Smart Bird House

Momen Abu Omar (30247), Tyreka Russell (26150), Sonika Sohal (28105)

Introduction

A smart bird house with computer vision can be used across various fields, like ornithology, for different purposes including species identification, behavioral patterns, feeding habits, migratory patterns and even population dynamics.

The aim of this project is to use the Arduino platform to allow the smart bird house to monitor bird activity remotely. This is done by capturing images when motion is detected. A small camera, combined with an ESP32 microcontroller, and a Passive Infra-Red (PIR) motion sensor which is compatible with an Arduino microcontroller can successfully carry out this task. All images captured are stored on a microSD card which can be easily accessed later. For this project to remain environmentally friendly, a solar panel is used to charge a Lithium Polymer (LiPo) charger which charges a battery. This is also connected to the microcontroller allowing it to be powered.

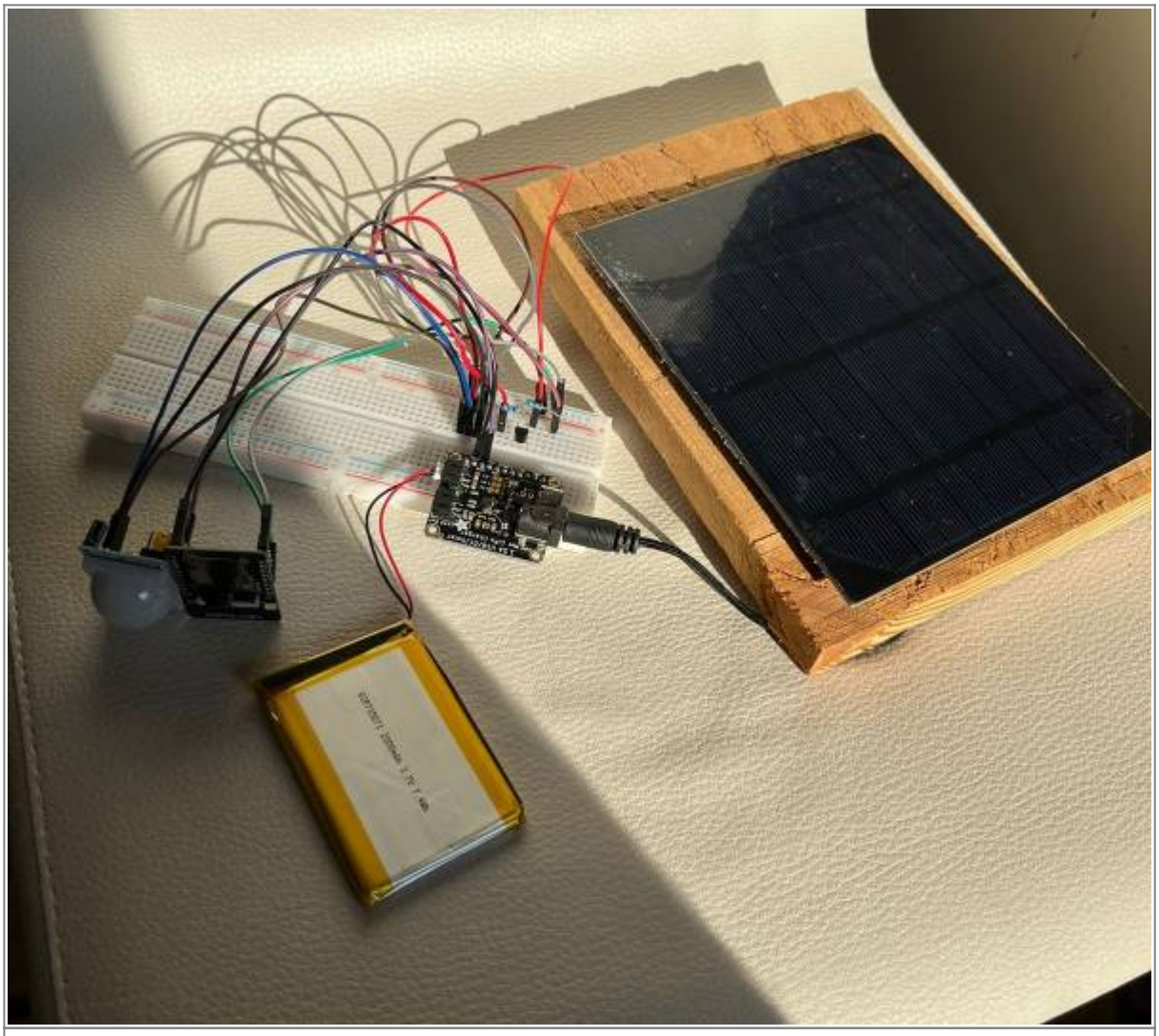


Figure 1. Complete setup of smart bird house

Hardware and Software Components

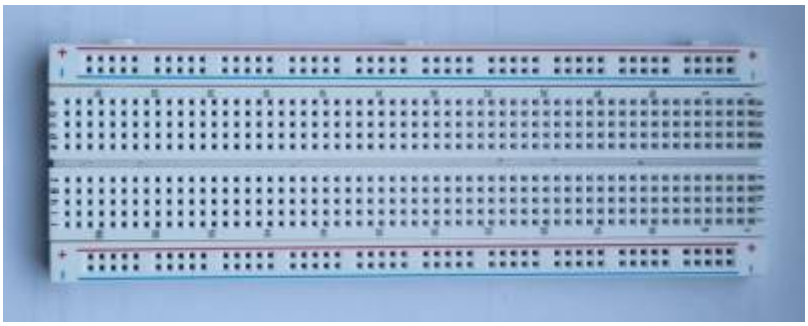
Arduino and Arduino IDE

Arduino is an ‘open-source electronics prototyping platform’ that combines the usage of both hardware and software components. It allows the building of innovative electronic systems and devices in a beginner-friendly manner.

