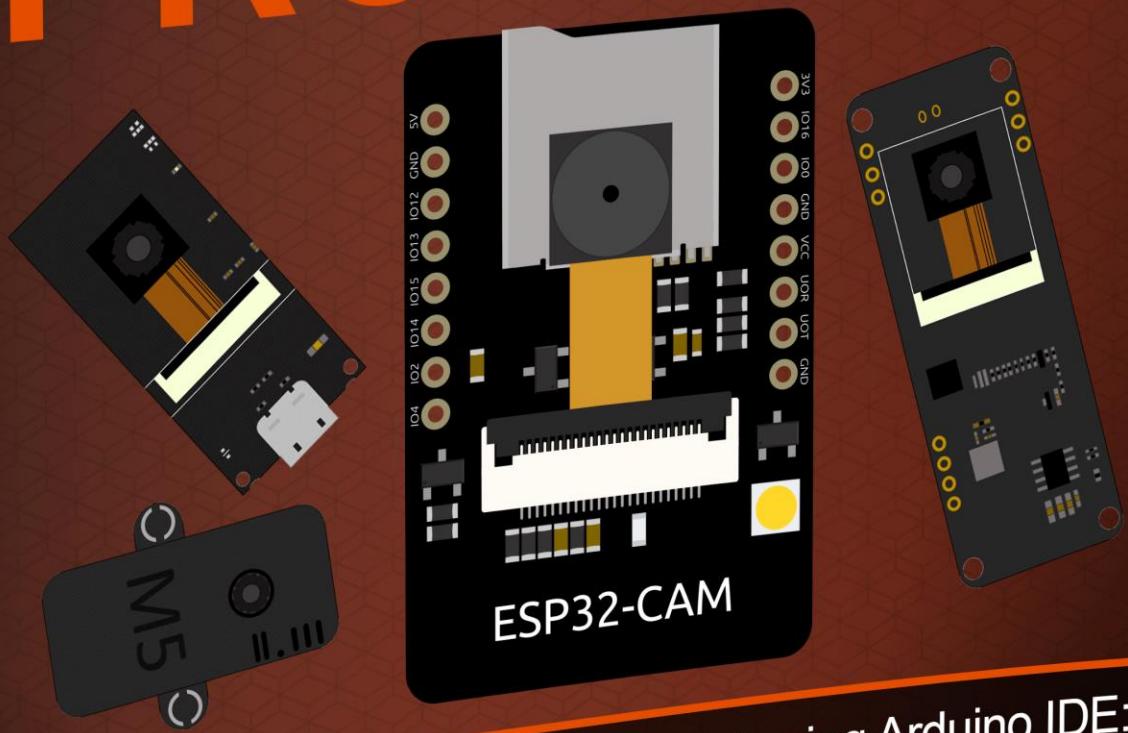


ESP32-CAM PROJECTS



Build Projects with the ESP32-CAM using Arduino IDE: Photo Capture, Web Servers, Email Notifications, Video Streaming, Face Detection, Face Recognition and more

RUI SANTOS and SARA SANTOS

Version 1.1

Security Notice

Sorry for having to write this notice, but the evidence is clear: piracy for digital products is over the entire internet.

For that reason, we've taken certain steps to protect our intellectual property contained in this eBook.

This eBook contains hidden random strings of text that only apply to your specific eBook version that is unique to your email address. You won't see anything different, since those strings are hidden in this PDF. We apologize for having to do that – but it means if someone were to share this eBook we know exactly who shared it and we can take further legal consequences.

You cannot redistribute this eBook. This eBook is for personal use and is only available for purchase at:

- <https://randomnerdtutorials.com/courses>
- <https://rntlab.com/shop>

Please send an email to the author (Rui Santos - hello@ruisantos.me), if you find this eBook anywhere else.

What we really want to say is thank you for purchasing this eBook and we hope you learn a lot and have fun with it!

Disclaimer

This eBook was written for information purposes only. Every effort was made to make this eBook as complete and accurate as possible. The purpose of this eBook is to educate. The authors (Rui Santos and Sara Santos) do not warrant that the information contained in this eBook is fully complete and shall not be responsible for any errors or omissions.

The authors (Rui Santos and Sara Santos) shall have neither liability nor responsibility to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by this eBook.

Throughout this eBook you will find some links and some of them are affiliate links. This means the authors (Rui Santos and Sara Santos) earn a small commission from each purchase with that link. Please understand that the authors have experience with all those products, and recommend them because they are useful, not because of the small commissions. Please do not spend any money on products unless you feel you need them.

Other Helpful Links:

- [Ask questions in our Forum](#)
- [Join Private Facebook Group](#)
- [Terms and Conditions](#)

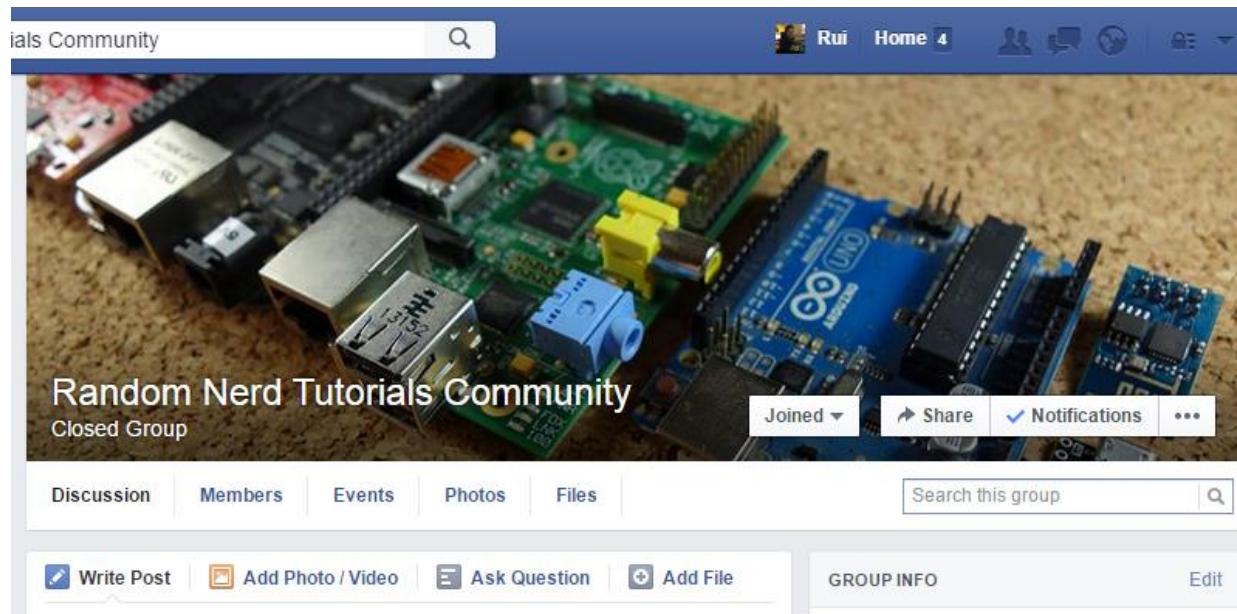
Join the Private Facebook Group

This course comes with an opportunity to join a private community of like-minded people. If you purchased this course, you can join our private Facebook Group today!

Inside the group, you can ask questions and create discussions about everything related to ESP32, ESP8266, Arduino, Raspberry Pi, etc.

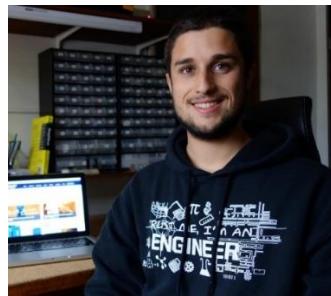
See it for yourself!

- Step #1: Go to -> <http://randomnerdtutorials.com/fb>
- Step #2: Click “Join Group” button
- Step #3: We’ll approve your request within less than 24 hours.



About the Authors

This eBook was developed and written by Rui Santos and Sara Santos. We both live in Porto, Portugal, and we know each other since 2009. If you want to learn more about us, feel free to read our [about page](#).



Hi! I'm Rui Santos, the founder of the Random Nerd Tutorials blog. I have a master's degree in Electrical and Computer Engineering from FEUP. I'm the author of "[BeagleBone For Dummies](#)", and Technical reviewer of the book "Raspberry Pi For Kids For Dummies". I wrote a book with Sara Santos for the [NoStarchPress publisher](#) about projects with the Raspberry Pi: "[20 Easy Raspberry Pi Projects: Toys, Tools, Gadgets, and More!](#)"



Hi! I'm Sara Santos! I started working at Random Nerd Tutorials back in 2015 as a hobby: I helped Rui with some simple tasks when he had a lot of work to do. Back then, I knew nothing about electronics, programming, Arduino, etc... Over time, I started learning everything I could about those subjects and I just loved it! At first, I helped Rui once a week on Saturdays, but then, I started working on the RNT blog alongside him, almost every day. Currently, I work full time at Random Nerd Tutorials and I love what I do!

