstrators are not just theoretical tools; they are designed to be built and experienced by students, promoting active and hands-on learning.

- **Model development:** Demonstrator models integrating renewable energy technologies are designed. These models serve as practical examples that students can replicate.

- **Construction guides:** EduDemos provides detailed guides for the construction of these demonstrators, ensuring that students and teachers can follow the step-by-step process. These guides are accessible and designed to be used in an educational setting, facilitating the learning of technical and scientific concepts in a practical way.

- **Teacher and student training:** Training programs are implemented for both teachers and students. These programs are designed to develop skills in both sustainability and digital competencies, preparing participants to face the challenges of climate change. The training includes workshops, courses, and educational resources that complement theoretical learning with practical application.

- **European collaboration:** The project promotes collaboration between various educational institutions and organizations across Europe. This cooperative network enables the exchange of knowledge, experiences, and resources, enriching the educational process. The project partners work together to adapt and improve the models and guides according to local and regional needs, fostering more inclusive and effective education throughout Europe.

1. 4. Partners

The EduDemos project has several key partners who contribute their expertise and resources to the success of the project:

- **Gerda Stetter Stiftung (Germany):** This foundation specializes in educational and technological projects, with a focus on sustainable development. Its participation ensures that the technical and pedagogical aspects of the project are of high quality.
- **Fundación Sergio Alonso (Spain):** A foundation that supports educational and social initiatives, especially in the Canary Islands. Its contribution is

crucial for the implementation of the project in Spain and for ensuring its local relevance.

- **FINNOVAREGIO (Belgium):** This organization focuses on regional innovation and sustainable development. Its role in EduDemos is key to integrating the project into European networks of innovation and sustainability.

- **GBS St.Gallen (Switzerland):** This Swiss educational institution brings its expertise in vocational and technical training, particularly in the field of renewable energies. It contributes to the development of the educational content and the project's methodology.

- **IES El Rincón (Spain):** A secondary education institute in Gran Canaria, Spain, that serves as one of the pilot centers where the project's sustainable demonstrators are implemented and tested. Its participation is essential for tailoring the project to the needs of students and teachers.

Each of these partners brings their specific expertise, enabling EduDemos to be a comprehensive project that addresses sustainability education from different perspectives and within various European contexts. The collaboration among these partners ensures that the resources developed are diverse, accessible, and applicable across different educational settings.

2. DEMONSTRATOR 3x1

2. 1. Description of the Models

2. 1. 1. Objectives

The objective of the demonstrators is to teach and raise awareness among students about how energy is generated from wind, sun, and water harvesting, promoting a practical understanding of renewable energies and the importance of environmental sustainability. These demonstrators aim to engage students in the construction and experimentation with sustainable technologies, fostering technical skills and sustainability-related competencies.

These demonstrators are primarily designed for students in vocational education, but they can also be used at the secondary education level. Specifically, they target students from compulsory secondary education to high school and vocational training, where students already have a foundation in science and technology, enabling them to understand and experiment with more advanced concepts such as wind or solar energy.

Their educational value lies in their ability to provide a practical and contextualized learning experience around renewable energies. Through the construction and use of the 3-in-1 Demonstrator, students gain a deeper understanding of how energy is generated from wind, sun, and water harvesting, while also fostering technical skills, digital competencies, and a greater commitment to environmental sustainability. This facilitates active learning and the development of key competencies in science and technology.

2. 1. 2. Educational Competencies

The demonstrators support the development of key competencies in students pursuing vocational training, particularly those studying technical professional families, as well as students from 1st to 4th year of secondary education. Through their construction and analysis, students gain essential skills for their growth, while also fostering teamwork, applying creativity, and critical thinking to solve problems. Additionally, they will understand the impact of sustainable technologies on society. This approach promotes comprehensive training that prepares students to face current and future challenges.

Educational Competencies to Be Acquired with the 3-in-1 Demonstrator for **1st to 3rd Year ESO:**

1. **Linguistic Communication Competency (CCL):**
   Development of communication skills: Students will learn to communicate their ideas and results using appropriate technical language. This applies both to presenting technological solutions and preparing basic technical documentation.

2. **Mathematical Competency and STEM (Science, Technology, Engineering, and Mathematics):**
   Application of scientific and technological knowledge: Students will apply basic physics and technology concepts to understand and explain the functioning of the 3-in-1 Demonstrator, including the interpretation and design of simple electrical circuits and diagrams.

3. **Digital Competency (CD):**
   Use of digital tools: The use of software for project design and simulation will be fundamental. Students will learn to use digital applications to represent diagrams, circuits, and design components of the 3-in-1 Demonstrator.

4. **Personal, Social, and Learning to Learn Competency (CPSAA):**
   Teamwork and self-regulation: Through collaboration in constructing and evaluating the Demonstrator, students will develop teamwork skills, self-regulation, and critical reflection on their learning process.

5. **Entrepreneurial Competency (CE):**
   Creativity and problem-solving: Students will be challenged to come up with creative and sustainable solutions to improve the performance of the 3-in-1 Demonstrator, fostering an entrepreneurial mindset that values innovation and sustainability.

6. **Cultural Awareness and Expression Competency (CCEC):**
   Cultural and technological appreciation: Students will reflect on the importance of sustainable technologies in different cultural contexts, recognizing the value of technological heritage and its impact on society.

These competencies are acquired through a practical and contextualized approach around the 3-in-1 Demonstrator, allowing students to develop essential skills that are relevant both in their academic and personal lives.

Educational Competencies to Be Acquired with the 3-in-1 Demonstrator for **4th Year ESO:**

1. **Linguistic Communication Competency (CCL):**
   Participation in collaborative projects: Students will develop the ability to effectively communicate their ideas and results in various formats (oral, written, visual) using appropriate and respectful technical language. The use of inclusive and non-sexist language in all project-related communication will also be promoted.

2. **Mathematical Competency and STEM (Science, Technology, Engineering, and Mathematics):**
   Design and manufacture of solutions: Students will apply knowledge of mathematics, science, and technology to design and build the Demonstrator, using computer-aided design (CAD) tools and digital manufacturing techniques (such as 3D printing). Additionally, they will analyze the product life cycle to ensure that solutions are sustainable and efficient.

3. **Digital Competency (CD):**
   Use of digital tools: Students will use digital applications and platforms to manage the project from ideation to final presentation. They will learn to validate and cross-check information using reliable digital sources, and configure digital tools as needed to optimize their learning and solve technological problems.

4. **Personal, Social, and Learning to Learn Competency (CPSAA):**
   Teamwork and project management: Throughout the project, students will be encouraged to work effectively as a team, sharing responsibilities and collaborating to solve problems. Self-reflection and continuous improvement of the learning process will also be promoted through critical evaluation of proposed solutions.

5. **Entrepreneurial Competency (CE):**
   Creativity and innovation: Students will be encouraged to develop innovative technological solutions that create value for the community, using problem-solving strategies and fostering an entrepreneurial spirit. The planning and execution of sustainable and accessible projects will be key in this process.

6. **Cultural Awareness and Expression Competency (CCEC):**
   Expression and dissemination of ideas: Students will learn to express and disseminate their technological proposals creatively and effectively, using various media and techniques. The importance of valuing artistic and cultural creation in technology will be emphasized, promoting an ethical and socially responsible approach to their projects.

These competencies are designed to ensure that students not only acquire technical knowledge but also develop transversal skills and a critical, sustainable mindset, essential in the context of secondary education.

2. 1. 3. Basic Concepts

- Wind demonstrator

The wind demonstrator is an educational tool that allows students to build a small wind turbine to understand the basic principles of wind energy and its application in generating electricity. Through interaction with the device, students learn how the kinetic energy of the wind is first transformed into mechanical energy by the movement of the blades, and then into electrical energy thanks to a generator.

This demonstrator includes essential components such as blades, a shaft, and a generator, facilitating practical understanding of concepts like blade aerodynamics, the relationship between wind speed and energy efficiency, and the environmental impact of wind energy compared to other energy sources. It also allows students to explore how the design and orientation of the blades influence the maximization of energy generation.

Inspired by César Manrique's "Wind Toys," this device merges art and educational functionality, connecting the cultural heritage of the Canary Islands with the

teaching of renewable energy. It is a tool that not only promotes technical learning but also celebrates the harmony between nature, technology, and local culture.

- Solar demonstrator

The solar demonstrator is an educational tool designed to help students understand how sunlight is converted into electricity through photovoltaic panels. Inspired by the movement of sunflowers, which optimize their position to maximize solar energy capture, this device allows students to observe how solar cells capture light and transform it into electrical current, practically illustrating the photovoltaic effect.

The demonstrator includes an adjustable solar panel, enabling students to explore how the panel's orientation and the angle of sunlight impact energy efficiency. Just as sunflowers follow the sun throughout the day to maximize exposure, the demonstrator emphasizes the importance of proper alignment of solar panels for optimal performance.

In addition to teaching the technical principles of photovoltaic energy, the demonstrator provides an educational experience that connects students with nature and fosters a practical and efficient perspective on harnessing sustainable resources.

- Water demonstrator

The water demonstrator is an educational tool that allows students to understand how rainwater can be efficiently captured and managed. Through this device, students observe the water collection process, from the capture on a roof-like surface to its storage in a reservoir. This helps them explore the water cycle and apply water conservation strategies in everyday life.

The demonstrator illustrates how rainwater, a renewable and free resource, can be utilized for various practical applications, such as plant irrigation. Additionally, students learn about the importance of keeping collection and storage systems clean, as well as the significance of sustainability in efficient water use, especially in areas with limited water resources.

Inspired by the aesthetics and functionality of the Bejeque, a native succulent of the Canary Islands, this water collector connects nature with modern technology, demonstrating how biomimetic solutions can help preserve water resources. Following the philosophy of artistic and natural integration promoted by César Manrique, the collector merges art, ecology, and educational technology, encouraging respect for the environment and local culture, while teaching students to value and apply nature-inspired sustainable solutions for the future.

3. PROGRAMMING OF THE DEMONSTRATORS

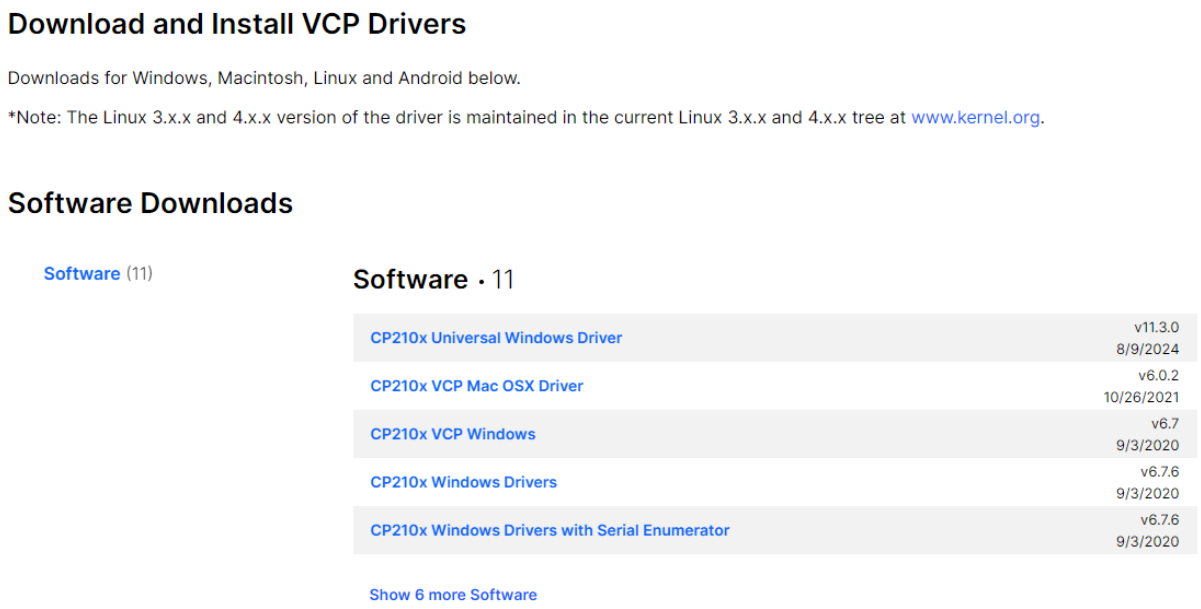
3. 1. Minimum Requirements

The demonstrators use a programmable board called ESP32. To begin programming this board from a PC, several key aspects must be considered.

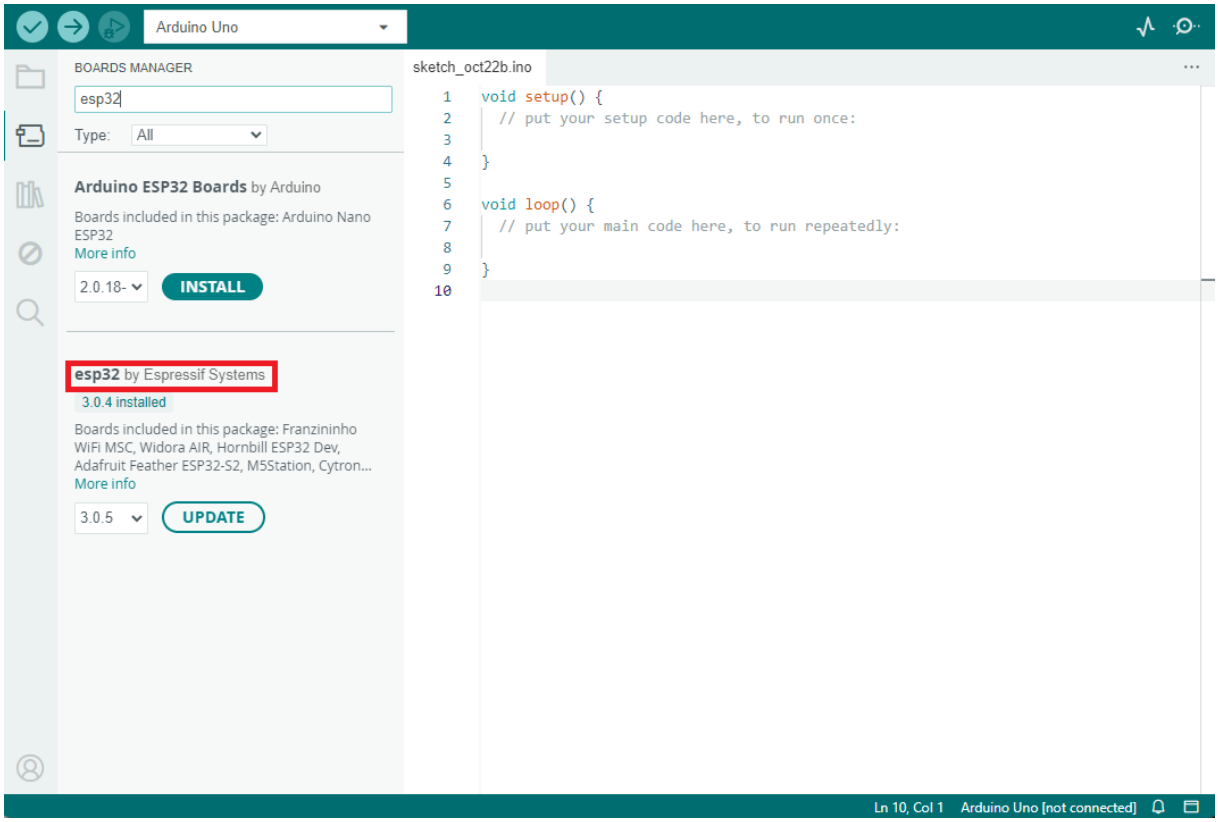
First, **we need to have the IDE** (Integrated Development Environment) installed, which serves as the interface that allows us to program our board. The ESP32 is compatible with the Arduino IDE, which is well-known and easy to use. To download the application, we need to visit the official Arduino website. For PC users, the easiest option is to select the appropriate version based on the operating system of the computer. This will download an executable file that will install the development tool.

Note: *in the Edudemos website* **› *demonstrators* › *3x1 demonstrator* › *guide*** *you will find the document* ***"Previous Requirements"****, where it explains step by step how to install the Arduino software, the drivers, libraries, and the main code.*

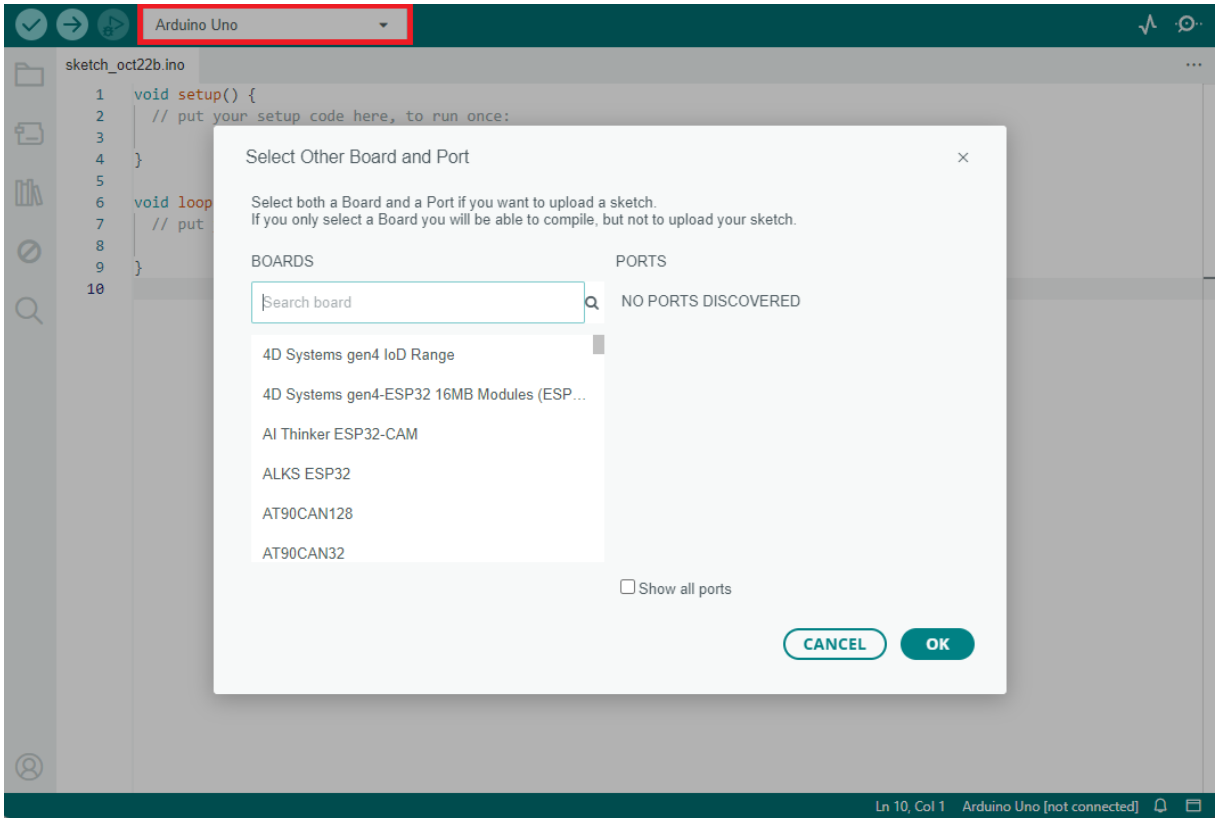
Secondly, the application communicates with the board through one of the computer's USB ports, using a serial communication protocol. To facilitate this communication, the board has a small integrated chip dedicated to this function. On the ESP32 board with ESP32, this function is performed by the CP2102 chip. If the computer does not automatically detect the board when connected via USB, **we will need to install the appropriate drivers**. These drivers can be downloaded from the manufacturer's website.



Thirdly, before being able to program the ESP32, **you need to add support for the ESP32 board** in the Arduino IDE. To do this, go to Tools > Board > Board Manager and search for "ESP32." Install the support package created by the microcontroller's manufacturer, "Espressif Systems," for that board. You should now have a list of boards available, including the "**ESP32-WROOM-DA**", which we will use throughout this guide.



Lastly, you only need to select the port where the board is connected to the computer **via a micro USB to USB Type-A cable**. Serial ports are selected at the top of the Arduino IDE under the Tools > Port tab. From there, choose the port where the board is connected.

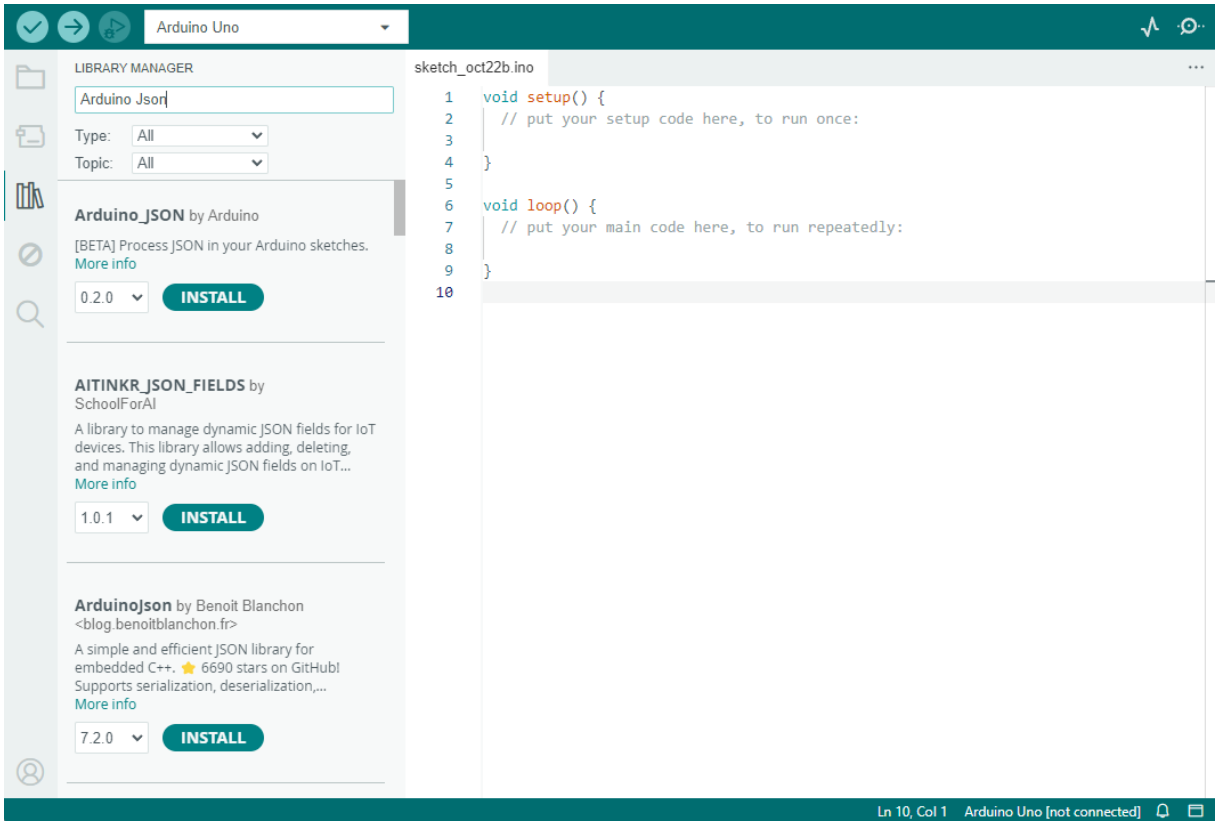


If you're unsure which port is assigned to the ESP32, you can disconnect and reconnect the board to identify it. On Windows, the port will appear as COMX (where X is the port number). On other operating systems, like macOS or Linux, the ports may be displayed as /dev/ttyUSBX or /dev/ttySX. By disconnecting and reconnecting the board, observe which port appears or disappears from the list to identify the correct one.

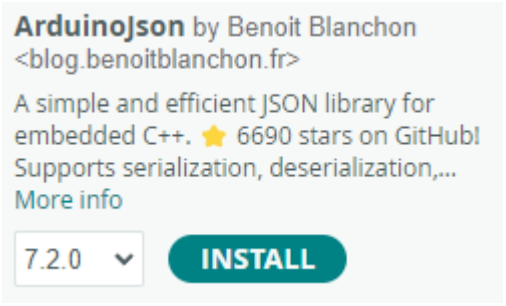
3. 2. Required Libraries

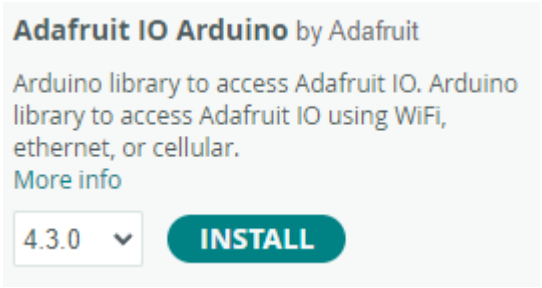
An Arduino IDE library is a set of pre-written code that simplifies programming by providing ready-to-use functions and procedures. These libraries group together commands and tools that allow interaction with specific components and modules, such as sensors, motors, displays, and more, without the need to write all the code from scratch. By including a library in your project, you can simplify complex tasks and make your code cleaner and easier to understand.

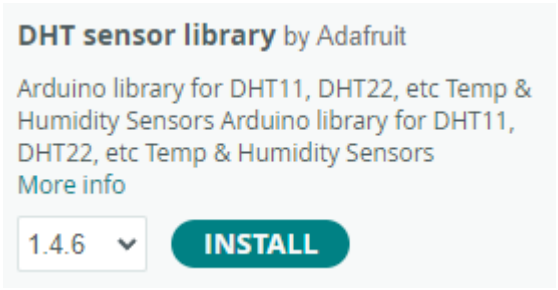
At this stage, you'll need to install several libraries that are necessary for the microcontroller code (we'll explain how to download it in the next section) to function correctly. These libraries will allow proper sensor readings and facilitate the connection of the Demonstrators to the internet.



Within the Arduino IDE, go to the Tools > Manage Libraries tab, search for the necessary libraries for the code: **ArduinoJson, Adafruit_WiFi, and DHT sensor library**, and install them.

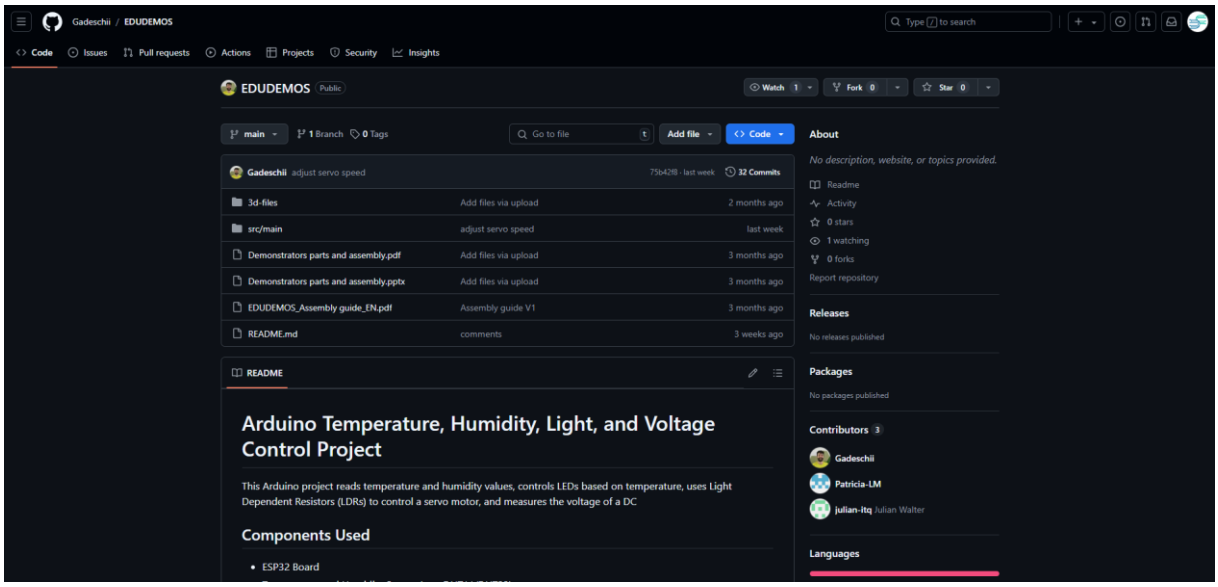




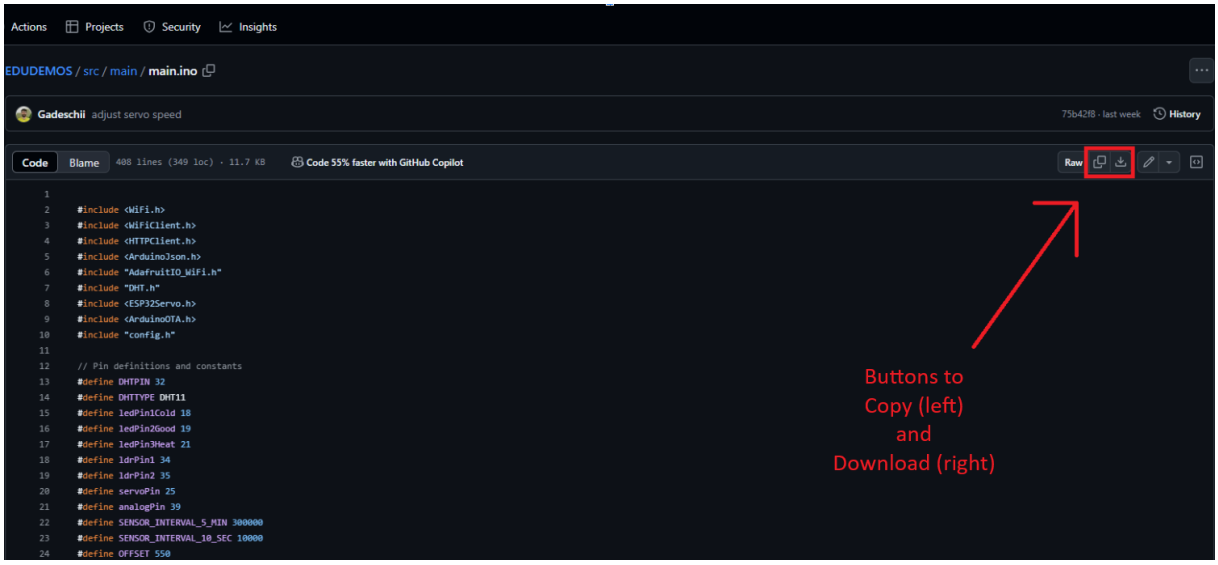


3. 3. Finding and Downloading the Code

The code for the Demonstrators can be found in the **official EDUDEMOS repository** on GitHub, an online platform that allows developers to store, manage, and collaborate on programming projects. User repositories serve to host their code, collaborate with other developers, and manage software versions.
Visit the official GitHub page and search for the EDUDEMOS repository. You should see a page like the following:



You can find the source code for the Demonstrators in the **src/main/** folder in the **main_2.ino** file. Within GitHub, you can open it to view the program. There are two ways to obtain the code: by downloading the file using the download button or by copying the lines of code and pasting them into an empty file in your Arduino IDE using the copy button.