The Arduino Integrated Development Environment (IDE) is an important software which is used to write and compile a code to be uploaded on most Arduino modules, including a microcontroller board. It was intended to make developing projects easier, even with a library of some already built-in sketches to use with sensors, actuators and other compatible components. The Arduino uses a programming language similar to C++ to enable these connections between hardware and software.

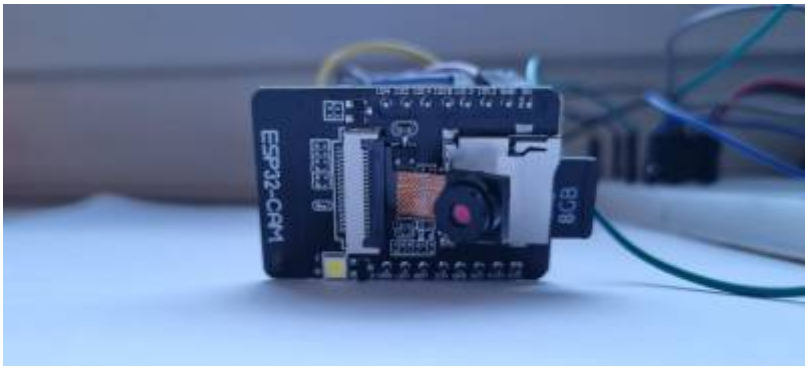
Breadboard

The breadboard helps with easy electrical connections to multiple electronic components, like the ESP32 camera, PIR motion sensor and the Solar LiPo Charger, without needing to continuously solder the necessary parts. This also allows the connections to be temporary and easy to disassemble and reassemble. The jumper wires also aid significantly in these connections as seen in the photo below.



ESP32 Camera

The ESP32 camera is a combination of an ESP32 microcontroller and an OV2640 2 Megapixels camera module. This combination is advantageous in this project because it is low cost, low power and easily adjustable. The camera is easily programmed using the Arduino IDE software to write and upload a set of instructions like capturing images and even videos. The microcontroller is equipped with Wi-Fi and Bluetooth potential which makes it easier for DIY or IoT projects. Even though the ESP32 camera had the ability to transmit images or videos to a secondary location using its Wi-Fi and Bluetooth features, it was not incorporated into this project. Since the camera module was equipped with a microSd card slot, which is featured on the ESP32 S chip, this was our preferred method of retrieving the data. Several GPIOs can also be found on the ESP32 camera to connect the sensors and peripherals.