Table of Contents

| | |
|---|------------|
| MODULE 0..... | 9 |
| Course Intro | 9 |
| Welcome to ESP32-CAM Projects! | 10 |
| MODULE 1..... | 12 |
| Getting Started | 12 |
| Unit 1: Getting Started with ESP32-CAM | 13 |
| Unit 2: Preparing Arduino IDE for the ESP32-CAM..... | 37 |
| Unit 3: Camera Web Server (Video Streaming and Face Recognition) | 44 |
| Unit 4: Troubleshooting Most Common Problems | 59 |
| Unit 5: ESP32-CAM Flashlight and External Pushbutton..... | 73 |
| MODULE 2..... | 84 |
| Take Photos: Time-lapse, Camera Settings, Web Server, SD Card Manager .. | 84 |
| Unit 1: Take Photos and Save to MicroSD Card (Time-lapse)..... | 85 |
| Unit 2: Photo Filename with Date and Time Saved to MicroSD Card..... | 102 |
| Unit 3: Change Camera Settings | 112 |
| Unit 4: Take Photo and Save to MicroSD Card with Pushbutton | 117 |
| Unit 5: Take Photo and Display in Web Server | 129 |
| Unit 6: Web Server SD Card Photo Manager: Capture, View and Delete | 146 |
| MODULE 3..... | 170 |
| Take Photos and Send Notifications | 170 |

| | |
|---|------------|
| Unit 1: Send Photos via Email..... | 171 |
| Unit 2: Motion Detector with Photo Capture and Email Notifications | 189 |
| Unit 3: Take and Email Photo with a Web Server | 202 |
| Unit 4: Take and Send Photo to Telegram App..... | 219 |
| Unit 5: Motion Detector with Photo Capture and Telegram Notifications | 241 |
| MODULE 4..... | 258 |
| Video Streaming, Pan and Tilt, Car Robot | 258 |
| Unit 1: Video Streaming Web Server (Start and Stop Buttons) | 259 |
| Unit 2: Video Streaming Web Server with Sensor Readings..... | 270 |
| Unit 3: Video Streaming IP Camera..... | 289 |
| Unit 4: Remote Controlled Car Robot with Camera (Web Server) | 302 |
| Unit 5: Pan and Tilt Video Streaming (2 Axis)..... | 321 |
| Unit 6: ESP32-CAM IP Camera Access from Anywhere in the World | 338 |
| MODULE 5..... | 365 |
| Face Detection and Face Recognition | 365 |
| Unit 1: Face Detection Video Streaming (Age, Face Expression, etc.)..... | 366 |
| Unit 2: Face Recognition (Comparing Two Photos) | 380 |
| EXTRA UNITS..... | 394 |
| ESP32-CAM Access Point..... | 395 |
| ESP32-CAM Static IP Address..... | 402 |
| APPENDIX..... | 406 |
| Pinout for ESP32 Camera Boards | 407 |

| | |
|-------------------------------|-----|
| List of Parts Required | 415 |
| Other RNT Courses/eBooks..... | 417 |

MODULE 0

Course Intro

Welcome to ESP32-CAM Projects!

Welcome to “ESP32-CAM Projects” eBook. This eBook is a collection of 20 practical projects with the ESP32-CAM using Arduino IDE.

After an introduction to the ESP32-CAM, you’ll build projects that cover: photo capture, save photos to microSD card, web servers, email and Telegram notifications with photos, video streaming, car robot with camera, pan and tilt, face detection, face recognition, and much more.

How to Follow this Course?

If you’re completely new to the ESP32-CAM, we recommend following the eBook in order. It is mandatory to complete Module 1 first. Otherwise, you won’t know how to upload code to the board or open the Serial Monitor. We start with a detailed getting started guide, and then we’ll introduce new concepts along the way.

If you already have some experience programming the ESP32-CAM with Arduino IDE, you can skip the most basic sections and go straight to the Modules you find more interesting.

Download Source Code and Resources



Each section contains the code, schematics, and all the resources you need to follow the projects. You can download each resource by clicking the “Code” box on each project, or you can [download the ESP32-CAM Projects eBook repository](#) and instantly download all the resources for this eBook.

Getting Parts

To follow the projects, you need some electronics components. In each section, we provide a list of the needed parts and links to [Maker Advisor](#), so that you can find the part you're looking for on your favorite store at the best price.



If you buy your parts through Maker Advisor links, we'll earn a small affiliate commission (you won't pay more for it). By getting your parts through our affiliate links you are supporting our work. If there's a component or tool you're looking for, we advise you to take a look at [our favorite tools and parts here](#).

Note: for the full parts list, consult the [Appendix here](#).

Leave Feedback

Your feedback is very important so that we can improve the eBook and our learning materials. Suggestions, rectifications, and your opinion is very important for us.

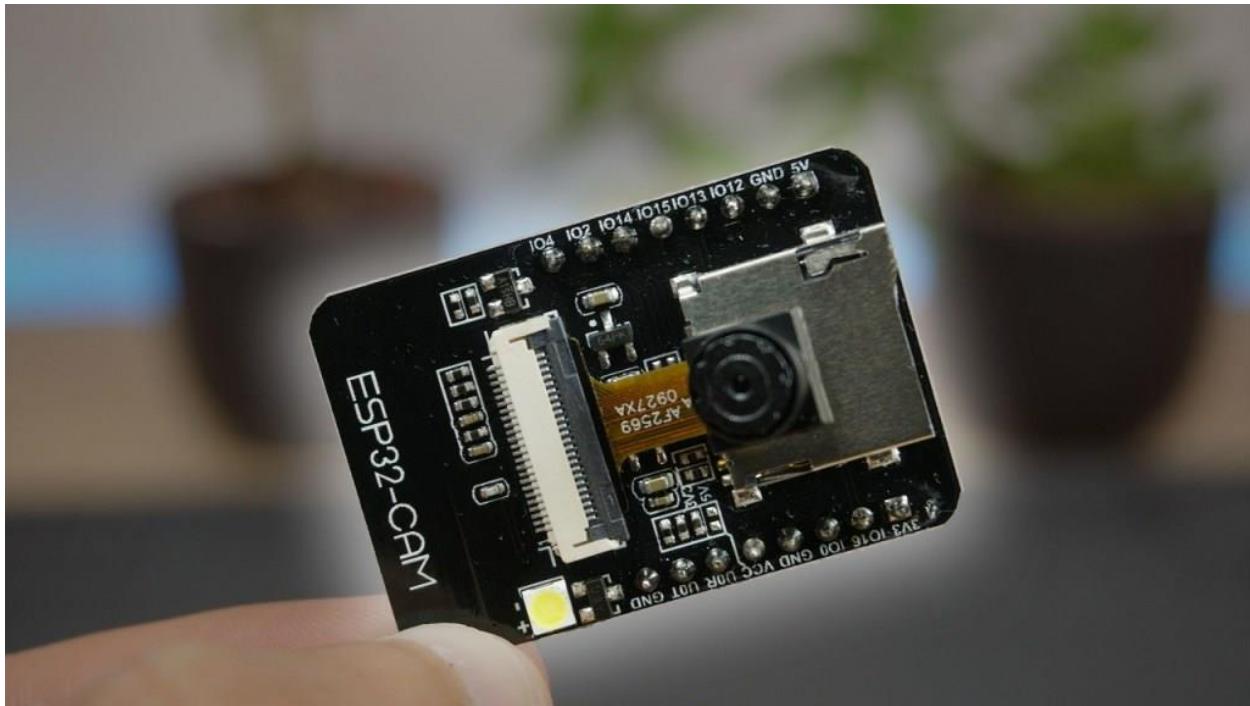
You can use the following channels to leave feedback:

- [Facebook group](#)
- [RNT Lab Forum](#)

MODULE 1

Getting Started

Unit 1: Getting Started with ESP32-CAM



Introducing the ESP32-CAM

The ESP32-CAM is a development board with an ESP32-S chip, an OV2640 camera, several GPIOs to connect peripherals and a microSD card slot that can be useful to store images taken with the camera or to store files to serve to clients.



It allows you to set up a video streaming web server, build a surveillance camera to integrate with your home automation system, do face detection and recognition, and many other applications in the Smart Home and Internet of Things (IoT) fields.



If you're not familiar with the ESP32, it's a dual core "chip" from Espressif that combines Wi-Fi and Bluetooth wireless capabilities.



There are many ESP32 development boards with the ESP32 chip. These differ in the number of accessible GPIOs, USB-to-UART interface (some boards have it, some don't), BOOT and RESET buttons, battery connector, and other extra features, like built-in OLED display, LoRa chips, SD card, etc. For a comparison of some of the most popular ESP32 development boards, read the following article:

- [ESP32 Development Boards Review and Comparison](#)

The ESP32-CAM is *just* another ESP32 development board with its own features:

- OV2640 camera;
- No USB-to-UART interface;
- Reset button (labelled as RST or EN);
- 10 accessible GPIOs;
- 4MB PSRAM;
- MicroSD card interface.



Like the regular ESP32 development board, the ESP32-CAM can be programmed using Arduino IDE and the Arduino core for the ESP32. Throughout this eBook, we'll be covering how to program the ESP32-CAM using Arduino IDE.