In the repository, you will find a file called README.md, which explains the code and the functions it uses. You can refer to it whenever you need.

3. 4. How the Code Works

The source code implemented for an ESP32 microcontroller in this case allows for the measurement of temperature and humidity, LED control based on temperature values, the use of an LDR to control a servo motor, and the measurement of a generated direct current voltage. Below, the code is broken down and explained in parts to provide an overview of its functionality.

In the *setup()* function, serial communications are initialized, the WiFi connection is configured, OTA (Over The Air) updates are enabled, and both the DHT sensor and the LEDs and servo motor are prepared for use. Additionally, FreeRTOS tasks are created to manage data readings from the sensor and the voltage generated by the motor.

```
void setup() {
  Serial.begin(115200); // Start serial communication at 115200 baud rate
  analogReadResolution(12); // ESP32 has a 12-bit ADC
  analogSetAttenuation(ADC_11db); // Set attenuation for higher sensitivity
  pinMode(4, INPUT);

  setupWiFi(); // Setup WiFi connection
  setupOTA(); // Setup OTA for remote updates
  setupDHT(); // Setup DHT sensor
  setupLEDsAndServo(); // Setup LED pins and servo

  // Create tasks
  xTaskCreatePinnedToCore(
    readSensorTask1,    // Task function
    "ReadSensorTask1", // Name of the task
    4096,              // Stack size
    NULL,               // Task input parameter
    1,                  // Priority of the task
    NULL,               // Task handle
    0                   // Core where the task should run
  );
```

```
  xTaskCreatePinnedToCore(
    readSensorTask2,    // Task function
    "ReadSensorTask2", // Name of the task
    4096,              // Stack size
    NULL,               // Task input parameter
    1,                  // Priority of the task
    NULL,               // Task handle
    1                   // Core where the task should run
  );

  xTaskCreatePinnedToCore(
    readDcMotorVoltageTask,    // Task function
    "readDcMotorVoltageTask", // Name of the task
    4096,              // Stack size
    NULL,               // Task input parameter
    1,                  // Priority of the task
    NULL,               // Task handle
    1                   // Core where the task should run
  );

  xTaskCreatePinnedToCore(
    readSolarVoltageTask,    // Task function
    "readSolarVoltageTask", // Name of the task
    4096,              // Stack size
    NULL,               // Task input parameter
    1,                  // Priority of the task
    NULL,               // Task handle
    1                   // Core where the task should run
  );
}
```

The main program cycle, the *loop()* function, is responsible for handling OTA updates and maintaining a continuous connection to Adafruit IO.

```
void loop() {
  handleOTA(); // Handle OTA updates
  io.run(); // Maintain connection to Adafruit IO
}
```

The *readDcMotorVoltage()* function is responsible for reading the voltage generated by a direct current motor through a voltage sensor, converting this measurement into microvolts. The data is sent to Adafruit IO after performing multiple readings to reduce error, and a minimum threshold is applied to filter out unwanted noise. Similarly, the *readSolarVoltage()* function is designed to measure the output from a voltage detector connected to a solar panel. As with the previous function, multiple readings are taken to minimize noise and improve data accuracy.

```
void readDcMotorVoltage() {
  // Take multiple readings to reduce noise
  const int numReadings = 1000;
  unsigned long sum = 0;

  for (int i = 0; i < numReadings; i++) {
    int reading = analogRead(DcMotorPin);
    sum += reading;
  }

  float averageReading = sum / (float)numReadings;
  Serial.print("Average ADC reading: ");
  Serial.println(averageReading);  // Display the average value

  // Calculate voltage
  float voltage = (averageReading / 4095.0) * 3.3;  // For 12-bit ADC (0-4095)
  Serial.print("Measured voltage (V): ");
  Serial.println(voltage);  // Display the measured voltage

  // Adjust for voltage divider
  float inputVoltage = voltage / (R2 / (R1 + R2));

  // Convert to microvolts
  float microvoltsMotorDc = inputVoltage * 1000000.0;
  Serial.print("Input voltage (µV): ");
  Serial.println(microvoltsMotorDc);

  // Apply a minimum threshold to filter noise
  const float threshold = 10.0;  // Threshold in microvolts
  if (microvoltsMotorDc < threshold) {
    microvoltsMotorDc = 0;
  }

  // Check if the microvolts value has changed
  if (microvoltsMotorDc != lastMicrovolts) {
```

```
    Serial.print("Measured voltage: ");
    Serial.print(microvoltsMotorDc, 2);  // Display with 2 decimal places
    Serial.println(" µV");

    // Send microvolts to Adafruit IO
    microvoltsMotorDcFeed->save(microvoltsMotorDc);

    // Update the last microvolts value
    lastMicrovolts = microvoltsMotorDc;
  }
}

void readSolarVoltage() {
  // Take multiple readings to reduce noise
  const int numReadings = 1000;
  unsigned long sum = 0;

  for (int i = 0; i < numReadings; i++) {
    int reading = analogRead(SolarPin);
    sum += reading;
  }

  float averageReading = sum / (float)numReadings;
  Serial.print("Average ADC reading: ");
  Serial.println(averageReading);  // Display the average value

  // Calculate voltage
  float voltage = (averageReading / 4095.0) * 3.3;  // For 12-bit ADC (0-4095)
  Serial.print("Measured voltage (V): ");
  Serial.println(voltage);  // Display the measured voltage

  // Adjust for voltage divider
  float inputVoltage = voltage / (R2 / (R1 + R2));

  // Convert to microvolts
  float microvoltsSolar = inputVoltage * 1000000.0;
  Serial.print("Input voltage (µV): ");
  Serial.println(microvoltsSolar);

  // Apply a minimum threshold to filter noise
  const float threshold = 10.0;  // Threshold in microvolts
  if (microvoltsSolar < threshold) {
    microvoltsSolar = 0;
  }
```

```
  // Check if the microvolts value has changed
  if (microvoltsSolar != lastMicrovolts) {
    Serial.print("Measured voltage: ");
    Serial.print(microvoltsSolar, 2);  // Display with 2 decimal places
    Serial.println(" µV");

    // Send microvolts to Adafruit IO
    microvoltsSolarFeed->save(microvoltsSolar);

    // Update the last microvolts value
    lastMicrovolts = microvoltsSolar;
  }
}
```

The program utilizes several FreeRTOS tasks. The *readSensorTask1()* task is executed periodically to read and process temperature and humidity data from the DHT sensor. These data are sent to Adafruit IO, and based on the measured temperature, the LEDs are controlled accordingly. Meanwhile, the *readSensorTask2()* task reads values from two LDRs (light-dependent resistors), calculates the light difference between them, and adjusts the servo motor accordingly. It also sends the average LDR values to Adafruit IO. Another task, *readDcMotorVoltageTask()*, is dedicated to periodically reading and processing the motor voltage and sending this data to Adafruit IO. Finally, *readSolarVoltageTask()* measures the voltage from the detector and updates it on the platform along with the rest of the information.

```
void readSensorTask1(void * parameter) {
  for (;;) {
    Serial.println("----------------------- Measured Temperature / Humidity -------
----------------  ");
    updateSensorData(); // Read and process sensor data
    vTaskDelay(SENSOR_INTERVAL_5_MIN / portTICK_PERIOD_MS); // Delay for 5 minutes
  }
}

void readSensorTask2(void * parameter) {
  for (;;) {
    Serial.println("---------------------------- Measured LDR and Servo ----------
-------------------  ");
    updateLDRAndServo(); // Read LDR values and control servo
    vTaskDelay(SENSOR_INTERVAL_10_SEC / portTICK_PERIOD_MS); // Delay for 10
seconds
```

```
  }
}

void readSolarVoltageTask(void * parameter) {
  for (;;) {
    Serial.println("----------------------------- Measured Solar (V) --------------
--------------  ");
    readSolarVoltage(); // Read and process voltage
    vTaskDelay(SENSOR_INTERVAL_10_SEC / portTICK_PERIOD_MS); // Delay for 10
seconds
  }
}

void readDcMotorVoltageTask(void * parameter) {
  for (;;) {
    Serial.println("----------------------------- Measured DC Motor (V) -----------
-----------------  ");
    readDcMotorVoltage(); // Read and process voltage
    vTaskDelay(SENSOR_INTERVAL_5_MIN / portTICK_PERIOD_MS); // Delay for 10 seconds
  }
}
```

Regarding connectivity, the *setupWiFi()* function connects the ESP32 to a WiFi network and maintains the connection with Adafruit IO. For remote updates, the *setupOTA()* function initializes the OTA update system using the ArduinoOTA library, defining the handlers to manage these events. The *handleOTA()* function is responsible for managing OTA updates by invoking *ArduinoOTA.handle()*.

```
void setupWiFi() {
  int retries = 0;
  const int maxRetries = 5;

  WiFi.begin(WIFI_SSID, WIFI_PASS);
  Serial.print("Connecting to WiFi");

  while (WiFi.status() != WL_CONNECTED && retries < maxRetries) {
    delay(1000);
    Serial.print(".");
    retries++;
  }

  if (WiFi.status() == WL_CONNECTED) {
    Serial.println("\nConnected to WiFi");
```

```
    // Send microvolts to Adafruit IO
    wifiFeed->save(WIFI_SSID);
  } else {
    Serial.println("\nWiFi connection failed. Restarting...");
    ESP.restart();
  }

  connectToAdafruitIO();
}

void handleOTA() {
  ArduinoOTA.handle(); // Handle OTA updates

  //check if there are new client
  // Verificar si hay un nuevo cliente telnet
  if (telnetServer.hasClient()) {
    if (!telnetClient || !telnetClient.connected()) {
      if (telnetClient) telnetClient.stop();
      telnetClient = telnetServer.available();
      Serial.println("New Telnet client connected");
    } else {
      telnetServer.available().stop();
    }
  }

  //Send data to the telnet client server
  if (telnetClient && telnetClient.connected()) {
    while (Serial.available()) {
      telnetClient.write(Serial.read());
    }
    while (telnetClient.available()) {
      Serial.write(telnetClient.read());
    }
  }

}

void setupOTA() {
  // Start telnet server
  telnetServer.begin();
  telnetServer.setNoDelay(true);
  // Initialize ArduinoOTA for over-the-air updates
  ArduinoOTA.setHostname("ESP32-OTA");
  ArduinoOTA.setPassword("admin"); // Uncomment to set a password for OTA updates
```

```
  // Define OTA event handlers
  ArduinoOTA.onStart([]() {
    String type;
    if (ArduinoOTA.getCommand() == U_FLASH) {
      type = "sketch";
    } else { // U_SPIFFS
      type = "filesystem";
    }
    Serial.println("Start updating " + type);
  });
  ArduinoOTA.onEnd([]() {
    Serial.println("\nOTA Update Finished");
  });
  ArduinoOTA.onProgress([](unsigned int progress, unsigned int total) {
    Serial.printf("Progress: %u%%\r", (progress / (total / 100)));
  });
  ArduinoOTA.onError([](ota_error_t error) {
    Serial.printf("Error[%u]: ", error);
    if (error == OTA_AUTH_ERROR) {
      Serial.println("Auth Failed");
    } else if (error == OTA_BEGIN_ERROR) {
      Serial.println("Begin Failed");
    } else if (error == OTA_CONNECT_ERROR) {
      Serial.println("Connect Failed");
    } else if (error == OTA_RECEIVE_ERROR) {
      Serial.println("Receive Failed");
    } else if (error == OTA_END_ERROR) {
      Serial.println("End Failed");
    }
  });
  ArduinoOTA.begin(); // Start the OTA service
  Serial.println("OTA Ready");
}
```

Within the code itself, there are constant variables defined that contain the credentials for the Wi-Fi network to be used and the corresponding keys for the Adafruit IO platform. You can modify the values of these variables from line 30 onward, as shown below, if needed.

```
// Set up your wifi Credencial
#define WIFI_SSID "wifi_name"
#define WIFI_PASS "Wifi_Password"
```

```
// Set up your API key for Adafruit IO
#define AIO_USERNAME "Adafuit_name"
#define AIO_KEY "Adafruit_Password"
```

The DHT sensor is initialized in the *setupDHT()* function, and the pins for the LEDs and the servo motor are configured in the *setupLEDsAndServo()* function, which also connects the motor to its corresponding pin.

```
void setupDHT() {
  dht.begin(); // Start the DHT sensor
}
```

Finally, there are two key functions for data updates: *updateSensorData()* reads the temperature and humidity values from the DHT sensor, sends this information to Adafruit IO, and controls the LEDs based on the temperature. The *updateLDRAndServo()* function reads the values from two LDRs, adjusts the servo motor according to the difference in light, and sends the average LDR values to Adafruit IO.

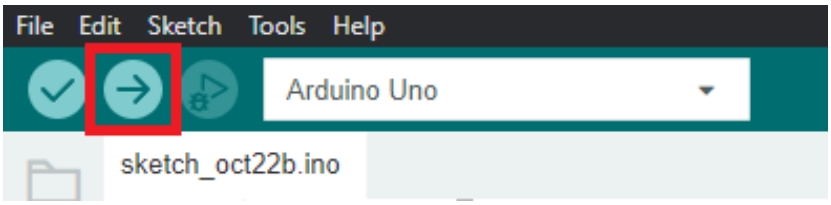
```
void updateSensorData() {
  // Read values from DHT sensor
  float h = dht.readHumidity();
  float t = dht.readTemperature();

  // Check if readings are valid
  if (isnan(h) || isnan(t)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  // Send values to Adafruit IO
  temperature->save(t);
  humidity->save(h);

  // Print sensor values
  Serial.print("Temperature: ");
  Serial.print(t);
  Serial.print(" °C, Humidity: ");
  Serial.print(h);
  Serial.println(" %");

  // Example LED control based on temperature
```

```
  if (t < 20) {
    digitalWrite(ledPin1Cold, HIGH);
    digitalWrite(ledPin2Good, LOW);
    digitalWrite(ledPin3Heat, LOW);
     // Send to the corresponding feed (Cold LED)
    ledColdFeed->save(1);  // 1 means the LED is ON
    ledGoodFeed->save(0);  // OFF
    ledHighFeed->save(0);  // OFF
    Serial.println("Cold is HIGH");

  } else if (t >= 20 && t < 25) {
    digitalWrite(ledPin1Cold, LOW);
    digitalWrite(ledPin2Good, HIGH);
    digitalWrite(ledPin3Heat, LOW);
    // Send to the corresponding feed (Good LED)
    ledColdFeed->save(0);  // OFF
    ledGoodFeed->save(1);  // ON
    ledHighFeed->save(0);  // OFF
       Serial.println("Good is HIGH");
  } else {
    digitalWrite(ledPin1Cold, LOW);
    digitalWrite(ledPin2Good, LOW);
    digitalWrite(ledPin3Heat, HIGH);
    // Send to the corresponding feed (Heat LED)
    ledColdFeed->save(0);  // OFF
    ledGoodFeed->save(0);  // OFF
    ledHighFeed->save(1);  // ON
    Serial.println("Heat is HIGH");
  }
}

void updateLDRAndServo() {
  // Read LDR values
  ldrValue1 = analogRead(ldrPin1) ;
  ldrValue2 = analogRead(ldrPin2)- OFFSET;

  // Invert readings so low values mean darkness and high values mean light
  ldrValue1 = 4095 - ldrValue1;
  ldrValue2 = 4095 - ldrValue2;
  averageLdrValue = (ldrValue1 + ldrValue2) / 2;

  // Print readings for debugging
  Serial.print("LDR 1: ");
  Serial.print(ldrValue1);
```

```
    Serial.print(" | LDR 2: ");
    Serial.println(ldrValue2);
    Serial.print("Average LDR: ");
    Serial.println(averageLdrValue);

    // Send the average value to Adafruit IO
    ldrFeed->save(averageLdrValue);
    ldr1Feed-> save (ldrValue1);
    ldr2Feed -> save (ldrValue2);

    // Compare LDR values and move the servo based on the difference
    if (ldrValue1 > ldrValue2 + tolerance) {
      if (servoPos > 0) {
        servoPos -= servoStep;  // Move the servo to the left
        Serial.println("Moving left");
      }
    } else if (ldrValue2 > ldrValue1 + tolerance) {
      if (servoPos < 180) {
        servoPos += servoStep;  // Move the servo to the right
        Serial.println("Moving right");
      }
    }

    // Ensure the servo stays within the [0, 180] range
    servoPos = constrain(servoPos, 0, 180);

    // Move the servo to the new position only if it has changed
    myServo.write(servoPos);

    // Print the servo angle
    Serial.print("Servo angle: ");
    Serial.println(servoPos);

    // Pause to give the servo time to move
    delay(30);  // Increase the delay for more response time for the servo
}
```

3. 5. Programming the Microcontroller

To program the ESP32 board with the provided code, the first step is to have the ESP32 connected to the computer using a Micro USB to USB Type-A cable. Then, as previously mentioned, you will select the corresponding COM port for your board and use the Upload button in the Arduino IDE, which compiles the program before sending it to the programmable board. Just to the left of this button, with the tick symbol, is the Verify button, which only compiles the code. Compilation converts the written code, in a programming language like C/C++, into a format the microcontroller can understand and execute, while checking for syntax errors or issues that could prevent the program from running properly during the process.



If you encounter any errors related to the uploading process or loading the code onto the board, you can check the following:

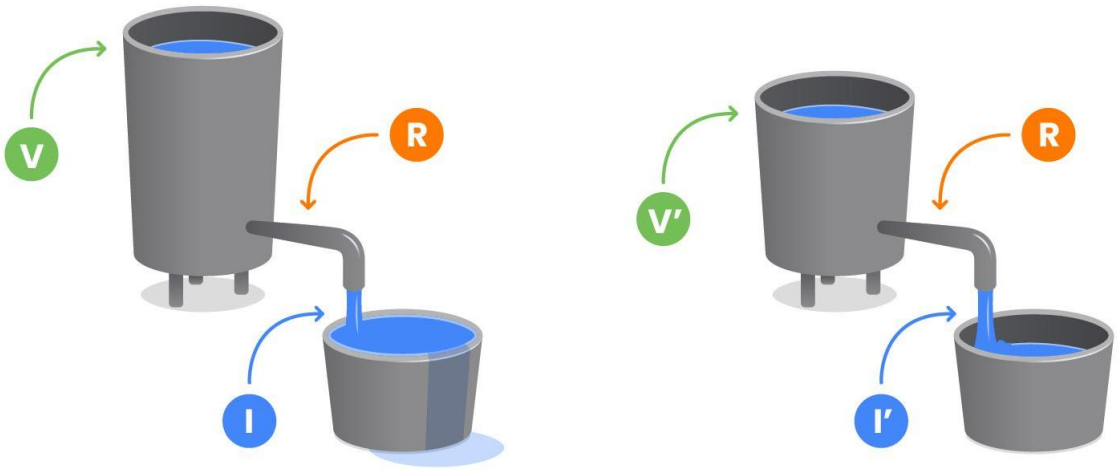
- Check that the USB cable is properly connected at both ends.
- Try using a different USB cable, as some cables are only for charging and do not transmit data.
- Try using a different USB port on your computer.
- Ensure that you have selected the correct COM port.
- Some ESP32 models require manual programming mode activation by pressing the BOOT button as the code upload process begins.
- Verify that the necessary driver for programming the ESP32 (CP210x) is correctly installed.

4. Introduction to Electronics

Electronics is a branch of physics and engineering that focuses on the study, design, and application of devices and systems that operate by controlling electrons and other electrically charged particles. Through electronics, circuits are built that enable signal manipulation, facilitating the creation of essential technologies such as computers, phones, cars, and a wide variety of everyday devices.

4. 1. Ohm's Laws

In the field of electronics and electricity, there is a key law known as **Ohm's Law**, which defines the relationship between the three most important quantities in a circuit: **voltage (tension), current (intensity), and resistance**. Understanding the meaning of these quantities and how they relate to each other may seem complex at first, but it becomes much easier when comparing an electrical circuit to a water system, using an analogy to explain its operation.



- **Voltage (Tension):** It is the water pressure in the pipes. Voltage is what "pushes" electrons (the current) to move through the circuit. It is measured in **volts (V)**.

- **Current (Intensity):** It is the flow of water through the pipes. In an electrical circuit, it is the flow of electrons driven by the voltage. It is measured in **amperes (A)**.

- **Resistance:** It is the size or narrowing of the pipe. Resistance is what "hinders" or "slows down" the flow of current. If the resistance is high, less current can pass through. It is measured in **ohms (Ω)**.

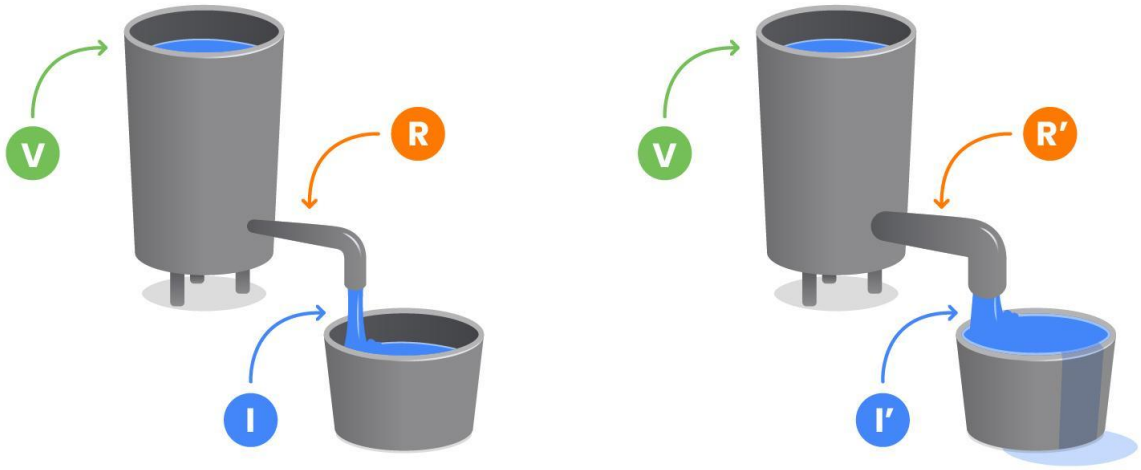
Ohm's Law states that the relationship between these three quantities follows a simple rule:

$$V = I \cdot R$$

Or in other forms:

- If you increase the voltage (the pressure), more current will flow.

- If you increase the resistance (narrow the pipe), less current will pass, unless you also increase the voltage.



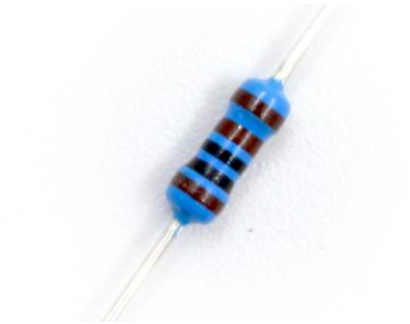
4. 2. Electronic Components

Electronic components are the basic building blocks that make up electrical and electronic circuits. Each one performs a specific function within the circuit, allowing control and manipulation of the flow of electric current. Among the most common components are resistors, which limit current flow; capacitors, which store and release energy; diodes, which allow current to flow in only one direction; and transistors, which act as switches or amplifiers. These elements, when combined in various configurations, enable the operation of the electronic devices we use daily.

4. 2. 1. Basic Components

- Resistors

In electronics, a resistor (R) is a component whose function is to limit the flow of electric current within a circuit. Its purpose is to regulate the amount of current (I) flowing through other components, protecting them and ensuring the proper functioning of the circuit.



The value of the resistor is indicated by color bands printed on the component itself. These color combinations allow you to identify the resistance value in ohms, following the pattern established in the following table:

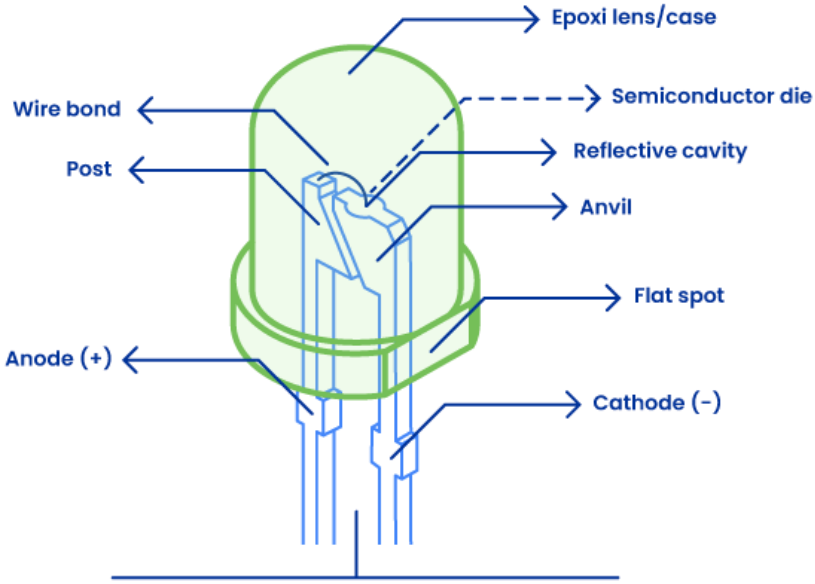
COLOR	BAND 1	BAND 2	BAND 3	MULTIPLIER	TOLERANCE
Black	0	0	0	1 Ω	-
Brown	1	1	1	10 Ω	± 1%
Red	2	2	2	100 Ω	± 2%
Orange	3	3	3	1 kΩ	-
Yellow	4	4	4	10 kΩ	-
Green	5	5	5	100 kΩ	± 0.5%
Blue	6	6	6	1 MΩ	± 0.25%
Purple	7	7	7	10 MΩ	± 0.10%
Gray	8	8	8	-	± 0.05%
White	9	9	9	-	-
Gold	-	-	-	-	± 5%
Silver	-	-	-	-	± 10%

Keep in mind that there are resistors with 3 color bands (Band 1 and Multiplier, in addition to Tolerance), others with 4 color bands (Band 1, Band 2, and Multiplier, along with Tolerance), and even some with 5 bands. The Tolerance refers to the margin of error between the actual resistance value and the value indicated by the color bands.

- Light Emitting Diodes (LEDs)

An LED (Light Emitting Diode) is a type of diode with the unique characteristic of being able to emit light. A diode is an electronic component that allows current to flow in only one direction.

It consists internally of a semiconductor material with a p-n junction. Therefore, when it is forward biased, meaning the anode has a higher voltage than the cathode, the diode conducts electricity; when it is reverse biased, it blocks the flow of current. This operating principle of the diode makes it somewhat similar to a voltage-controlled switch. The LED indicates the polarity of the connection at a glance by the size of its connection leads. The longer lead corresponds to the anode (positive), while the shorter lead is the cathode (negative). In the case of the LED, when it conducts electricity, its semiconductor material releases energy in the form of photons, which causes it to emit light.