For more technical data

PIR Motion Sensor

The Passive Infrared (PIR) motion sensor is a module used to detect changes in infrared radiation which is emitted by humans and animals. This sensor is called ‘passive’ because it simply detects the changes in infrared radiation and does not emit it themselves. The digital output signal provided by the sensor is HIGH (1) when it detects motion and LOW (0) when no motion can be detected. The PIR motion sensor used at first has two noticeable potentiometers: one to adjust sensitivity and the other to adjust the delay time. The Seeedstudio mini PIR motion sensor produced a better output for this project. Its 20mm x 20mm x 12mm size made it easier and more flexible to handle. The most optimal detection distance is 2m but this sensor has a range of up to 5m.

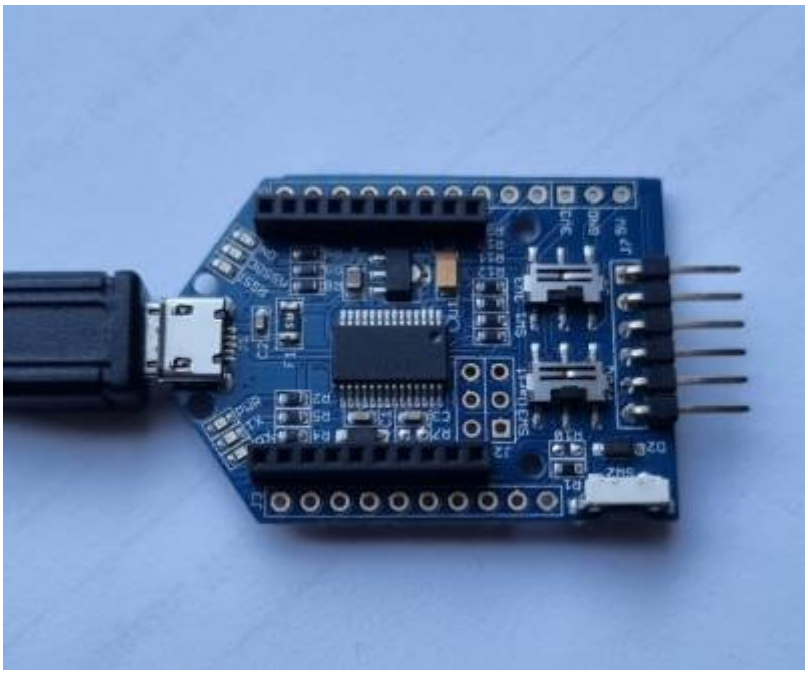


Important technical details

UartSBee V5

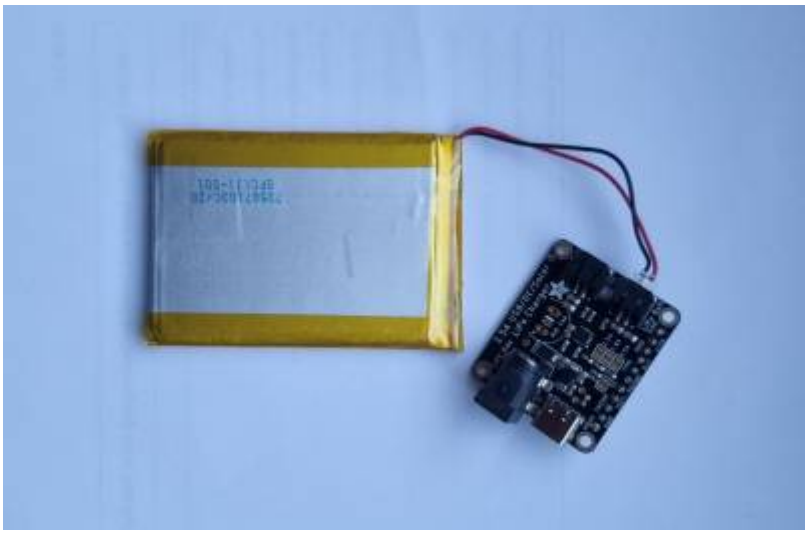
In order to configure the ESP32 camera with Arduino, the UartSBee V5 was used. It is a device that establishes communication between the USB port of a computer and a serial (UART) interface of a component or system. This facilitates programming, debugging, monitoring and uploading of the

necessary software for the interaction to occur. The UartSBee V5 configuration of the ESP32 camera with Arduino therefore allowed the code required to be saved and aided in the connection of the motion sensor.