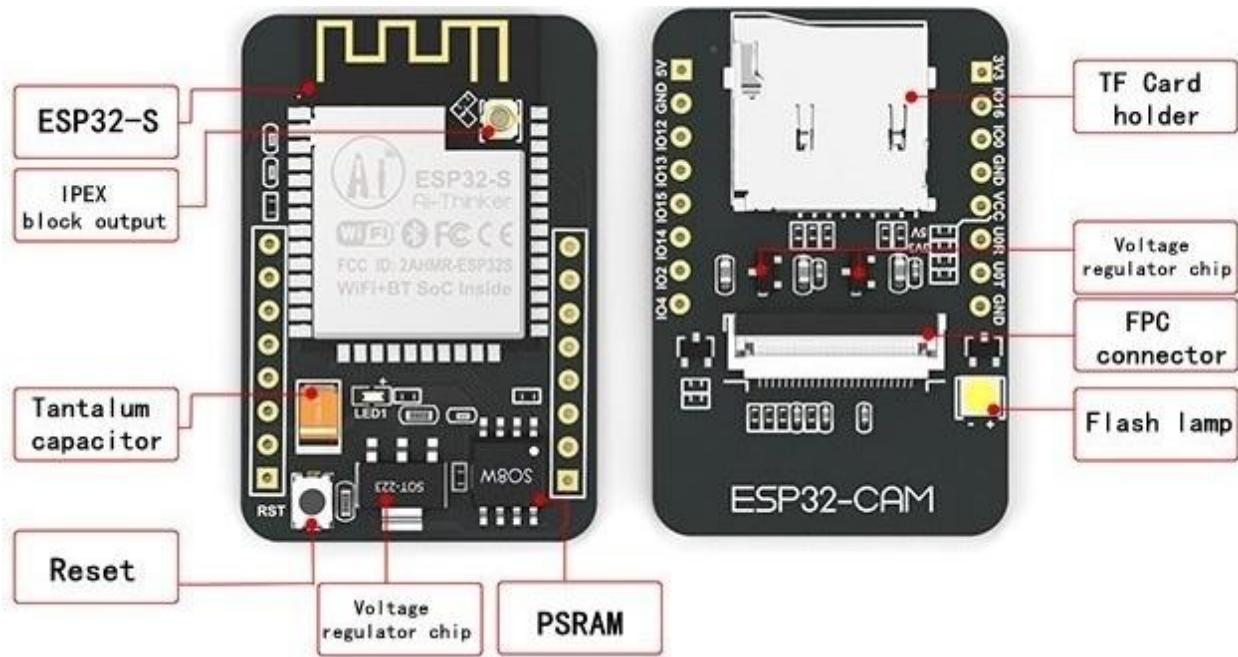
ESP32-CAM Specifications

The following table shows the ESP32-CAM specifications. *Table adapted from the ESP32-CAM datasheet.*

| | |
|------------------------------|--|
| SPI Flash | Default 32 Mbit |
| RAM | 520 KB SRAM + 4M PSRAM |
| Bluetooth | Bluetooth 4.2 and Bluetooth Low Energy (BLE) |
| Wi-Fi | 802.11 b/g/n |
| Support Interface | UART, SPI, I2C, PWM |
| IO port | 10 |
| UART Baud Rate | Default 115200 bps |
| Image Output Format | JPEG (OV2640 support only), BMP, GRayscale |
| Spectrum Range | 2412~2484MHz |
| Antenna | Onboard PCB, and IPEX connector |
| Transmit Power | 802.11b: 17+/- dBm (@11Mbps) 802.11g: 14+/- dBm(@54Mvps) 802.11n:3+/- dBm (@MCS7) |
| Receiving Sensitivity | CCK, 1 Mbps: -90dBm CCK, 11 Mbps: -85dBm 6 Mbps (1/2 BPSK): -88dBm 54 Mbps, 72.2 Mbps): -67dBm |
| Power Dissipation | Flashlight off: 180mA@5V Flashlight on with maximum brightness: 310mA@5V Deep sleep: minimum power consumption 6mA@5V Modem sleep: minimum up to 20mA@5V Light sleep: minimum up to 6.7mA@5V |
| Security | WPA/WPA2/WPA2-Enterprise/WPS |
| Power Supply Range | 5V |

ESP32-CAM Features

The ESP32-CAM (AI-Thinker) has most of the specs of a regular ESP32 development board but with its own tweaks and features.



To keep in mind about the ESP32-CAM features:

- The ESP32-CAM comes with an **OV2640 camera**. There are camera probes sold separately with longer flex cables and with fish-eye lens. You connect the camera probes in the FPC connector.
- The **TF card holder** supports microSD card interface. The ESP32-CAM datasheet mentions that it only supports microSD cards up to 4GB. However, we've experimented with microSD cards with other sizes (8GB and 16GB) and these worked well. Probably, they mean that it only supports writing up to 4GB (but we haven't tested this yet – you need a lot of photos with the ESP32-CAM to occupy 4GB).

- **No USB-to-UART interface:** this means that you can't connect the ESP32-CAM directly to your computer using an USB cable. You need to use a TTL to USB converter (FTDI programmer).
- **On-board RESET button:** press this button to reset (restart) the ESP32-CAM.
- **On-board antenna and IPEX connector for external antenna:** the ESP32-CAM board comes with an on-board antenna and with an IPEX connector that allows you to use an external antenna to improve Wi-Fi communication range (more information about the antenna in the next sections).
- **On-board LED (flashlight):** the ESP32-CAM comes with an on-board and very bright LED. It can be really useful to add some light to your pictures or video streaming. However, the flashlight shares its GPIO with one of the microSD card GPIOs. This means that when using functions related with the microSD card, the LED will light up occasionally, even when you don't want.
- **Accessible GPIOs:** the ESP32-CAM comes with 10 accessible GPIOs. However, not all GPIOs can be used. For example, GPIO 0 is used to put the board in flashing mode, so you can't use it for other tasks. Additionally, some of the exposed pins are being used either by the camera or by the microSD card. So, you need to be careful with which GPIOs you'll use.
- The ESP32-CAM has **PSRAM:** PSRAM is used for buffering images from the camera into video streaming or other tasks and allows you to use higher quality in your pictures without crashing the ESP32.

Buying an ESP32-CAM

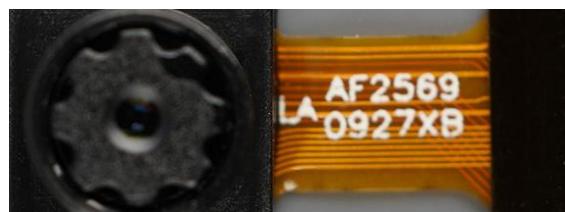
There are several different boards that feature an ESP32 chip and a camera.

AI-THINKER ESP32-CAM

Throughout this course, we'll be using the ESP32-CAM shown below labelled as "ESP32-S AI-THINKER".



There are other similar ESP32-CAM boards that look exactly like this one but don't have the AI-THINKER label. These should be perfectly suited too. Also, the camera probe that comes with our ESP32-CAM boards has a label with the following reference "**LA AF2569 0927XB**". Cameras with other labels should also work as long as they are OV2640 cameras.



Besides the AI-THINKER ESP32-CAM, there are other ESP32 camera development boards that can also be used with this course. However, keep in mind that if you're using a different camera, you might need to modify your code to make it work properly, for example:

- If your camera model doesn't support microSD card, so projects that use microSD card, won't work. Alternatively, you can use the ESP32 SPIFSS to store photos instead of using microSD card in some of the projects;
- If your camera doesn't have accessible GPIOs, you won't be able to connect peripherals like sensors and actuators (motors, relays, sensors, etc.);
- You need to double-check the pinout for the camera board you're using. Different cameras use different GPIOs and if you don't assign them properly in your code, it won't work (refer to the [Pinout guide](#));
- If you're using a camera without PSRAM, you won't be able to use face recognition and detection. Additionally, those cameras may have troubles taking pictures with resolution higher than SVGA (800×600).

Nonetheless, you might want to get a different ESP32 camera development board, because it may be more suitable for the kind of project you want to build. Here's a brief description of some of the most popular ESP32 camera development boards.

ESP-EYE



The ESP-EYE is an ESP32-based board dedicated to artificial intelligence (AI) with voice wake-up and face recognition. Just with 21mm by 41mm, it is equipped with an OV2640 camera, on-board microphone, reset, boot, and function buttons and two LEDs. It features 4MB Flash, 8MB PSRAM, and Micro USB type-C connector (easy to upload code). It comes with on-board antenna and IPEX connector if you want to add an external antenna.

One of the greatest advantages of this board is the USB type-C connector – fast and easy to upload code to the board; the microphone allows you to add voice features to your projects; and 8MB PSRAM ensures that your board doesn't crash when using higher image quality settings. The major drawback is the price.

More details about ESP-EYE:

- [ESP-EYE: ESP32-based board for AI \(voice wake-up and face recognition\)](#)

TTGO T-Journal



The [TTGO T-Journal](#) is a \$12-\$15 ESP32 Camera Development Board with an OV2640 camera, an external antenna, an I2C SSD1306 0.91 inch OLED display, some exposed GPIOs, function button, battery connector and a micro-USB interface.

The OLED display is a great addition to the board. You can display the board IP address, or any errors while debugging. There are four accessible GPIOs. Two of them are for I2C communication and other two are perfect to connect servo motors (you can also connect other peripherals).

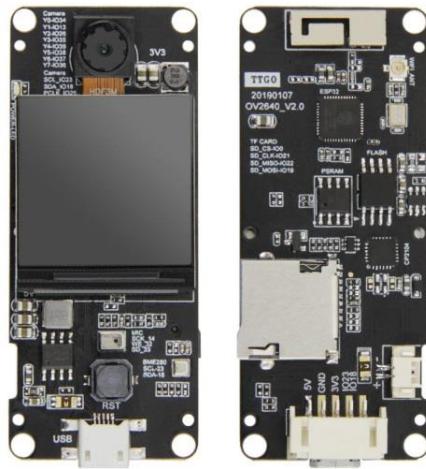
This board also comes with a connector to connect a 3.7V lithium battery. Uploading code is easy with this board because it comes with a USB connector that can be used both for uploading code or for power. You also have a function button connected to GPIO 32 that you program in your code to trigger any task you want.

Its major drawbacks are: it doesn't have microSD card support neither external PSRAM.

More details about the TTGO T-Journal ESP32 board:

- [TTGO T-Journal ESP32 Camera Development Board Review](#)
- [Getting Started with the TTGO T-Camera \(with examples\)](#)

TTGO T-Camera Plus



The TTGO T-Camera Plus comes with all the functionalities we would want in such development board and for a very reasonable price. The board comes with microSD card support, microphone, support for a 3.7V lithium battery as well as battery

management circuit, 1.3 TFT display (color screen), microUSB interface and on-board reset button.