When an LED conducts electricity, it offers very little resistance to the flow of current, which means the power supply will provide as much current as possible uncontrollably, effectively creating a short circuit that poses a **significant risk of burning out the LED**. To prevent this, we should always connect a resistor of appropriate value (typically 220 ohms or higher) in series with the LED. This way, we can control the current flow and avoid damaging the LED.

- Capacitors

A capacitor is an electronic component that stores energy in the form of electric charge. It consists of two conductive plates separated by an insulating material called a dielectric. When a voltage is applied across the plates, charge accumulates on them, creating an electric field.

Capacitors are used in various applications, such as in filters, power supplies, timing circuits, and for smoothing electrical signals. Their ability to store and release energy quickly makes them essential in the design of electronic circuits. Capacitors are measured in farads (F), although smaller units like microfarads (μF) and nanofarads (nF) are commonly used.

4. 2. 2. Powering a Circuit

Powering an electronic circuit refers to the process of providing the necessary energy for the circuit to function correctly. This energy can come from various sources, such as batteries, power adapters, solar panels, or mains power supplies. The key aspects related to powering an electronic circuit are:

- **Voltage (Tension):** It is the force that drives electrons through the circuit. Each electronic component has a specific voltage range it requires to operate. For example, some microcontrollers operate at 5V, while others may need 3.3V.

- **Current (Amperage):** It is the amount of electrons flowing through the circuit. The power supply must be capable of providing sufficient current to meet the needs of all the components in the circuit.

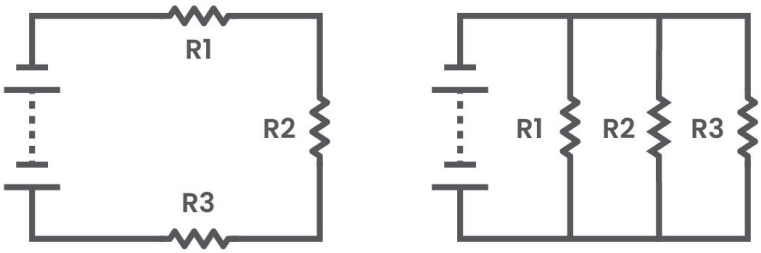
- **Stability:** The power supply must be stable and constant to avoid fluctuations that could damage electronic components or affect their performance. This is especially important in sensitive circuits.

- **Regulation:** Some circuits require a specific and constant voltage, so voltage regulators are used to maintain the voltage at the appropriate level, regardless of variations in load or power supply.

- **Types of Power Supply:** Circuits can be powered by direct current (DC), such as that provided by batteries, or alternating current (AC), like that obtained from the electrical grid. Some circuits may require a converter to transform alternating current into direct current.

4. 2. 3. Configuration of Components in a Circuit

In electronics, components in a circuit can be connected in various ways, and each configuration affects the behavior of the circuit. The most common configurations are **series**, **parallel**, and **mixed**.



Series circuit **Parallel circuit**

In a **series** configuration, components are connected one after another, forming a single path for the current. This means that the same current flows through all components.

In a **parallel configuration**, components are connected so that each one forms its own path for the current.

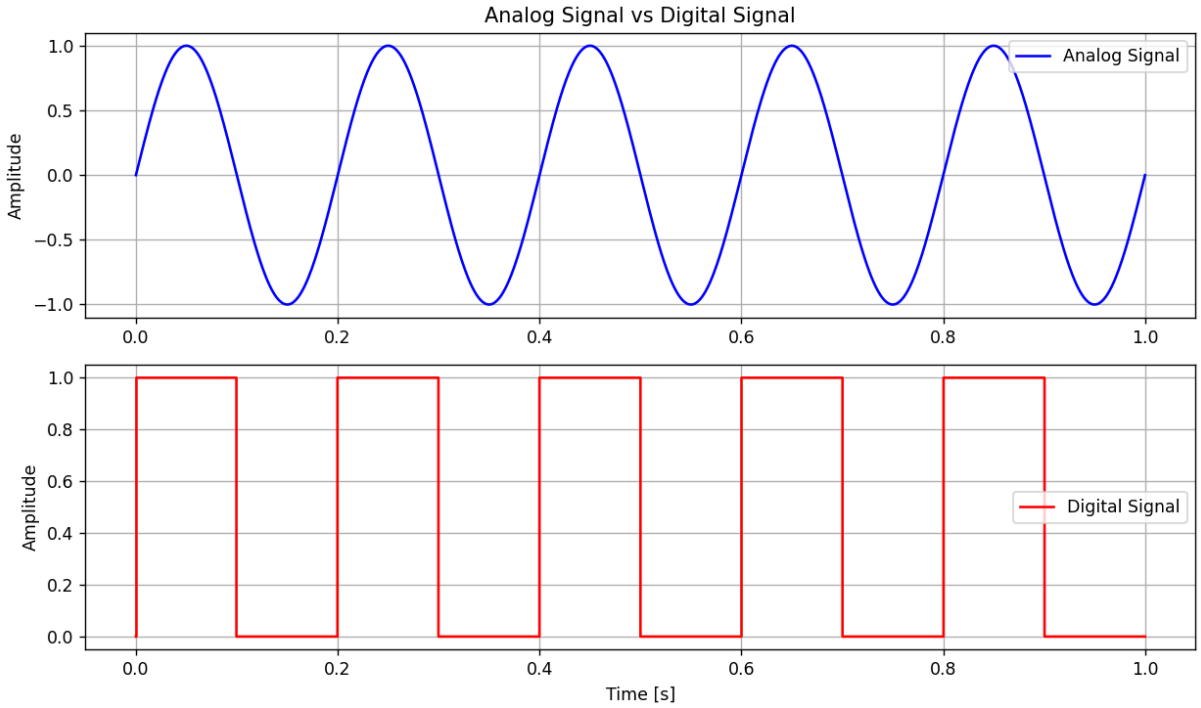
In a **mixed configuration**, components are connected in a combination of series and parallel. This setup allows for greater flexibility in circuit design and is used to meet specific voltage and current requirements.

4. 2. 4. Sensors

A sensor is a device that detects changes in the environment and converts that information into an electrical signal or data that can be interpreted by an electronic system. Sensors are essential in a wide range of applications, from automation and robotics to measurement and monitoring devices.



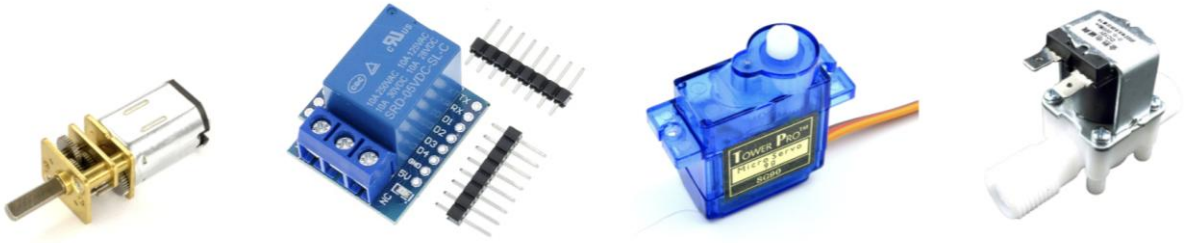

Sensors are capable of detecting different types of stimuli, such as light, temperature, pressure, humidity, motion, sound, and more. Each type of sensor is designed to measure a specific magnitude, acting as transducers, meaning they convert one form of energy (such as thermal, mechanical, or luminous energy) into an electrical signal that can be processed by other devices, like microcontrollers. The generated signal can be either analog (a continuous variation, such as a voltage) or digital (a discrete value, such as "yes" or "no").



Sensors are used in a wide range of applications, such as in home automation to control the climate of a house, in cars for security systems, in wearable devices to monitor health, and in industry for process control. For example, a DHT-11 or DHT-22 is a type of temperature sensor that provides measured values with a digital signal, while a photoresistor is a type of sensor that varies its electrical resistance based on the amount of light incident upon it, allowing for an analog signal to be obtained.

4. 2. 5. Actuators

An actuator is a device that converts a control electrical signal into movement or a physical action within a system. Actuators work in conjunction with sensors in control systems. While sensors detect and measure changes, actuators respond to that information to perform actions and modify the environment or the controlled system through signals that can be either analog or digital, sent by a control system such as a microcontroller or a computer. These signals indicate to the actuator what task it should perform, whether it's opening a valve, moving a robotic arm, or rotating a motor.



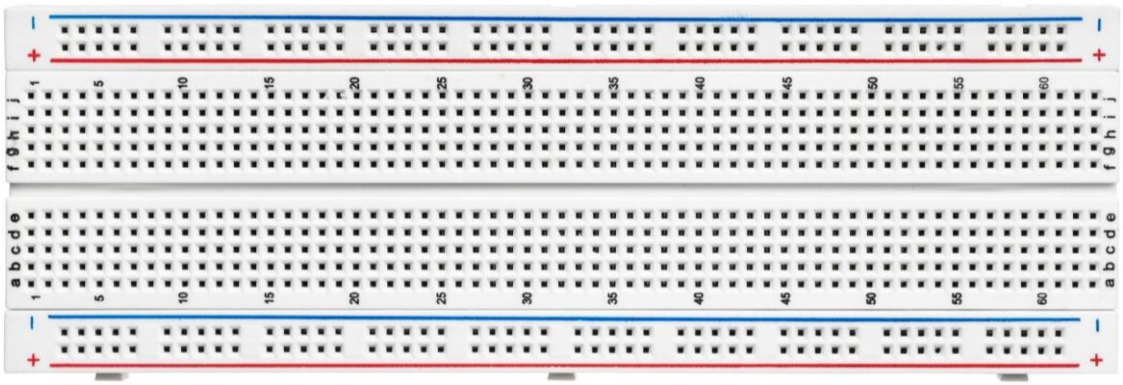
There are different types of actuators:

- **Electric Actuators:** They use electrical energy to generate movement, such as electric motors, solenoids, and servomotors.
- **Hydraulic Actuators:** They operate using fluid pressure to move objects, commonly used in systems requiring significant force, such as heavy machinery.
- **Pneumatic Actuators:** They employ compressed air to generate movement, often found in industrial applications and automation systems.

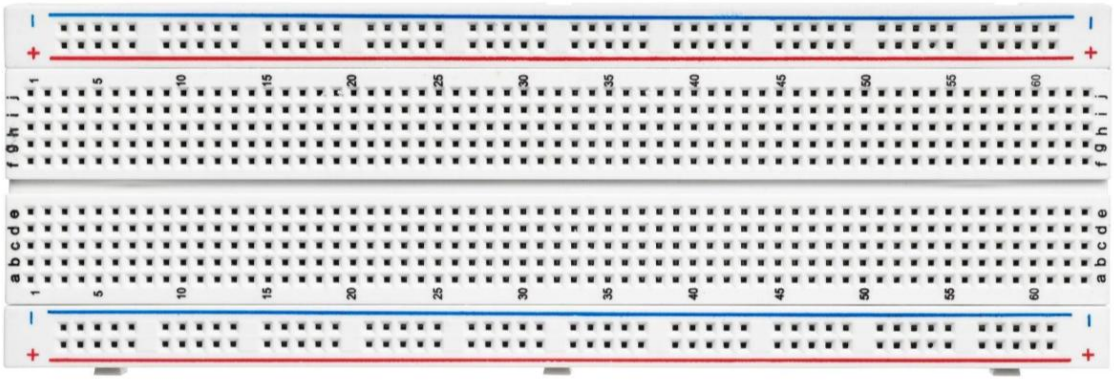
Actuators are essential in a wide variety of applications. In robotics, for example, they enable the movement of arms or wheels, while in home automation, they control elements such as blinds or automatic doors. They are also present in automobiles, where they manage the movement of seats or the opening of valves.

4. 3. Breadboard

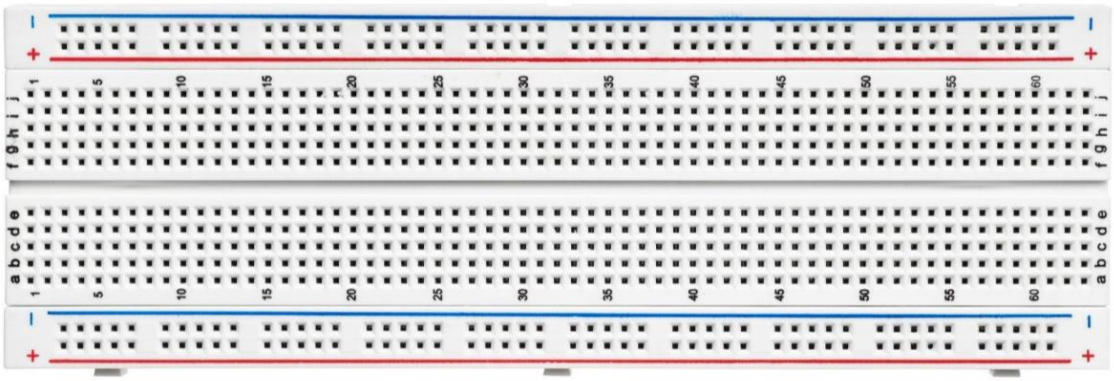
A breadboard is a type of "workspace" for creating electronic circuits. It is a board filled with small holes where you can insert various components such as resistors, LEDs, wires, etc. These holes are internally connected in such a way that when a component is placed in one of them, it automatically connects with other holes following an established pattern.



The breadboard allows you to experiment with different circuit configurations without the need to solder the components permanently. You can easily move, replace, or remove elements without causing any damage.



The image shows how the holes in a breadboard are internally connected. The copper strips inside allow certain groups of holes to be connected to each other. In this case, the horizontal lines, which are typically the power rails located along the edges of the board, are electrically connected along their entire length. This means that any component or wire inserted into those holes will share the same power or ground connection.

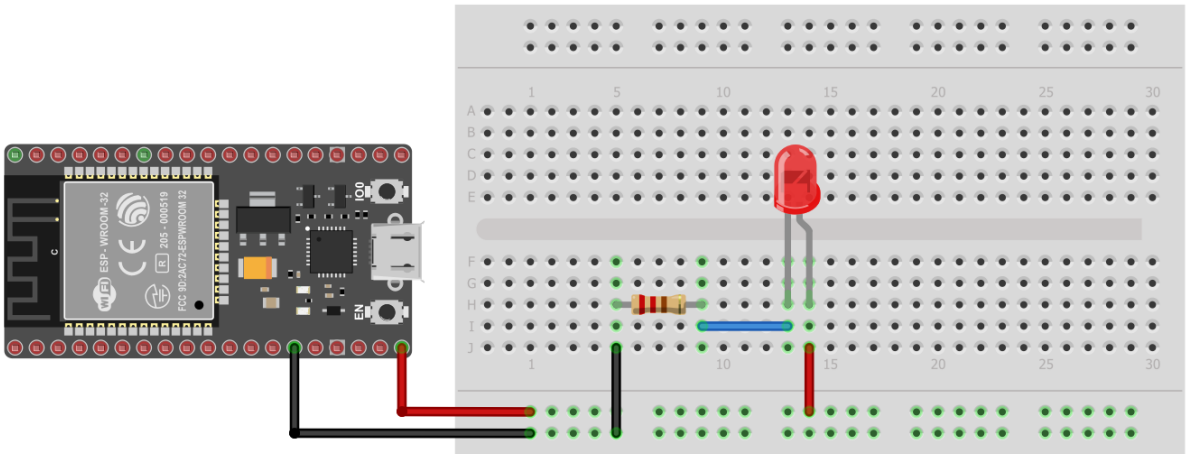


On the other hand, the groups of vertical lines found in the central area of the breadboard are also electrically connected to each other. These connection groups are isolated from one another, but within each group, the components you place will share the same connection, making it easier to create complex circuits without the need for soldering. This internal connection design allows for easy manipulation and modification of experimental circuits on the breadboard.

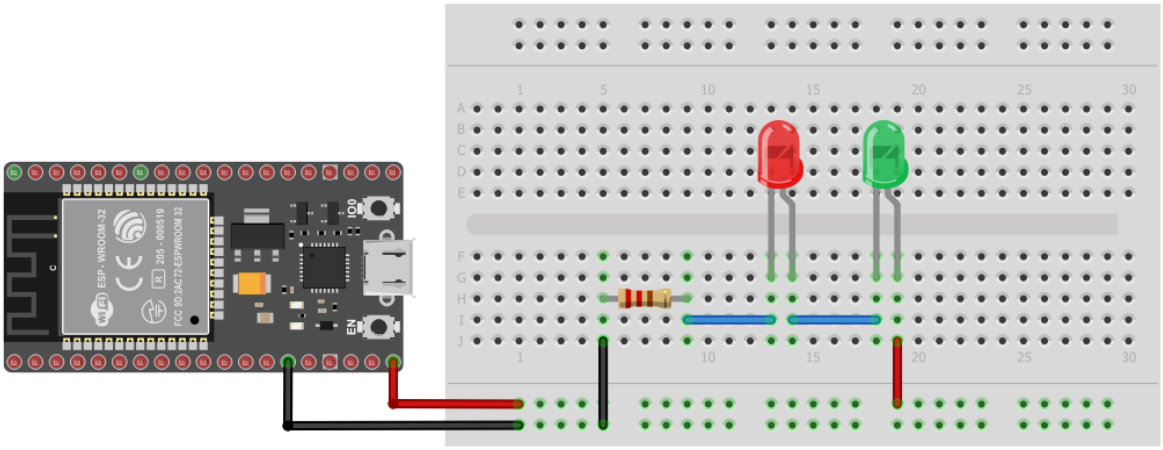
4. 3. 1. Activities with a Breadboard

In this first activity, the objective is to learn how to use the breadboard and understand the flow of current in a simple circuit. To achieve this, you can use the VIN and GND pins to power the circuit on the breadboard with the side power rails, noting that VIN is the positive terminal and GND is the negative. You will then need an LED as a visual indicator of power and a resistor to limit the current. Connect the LED in series with the resistor in any area of the breadboard, ensuring that the LED diode is connected in the correct orientation. Once the circuit is complete and the ESP32 is connected to the computer, the LED should light up, indicating that current is flowing correctly.

As a review element, could you tell me the value of the resistor used in the schematic?

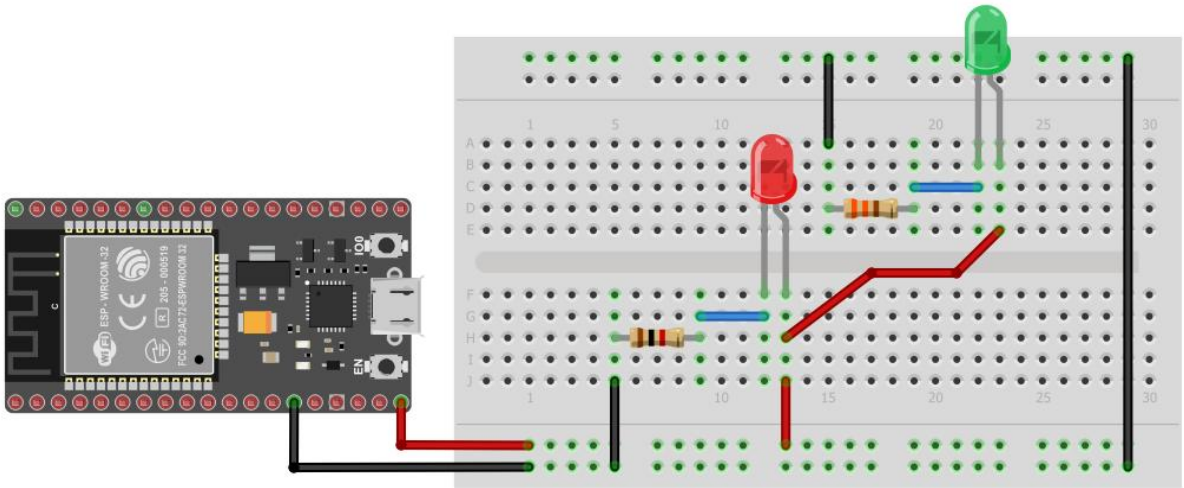


In this next activity, the goal is to connect two LEDs in series along with their current-limiting resistor. Connect the negative terminal of one LED to the positive terminal of the next, and then place the resistor, connecting both ends of the circuit to the power rails. Be sure to pay attention to the orientation of both LEDs.



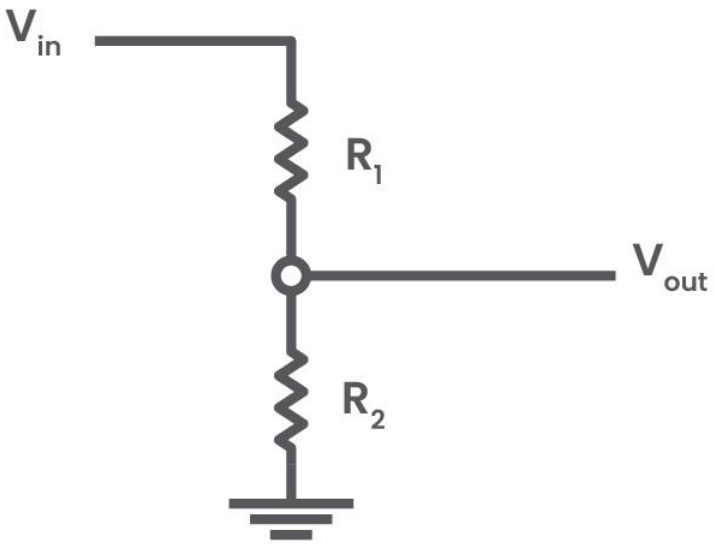
To continue practicing with the breadboard, we will set up the same two LEDs from the previous exercise, but in parallel, each with different values of current-limiting resistors.

Could you tell me the values of the resistors used this time by looking at the schematic?



4. 3. 2. Voltage Divider

A voltage divider is a very simple and useful circuit that allows you to obtain a fraction of the input voltage using two resistors connected in series. It is commonly used in applications where it is necessary to reduce a voltage to a level that can be used by other components.



When two resistors are connected in series to a voltage source, the current that flows through them is the same, but the voltage is divided between the resistors according to their values. The voltage divider takes advantage of this property to "divide" the input voltage into a smaller portion. The voltage at the output of the divider can be calculated using the following expression:

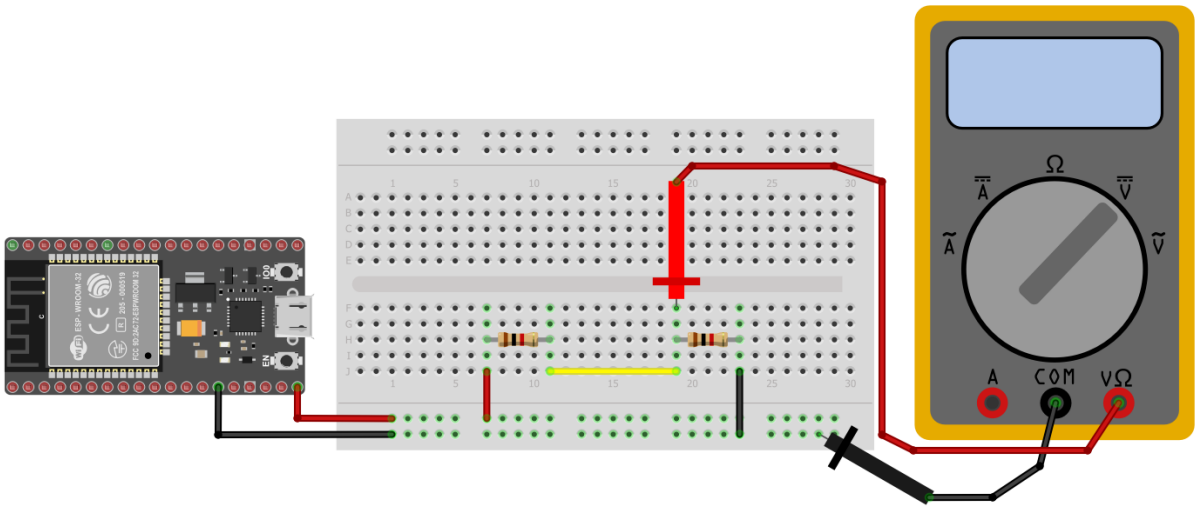
$$V_{out} = \frac{R_2}{R_1 + R_2} * V_{in}$$

If you have a 5V source and use two resistors in series of 1 kΩ (R1) and 2 kΩ (R2), the output voltage of the voltage divider can be calculated as follows:

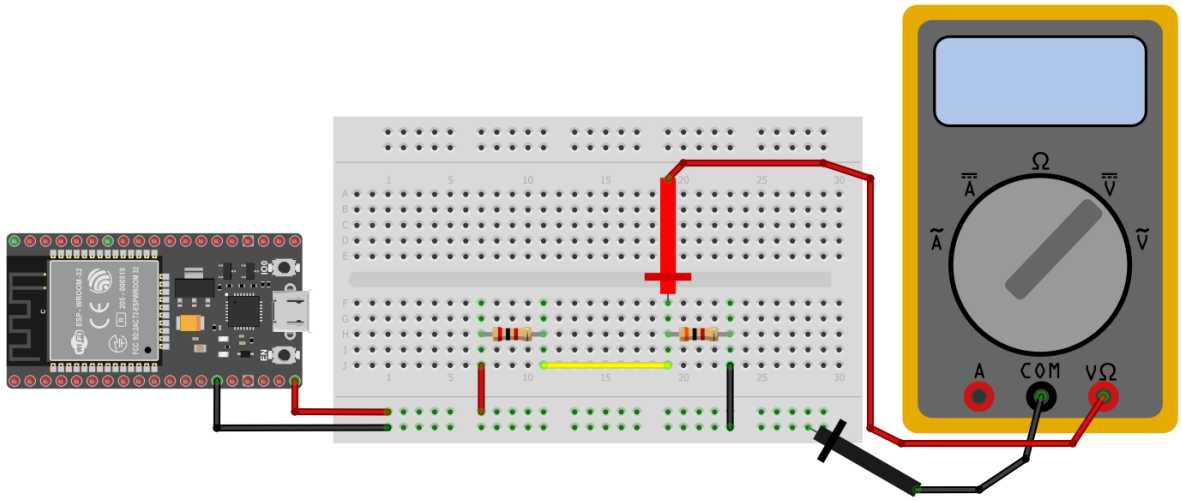
$$V_{out} = 5 * \frac{2k\Omega}{2k\Omega + 1k\Omega} = 3{,}33V$$

4. 3. 3. Activities with Voltage Dividers

To practice the concept of a voltage divider, set up two 1 kΩ resistors in series on the breadboard and use the ESP32 again to power the circuit (remember that VIN is the positive supply and GND is the negative), as shown in the following schematic. Measure the voltage at the midpoint between the two resistors and verify that it matches the value calculated using the voltage divider formula.



Now create a circuit with a voltage divider using a 2 kΩ resistor (R1) and a 3 kΩ resistor (R2) connected in series to the ESP32's power supply. Calculate the output voltage of the divider and verify its accuracy by measuring the voltage at the indicated point. Below is the schematic for the voltage divider for the exercise.



5. ASSEMBLY OF THE 3-IN-1 DEMONSTRATOR

The assembly of the 3x1 Demonstrator will be carried out modularly, addressing each section separately in the corresponding sections of this guide. This approach aims to simplify the process and enhance understanding, ensuring that the assembly is clear and accessible for those undertaking it. Each step is designed to guarantee a functional and effective outcome, even for individuals with no prior experience in assembling demonstrators.

One of the main goals of this project is to promote the autonomy of educational institutions, enabling them to create their own demonstrators if they have 3D printers on-site. For this reason, we have made the necessary printing files available for the fabrication of the components, which can be downloaded via the following link: 3D Printing Files. With this tool, we aim to empower educational institutions to independently and efficiently integrate these demonstrators into their activities.

5.1. List of Materials

- 1 x **ESP32 DEVKIT V1** with 30 pins
- 2 x **Mini breadboards** with 170 tie-points
- 1 x **DHT11 module**, maximum width 14 mm, with or without built-in LED
- 10/12* x **Male-to-male jumper wires**, 10 cm
- 5 x **Male-to-male jumper wires**, 20 cm
- 3 x **Male-to-female jumper wires**, 10 cm
- 10 x **Male-to-female jumper wires**, 20 cm
- 5 x **1k ohm resistors**
- 1 x **Capacitor**, **100nF** or **1uF**
- 3 x Colored **LEDs** (preferably Red, Green, and Blue)
- 1 x **Solar panel**, maximum dimensions: 45 mm x 60 mm
- 1 x **Water level sensor**, 6 cm x 2 cm
- 1 x 5V **DC motor**, maximum dimensions: 28 mm diameter and 45 mm length
- 1 x **SG-90 Servo motor**
- 1 x **Voltage detector**
- 2 x **Photoresistors** standalone GL5528
- 1 x **6203 bearing**, either ZZ or 2RS type: 40 mm outer diameter, 17 mm inner diameter, and 12 mm height
- 3 x **Wooden dowels**: 8 mm diameter and 40 mm length
- 1 x **Motor gear**: 2 mm inner diameter, 10 teeth, and 5 mm height
- 1 x **Data transfer cable USB to microUSB**
- 1 x **Aluminum tape**, 5 cm wide
- 1 x **Double sided tape**, 3 cm de wide

* If the Motor DC doesn't come with wires, you will need 12 male-to-male jump wires of 10 cm.