1.5 USB/DC/Solar LiPo Charger

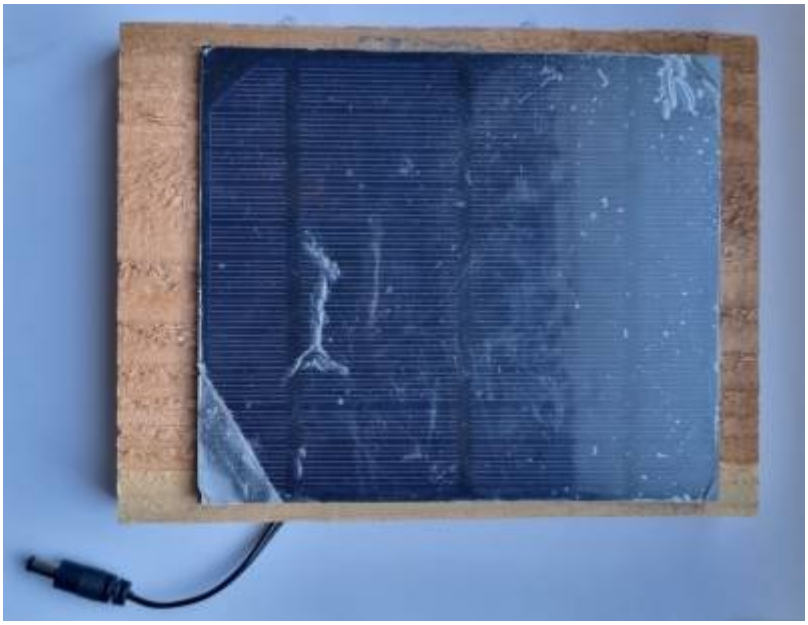
The Solar LiPo Charger charges the Lithium Polymer battery using a few different power sources like USB, DC power supply or solar panels. Keeping with an environmental theme for this project, a solar panel was used. This charging module is therefore charged by utilizing solar energy from the Sun and this makes it suitable for this outdoor birdhouse. These chargers also contain built-in protection systems to prevent overcharging or over-discharging of the battery.



Further features and specifications

Solar Panel

Solar panels, or photovoltaic panels, harness the sunlight in order to convert it into electricity. The solar panel consists of multiple solar cells which are all connected to increase the overall current and voltage output to be used to power electronic devices or even charging batteries, as is done for this project. This source of sustainable and renewable energy reduces and mitigates severe environmental impact.

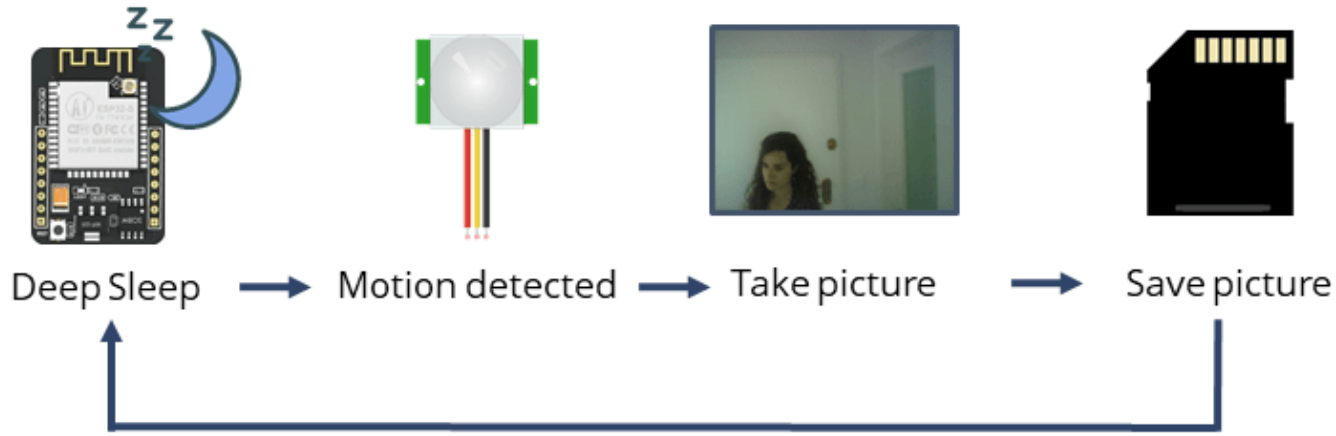


MicroSD card

The SD card is a small, compact memory card that can be easily removed from electronic devices. This card comes in a variety of storage capacity for data. The SD card used for this project is 8 GB. It is enough so that after a couple of days, the card can be removed and read on a computer using an adapter.