It comes with some GPIOs exposed that were used to connect an on-board BME280 sensor. However, the sensor would get really hot on the board, so the manufacturer decided to remove the sensor but you still get access to the GPIOs, so you can connect other I2C peripherals. These GPIOs are also accessible via grove connector.

Finally, the board has an on-board antenna, but also an IPEX connector if you want to add an external antenna.

More details about the TTGO T-Camera Plus:

- [TTGO T-Camera Plus ESP32 Camera Development Board Review](#)

TTGO T-Camera with PIR Motion Sensor



This camera features an OV2640 camera, a 0.96 inch SSD1306 OLED display, a grove connector (ideal to connect I2C devices), battery connector, a PIR motion sensor, on-board RESET button and function button connected to GPIO 34. Like the previous board, it also features 8MB PSRAM, but it doesn't support microSD card.

M5-Camera Model A/B



There are several different versions of M5-Stack ESP32 boards with cameras. The M5-Camera A or M5-Camera B like all the other boards featured here, come with the OV2640 camera. It has 4MB PSRAM – so, you shouldn't have problems taking pictures and streaming with higher quality.

It comes inside a LEGO-style enclosure and features a grove connector ideal to connect other M5-Stack expansions like: microphone, the MPU6050 gyroscope/accelerometer or BME280 temperature, humidity and pressure sensor.

The USB type-C connector makes it easy and fast to upload new code to the board. It comes with an external RST button to restart the board and doesn't have exposed GPIOs.

The greatest advantage of this board is its finished project look when comparing with the other boards. It doesn't have visible electronics components or wires and if you want to add a BME280 sensor, a gyroscope or a microphone, you just need to use the grove connector.

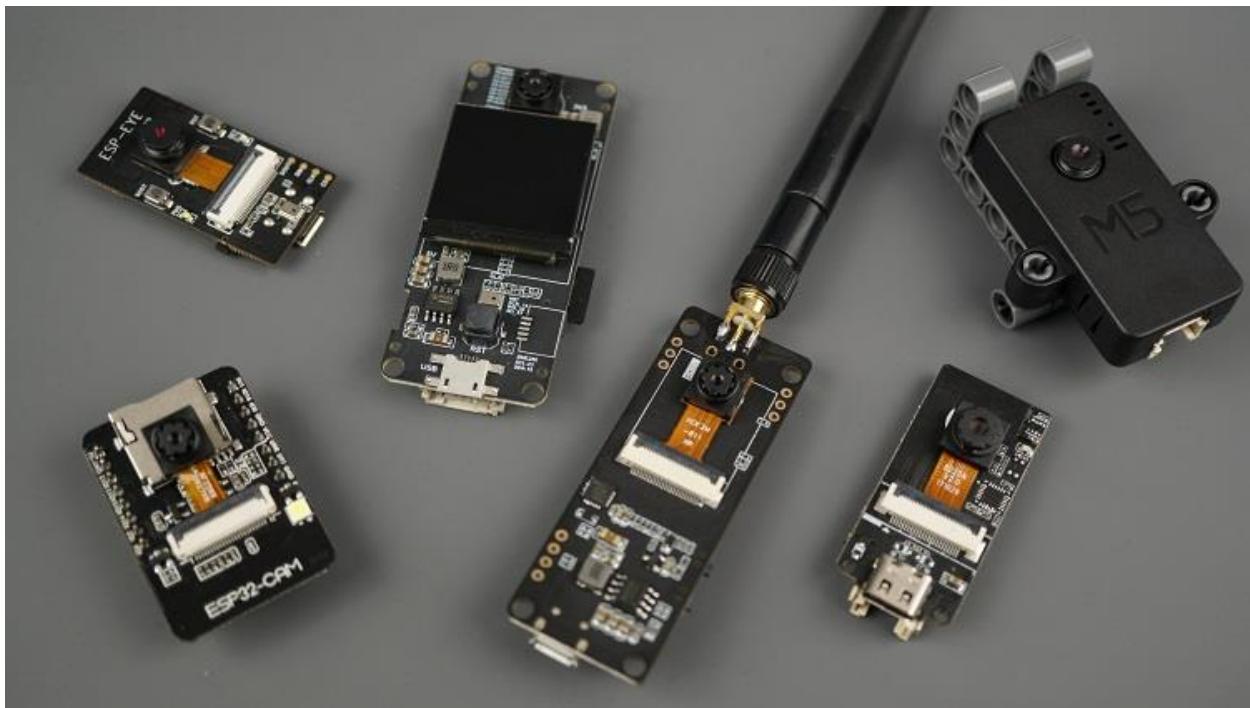
M5-Stack Camera without PSRAM



This is another ESP32 board with camera from M5-Stack. This ESP32-Camera as they called it, doesn't have PSRAM. In practical terms, this means the camera is not able to do face recognition and detection and doesn't support picture resolution higher than SVGA (800×600). You may also have a hard time with video streaming. Some people reported that his camera heats up very fast with video streaming. Usually, when you get one of these boards, you also get a heat-sink precisely because of that.

Like other M5-Stack boards, it has a grove connector, so it is easy to add M5-Stack expansions like microphone, accelerometer or BME280 sensors. Additionally, it has a USB-C connector that you can use to upload code or apply power.

ESP32 Camera Boards Comparison



To help you choose the best ESP32 camera dev board for your project, we've put together a comparison table with the most relevant specs.

| | ESP32-CAM AI-Thinker | ESP-EYE | T-Journal | T-Plus | T-Camera (with PIR) | M5 (A and B) | M5 (without PSRAM) |
|-----------------|-------------------------|---------|-----------------------|-------------------------|------------------------|-----------------|--------------------------|
| PSRAM | ✓ (4MB) | ✓ (8MB) | ✗ | ✓ (8MB) | ✓ (8MB) | ✓ (4MB) | ✗ |
| Screen | ✗ | ✗ | ✓ 0.91inch OLED | ✓ 1.3 TFT display | ✗ | ✗ | ✗ |
| MicroSD Card | ✓ | ✗ | ✗ | ✓ | ✗ | ✗ | ✗ |
| Microphone | ✗ | ✓ | ✗ | ✓ | ✗ | ✓* | ✓* |

| | | | | | | | |
|----------------------------|----|---|---|---|---|---|---|
| Function button | x | ✓ | ✓ | x | ✓ | x | ✓ |
| Battery connector | x | x | ✓ | ✓ | ✓ | x | ✓ |
| Built-in programmer | x | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| GPIOs | 10 | x | 4 | 2 | x | x | x |
| Motion sensor | x | x | x | x | ✓ | x | x |
| Grove connector | x | x | x | ✓ | ✓ | ✓ | ✓ |

* supports microphone (additional hardware sold separately).

For a more in-depth comparison of the ESP32 Camera Development Boards, read:

- [ESP32 Camera Dev Boards Review and Comparison \(Best ESP32-CAM\)](#)

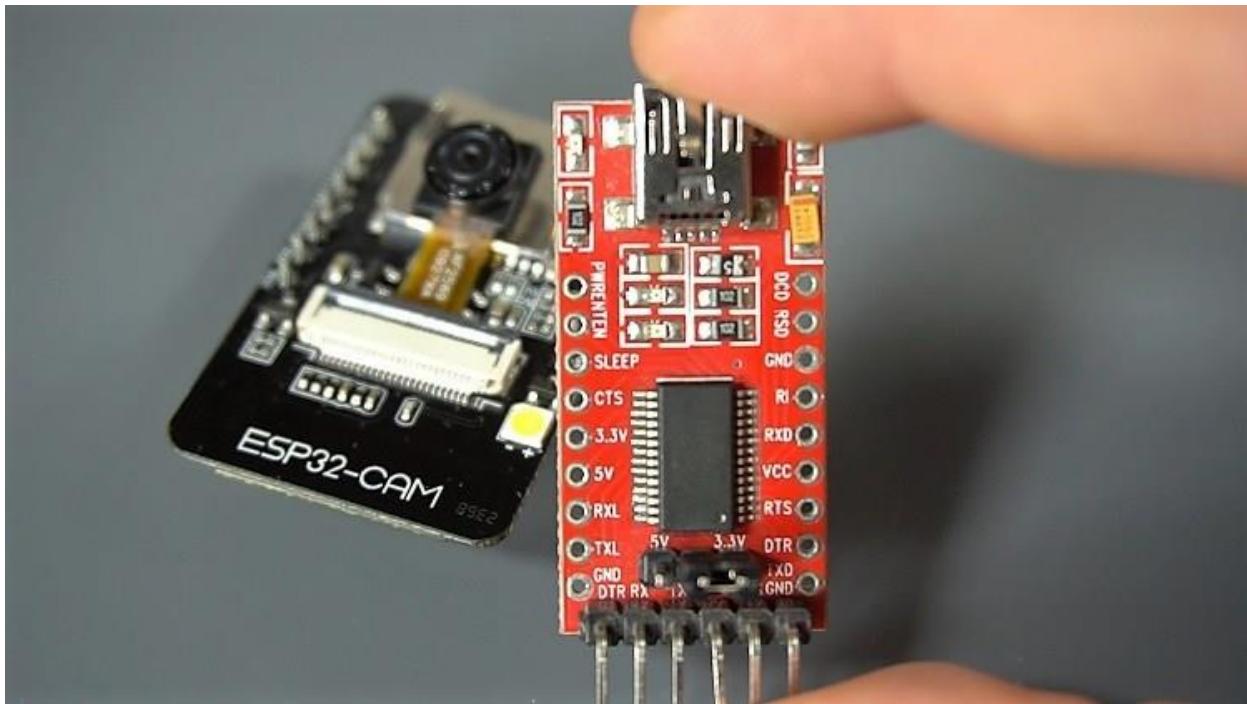
For projects that don't required microSD or external peripherals, you can use any of the previous boards with small changes in the code (you just need to change the pin assignment).