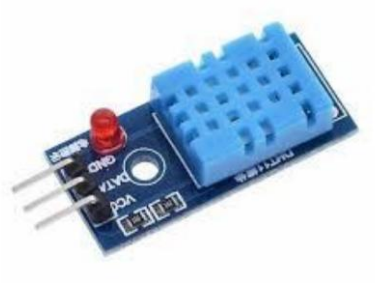
Tools you might need:

- **Soldering iron**: if the Motor DC or the Solar Panel don't come with wires.
- **Flat head screw driver 2 mm**: for the voltaje sensor.

5. 2. Assembly of the Water and Solar Demonstrator

5. 2. 1. Electronic Assembly (Water Component)

We will begin by assembling the water collector. To complete this assembly, you will need the following materials:



1 x DHT-11 Sensor

Responsible for measuring ambient temperature and humidity.

3 x 1 kΩ Resistors

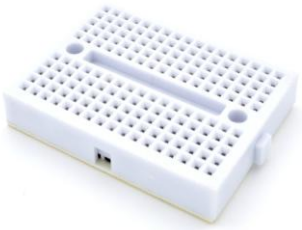
Used to limit the current flowing through LED diodes.

3 x Colored LEDs (Preferably Red, Green, and Blue)

Used as visual indicators of the measured temperature range.

1 x Dupont Cables (Male-Male) 10cm

Necessary for making connections between the different components on the breadboard.



2 x Mini Breadboard

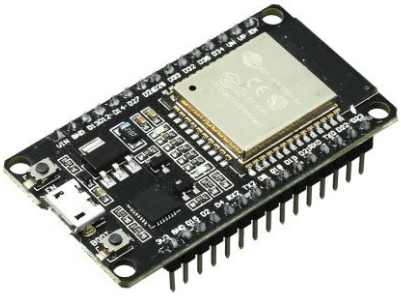
Component necessary to connect the microcontroller board to the different components.

6 x Dupont Cables (Male-Female) 20cm

Necessary for making connections between the different components on the breadboard.

1 x Water level sensor

Used to measure the water level in the tank.

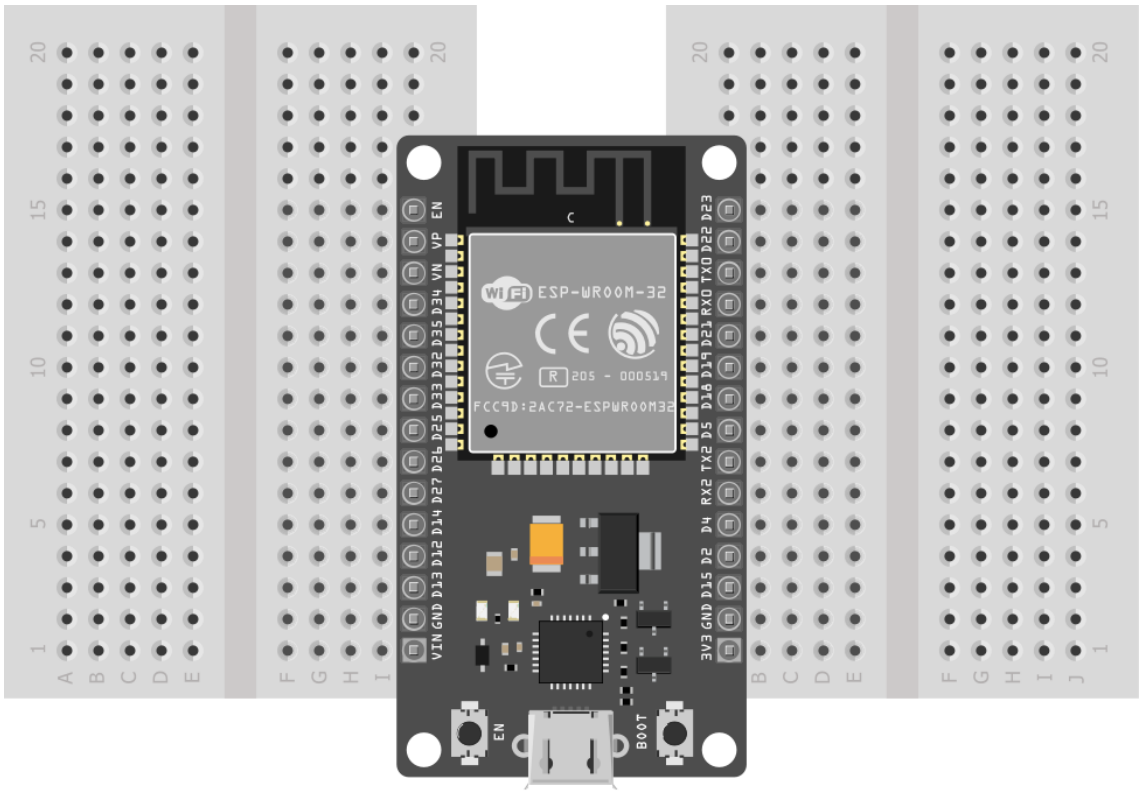


1 x ESP32 DEVKIT V1 (30 pins)

Responsible for executing the programming that controls the operation of the electronics.

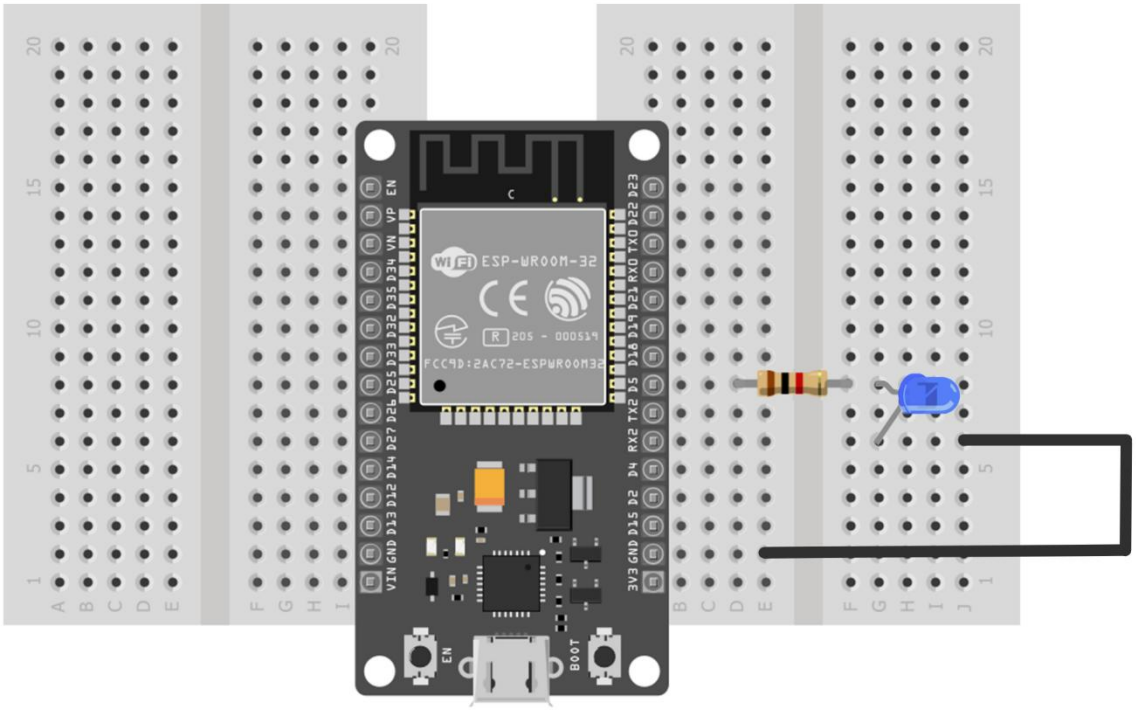
The objective of the assembly is to obtain the values of ambient temperature and humidity, and based on the temperature value, to light up a specific LED as indicators of danger for high or low temperatures. In this way, if the temperature is below 20°C, the Blue LED will light up; if it is above 25°C, the Red LED will light up; and if it is between those values (between 20 and 25°C), the Green LED will light up.

The first step is to place the ESP32 board on two small breadboards to have enough space to connect the components we will need later. Below, you can see the result:



Once we have the microcontroller placed on the two breadboards, we will start connecting the colored LEDs with their current-limiting resistors. We will begin by placing the Blue LED and one of the 1 kΩ resistors in series. We will connect the positive of the LED to **pin D5** and use any GND pin to connect the negative of the circuit. Below, you can see the indicated assembly.

Note: *The LED diode indicates the polarity of the connection at a glance by the length of its leads. The longer lead corresponds to the anode, or positive, and the shorter one is the cathode, or negative. In the diagram, the longer lead is the one connected in series with the resistor.*

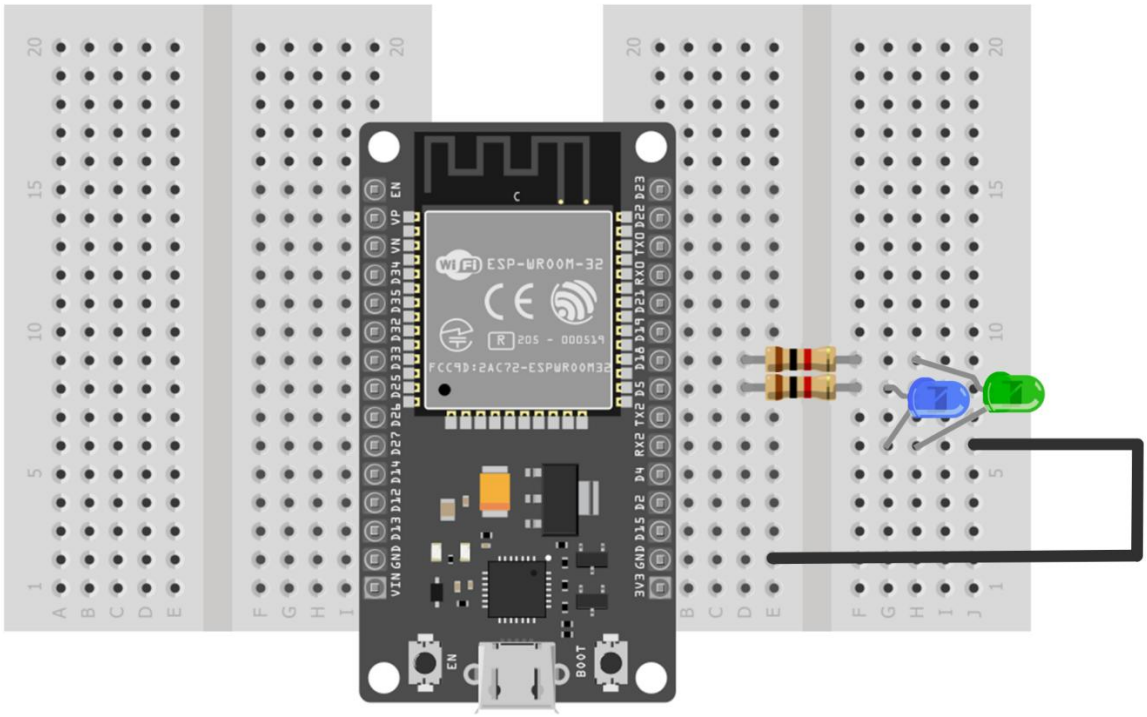


To verify the correct functioning of the assembly, we can upload the following code to the ESP32:

```
#define ledPin1Cold 5 // Define the pin where the Blue LED is connected

// Initialization block
void setup() {

    // We configure the operating mode of the LED pin as an output
    pinMode(ledPin1Cold, OUTPUT);

}

// Main loop of the code
void loop() {

    digitalWrite(ledPin1Cold, LOW); // We turn off the LED
    delay(1000); // We wait 1 second = 1000 milliseconds
    digitalWrite(ledPin1Cold, HIGH); // We turn on the LED
    delay(1000); // We wait 1 second

}
```

This small program makes the Blue LED blink, changing its state between on and off every second. If it does not work correctly, check that the LED is oriented properly (remember that the longer lead is the positive terminal, while the shorter one is the negative) and connected appropriately with the current-limiting resistor.

The next step in the assembly will be to add a second LED. This time, we will place and connect the Green LED to **pin D18** with its resistor. You can see the indicated assembly schematic in the following image.

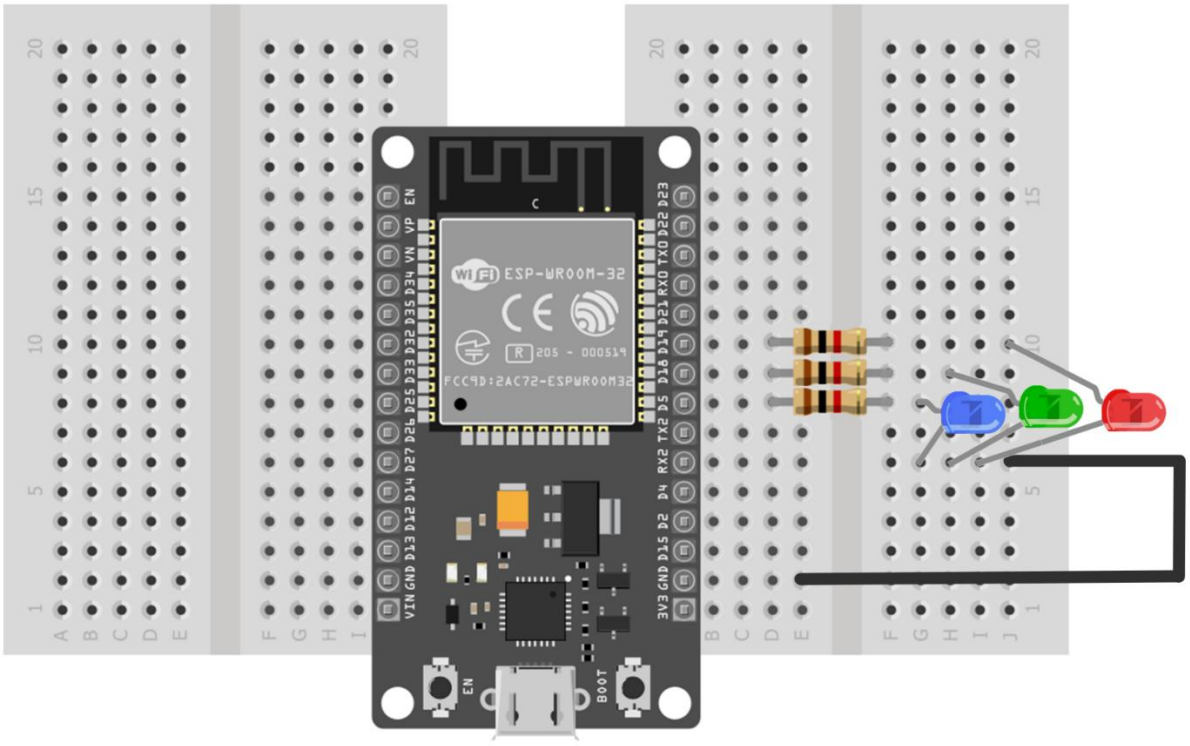


Let's verify that the assembly is correct again. We will use the code below to check that the new LED is also properly placed.

```
#define ledPin1Cold 5 // Define the pin where the Blue LED is connected
#define ledPin2Good 18 // Define the pin where the Green LED is connected

// Initialization block
void setup() {

    // We configure the operating mode of the LEDs pins as outputs
    pinMode(ledPin1Cold, OUTPUT);
    pinMode(ledPin2Good, OUTPUT);

}

// Main loop of the code
void loop() {

    digitalWrite(ledPin1Cold, LOW); // We turn off the Blue LED
    digitalWrite(ledPin2Good, LOW); // We turn off the Green LED
    delay(1000); // We wait 1 second = 1000 milliseconds
    digitalWrite(ledPin1Cold, HIGH); // We turn on the Blue LED
    digitalWrite(ledPin2Good, HIGH); // We turn on the Green LED
    delay(1000); // We wait 1 second

}
```

As you can see, the objective of the code remains the same: to turn both LEDs on and off, changing their state every second. If the Blue and Green LEDs are changing states correctly, we can consider the current assembly valid and proceed to the next step.

To complete the assembly of the colored LEDs needed to fulfill the established objective for this part of the Demonstrator, we need to connect the last LED, which is Red, to **pin D19**. Proceed to assemble the LED diode and the corresponding last resistor, as shown in the following image.

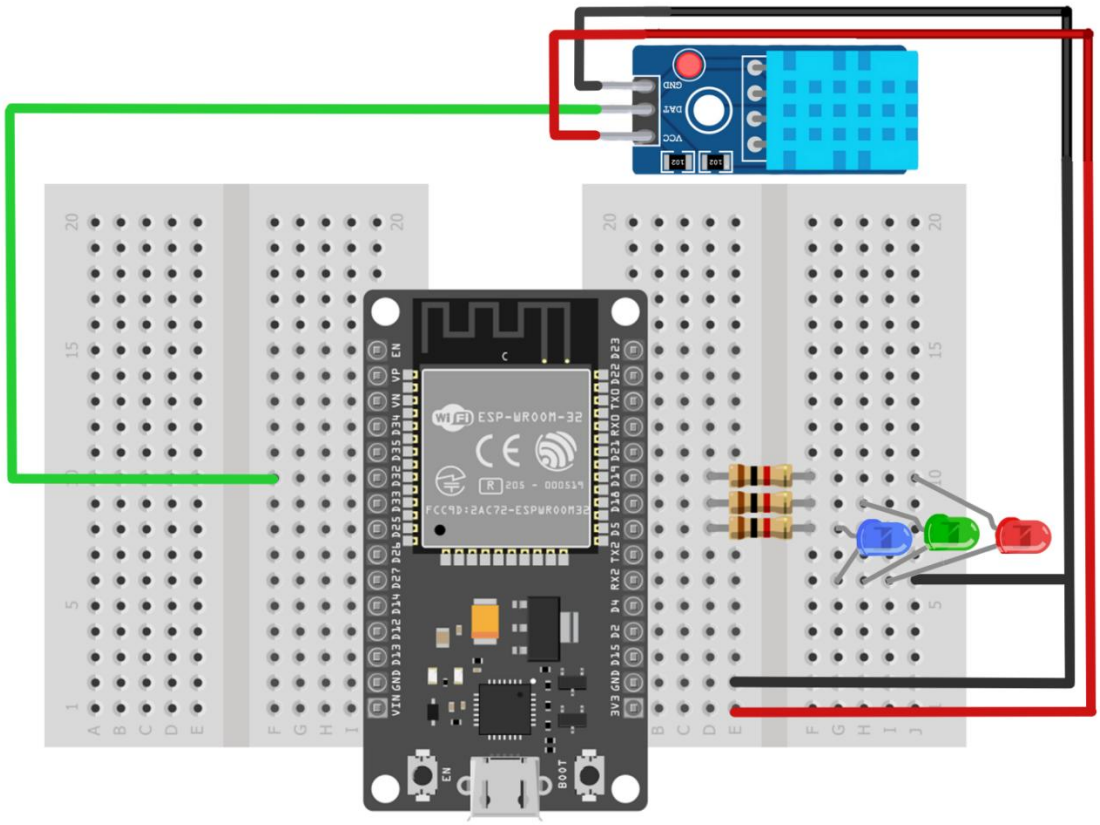


We will check that it is correct by using the same programming as in the previous cases, generating a blink in all three colored LEDs. The code for this is as follows:

```
#define ledPin1Cold 5 // Define the pin where the Blue LED is connected
#define ledPin2Good 18 // Define the pin where the Green LED is connected
#define ledPin3Heat 19 // Define the pin where the Red LED is connected

// Initialization block
void setup() {

    // We configure the operating mode of the LEDs pins as outputs
    pinMode(ledPin1Cold, OUTPUT);
    pinMode(ledPin2Good, OUTPUT);
    pinMode(ledPin3Heat, OUTPUT);

}

// Main loop of the code
void loop() {

    digitalWrite(ledPin1Cold, LOW); // We turn off the Blue LED
    digitalWrite(ledPin2Good, LOW); // We turn off the Green LED
    digitalWrite(ledPin3Heat, LOW); // We turn off the Red LED
    delay(1000); // We wait 1 second = 1000 milliseconds
```

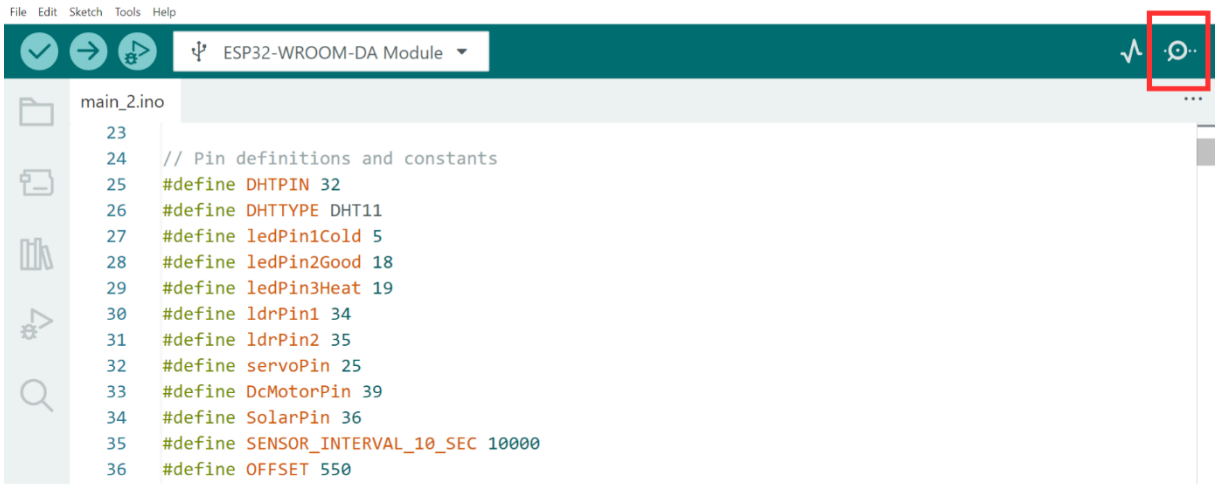
```
    digitalWrite(ledPin1Cold, HIGH); // We turn on the Blue LED
    digitalWrite(ledPin2Good, HIGH); // We turn on the Green LED
    digitalWrite(ledPin3Heat, HIGH); // We turn on the Red LED
    delay(1000); // We wait 1 second

}
```

The next step is to add and connect the DHT11 sensor to **pin D32**, either as an individual component or on a module. Below is the assembly completed so far along with the connections for the DHT11 sensor.



Note: *Make sure that the GND and VCC connections of the DHT11 sensor match the actual pinout of the component, as they may differ from what is shown in the diagram.*

We will now check the correct functioning of the DHT-11 sensor. Below is a test code for the entire assembly completed for this part of the project. Upload the program, and you will see that, in addition to the colored LEDs blinking as they have done so far, the ESP32 board will send the obtained measurements of ambient temperature (in ºC) and relative humidity (in %) to the Serial Monitor every couple of seconds (you can see the button to open it in the following image).

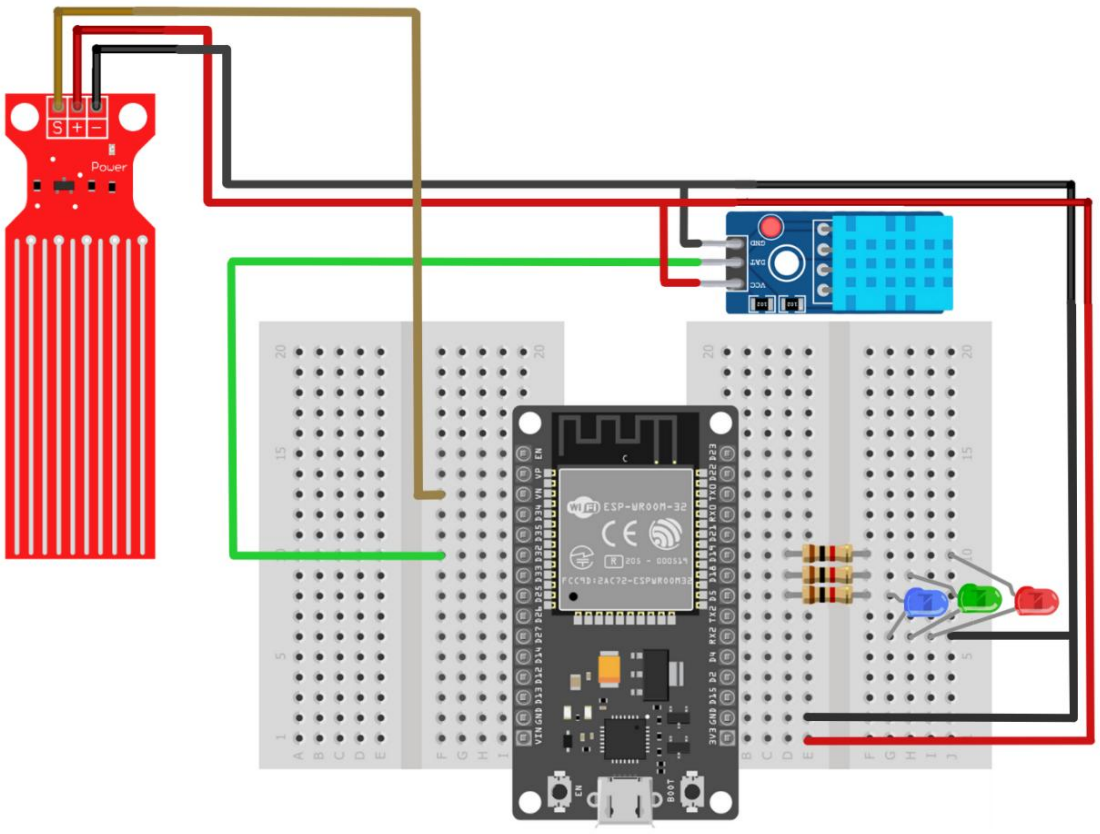



```
#include <DHT.h> // We include the necessary library to control DHT.

#define ledPin1Cold 5 // Define the pin where the Blue LED is connected
#define ledPin2Good 18 // Define the pin where the Green LED is connected
#define ledPin3Heat 19 // Define the pin where the Red LED is connected
#define DHTPIN 32 // Define the pin to which the DHT is connected
#define DHTTYPE DHT11 // Define the type of DHT sensor to use

// Define the type of DHT sensor to be usedCreate an object of type DHT with the corresponding
configuration
DHT myDHT11(DHTPIN, DHTTYPE);

// Initialization block
void setup() {

    // We configure the operating mode of the LEDs pins as outputs
    pinMode(ledPin1Cold, OUTPUT);
    pinMode(ledPin2Good, OUTPUT);
    pinMode(ledPin3Heat, OUTPUT);

    // Initialize the Serial monitor
    Serial.begin(115200);

    // Initialize the DHT sensor
```

```
    myDHT11.begin();

}

// Main loop of the code
void loop() {

    digitalWrite(ledPin1Cold, LOW); // We turn off the Blue LED
    digitalWrite(ledPin2Good, LOW); // We turn off the Green LED
    digitalWrite(ledPin3Heat, LOW); // We turn off the Red LED
    delay(1000); // We wait 1 second = 1000 milliseconds
    digitalWrite(ledPin1Cold, HIGH); // We turn on the Blue LED
    digitalWrite(ledPin2Good, HIGH); // We turn on the Green LED
    digitalWrite(ledPin3Heat, HIGH); // We turn on the Red LED
    delay(1000); // We wait 1 second

    //  We obtain the DHT sensor measurements
    float temperature = myDHT11.readTemperature();
    float humidity = myDHT11.readHumidity();

    // We show the values per monitor Serial
    Serial.print("Temperature: ");
    Serial.println(temperature);
    Serial.print("Humidity: ");
    Serial.println(humidity);

}
```

With the monitor open, make sure that the dropdown located furthest to the right on the Monitor is set to "115200". Now you will be able to see the data sent by the ESP32 from the DHT-11 constantly and verify that the assembly up to this point is functioning correctly.

The next component to be added is the water level sensor module. Connect the sensor to **pin VN**. In the following image you can see how the assembly will look like:



Now we are going to verify the operation of the water level sensor by constantly reading and displaying the measured value of the sensor on the serial monitor:

```
//Water sensor calibration
const int WaterSensorPin = 39; // Analog Value from VN PIN

void setup() {
Serial.begin(115200);
}
void loop() {
  int sensorValue; // Variable to store sensor values
  sensorValue = analogRead(WaterSensorPin); // Read the value from the sensor
  Serial.println(sensorValue); // Print value in serial monitor
  delay(100);
}
```

5. 2. 2. Electronic Assembly (Solar Component)

We will begin by assembling the solar component. To complete this assembly, you will need the following materials:

1 x SG-90 Servo Motor

Used to move the solar demonstrator and orient it toward sunlight.