Execution

1. The ESP32 camera goes into deep sleep mode until it receives another signal stating that motion is being detected.
2. Motion is detected by the PIR motion sensor. It consequently sends a signal to the ESP32 camera to be activated.
3. When the camera receives the signal, it wakes up and takes a photo.
4. The photo is saved to the 8 GB microSD card which is later accessed.



Deep sleep mode of the ESP32 reduces the amount of power consumed while in operation, but enough power is left to keep the processor and RTC peripherals running. When in deep sleep mode, the CPU, Wi-Fi and bluetooth are disabled. This power-saving feature is quite useful in extending the battery life. When the ESP32 camera is stimulated, in this case by the motion sensor, it will wake up and function as normal.

Code

The following code, which is modified from Random Nerd Tutorials, is an Arduino sketch specifically for the ESP32 camera to capture images and save it to an SD card while also utilizing the deep sleep mode. This code includes several libraries in order to support the functioning of the camera, including 'esp_camera.h'.

- In order for the code to be uploaded, it was important to select the 'AI Thinker ESP32-CAM' board.
- GPIO 0 must be connected to GND in order to successfully upload the sketch.
- Baud rate of 115200 bits per second was used to initialize the serial communication.

```
//Libraries to support the functioning of the camera, access to the microSD
card and its power-saving feature.
#include "esp_camera.h"
#include "Arduino.h"
#include "FS.h"                // SD Card ESP32
#include "SD_MMC.h"            // SD Card ESP32
#include "soc/soc.h"           // Disable brownout problems
#include "soc/rtc_cntl_reg.h"  // Disable brownout problems
#include "driver/rtc_io.h"
#include <EEPROM.h>            // read and write from flash memory

#define EEPROM_SIZE 1          // defined to specify the number of bytes to
be accessed
```

```
RTC_DATA_ATTR int bootCount = 0;

// Pin definition for CAMERA_MODEL_AI_THINKER
#define PWDN_GPIO_NUM     32
#define RESET_GPIO_NUM    -1
#define XCLK_GPIO_NUM      0
#define SIOD_GPIO_NUM     26
#define SIOC_GPIO_NUM     27
#define Y9_GPIO_NUM       35
#define Y8_GPIO_NUM       34
#define Y7_GPIO_NUM       39
#define Y6_GPIO_NUM       36
#define Y5_GPIO_NUM       21
#define Y4_GPIO_NUM       19
#define Y3_GPIO_NUM       18
#define Y2_GPIO_NUM        5
#define VSYNC_GPIO_NUM    25
#define HREF_GPIO_NUM     23
#define PCLK_GPIO_NUM     22

int pictureNumber = 0;
void setup() {
  WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0); //disable brownout detector
  Serial.begin(115200);                    // the baud rate (115200)
determines the rate that the signal changes (data transmission) in the
initializationof serial communication.
  Serial.setDebugOutput(true);

  camera_config_t config;     // camera is configured using this in order to
specify the pins and the desirable settings for the camera.
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sscb_sda = SIOD_GPIO_NUM;
  config.pin_sscb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.pixel_format = PIXFORMAT_JPEG;
  pinMode(4, INPUT);
```

```
  digitalWrite(4, LOW);
  rtc_gpio_hold_dis(GPIO_NUM_4);

  if(psramFound()){
    config.frame_size = FRAMESIZE_UXGA; // FRAMESIZE_ +
QVGA|CIF|VGA|SVGA|XGA|SXGA|UXGA
    config.jpeg_quality = 10;
    config.fb_count = 2;
  } else {
    config.frame_size = FRAMESIZE_SVGA;
    config.jpeg_quality = 12;
    config.fb_count = 1;
  }

  // Init Camera
  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
  }

  Serial.println("Starting SD Card");

  delay(500);
  if(!SD_MMC.begin()){    // Initializes and checks that the SD card is
mounted well
    Serial.println("SD Card not mounted");
    //return;
  }

  uint8_t cardType = SD_MMC.cardType();
  if(cardType == CARD_NONE){
    Serial.println("No SD Card found");
    return;
  }
  camera_fb_t * fb = NULL;

  // Take Picture with Camera
  fb = esp_camera_fb_get();
  if(!fb) {
    Serial.println("Camera capture error");
    return;
  }
 // This library is used to create a counter value in order to keep track of
the number of pictures that were taken.
  EEPROM.begin(EEPROM_SIZE);
  pictureNumber = EEPROM.read(0) + 1;

  // Path where new picture will be saved in SD Card
  String path = "/picture" + String(pictureNumber) +".jpg";
```

```
  fs::FS &fs = SD_MMC;
  Serial.printf("Picture file name: %s\n", path.c_str());

  File file = fs.open(path.c_str(), FILE_WRITE);  // Allows the photo to be
saved with  file name and picture number.
  if(!file){
    Serial.println("Failed to open file in writing mode");
  }
  else {
    file.write(fb->buf, fb->len); // payload (image), payload length
    Serial.printf("Saved file to path: %s\n", path.c_str());
    EEPROM.write(0, pictureNumber);
    EEPROM.commit();
  }
  file.close();
  esp_camera_fb_return(fb);
  delay(1000);
  // Turns off the ESP32-CAM white on-board LED (flash) connected to GPIO 4
  pinMode(4, OUTPUT);
  digitalWrite(4, LOW);
  rtc_gpio_hold_en(GPIO_NUM_4);

  esp_sleep_enable_ext0_wakeup(GPIO_NUM_13, 0);

  Serial.println("Deep sleep mode loading");
  delay(1000);
  esp_deep_sleep_start();     // After saving a photo the camera goes into
deep sleep mode and is only activated by an external interruption to wake it
up again.
  Serial.println("Sleeping");
}

void loop() {   // Loop function is not utilized here because deep sleep
mode is activated in the code and therefore does not need to be looped after
the first run.

}
```