Many of the projects in this eBook include microSD card or connecting external peripherals. So, we recommend having at least one ESP32-CAM AI-Thinker to be able to experiment with all the projects presented in this eBook.

- [Click here to grab an ESP32-CAM with External Antenna](#) (recommended)
- [Click here to grab an ESP32-CAM board](#)

FTDI Programmer

As we've mentioned previously, the ESP32-CAM AI-Thinker doesn't come with an USB connector, so you need an FTDI programmer to upload code to the board through the U0R and U0T pins (serial pins).



We've been using the FTDI programmer shown in the previous picture and it's been working flawlessly. So, we recommend getting one like this. If you already have a different FTDI programmer, it should also work. However, some of our readers reported some issues when using different FTDI programmers.

- [Click here to grab an FTDI programmer](#)

To connect the FTDI programmer to your computer you need an USB cable with data wires. Some USB cables from power banks or phone chargers don't have data wires – these won't work.

Our FTDI programmer uses a mini-USB interface, other versions support micro-USB. Make sure you have a proper USB cable, otherwise, you won't be able to program your board.

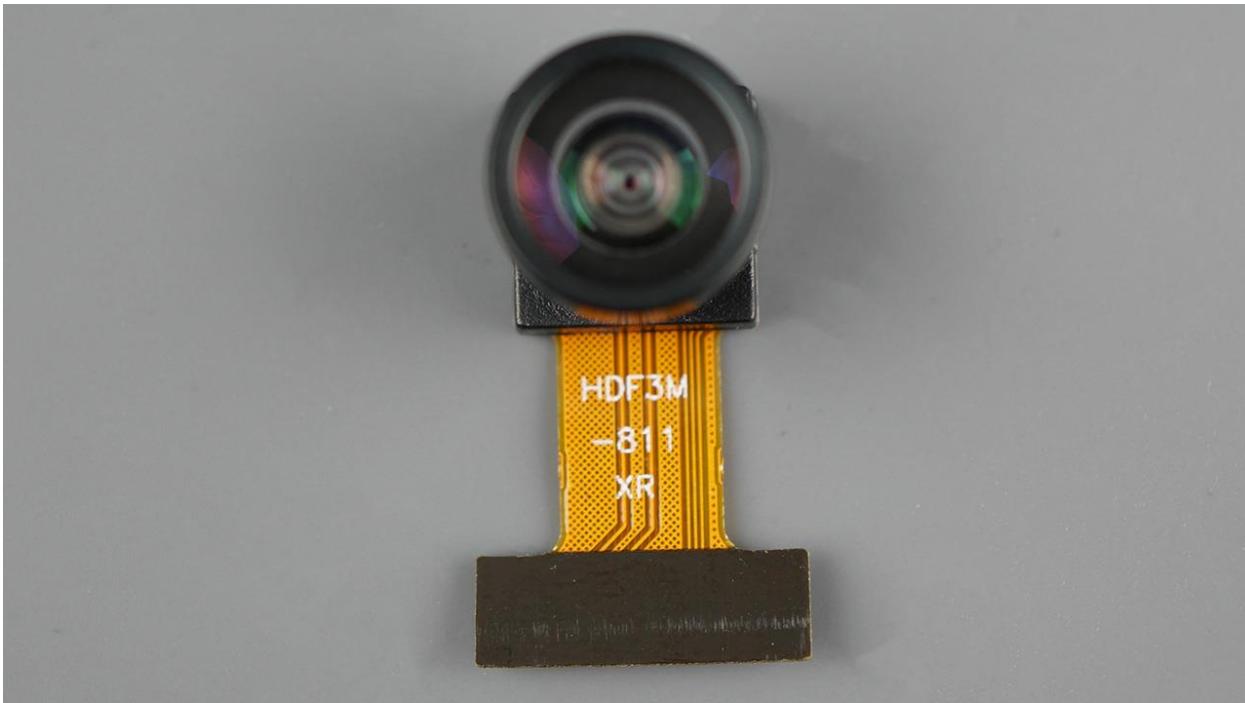
Note: the other ESP32 camera development boards don't require an FTDI programmer.

Different Camera Probes

When you first get an ESP32-CAM it comes in two parts: the main board and a separate OV2640 camera.



There are OV2640 cameras with fish-eye lens for the ESP32-CAM that capture a wider area. The following picture shows an OV2640 camera with fish-eye lens.

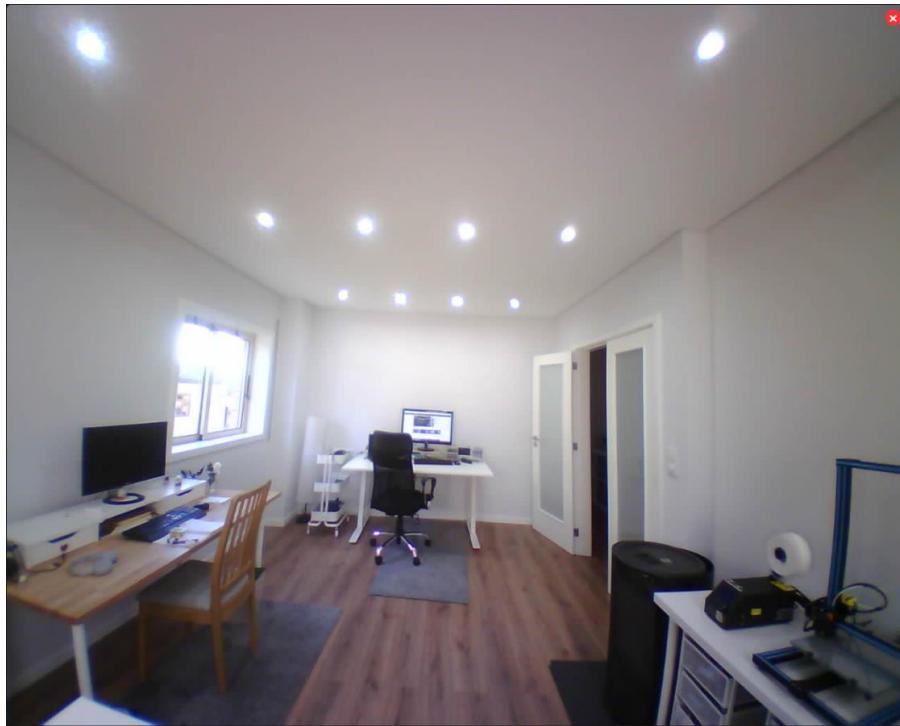


The flex cable that connects to the main board is very short, about 8mm long (0.3in). You can get camera probes separately with longer ribbons – these are available both with the normal and fish-eye lens as shown below.

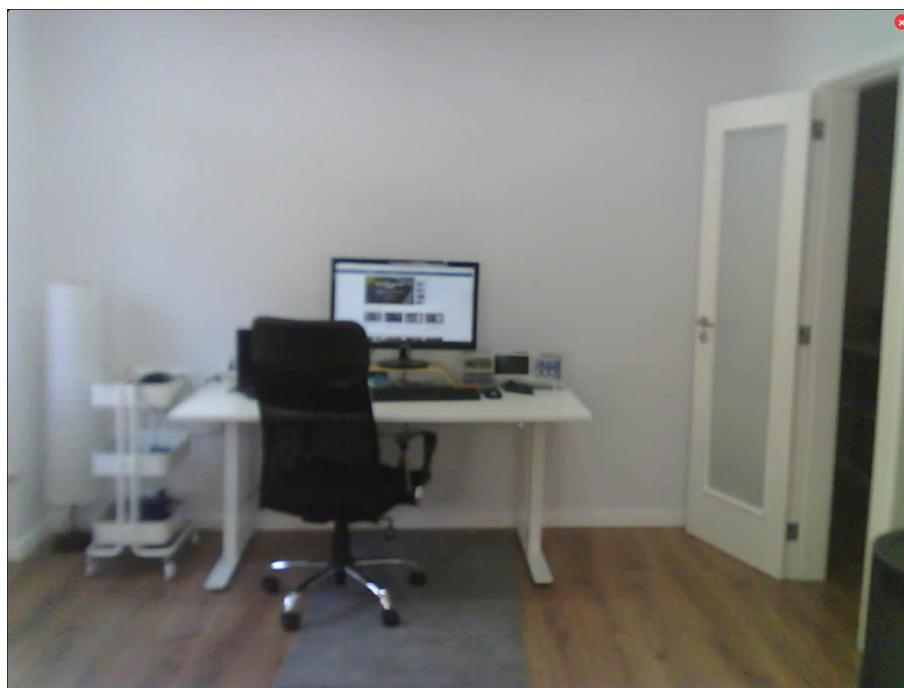


To give you an idea of the differences between the normal camera lens and the fish-eye lens, we took a photo on the exact same place using each camera.

This is the result for a camera with an OV2640 fish-eye lens.



And this is the result for a normal OV2640 camera lens.



If you want to catch a wider area with your ESP32-CAM, getting a camera with fish-eye lens is the best option.

- [Buy an OV2640 camera with longer ribbon or fish-eye lens](#)

Heat Sink

Some people reported issues with overheating when the ESP32-CAM is streaming continuously. If you end up getting heating problems streaming continuously or if you plan to put it inside an enclosure it may be a good idea getting a heat sink or a fan depending on the project application.