2 x Photoresistors

Used to measure the level of received light intensity.

2 x 1 kΩ Resistors

Used only with the independent LDRs to assemble a voltage divider controlled by light intensity.

9 x Dupont Cable (Male-Male) 10 cm

Necessary for making connections between the different components on the breadboard.

5 x Dupont cable (Male-Male) 20cm

Necessary for making connections between the different components on the breadboard.

3 x Dupont cable (Male-Female) 10cm

Necessary for making connections between the different components on the breadboard.

4 x Dupont cable (Male-Female) 20cm

Necessary for making connections between the different components on the breadboard.

1 x Voltage sensor

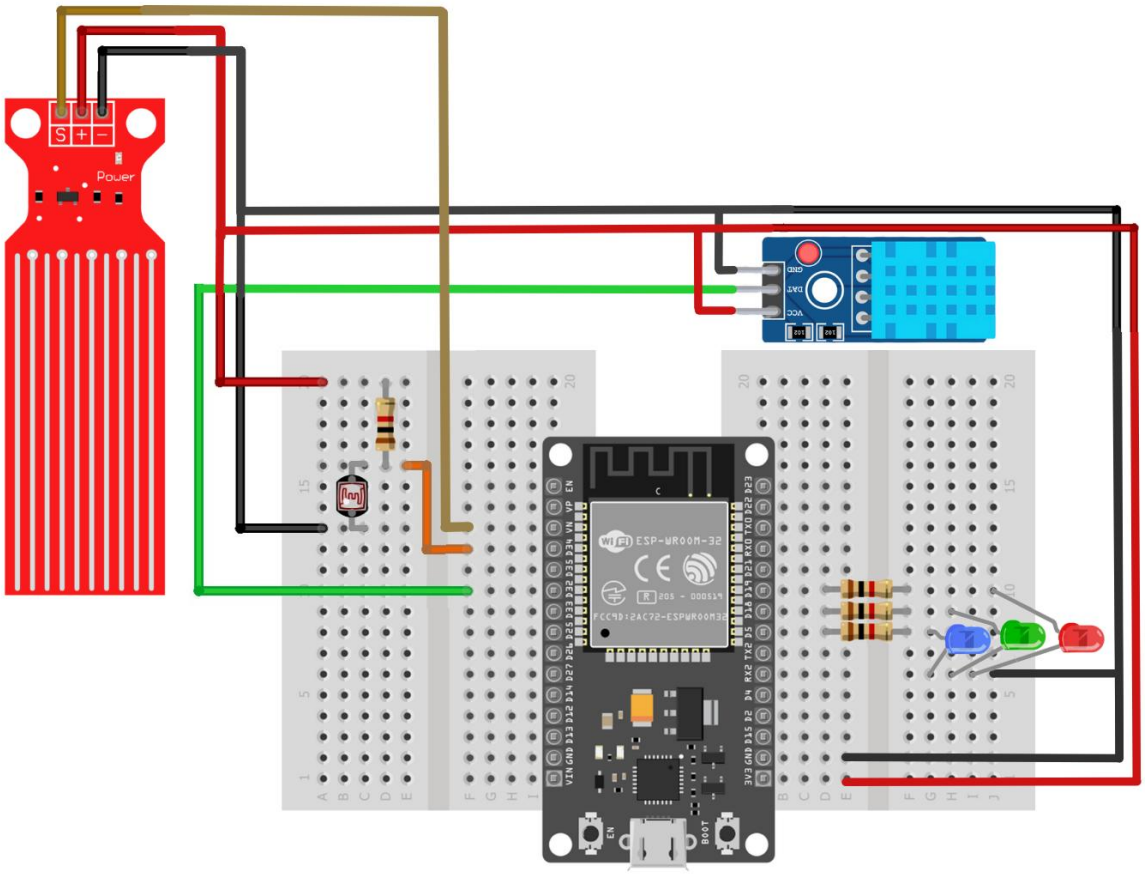
Used alongside the solar panel to detect sunlight.

1 x Solar Panel

Used to detect sunlight.

The objective of the assembly will be to determine the orientation of the incidence of sunlight with the LDRs and rotate the Demonstrator to maximize the amount of sunlight hitting the panel.

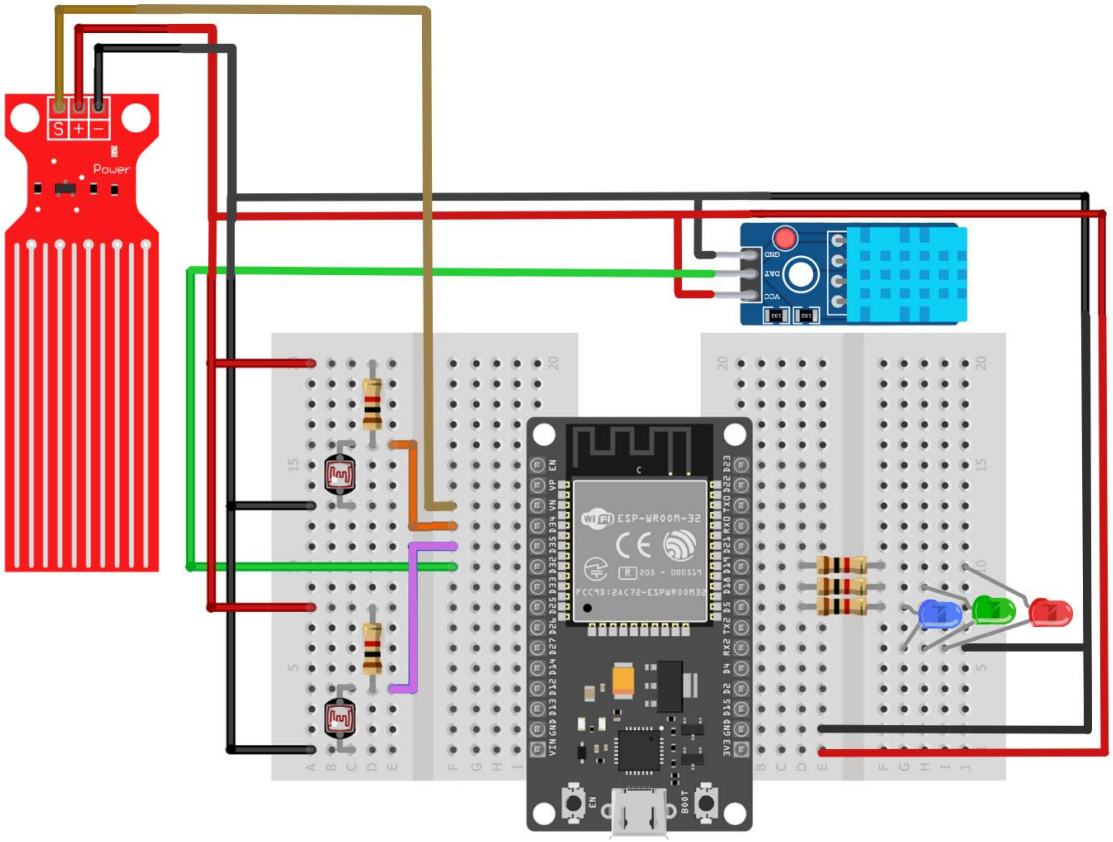
The assembly of the solar component will begin from the last assembly seen in the previous section. In this assembly, an LDR and a resistor will be added in series, forming a voltage divider controlled by light intensity. To obtain the value from the sensor, the digital output will be connected to **pin D34** of the ESP32. The assembly can be seen below:



To check the functionality of the new elements added to the assembly, we will use a code that only reads the LDR value to light up the Red LED with greater or lesser intensity based on the brightness measured by the photoresistor. You can see the code to upload below:

```
#define ldrPin1 34 // Define the pin to which the LDR is connected

// Initialization block
void setup() {

    // Initialize the serial monitor
    Serial.begin(115200);

}

// Main loop of the code
void loop() {

    // We take the reading of the pin where the LDR is located.
    int reading_ldr_1 = analogRead(ldrPin1);
    // Display the value on the Serial monitor
    Serial.print("Luminosity 1: ");
    Serial.println(reading_ldr_1);

}
```

Next, the second LDR with its resistor should be added. To obtain the reading value from this sensor, the digital output will be used, as in the previous case, but this one will be connected to **pin D35**. Below, you can see the assembly with both LDRs for the current part of the Demonstrator.

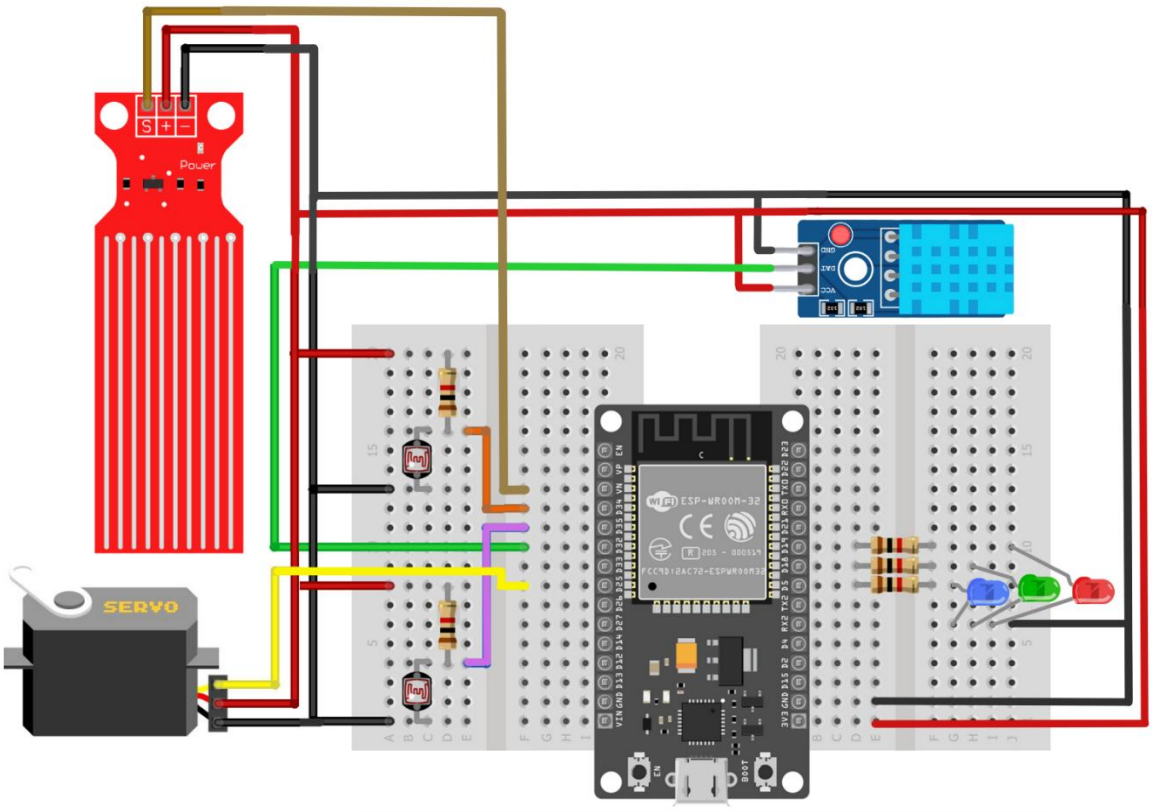


Let's verify that the connections have been made correctly using the following code, which is very similar to the previous one but displays the measurement values from both photoresistors.

```
#define ldrPin1 34 // Define the pin to which the first LDR is connected
#define ldrPin2 35 // Define the pin to which the second LDR is connected

// Initialization block
void setup() {

    // Initialize the serial monitor
    Serial.begin(115200);

}

// Main loop of the code
void loop() {

    // We take the readings of the pins where the LDRs are located
    int reading_ldr_1 = analogRead(ldrPin1);
```

```
    int reading_ldr_2 = analogRead(ldrPin2);
    // We display the first value on the Serial monitor
    Serial.print("Luminosity 1: ");
    Serial.println(reading_ldr_1);
    // We display the second value on the Serial monitor
    Serial.print("Luminosity 2: ");
    Serial.println(reading_ldr_2);

}
```

The next component to be added to the assembly is a servo motor. This will be connected to both power rails (the red wire to the positive and the black wire to the negative), and the yellow wire (which is the control signal) will be connected to **pin D25**.

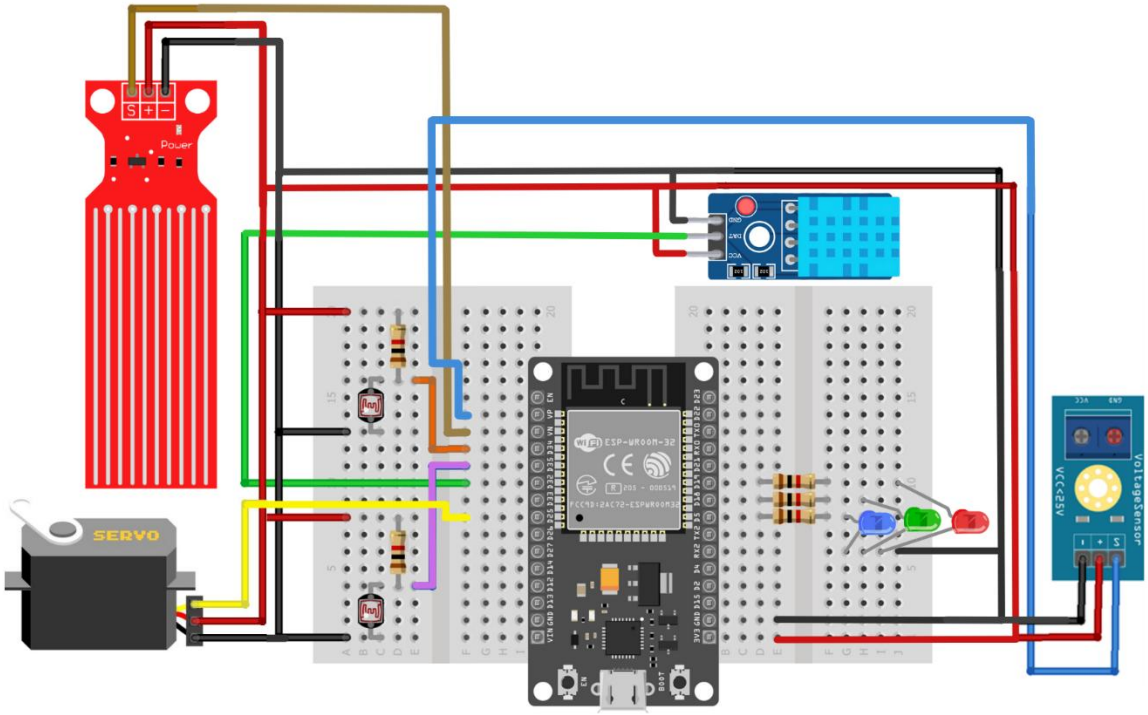


We will check the operation of the servo motor by uploading the following code to the ESP32, which sweeps the servo between the positions of 0°, 90°, and 180° in a loop repeatedly.

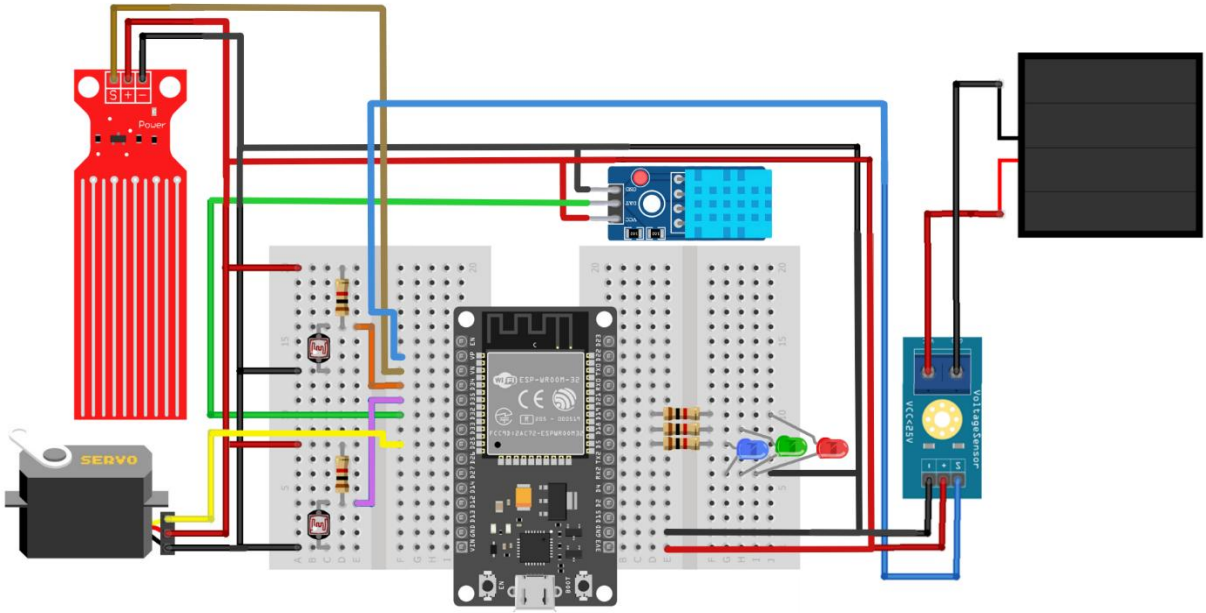
```
#include <ESP32Servo.h> // Include the necessary library to control the servomotor

#define servoPin 25 // Define the pin where the servomotor is connected

// We create an object of type Servo to control it
Servo myServo;

// Initialization block
void setup() {

    // We associate the servo we created to the pin where it is connected
    myServo.attach(servoPin);
    // We place the servo in an initial position of 0°
    myServo.write(0);

}

// Main loop of the code
void loop() {

    myServo.write(0); // We move the servomotor to 0°
    delay(1000); // We wait 1 second = 1000 milliseconds
    myServo.write(90); // We move the servomotor to 90°
    delay(1000); // We wait 1 second
    myServo.write(180); // We move the servomotor to 180°
    delay(1000); // We wait 1 second

}
```

Note: Attach the helix to the servo. Screws are not needed to secure it.

The last component we will connect to the ESP32 will be a voltage detector. This module should be powered through both power rails (Vcc and GND). The signal line will be connected to **pin VP**.



The final step to complete the electronic assembly is to connect the solar panel to the voltage detector. Below, you can see how the complete assembly will look.



We will check the functionality of the complete solar detector by using the following code, which simply reads the signal from the voltage detector module and turns on the Blue LED if it detects voltage.

```
#define ledPin1Cold 5 // Define the pin where the Blue LED is connected
#define detectorPin 36 // Define the pin where the Voltage Detector is connected

// Initialization block
void setup() {

    // We configure the operating mode of the LED pin as an output
    pinMode(ledPin1Cold, OUTPUT);

}

// Main loop of the code
void loop() {

    // We take the reading of the Detector Module
    bool tension_detectada = digitalRead(detectorPin);
    // We check if there is tension or not
    if(tension_detectada) { // Yes, there is tension

        digitalWrite(ledPin1Cold, HIGH); // We turn on the Blue LED

    } else { // No tension

        digitalWrite(ledPin1Cold, LOW); // We turn off the Blue LED

    }

}
```

Below is a test code to verify that all the sensors are getting readings and working correctly:

```
//include the necessary libraries
#include <ESP32Servo.h>
#include "DHT.h"

// Pin definitions and constants
#define ldrPin1 34 // set ldr 1 Analog input pin of East ldr as an integer
#define ldrPin2 35 // set ldr 2 Analog input pin of West ldr as an integer
#define DHTPIN 32
#define DHTTYPE DHT11
```

```
#define ledPin1Cold 5
#define ledPin2Good 18
#define ledPin3Heat 19
#define SolarPin 36

// Global variables
DHT dht(DHTPIN, DHTTYPE);
Servo myservo;
int valldr1 = 0 ;
int valldr2 = 0 ;
int averageLdrValue = 0;
int servopos = 90; // initial position of the Horizontal movement controlling servo motor
int tolerance = 20; // allowable tolerance setting - so solar servo motor isn't constantly in motion
float lastMicrovolts = 0;
const int WaterSensorPin = 39; // Analog Value from VN PIN
int sensorValue; // Variable to store sensor values

void setup(){
Serial.begin(115200);
analogReadResolution(12); // ESP32 has a 12-bit ADC
analogSetAttenuation(ADC_11db); // Set attenuation for higher sensitivity
pinMode(4, INPUT);

dht.begin(); // Start the DHT sensor

pinMode(ledPin1Cold, OUTPUT);
pinMode(ledPin2Good, OUTPUT);
pinMode(ledPin3Heat, OUTPUT);

myservo.attach(25); // attaches the servo on digital pin 2 to the horizontal movement servo motor

pinMode(ldrPin1, INPUT); //set East ldr pin as an input
pinMode(ldrPin2, INPUT); //set West ldr pin as an input
myservo.write(servopos); // write the starting position of the horizontal movement servo motor

delay(1000); // 1 second delay to allow the solar panel to move to its staring position before commencing solar tracking
}

void loop(){
```

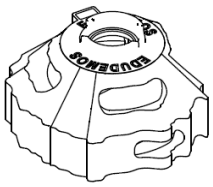
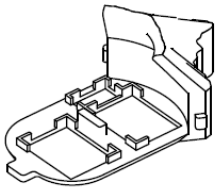
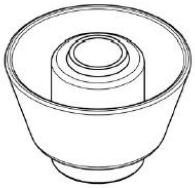
```
  Serial.println("----------------------------- Measured Solar (V) ------------------
----------  ");
   int solarVoltagedV = analogRead ( SolarPin); //dV
 int solarVoltage = solarVoltagedV * 10;

  Serial.print("Voltage of Solar Panel: ");
  Serial.println(solarVoltage);

  // Calculate voltage
  float voltage = (solarVoltage / 4095.0) * 3.3;  // For 12-bit ADC (0-4095)
  Serial.print("Measured voltage (V): ");
  Serial.println(voltage);  // Display the measured voltage

  // Convert to microvolts
  float microvoltsSolar = voltage * 1000000.0;
  Serial.print("Input voltage (µV): ");
  Serial.println(microvoltsSolar);

  // Apply a minimum threshold to filter noise
  const float threshold = 10.0;  // Threshold in microvolts
  if (microvoltsSolar < threshold) {
    microvoltsSolar = 0;
  }

  // Check if the microvolts value has changed
  if (microvoltsSolar != lastMicrovolts) {
    Serial.print("Send voltage: ");
    Serial.print(microvoltsSolar, 2);  // Display with 2 decimal places
    Serial.println(" µV");

    // Update the last microvolts value
    lastMicrovolts = microvoltsSolar;
  }

  Serial.println("----------------------- Measured Temperature / Humidity -----------
------------  ");
   // Read values from DHT sensor
  float h = dht.readHumidity();
  float t = dht.readTemperature();

  // Check if readings are valid
  if (isnan(h) || isnan(t)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
```

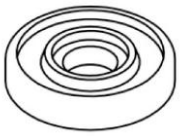
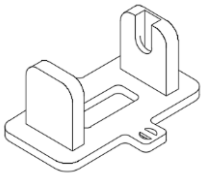
```
  }

  // Print sensor values
  Serial.print("Temperature: ");
  Serial.print(t);
  Serial.print(" °C, Humidity: ");
  Serial.print(h);
  Serial.print(" %");

  // Example LED control based on temperature
  if (t < 20) {
    digitalWrite(ledPin1Cold, HIGH);
    digitalWrite(ledPin2Good, LOW);
    digitalWrite(ledPin3Heat, LOW);
    Serial.println("Cold is HIGH");

  } else if (t >= 20 && t < 25) {
    digitalWrite(ledPin1Cold, LOW);
    digitalWrite(ledPin2Good, HIGH);
    digitalWrite(ledPin3Heat, LOW);
    Serial.println("Good is HIGH");
  } else {
    digitalWrite(ledPin1Cold, LOW);
    digitalWrite(ledPin2Good, LOW);
    digitalWrite(ledPin3Heat, HIGH);
    Serial.println("Heat is HIGH");
  }

  Serial.println("----------------------------- Measured Water Level ----------------
-------------  ");
  int sensorValue; // Variable to store sensor values
  sensorValue = analogRead(WaterSensorPin); // Read the value from the soil moisture
sensor
  //Check watersensor calibration first
  if (sensorValue >= 2000) { //These value represents de max value given
    sensorValue = 1980; //This value would return a 99% value
  }
  float waterlevel = (sensorValue*100)/2000 ; //Calculation to return the %

  Serial.print("Water level: "); // Print value in serial monitor
  Serial.print(waterlevel); // Print value in serial monitor
  Serial.println(" %"); // Print value in serial monitor
```

```
  Serial.println("---------------------------- Measured LDR and Servo --------------
---------------  ");
    // Read LDR values
  valldr1 = analogRead(ldrPin1); // read the value of ldr 1
  valldr2 = analogRead(ldrPin2); // read the value of ldr 2
  // Invert readings so low values mean darkness and high values mean light

  averageLdrValue = (valldr1 + valldr2) / 2;

  // Print readings for debugging
  Serial.print("LDR 1: ");
  Serial.print(valldr1);
  Serial.print(" | LDR 2: ");
  Serial.println(valldr2);
  Serial.print("Average LDR: ");
  Serial.println(averageLdrValue);

if((abs(valldr1 - valldr2) <= tolerance) || (abs(valldr2 - valldr1) <= tolerance)) {
//no servo motor horizontal movement will take place if the ldr value is within the
allowable tolerance
}else {
if(valldr1 > valldr2) // if ldr1 senses more light than ldr2
{
servopos = servopos+10; // decrement the 90 degree position of the horizontal servo
motor - this will move the panel position Eastward
}
if(valldr1 < valldr2) // if ldr2 senses more light than ldr1
{
servopos = servopos-10; // increment the 90 degree position of the horizontal motor -
this will move the panel position Westward
 }
}

if(servopos > 180) {servopos = 180;} // reset the horizontal position of the motor to
180 if it tries to move past this point
if(servopos < 0) {servopos = 0;} // reset the horizontal position of the motor to 0 if
it tries to move past this point
myservo.write(servopos); // write the starting position to the horizontal motor

  // Print the servo angle
  Serial.print("Servo angle: ");
  Serial.print(servopos);
  delay(5000);
}
```

5. 2. 3. Mechanical Assembly of the Water and Solar Demonstrator

The elements that make up this demonstrator are as follows:

Element	Quantify	Description	Image
Base	1	It is the upper part that secures the moving components, such as the blades and the motor shaft. It provides stability and allows the system to operate smoothly, facilitating the correct transfer of energy from the blades to the generator.	
Drawer	1	It is the base that connects the different sections of the device, providing structural support and ensuring proper alignment between the motor, the blades, and other essential components for its operation.	
Water container	1	This part is capable of storing the water that falls into it.	
Leaf	8	A sheet that allows liquids to enter the water collection container.	
Small disc	3	It is a static component that does not move. Its primary function is to act as part of the structural support of the device, helping to maintain proper alignment and	

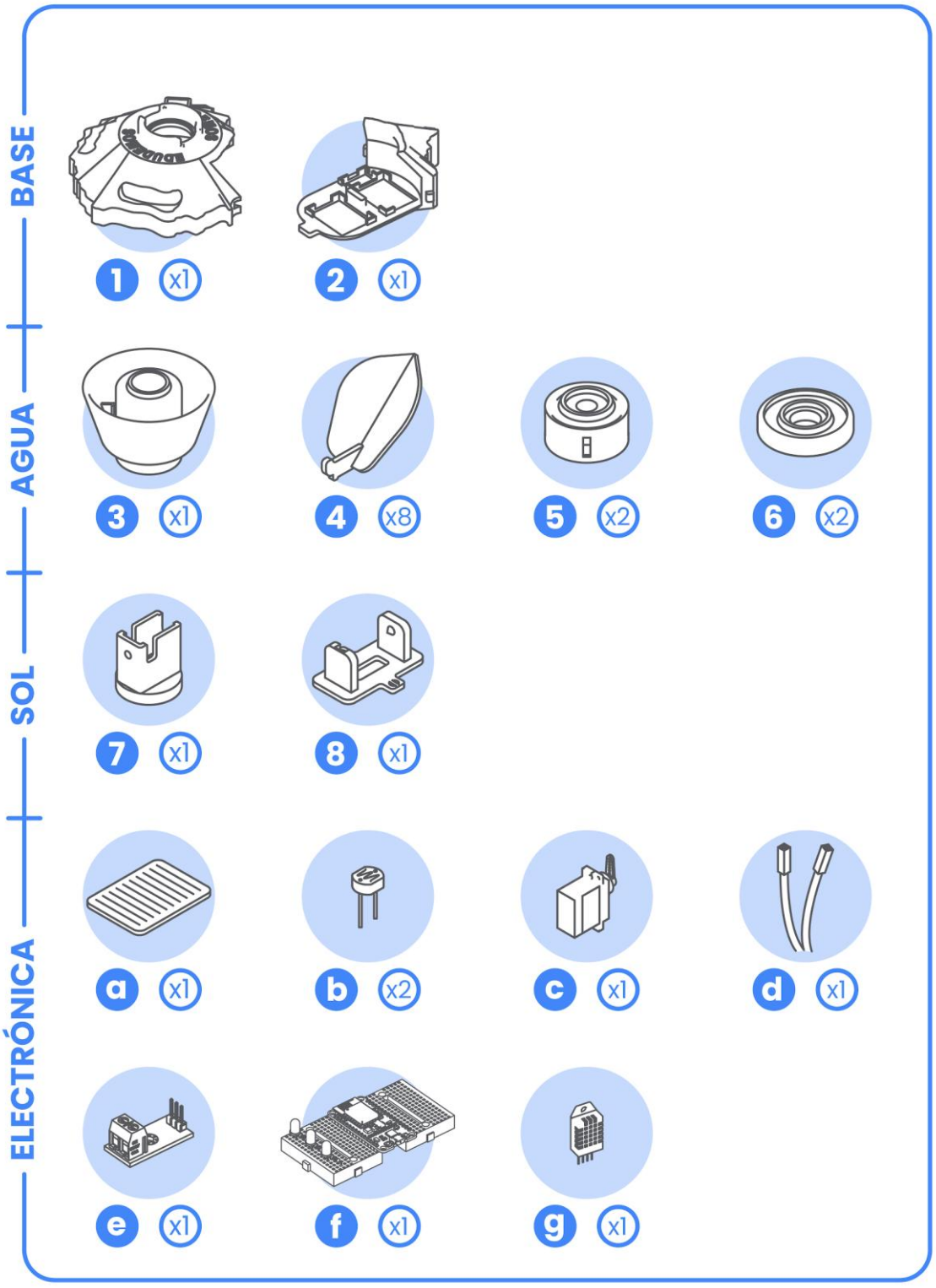
		stability of the elements.	
Static disc	2	A disc that remains fixed to cover and maintain the stability of the structure.	
Solar spin	1	A part that allows for the movement of the solar panel for proper orientation.	
Solar disc	1	A disc that connects the solar rotator with an installed servo motor and joins it with the rest of the structure.	