Results



In the end, the system worked harmoniously only when charged using the solar panel but overheated when charged with a computer. The figure above shows some photos that were taken when motion was detected. It was also clearly seen that the photo counter worked well as the images were labelled in its respective order. It was also noted that there was a delay with the sensor causing a delay in the photos taken even though movement was constant. There could have been an error due to faulty components or that the general setup had a wiring mishap. With more time and more discussions, the bird house can function without errors.

Challenges and Improvements

The ESP32 camera and PIR motion sensor was first configured using two separate microcontrollers, i.e. the Arduino Uno board, and then later put together using the breadboard. This was one of our biggest challenges because most of the pins of the microcontroller were used, and transferring it onto the breadboard required a different approach with a different code and a more complicated setup. Additionally, it was needed to program around 3 different brands of motion sensor because they were not working. In the end, it was still difficult to get the motion sensor to function continuously. It could also be that the components used were defective.

References

- https://randomnerdtutorials.com/esp32-cam-pir-motion-detector-photo-capture/
- https://www.adafruit.com/category/17
- https://randomnerdtutorials.com/esp32-cam-take-photo-save-microsd-card/
- https://RandomNerdTutorials.com/esp32-cam-pir-motion-detector-photo-capture/
- https://cpham.perso.univ-pau.fr/LORA/HUBIQUITOUS/solution-lab/arduino-lora-tutorial/introduction_arduino_ide/introduction_arduino_ide/
- https://arduinointro.com/articles/projects/learn-how-to-use-a-breadboard?utm_content=cmp-true
- https://www.berrybase.de/en/esp32-cam-development-board-inkl.-ov2640-kameramodul
- https://energysavingtrust.org.uk/advice/solar-panels/
- https://lastminuteengineers.com/esp32-sleep-modes-power-consumption/

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

Permanent link:
https://student-wiki.eolab.de/doku.php?id=amc:ss2023:group-j:start&rev=1690319992

Last update: **2023/07/25 23:19**