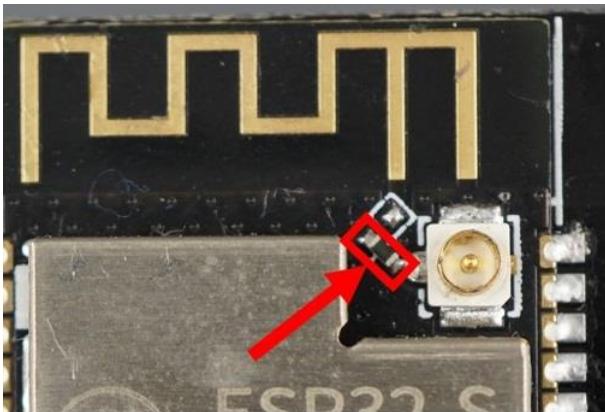
ESP32-CAM Antenna

The ESP32-CAM has the option to use either the built-in PCB antenna or an external antenna.

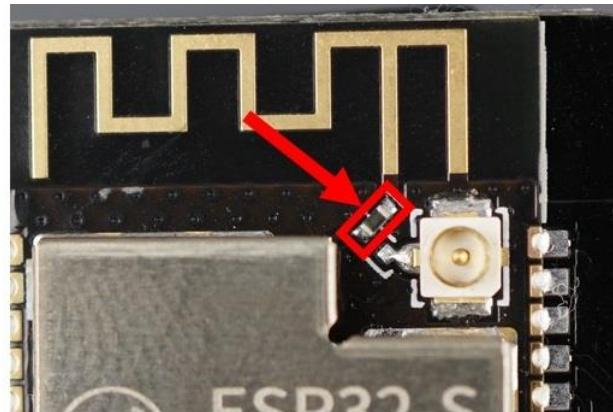
Next to the IPEX connector there are three little white squares laid out like a “<” with the middle position being common. There is a resistor selecting the desired antenna.

Here's the two configurations:

- To use the IPEX connector with an external antenna, the resistor must be on the bottom position, like this “\”. See illustration below;
- To use the PCB antenna (on-board antenna), the resistor must be on the top position, like this “/”.



External Antenna



On-board Antenna

Take a look at your board to see if it is set to use the on-board antenna or the IPEX connector. Using the on-board antenna works well if you are close to your router.

We recommend using the IPEX connector with an external antenna for better results.

Warning: projects with video streaming crash frequently when you don't use an external antenna due to poor connectivity. So, make sure you get one to get your projects working reliably.

To enable or disable the on-board antenna, you just need to unsolder that resistor and solder it in the desired configuration. You can also drop some solder to connect those points (you don't necessarily need to add the resistor as long as the pads are connected).

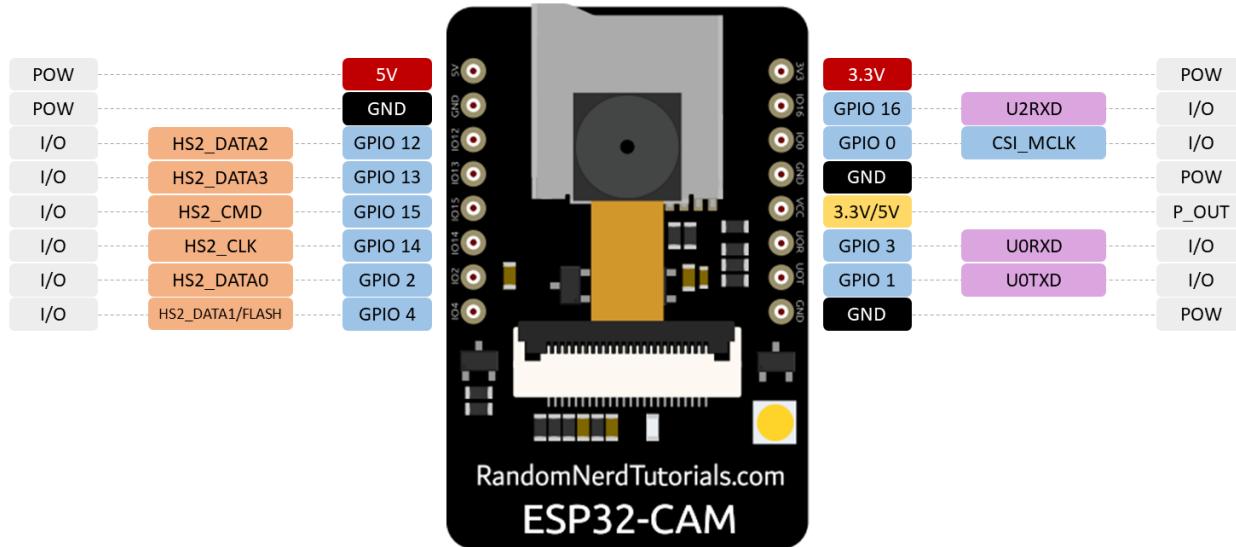
You can't use the two antennas at the same time, so you can only have one connection for the antenna.

When getting an ESP32-CAM, there are stores that offer the package with an external antenna.

- [Buy an ESP32-CAM with external antenna](#) (recommended)

ESP32-CAM Pinout

The following figure shows an overview of the ESP32-CAM pinout.



There are three **GND** pins and two pins for power: **3.3V** and **5V**.

GPIO 1 and **GPIO 3** are the serial pins (TX and RX, respectively). These pins are used to communicate with the FTDI programmer to upload code to your board.

GPIO 0 determines whether the ESP32 is in flashing mode or not. When GPIO 0 is connected to GND, the ESP32 goes into flashing mode and you can upload code to the board.

Camera Connections

The connections between the camera and the ESP32-CAM Ai-Thinker are shown in the following table.

| OV2640 CAMERA | ESP32 |
|---------------|--------|
| D0 | GPIO 5 |

| | |
|-----------|---------|
| D1 | GPIO 18 |
| D2 | GPIO 19 |
| D3 | GPIO 21 |
| D4 | GPIO 36 |
| D5 | GPIO 39 |
| D6 | GPIO 34 |
| D7 | GPIO 35 |
| XCLK | GPIO 0 |
| PCLK | GPIO 22 |
| VSYNC | GPIO 25 |
| HREF | GPIO 23 |
| SDA | GPIO 26 |
| SCL | GPIO 27 |
| POWER PIN | GPIO 32 |

Note: the connections in the previous table are for the ESP32-CAM AI-Thinker Module. If you're using another camera board, look for the board pinout on the [Appendix](#). Different ESP32 boards with cameras use different pins to connect the camera.

SD Card Connections

The following pins are being used by the microSD card module in the ESP32-CAM AI-Thinker.

| SD Card | ESP32 |
|--------------------|---------|
| CLK | GPIO 14 |
| CMD | GPIO 15 |
| DATA0 | GPIO 2 |
| DATA1 / Flashlight | GPIO 4 |
| DATA2 | GPIO 12 |
| DATA3 | GPIO 13 |

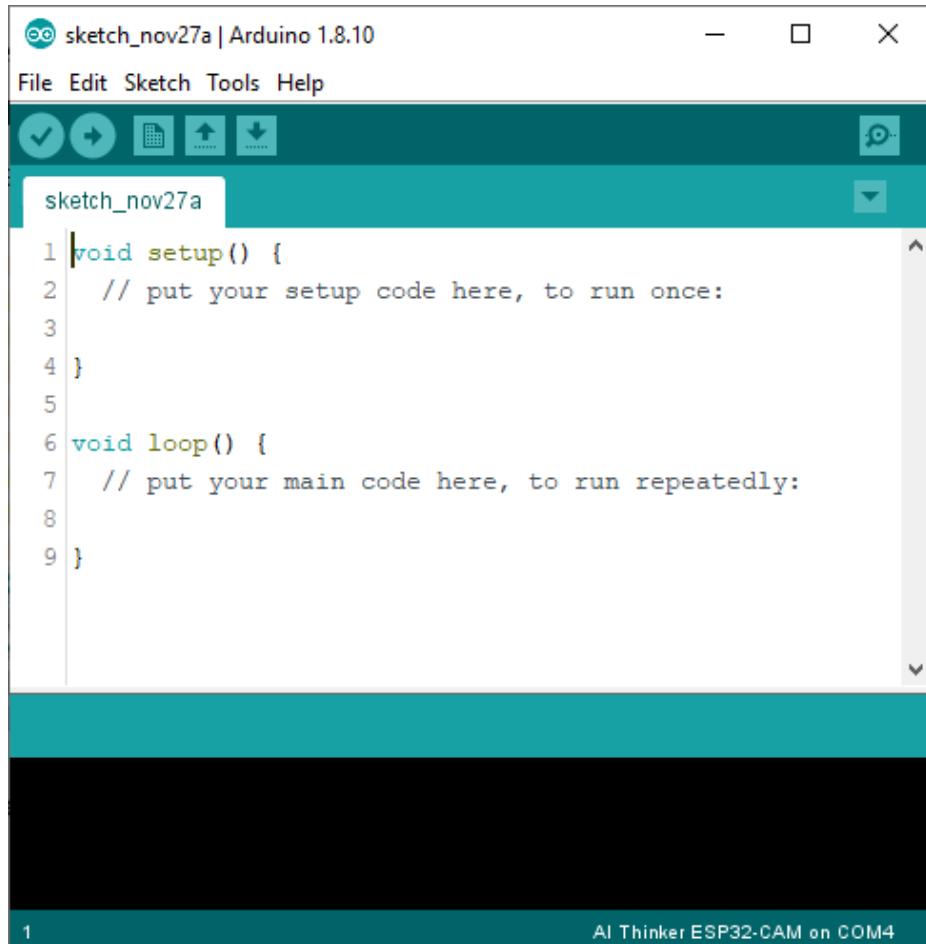
Wrapping Up

In this Unit we've taken a look at the ESP32-CAM AI-Thinker board. To proceed with the projects in this eBook, we recommend getting:

- [ESP32-CAM AI-Thinker Module and external antenna](#) (recommend)
- [FTDI programmer \(5V\)](#)
- [Other OV2640 camera probes: fish-eye lens, longer flex cable](#) (optional)
- [Female-to-female jumper wires](#)

Continue to the next Unit to install and prepare your Arduino IDE to work with the ESP32-CAM.