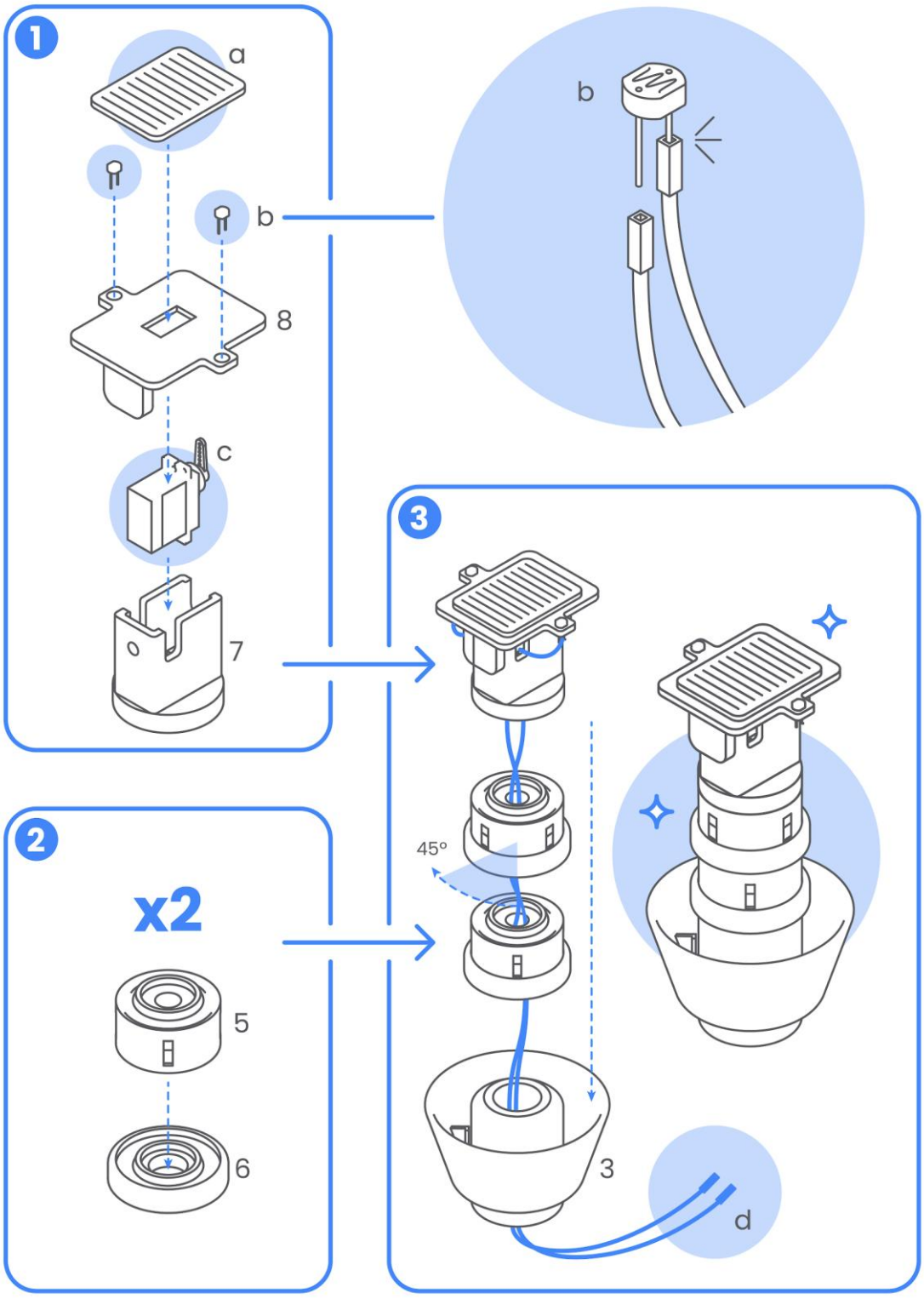
Next, the assembly of the demonstrator is shown:

Note: *The aluminum tape is used on the leaves' surface to optimise the water collection. The double sided tape is used on the back of the solar panel to stick it on the "solar spin" part.*

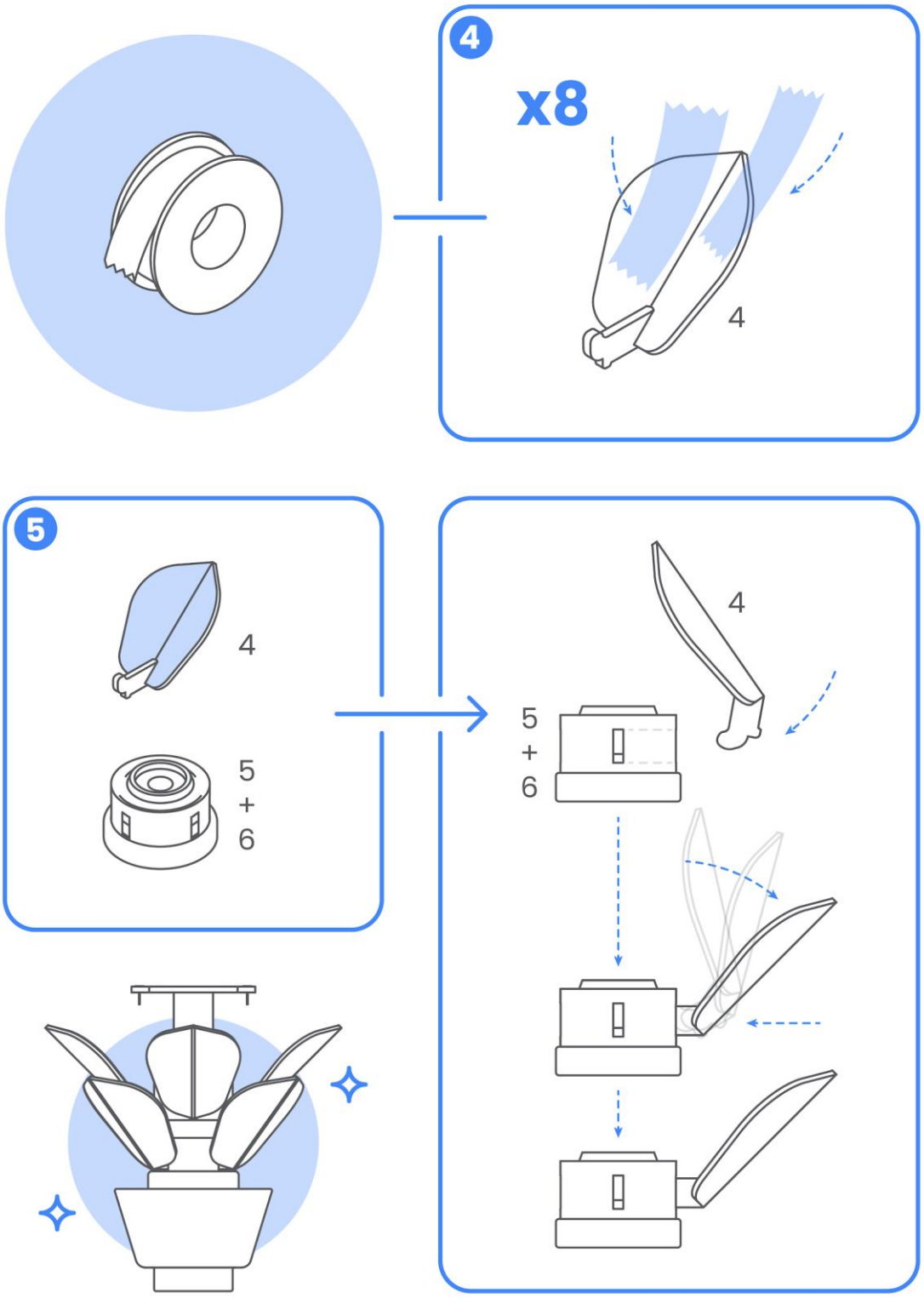
The water level sensor is placed in some slots inside the "water container" part.

When placing the electronic components into the mechanical assembly, it is necessary to use additional wires to extend the connections of the LDRs and the solar panel.

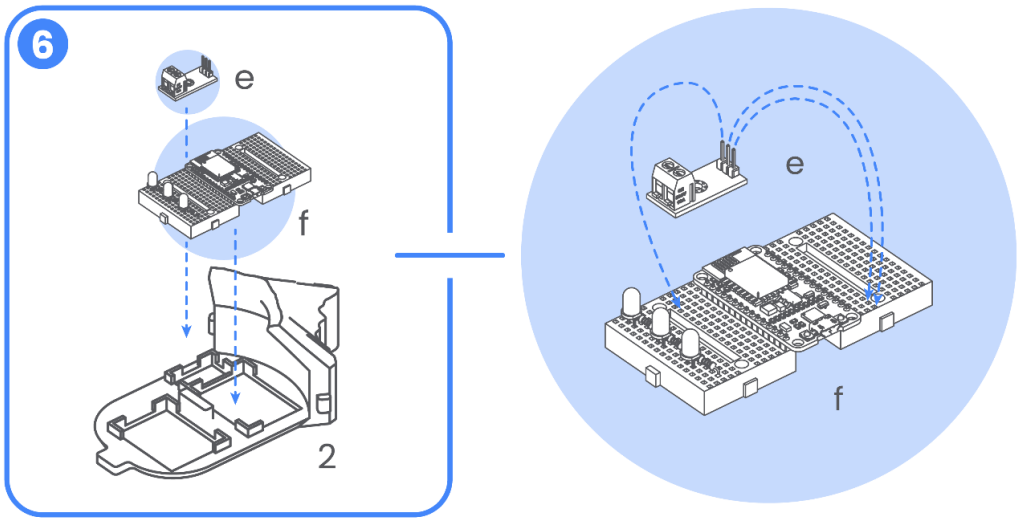

BASE
AGUA
SOL
ELECTRÓNICA
1 x1
2 x1
3 x1
4 x8
5 x2
6 x2
7 x1
8 x1
a x1
b x2
c x1
d x1
e x1
f x1
g x1

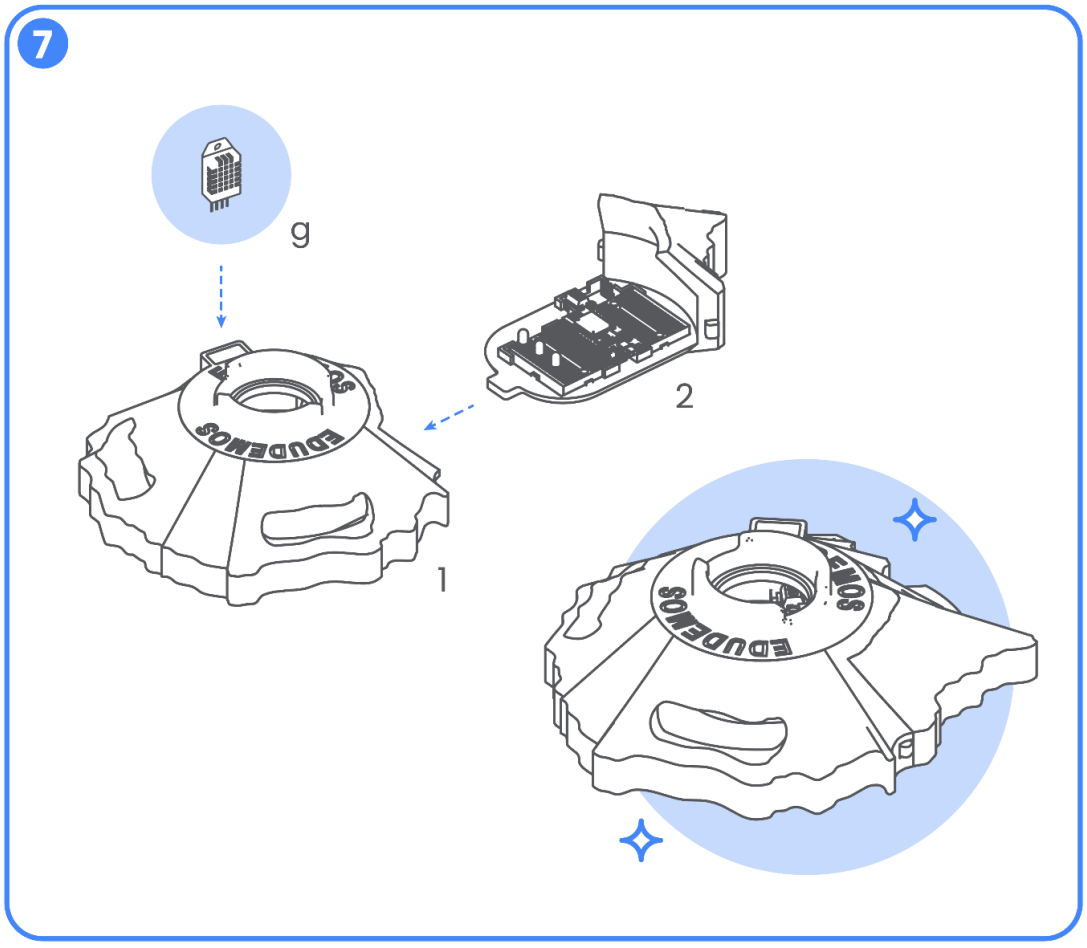


Funded by the European Union
Gerda Stetter Stiftung
Technik macht Spaß!
FUNDACIÓN SERGIO ALONSO
Finnova
gbs sg.ch

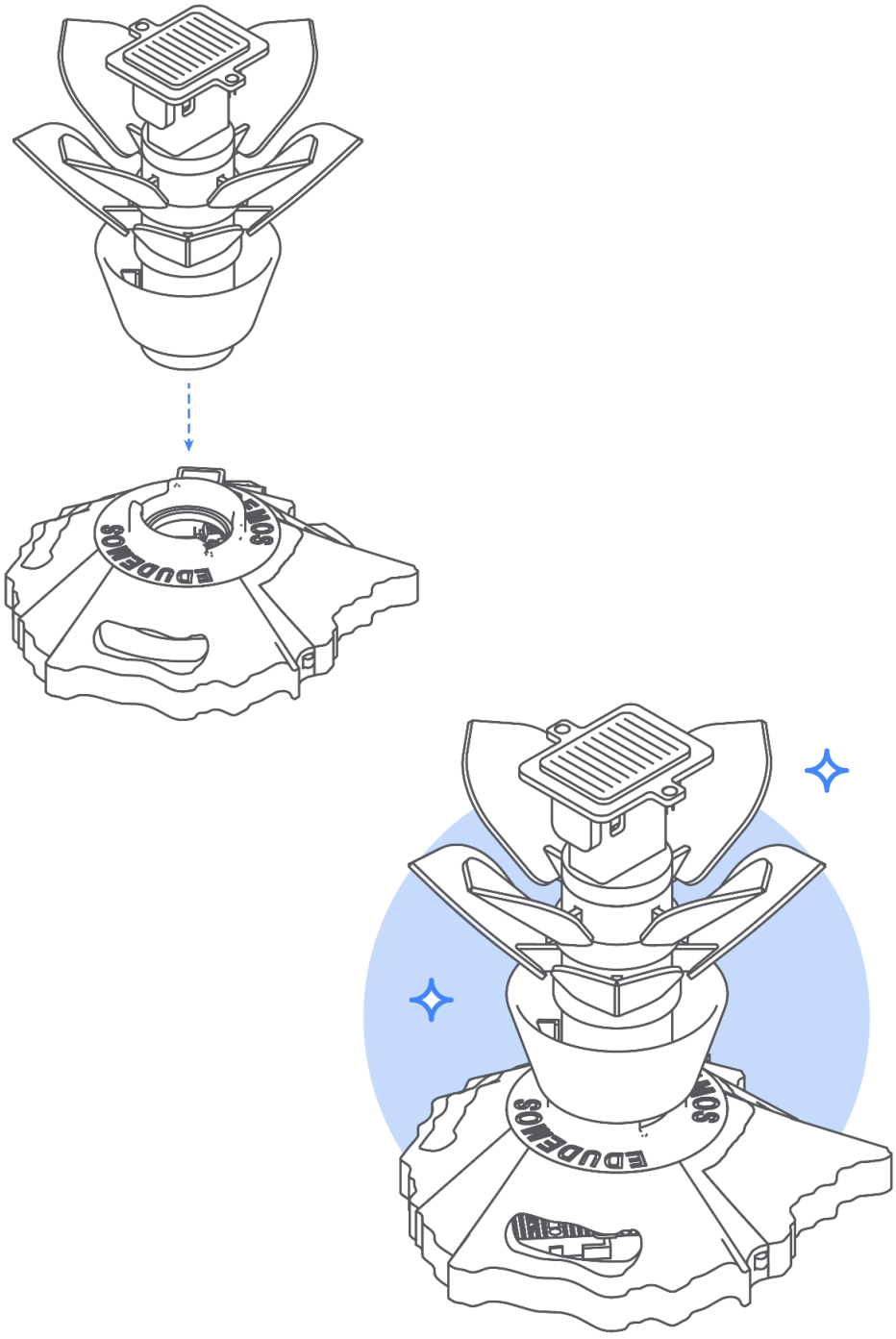


Funded by the European Union
Gerda Stetter Stiftung
Technik macht Spaß!


6
e
f
2
e
f


7
g
2
1

8



Funded by the European Union
Gerda Stetter Stiftung
Technik macht Spaß!
FUNDACIÓN SERGIO ALONSO
Finnova
gbs sg.ch

5. 3. Assembly of the Wind Demonstrator

5. 3. 1. Electronic Assembly

We will start by performing the assembly of the Wind Demonstrator. To perform this complete assembly you will need the following material:

1 x 5V DC Motor

Used to generate voltage with wind power.

1 x 100nF / 1uF Capacitor

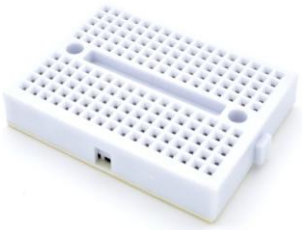
Used to eliminate unwanted noise generated by the DC motor.

2 x 1 kΩ Resistors

Used in the voltage divider that allows reading the voltage generated by the DC motor.

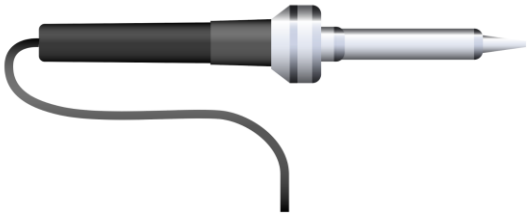
6 x Dupont Cable (Male-Male) 10 cm

Necessary for making connections between the different components on the breadboard.



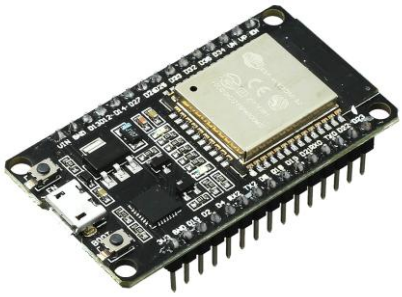
2 x Mini Breadboard

Component necessary to connect the microcontroller board to the different components.



Soldering iron

Necessary if the motor doesn't have wires included

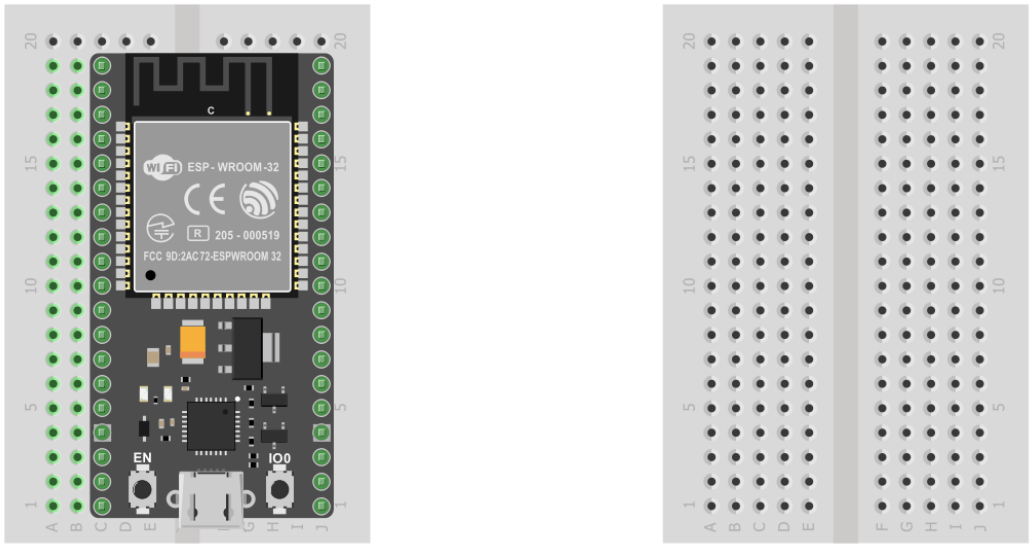


1 x ESP32 DEVKIT V1 (30 pins)

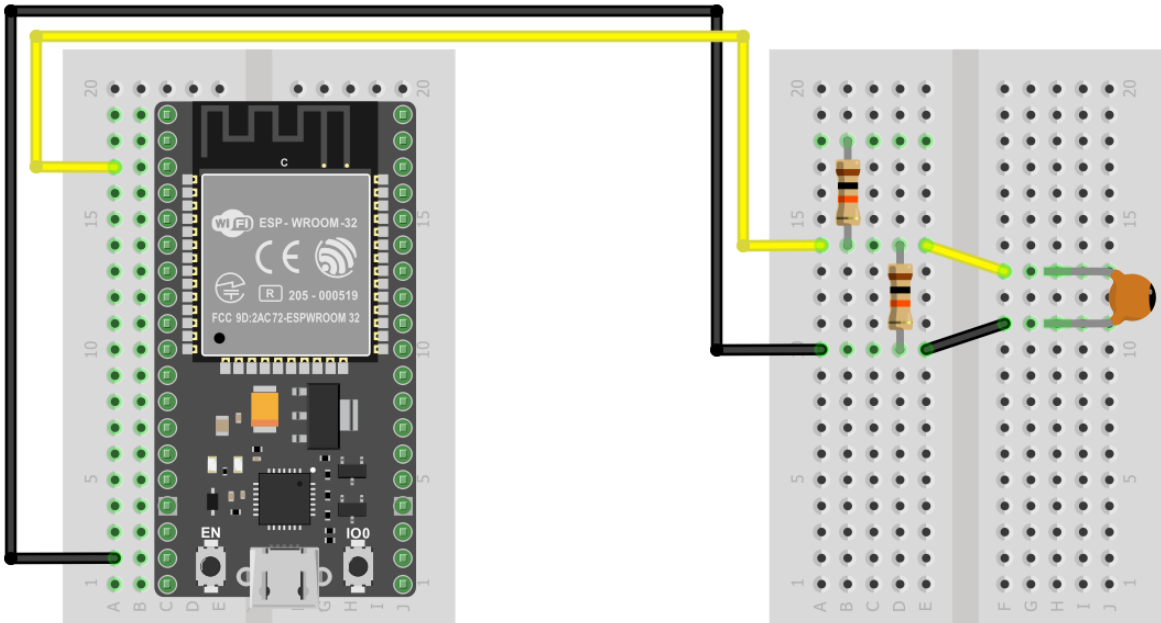
Responsible for executing the programming that controls the operation of the electronics.

The objective of the setup will be to measure the amount of voltage generated by the motor, which will be attached to a mechanical structure that will rotate due to wind power. To measure the voltage obtained from the movement of the blades of the Demonstrator using the ESP32, we need to assemble a voltage divider to reduce the maximum voltage level (5V) to one suitable for the ESP32 (which operates at 3.3V). This way, we will be able to determine how much voltage is being generated by the wind.

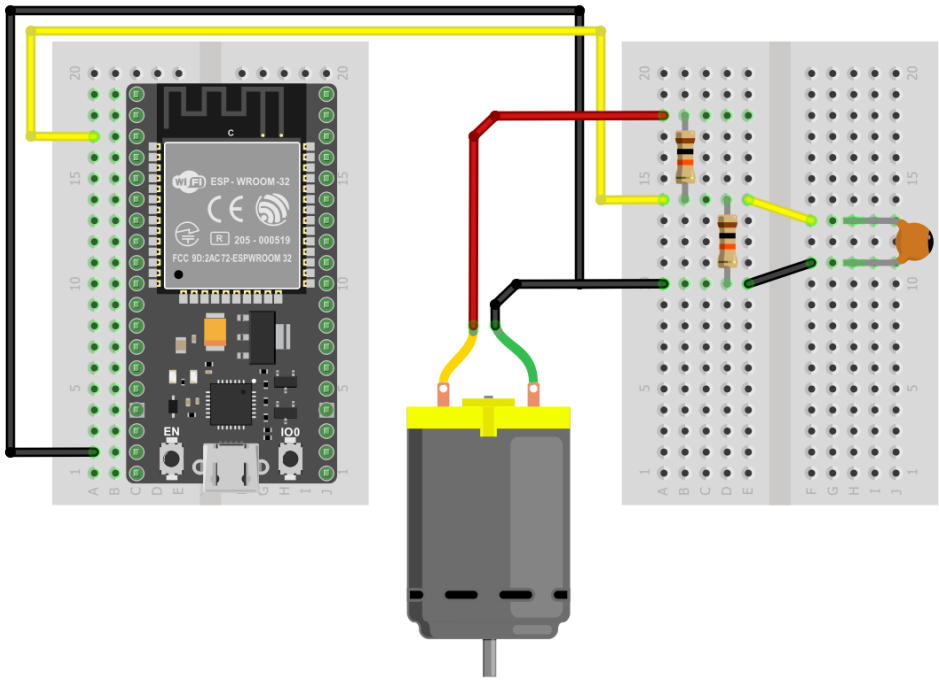
For this new setup, it will be necessary to create a circuit from scratch. We will begin by placing the ESP32 on two mini breadboards. The corresponding setup is shown below.



Next, a voltage divider will be assembled. The midpoint of this divider will be connected to **pin VN**. Additionally, a capacitor must be connected between the terminal of this pin and GND, which will act as a filter for unwanted variations in the measurement, as shown in the image below.



The final step in the electronic assembly of this demonstrator is to connect the motor to the previous setup. One of its terminals should be connected to GND, and the other terminal should be connected to the input of the voltage divider assembled earlier. The completed setup is presented below.

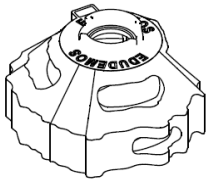
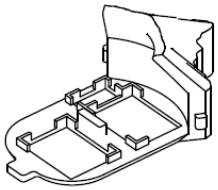
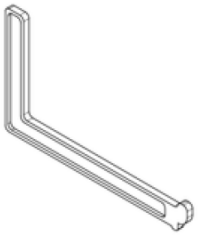


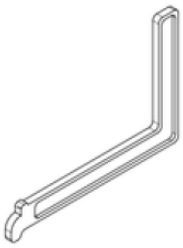
Let's check that the setup of this Demonstrator works correctly by using the following code snippet, where we only read the output of the voltage divider connected to the ESP32. By rotating the motor's rotor, we will see how the measurement gives different values through the Serial Monitor.

```
// Initialization block
void setup() {
    // We initialize the Serial Monitor
    Serial.begin(115200);
}
// Main loop of the code
void loop() {
    // We take the measurement from the pin connected to the output of the voltage
    //divider
    int medicion = analogRead(39);
    // We display the measurement on the Serial Monitor
    Serial.println(medicion);
}
```

5. 3. 2. Mechanical Assembly

The components of this demonstrator are as follows:

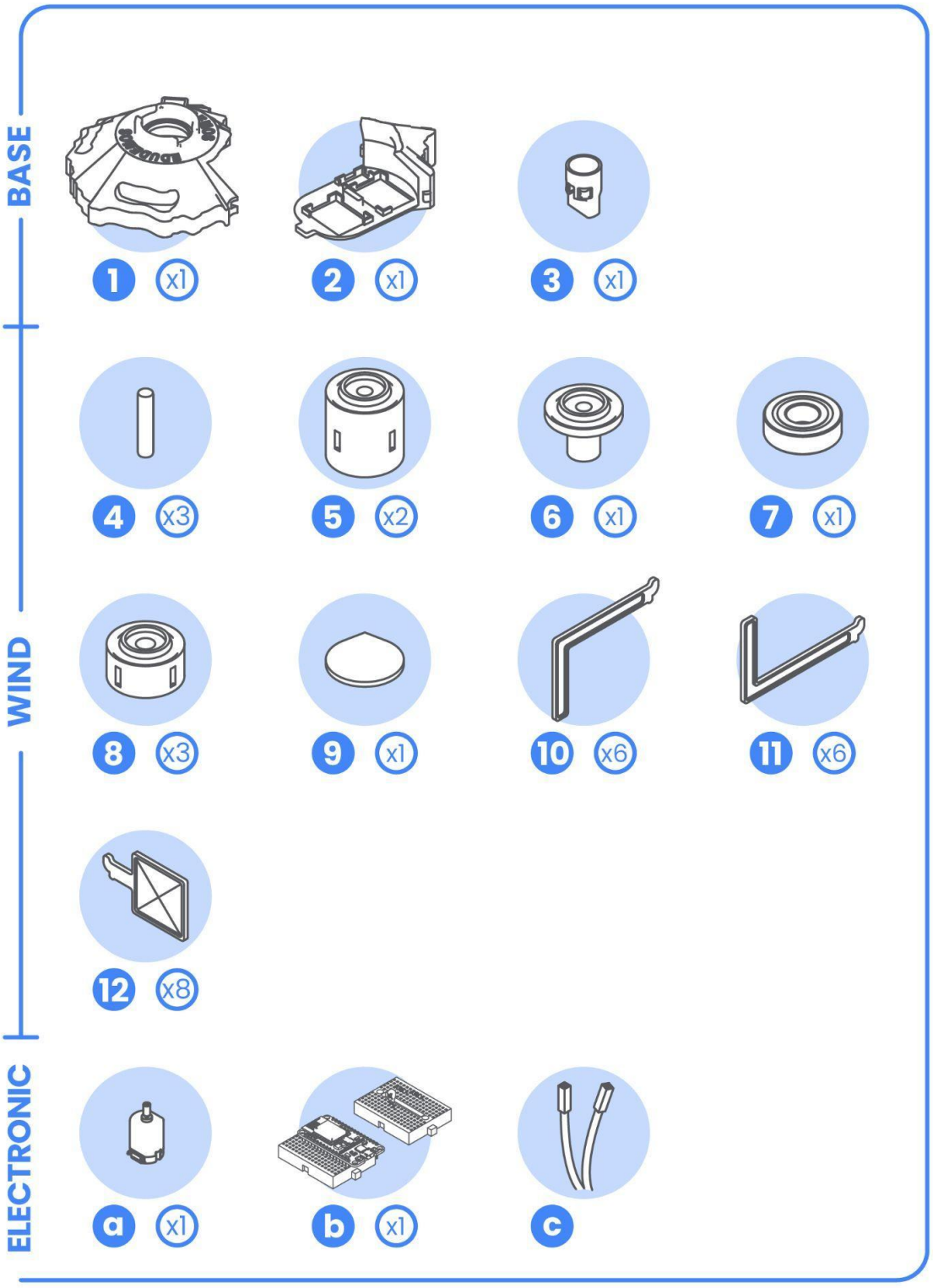
Elemento	Cantidad	Descripción	Imagen
Base	1	It is the top section that secures the moving components, such as the blades and the motor shaft. It provides stability and allows the system to operate smoothly, facilitating the proper transfer of energy from the blades to the generator.	
Drawer	1	It is the base that connects the different sections of the device, providing structural support and ensuring proper alignment between the motor, blades, and other essential components for its operation.	
Motor socket	1	This component will support and stabilize the motor, whose shaft will be connected to the bearing disc. This design will allow the connection of output pins, which will emit signals indicating the presence of movement.	
Big L blade	6	It is a fixed blade that does not move. It is designed to be part of the static structure of the demonstrator, contributing to the overall stability and balance of the system.	

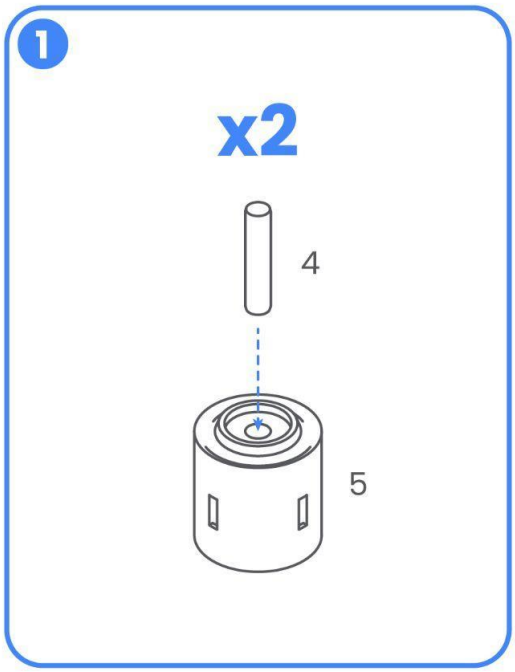
Small L blade	6	It is a fixed blade that does not move, smaller than the previous piece. It is designed to be part of the static structure of the demonstrator, contributing to the overall stability and balance of the system.	
Square blade	8	It is one of the key moving blades that rotates when driven by the wind. Its square design efficiently captures the wind's kinetic energy and converts it into rotational movement.	
Big disc	2	It is a key component that serves as a structural support for the moving blades. This disk is designed to rotate when the blades capture the wind, transferring the rotational movement to the system's central shaft.	
Small disc	3	It is a static component that does not move. Its main function is to act as part of the device's structural support, helping to maintain the proper alignment and stability of the moving elements, such as the blades and large disks.	
Bearing disc	1	It is a crucial component used to reduce friction between the system's moving parts. This disk acts as a rotational support, allowing components like the Large Disk and the blades to spin smoothly and efficiently around the central shaft.	

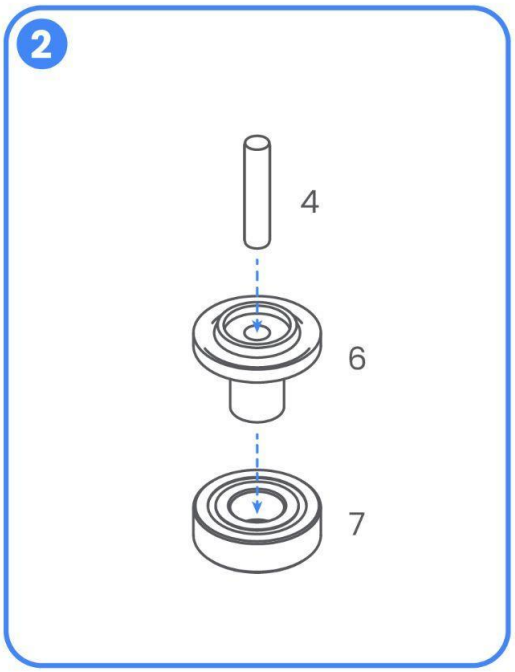
Top	1	It is a structural component placed at the top of the assembly, ensuring that all internal elements are protected and properly aligned.	
Bearing	1	It is a mechanical component that allows rotating elements, such as the disks and blades, to move with minimal friction.	
Wood joints	3	It is a structural component used to connect and secure the different wooden parts of the device.	

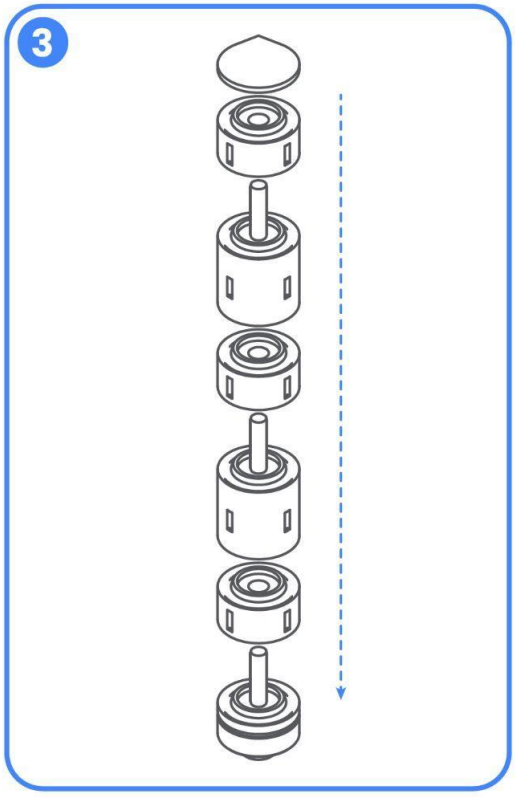
The following shows how the demonstrator assembly is carried out:

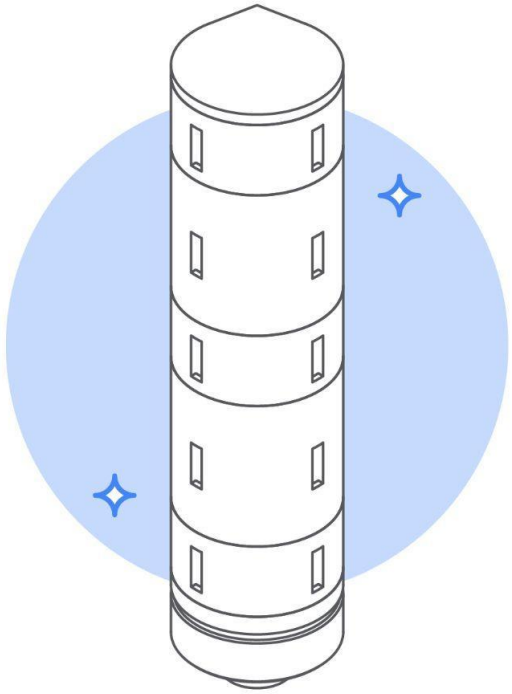
Note: It is necessary to insert the 10-tooth gear onto the DC motor shaft before starting the assembly of the demonstrator.

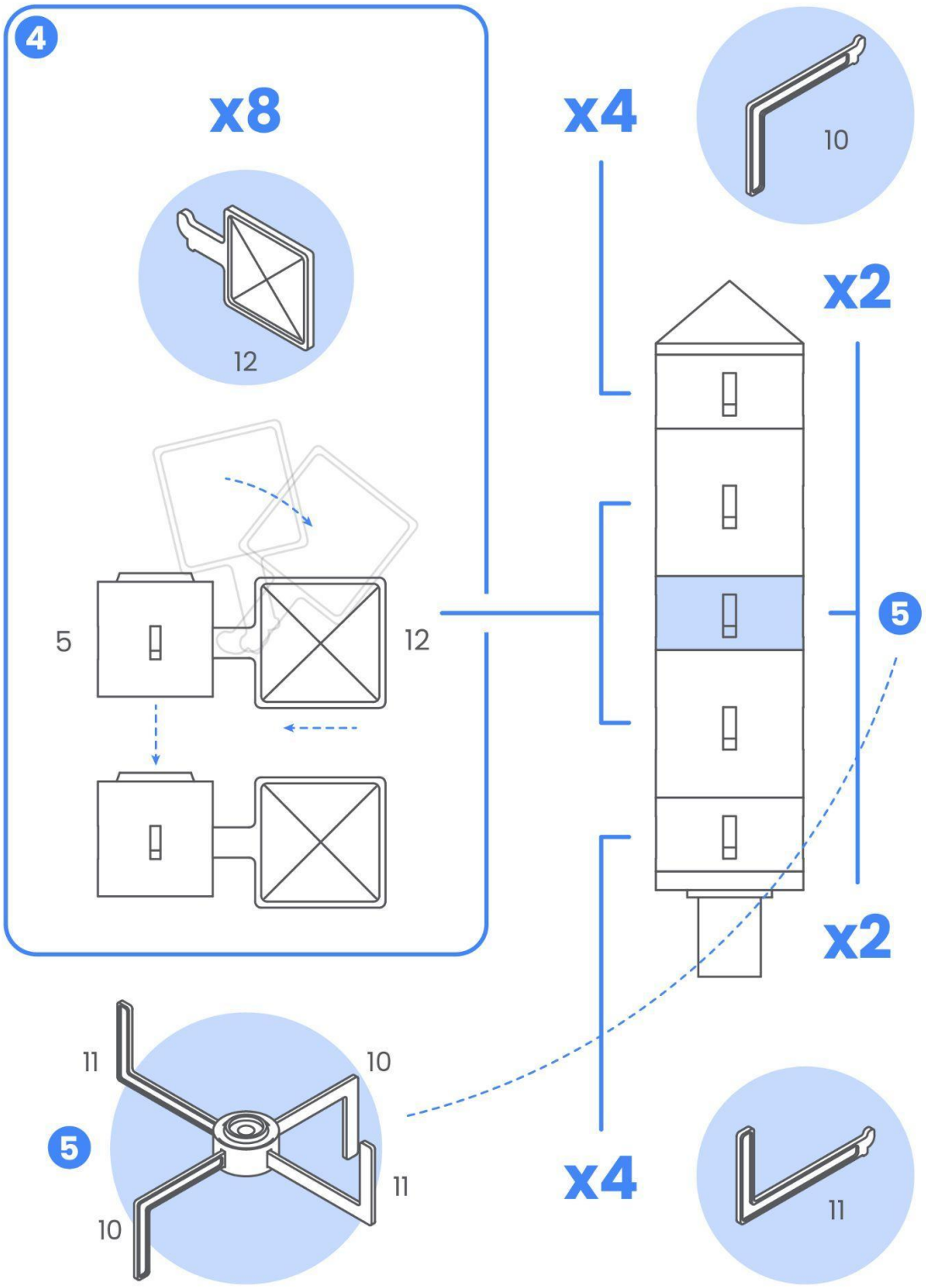

BASE
WIND
ELECTRONIC
1 x1
2 x1
3 x1
4 x3
5 x2
6 x1
7 x1
8 x3
9 x1
10 x6
11 x6
12 x8
a x1
b x1
c


1
x2
4
5

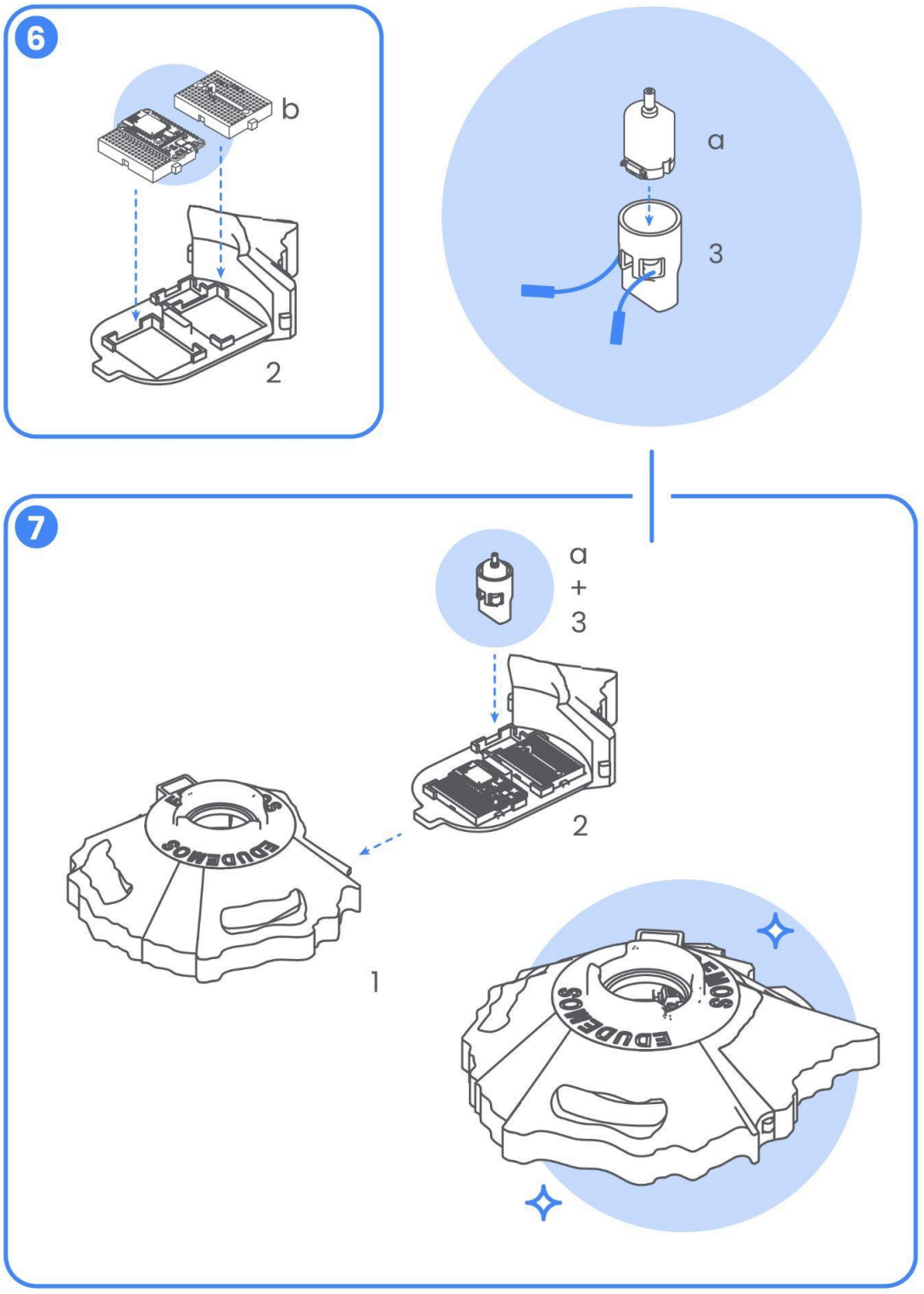

2
4
6
7


3



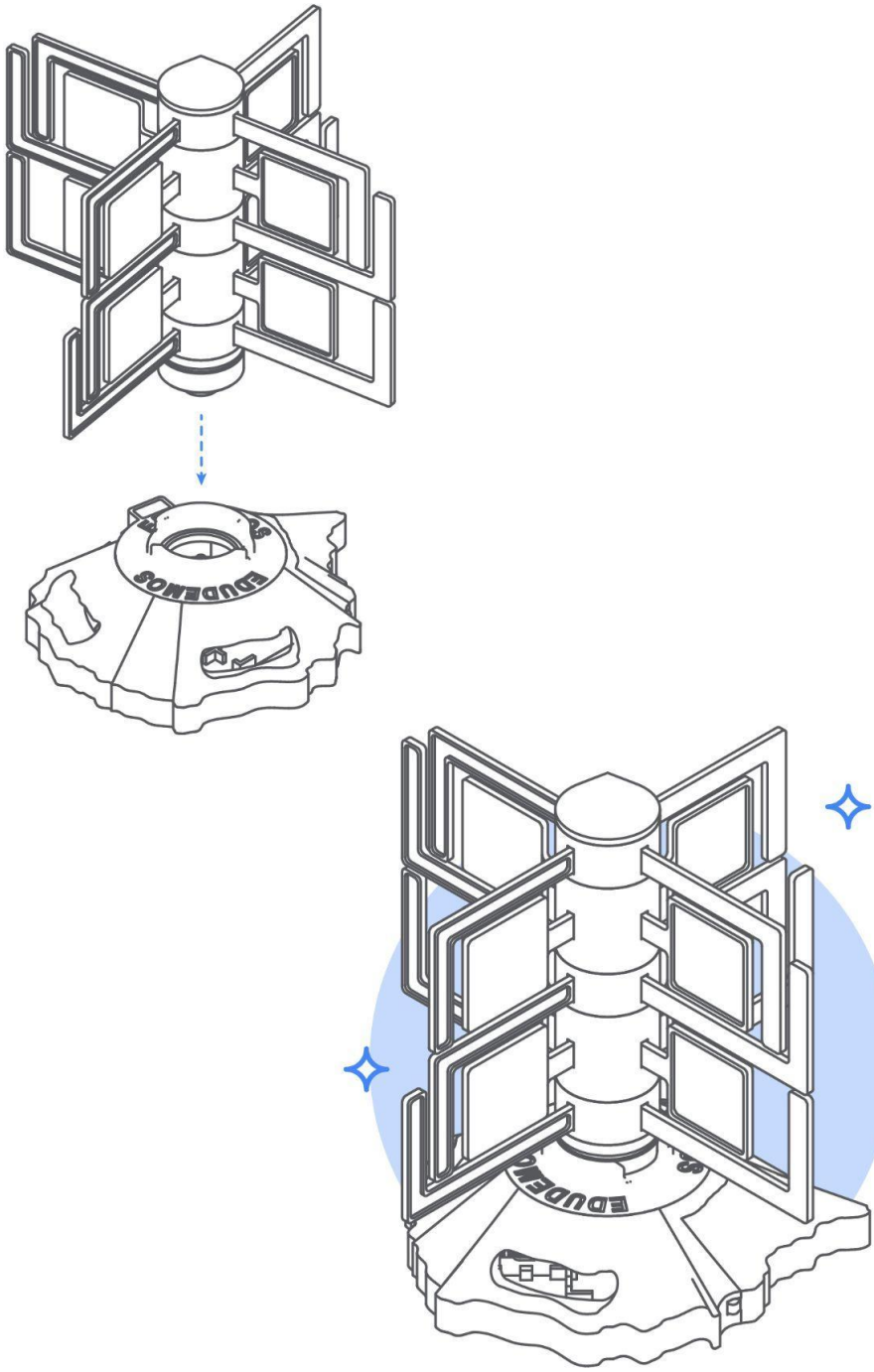


Funded by the European Union
Gerda Stetter Stiftung
Technik macht Spaß!
FUNDACIÓN SERGIO ALONSO
Finnova
gbs sg.ch



Funded by the European Union
Gerda Stetter Stiftung
Technik macht Spaß!
FUNDACIÓN SERGIO ALONSO
Finnova
gbs sg.ch

8

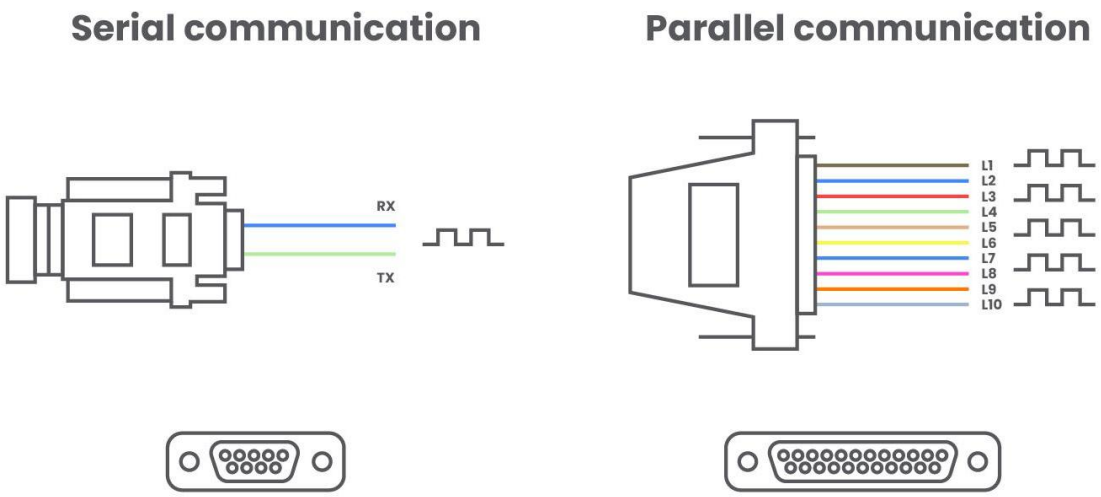


Funded by the European Union
Gerda Stetter Stiftung
Technik macht Spaß!
FUNDACIÓN SERGIO ALONSO
Finnova
gbs sg.ch

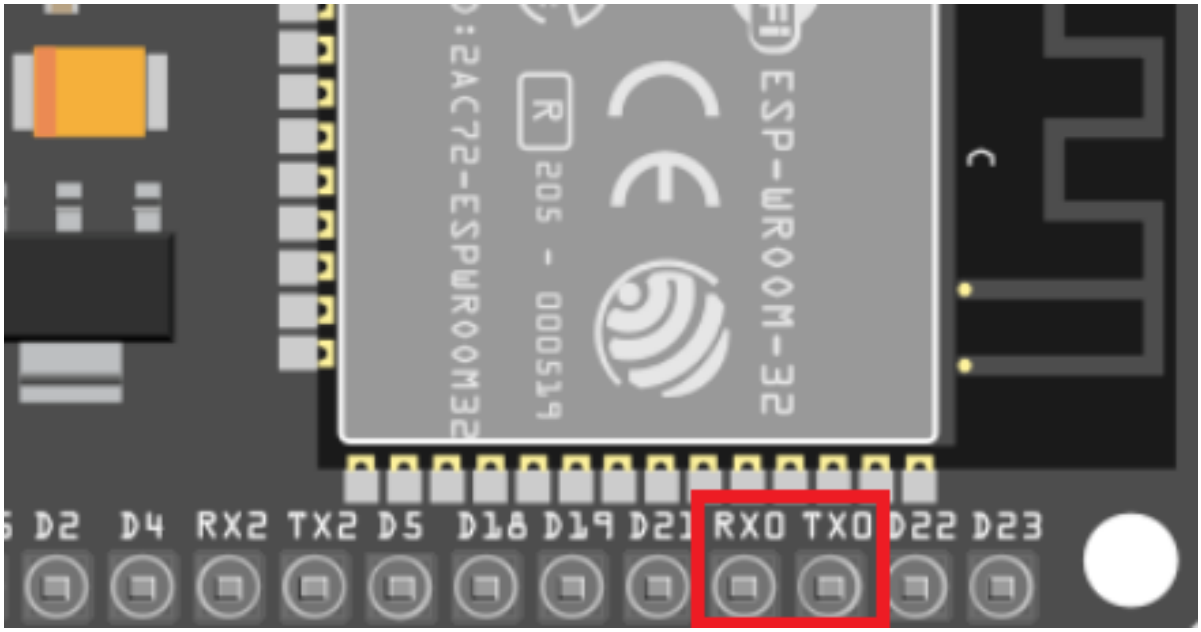
6. DATA FROM THE SENSORS

6. 1. Serial Port

Communication through the serial port is one of the most commonly used methods for devices to connect with each other. It is also the primary means by which the ESP32 microcontroller, the ESP32, communicates with both the computer and other components. In a serial port, data is transmitted and received sequentially, sending bits one at a time through a single channel. In contrast, a parallel port transmits information simultaneously across multiple channels.

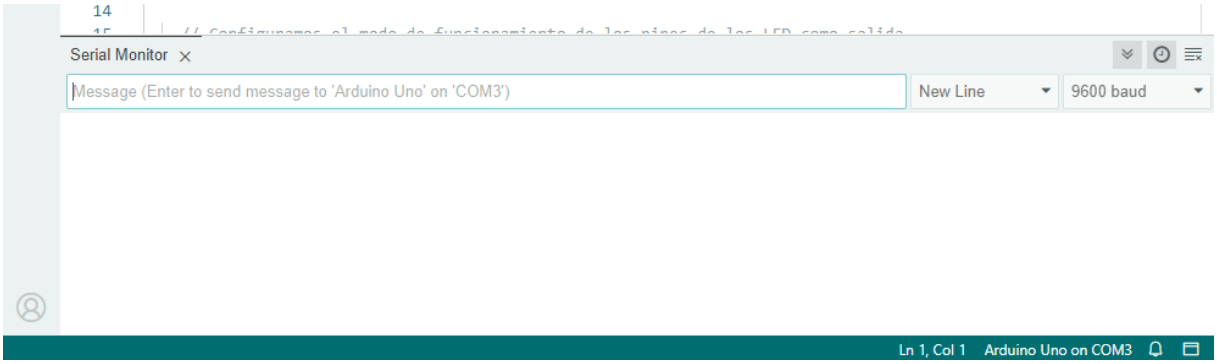


There are several types of serial ports, with the most popular being the USB (Universal Serial Bus) port. However, the older RS-232 serial port can also be found, as well as other serial communication protocols such as RS-485, I2C, SPI, among others. Serial communication is fundamental in microcontrollers, as all of them have at least one UART (Universal Asynchronous Receiver/Transmitter) unit. This type of serial port uses two channels: one for transmitting data (TX) and another for receiving it (RX).



The ESP32 board can communicate with the computer through a USB port, which is the same port used for programming it. This establishes a bidirectional communication channel that allows for both sending and receiving data. The tool that allows us to view the data sent from the ESP32 and, in turn, send information from the computer to the ESP32 is **the Serial Monitor, which we have mentioned and used previously**. This interface is integrated into the Arduino IDE and can be found in the upper right corner of the program, represented by the following icon:

Upon opening it, the Serial Monitor in the Arduino IDE appears as follows:



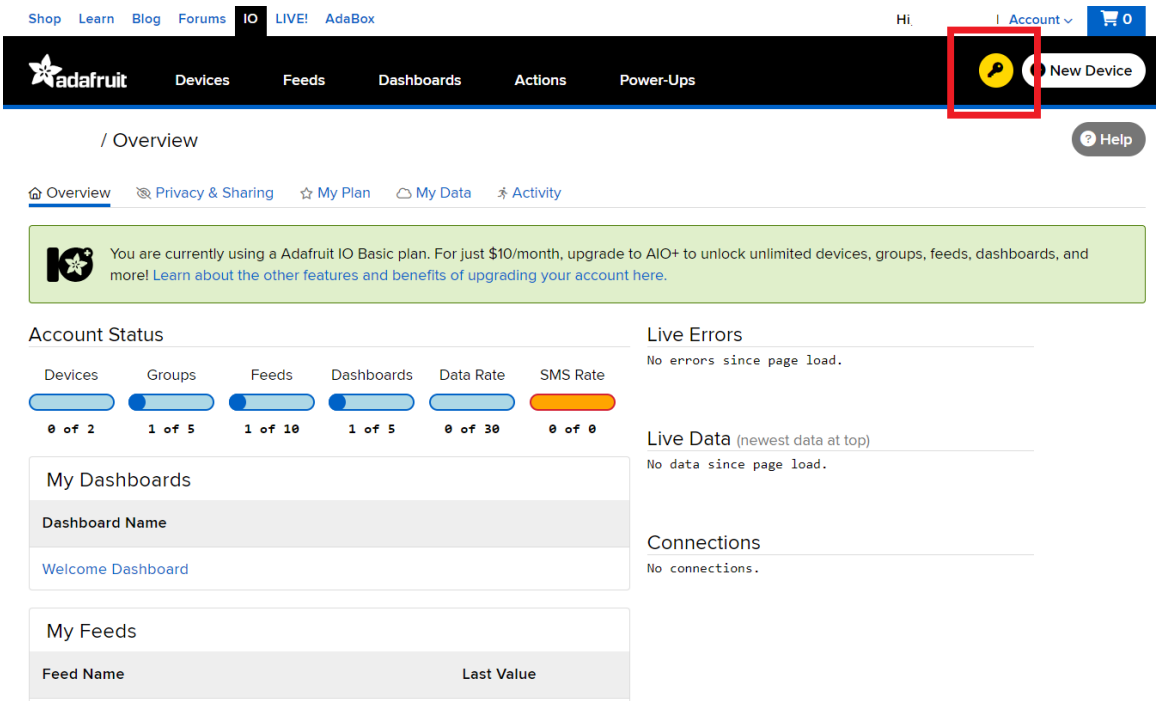
As you can see, the interface is quite simple and easy to use, with the following main elements:

- **Sending Data:** In the upper blank field, you can type and send data from the PC to the ESP32.
- **Receiving Data:** Most of the monitor is dedicated to displaying the data received from the ESP32.
- **Autoscroll:** Allows you to enable or disable automatic scrolling in the data reception window.
- **Show Timestamp:** Option to see the exact time each piece of data is received.
- **Line Ending:** Adds a type of line ending to the sent message.
- **Baud Rate:** Defines the transmission speed in baud (symbols per second). It is essential that the baud rate is the same for both the sender and the receiver for correct communication.

6. 2. Adafruit IO

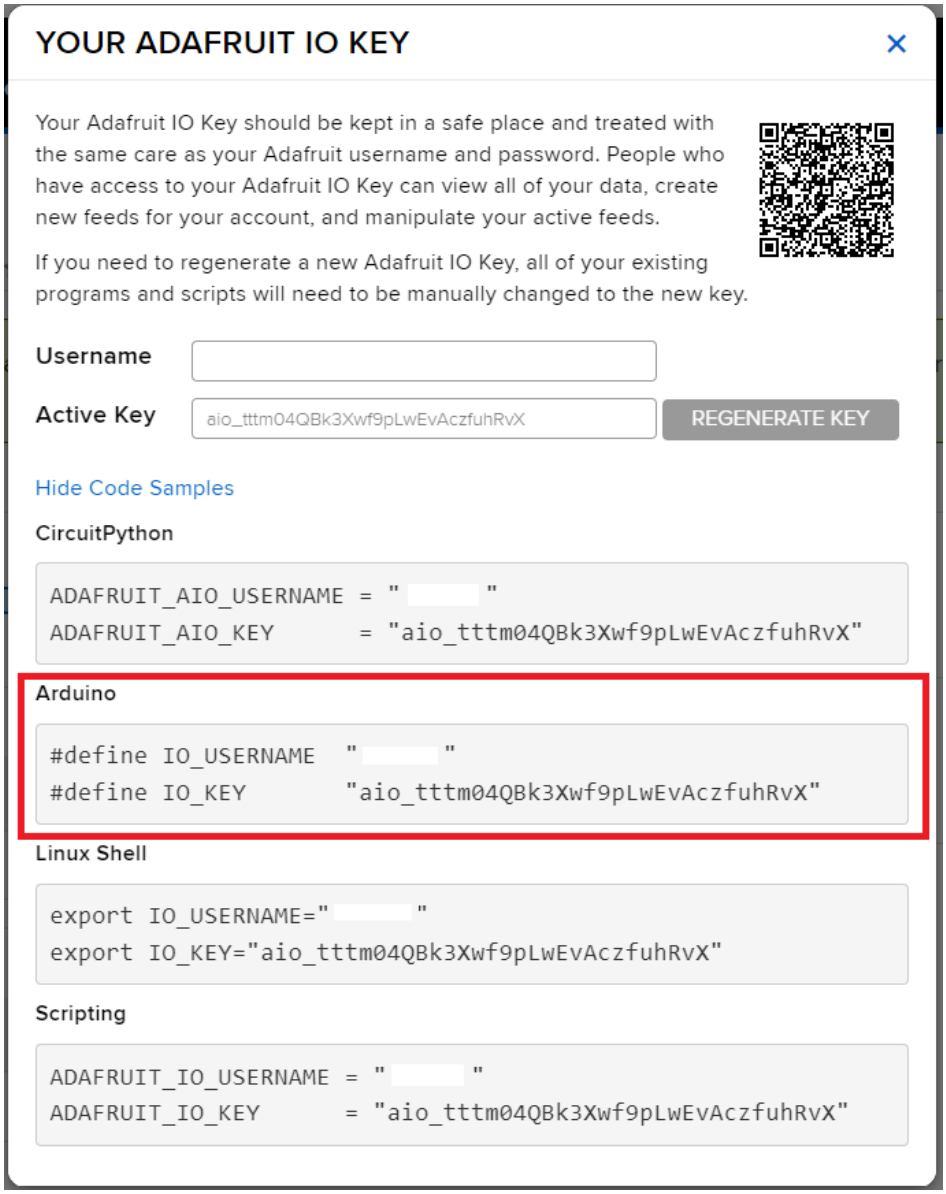
Adafruit IO is an Internet of Things (IoT) platform that allows users to collect, visualize, and manage data from devices connected to the network. Offered by Adafruit, a company known for developing open-source hardware and software, Adafruit IO simplifies the connection and control of hardware projects like microcontrollers (Arduino, ESP32, Raspberry Pi, etc.) with the cloud.

To use this platform, we need to create a user account on its website. Once we have an account, we will click on the "IO" section at the top, next to Shop, Blog, and Forum. We will see the following window, and we should click on the icon with the yellow key:

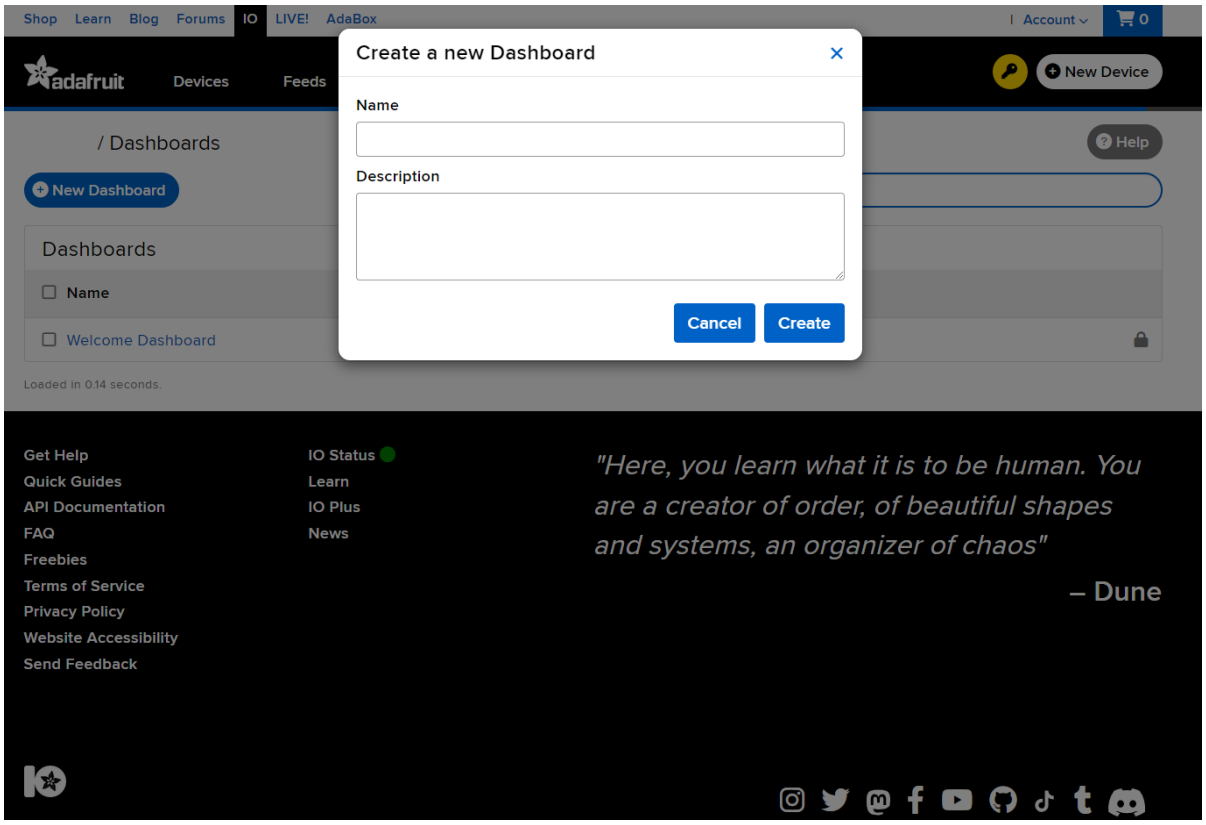


When we click there, a window will open like the one in the following image, where we need to go to the "**Arduino**" section to obtain the data displayed there and modify

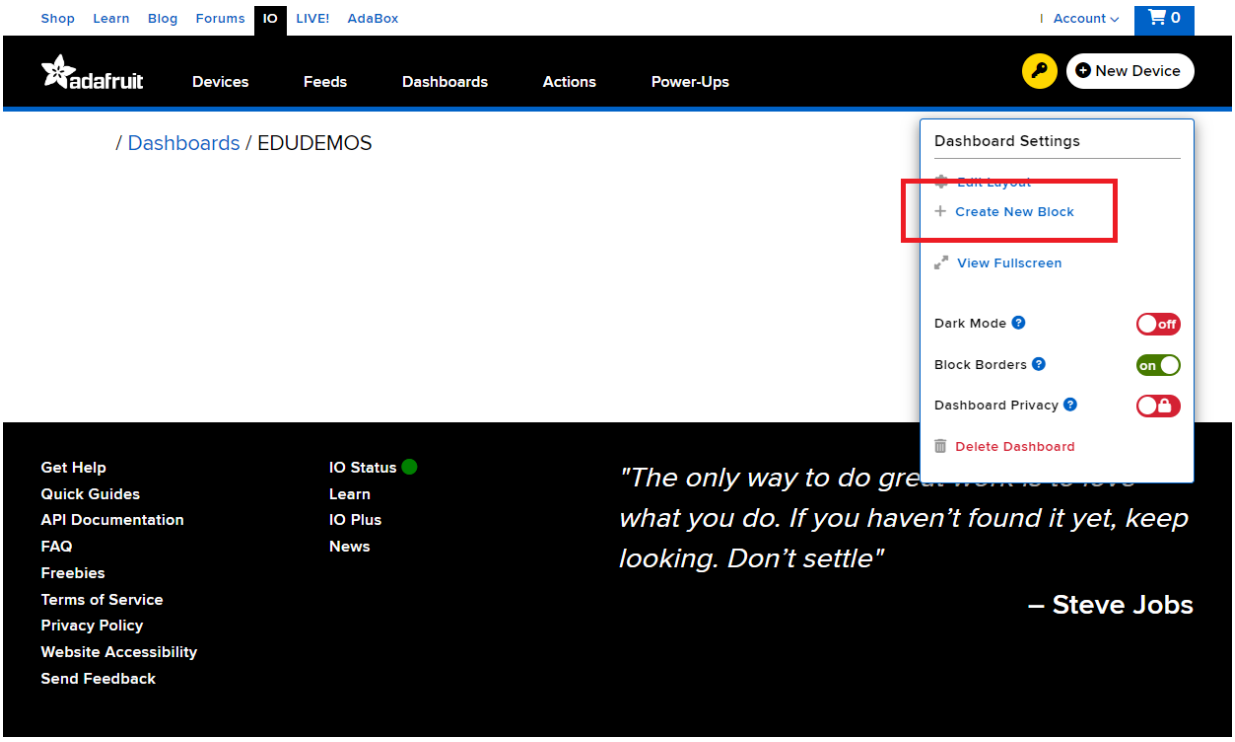
the same variables in the configuration code (config.h) of the Demonstrators mentioned earlier.



The next step is to create a **Dashboard**. To do this, we will click on the "Dashboards" menu at the top and then on the large blue button that says "New Dashboard." We will then see a pop-up window asking us to provide a name and a description for our new Dashboard. We can create one with any name we choose and add a description of what we will use it for.



Now we will proceed to enter the Dashboard we created, where we will see a nearly empty page. To add representations of the data that will arrive from our ESP32 in the Demonstrator, we need to click on the gear icon located in the upper right corner, which will open a menu where we will have the option to "**Create New Block**".



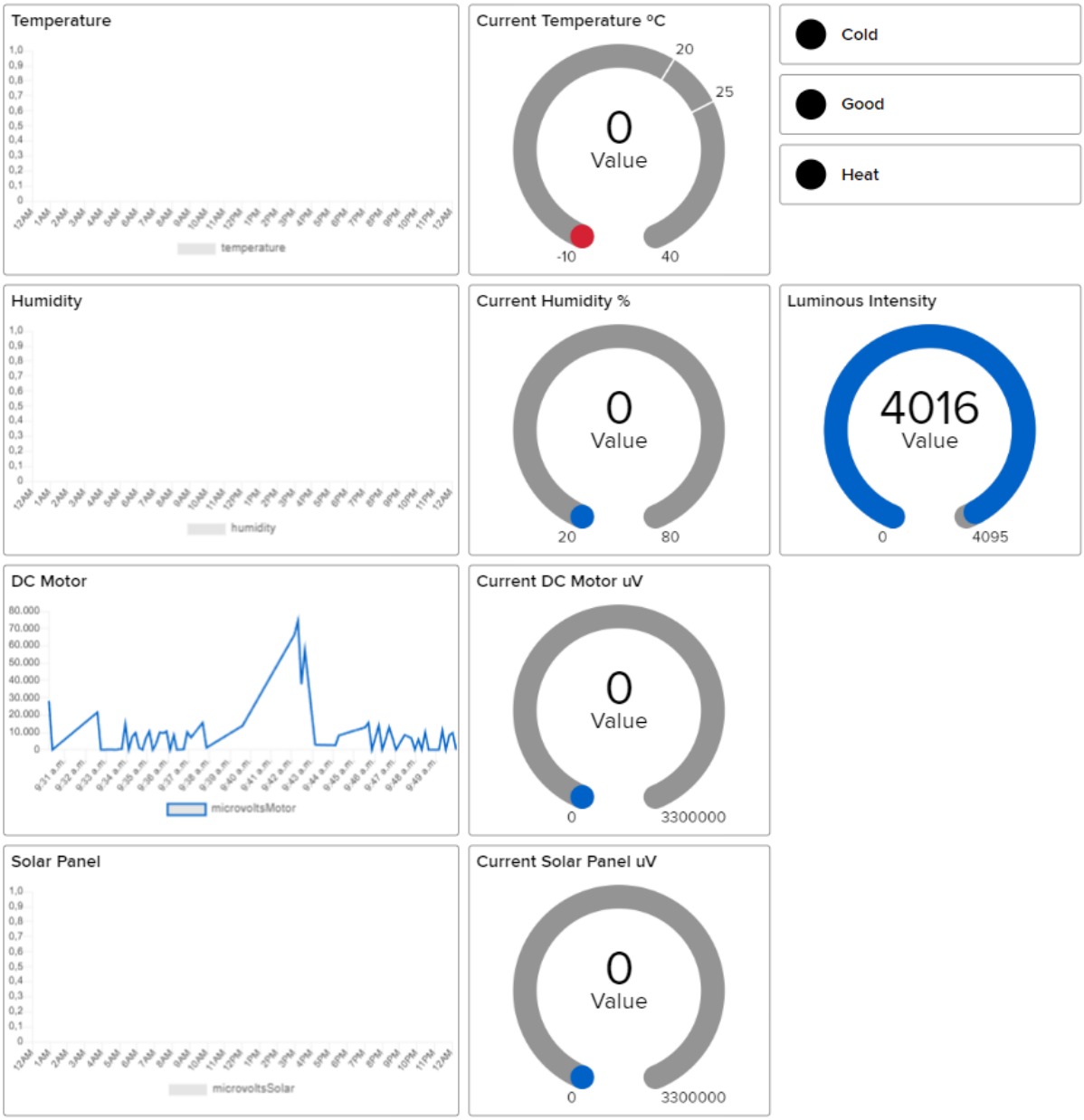
When we attempt to create a new block, a list of possible charts and indicators opens that can be included in our Dashboard. To represent the data in Adafruit IO, we need to create what they call Feeds, or data sources. These store the information sent or received from different devices connected to the platform. The Feeds we will need are: **temperature, humidity, ldr, ldr1Feed, ldr2Feed, microvoltsMotor, microvoltsSolar, ledHighFeed, ledGoodFeed, and ledColdFeed**.

We will use linear graphs, gauges, and indicators to represent the information received from the ESP32 of the ESP32 we are using as follows:

- Linear graph to represent Temperature in ºC over time.
- Linear graph to represent Relative Humidity in % over time.
- Linear graph to represent the voltage generated by the DC Motor in µV over time.
- Linear graph to represent the voltage generated by the Solar Panel in µV over time.
- Gauge to represent the current Temperature in ºC.
- Gauge to represent the current Relative Humidity in %.
- Gauge to represent the current voltage generated by the DC Motor in µV.

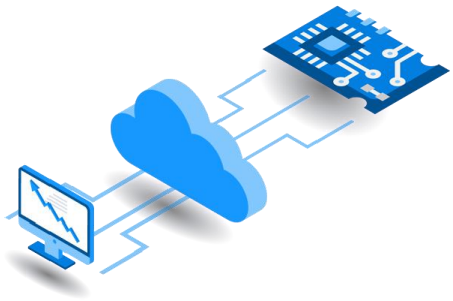
- Gauge to represent the current voltage generated by the Solar Panel in µV.
- Gauge to represent the current Average Light Intensity.
- Indicator to represent the status of the Cold LED.
- Indicator to represent the status of the Good LED.
- Indicator to represent the status of the Heat LED.

The goal will be to obtain a Dashboard at the end like the following:

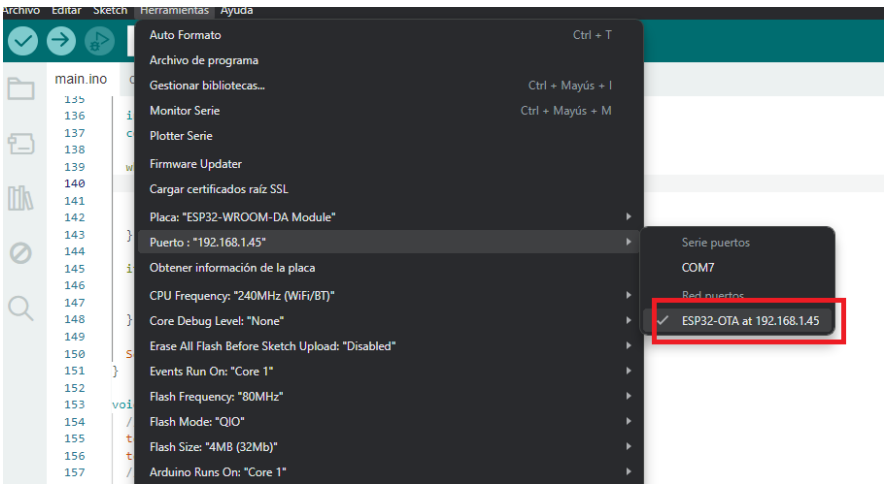


7. OTA (Over-The-Air) UPDATES

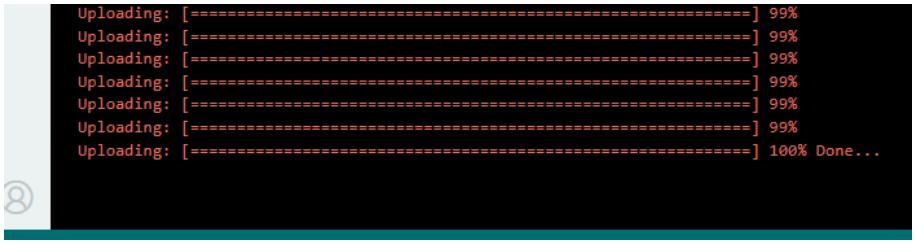
OTA updates allow for **the wireless updating of a device's firmware** without the need to physically connect it to a computer. This is particularly useful in IoT devices, such as the ESP32, which are often located in remote or hard-to-access areas. With OTA updates, you can upload new code or improvements to the device without having to disassemble it or connect it via a USB cable.



To use OTA updates, we must first upload the complete code of the Demonstrators to our ESP32 board. Once uploaded and the connection to the Wi-Fi network and the Adafruit IO platform is established, we will be able to select an OTA port in the Arduino IDE (under Tools > Port) corresponding to the ESP32 we are using (it will have a name like "**ESP32-OTA at**" followed by the device's IP address on the Wi-Fi network).



Now, if you want to upload code via OTA updates, simply click the upload button. You will be prompted for a password (which is "admin" by default) and the program will then begin compiling. The first time you use OTA, the Operating System may request permission to carry out the task. Simply allow the application to run, and you will be able to upload code to your ESP32 board connected to the WiFi network without needing USB cables whenever you need.



If at any point you need to change the name of the board when it is listed in the available ports or the password for performing the remote update, simply go to the source code of the Demonstrators and replace the values inside the parentheses and double quotes that you can see in the following lines (located within the ***setupOTA()*** function):

```
// Initialize ArduinoOTA for over-the-air updates
ArduinoOTA.setHostname("ESP32-OTA");
ArduinoOTA.setPassword("admin");
```

8. TYPICAL PROBLEMS

Problem	Verification	Solution
I can't upload the program to the ESP32 board.	Check that the USB cable is properly connected at both ends.	Disconnect and reconnect the cable properly at both ends.
	Check that the cable you're using is not only for charging.	Replace the cable with one that allows data transfer.
	Check that your computer's USB port is functioning correctly.	Change the USB port.

I get an error when compiling the program.	Check that you have selected the correct COM port for the ESP32 board.	Select the COM port where the ESP32 is connected.
	Check that you have installed the necessary drivers (CP210X).	Install the drivers as indicated in section 3.1 Minimum Requirements.
	Check if the error is related to a library.	Install the necessary libraries as indicated in section 3.2 Necessary Libraries.
	Check if the error is related to the syntax of the code.	Review the lines where you encounter the error and make corrections by comparing with the original code on GitHub.
The ESP32 doesn't connect to the WiFi network.	Check the credentials defined to connect to the WiFi.	Edit the WiFi network credentials (SSID and password) to the ones you want to use.
The ESP32 doesn't send data to Adafruit IO.	Check the credentials defined to connect to the platform.	Edit the platform credentials (Key and username) to the ones you want to use.
The ESP32 board doesn't turn on.	Check that it is connected by cable to an appropriate power source.	Connect the board to a USB port or another 5V power source.
The LEDs don't work properly.	Check that they are properly connected on the breadboard.	Connect each LED terminal in the correct place as indicated in the diagrams.
	Check that they are properly polarized.	Connect the longer terminal of the LED to the positive side and the shorter to the negative or GND.
	Check that the limiting resistor is properly connected.	Connect a resistor of at least 220 ohms in series with the LED.
The DHT sensor doesn't provide readings.	Check that it is properly powered.	Power the DHT with 3.3V on the positive terminal and GND on the negative.
	Check that the data terminal is connected to the same pin on the board as indicated in the program.	Connect the data terminal to the corresponding pin (D32)..

The LDR sensors don't provide reasonable measurements.	Check that the model of the DHT sensor is correct.	Use a DHT-22 sensor or change the DHT sensor type in the program.
	Check that the LDRs are properly connected.	Properly connect the LDRs in a voltage divider if using them independently or directly to the appropriate board pins if using a module.
	Check that the LDRs are connected to the correct board pins.	Connect the LDRs to the board pins that allow analog readings (D34 and D35).
The Servomotor doesn't work.	Check the power supply to the Servomotor.	Connect the positive power of the Servomotor to the 5V pin and the negative to GND.
	Check that the Servomotor is properly connected to the control pin.	Connect the control signal terminal of the Servomotor properly to D25.
The voltage detector doesn't provide reasonable measurements.	Check that the module is properly powered.	Power the voltage detector module with 3.3V and GND.
	Check that the data terminal is properly connected.	Correctly connect the analog output signal of the module to pin D36.
The measurements obtained from the solar panel are not logical.	Check the orientation of the solar panel.	Properly connect the solar panel to the voltage detector module, matching the positive and negative terminals.

MECHANICAL PROBLEMS		
The L-shaped blades do not move independently.	Check the spacing between the disks.	If necessary, slightly pull out the wooden dowels from the large disks to reduce friction.
I want to disassemble the blades and parts of my demonstrator.	Check how to lift the pieces and then remove them.	To disassemble the blades and parts of the demonstrators, follow the same steps you used for assembly but in reverse order.
The base does not fit with	Check that the drawer is properly aligned with	Align the drawer with the slot

the demonstrator's drawer.	the base.	in the base to fit them together.
The motor doesn't have wires to connect it to my setup.	Check if the motor has terminals for soldering wires.	Solder two wires with a soldering iron, one to each terminal of the DC motor..
The motor doesn't generate any voltage when it moves.	Check that it is connected properly.	Properly connect the motor by attaching one terminal to the input of the voltage divider and the other terminal to GND.

EDUDEMOS

EDUcating through Sustainable DEMOnstrators

Funded by
the European Union

info@edudemos.eu

@edudemos

@edudemos

www.edudemos.eu