Unit 2: Preparing Arduino IDE for the ESP32-CAM



Throughout this eBook we'll program the ESP32-CAM using Arduino IDE. So, you must follow this Unit first to download, install and prepare your Arduino IDE to work with the ESP32-CAM.

Requirements

You need JAVA installed in your computer. If you don't, go to this website: <http://java.com/download>, download and install the latest version.

Downloading Arduino IDE

To download the Arduino IDE, visit the following URL:

- <https://www.arduino.cc/en/Main/Software>

Select your operating system and download the software. For Windows, we recommend downloading the “Windows ZIP file for non admin install”.



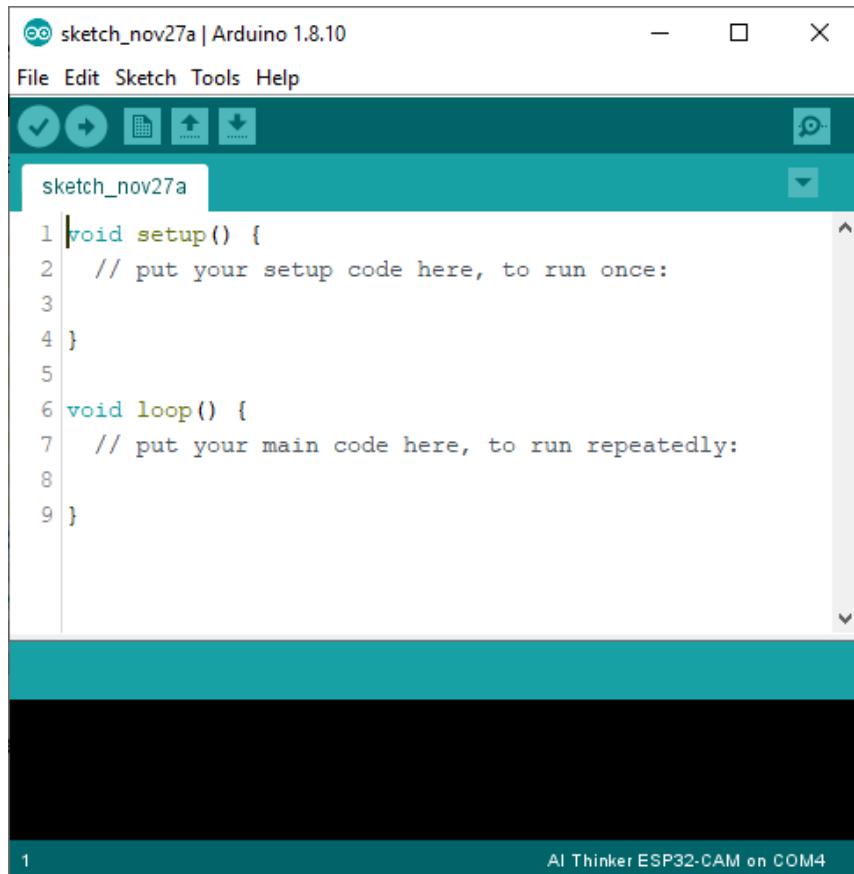
Note: if you have a previous version of the Arduino IDE installed, we recommend upgrading to the most recent version. At the time of writing this Unit, the most recent version is Arduino 1.8.12 and that's the one we'll use throughout this eBook.

Installing Arduino IDE

Grab the folder you've just downloaded and unzip it. Run the file highlighted below.

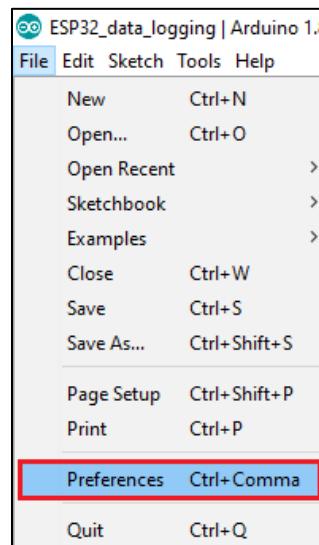
| | | | |
|-----------------------|--------------------|-----------------------|--------|
| examples | 11/28/2019 3:16 PM | File folder | |
| hardware | 11/28/2019 3:16 PM | File folder | |
| java | 11/28/2019 3:17 PM | File folder | |
| lib | 11/28/2019 3:17 PM | File folder | |
| libraries | 11/28/2019 3:17 PM | File folder | |
| reference | 11/28/2019 3:17 PM | File folder | |
| tools | 11/28/2019 3:18 PM | File folder | |
| tools-builder | 11/28/2019 3:18 PM | File folder | |
| arduino.exe | 11/28/2019 3:16 PM | Application | 395 KB |
| arduino.l4j.ini | 11/28/2019 3:16 PM | Configuration sett... | 1 KB |
| arduino_debug.exe | 11/28/2019 3:16 PM | Application | 393 KB |
| arduino_debug.l4j.ini | 11/28/2019 3:16 PM | Configuration sett... | 1 KB |

The Arduino IDE window should open.

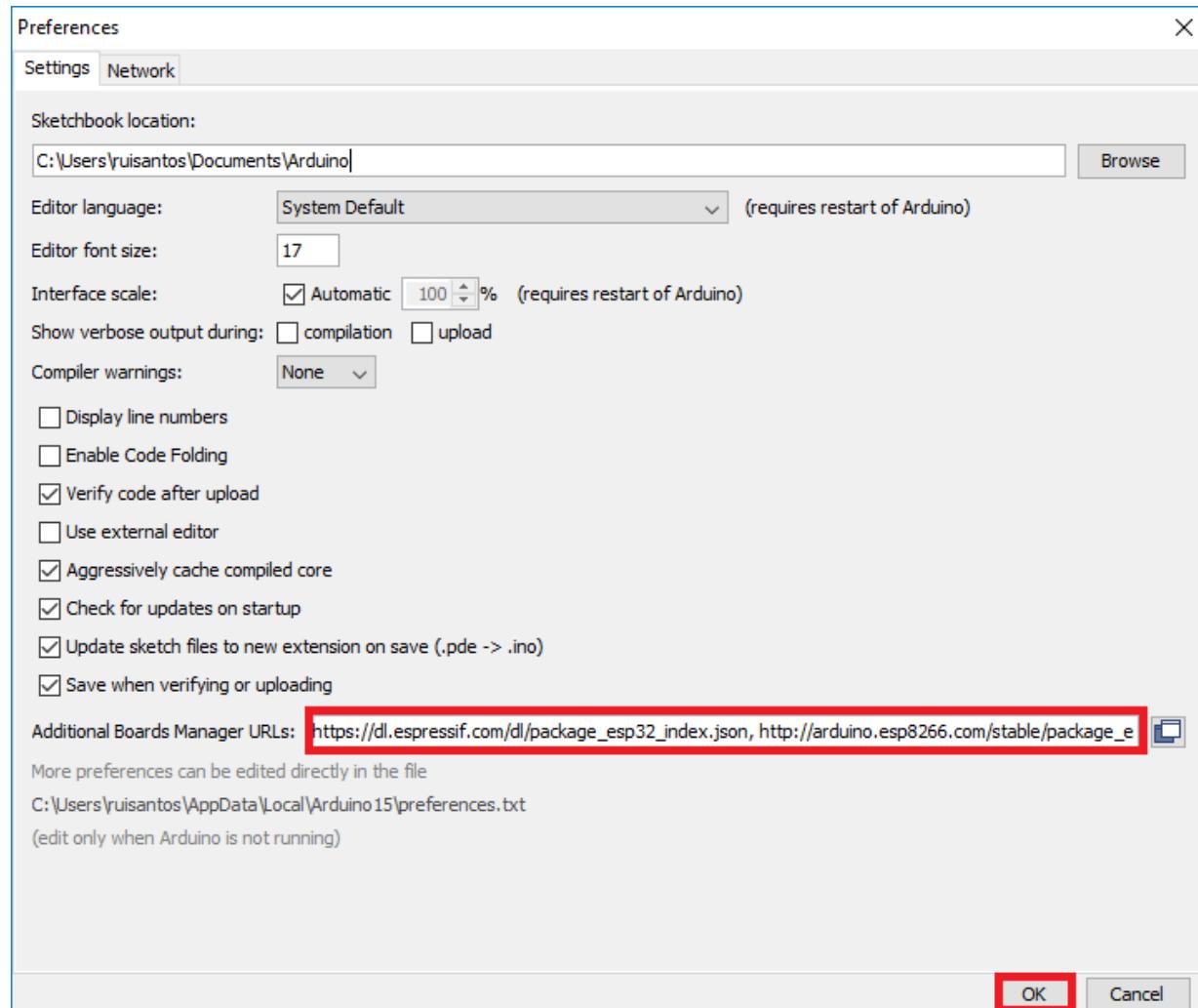


Installing the ESP32 add-on for Arduino IDE

Open the **Preferences** window in the Arduino IDE. Go to **File > Preferences**:



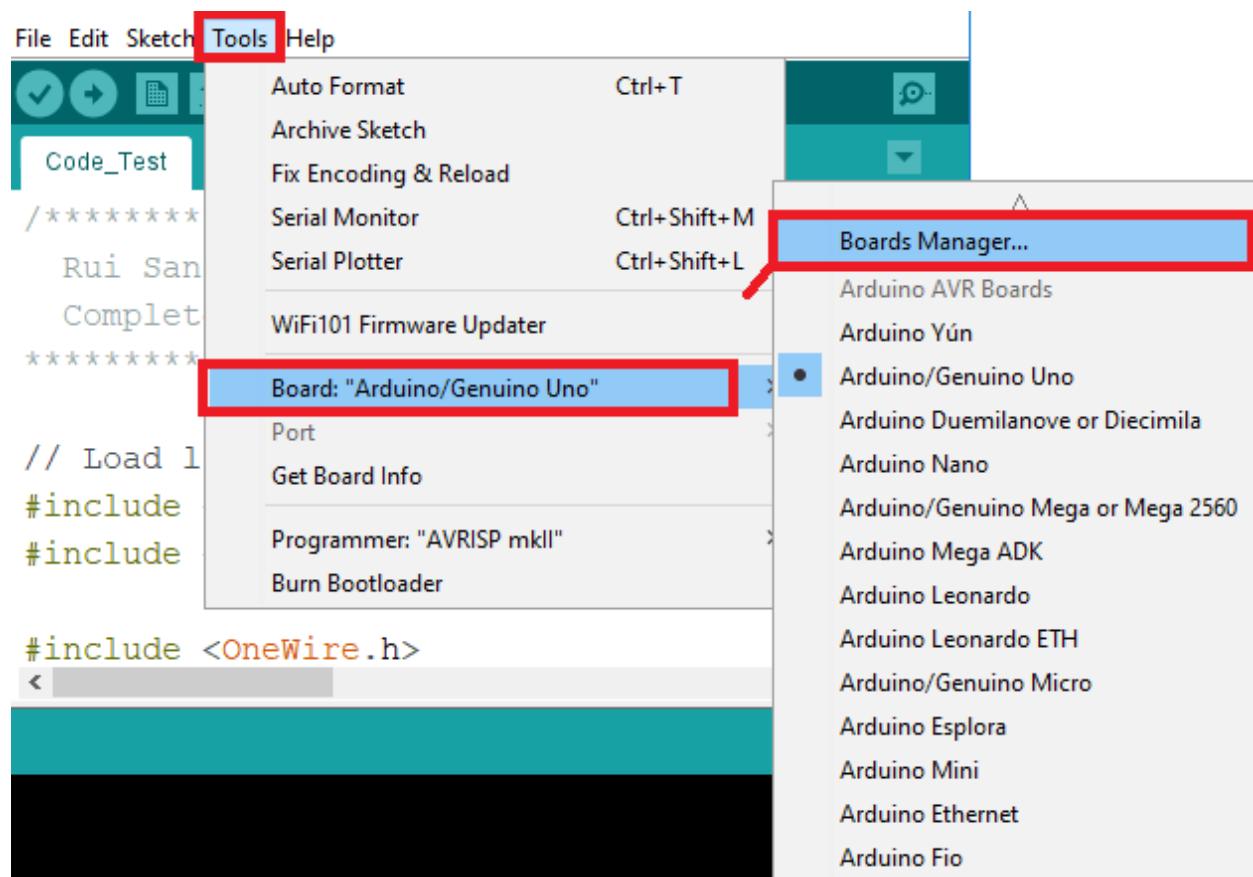
Enter https://dl.espressif.com/dl/package_esp32_index.json into the “**Additional Board Manager URLs**” field as shown in the figure below. Then, click the “**OK**” button.



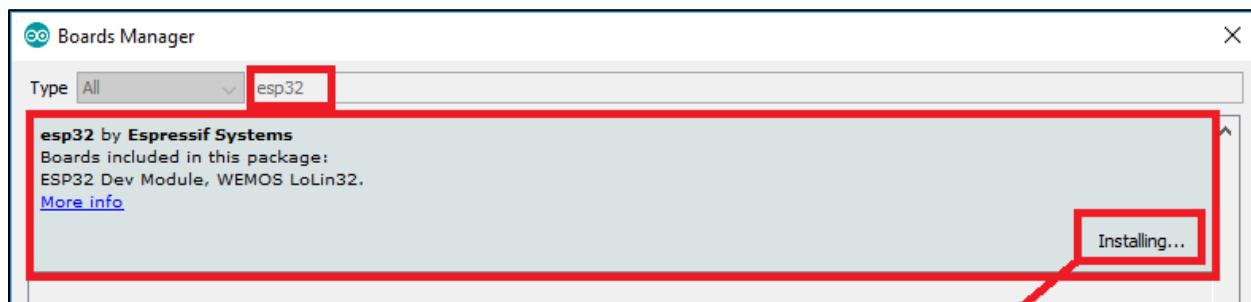
If you already have the ESP8266 boards URL, you can separate the URLs with a comma, as follows:

https://dl.espressif.com/dl/package_esp32_index.json,
http://arduino.esp8266.com/stable/package_esp8266com_index.json

Open the Boards Manager. Go to Tools > Board > Boards Manager...



Search for **ESP32** and press install button for the “**ESP32 by Espressif Systems**”:

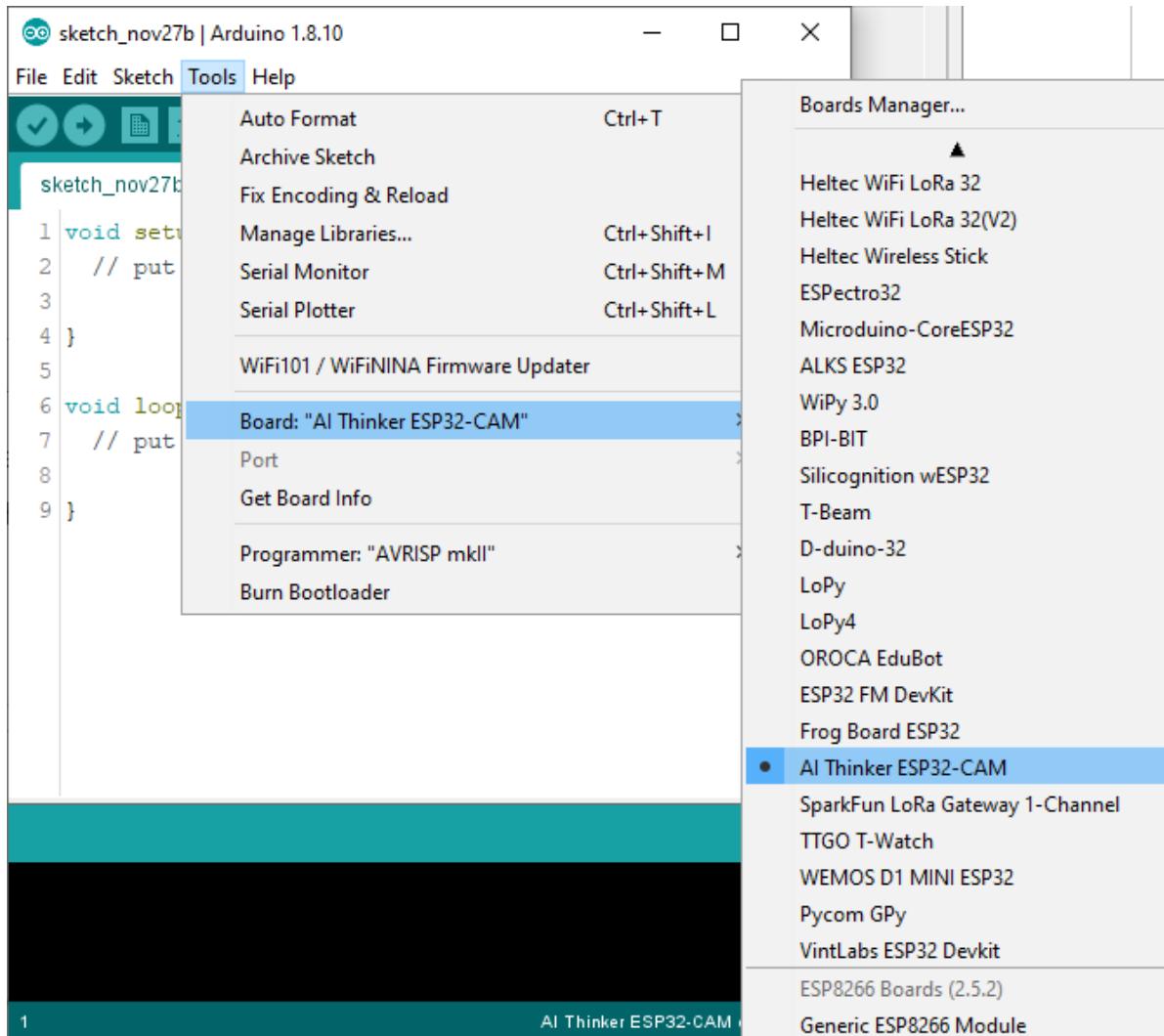


That's it. It should be installed after a few seconds.



Note: if you already have the ESP32 add-on installed, you should update to the newest version. Throughout this eBook, we'll use version 1.0.4 (if there's a newer version, update to the latest version).

After this, restart your Arduino IDE. Then, go to **Tools** ▶ **Board** and check that you have ESP32 boards available.



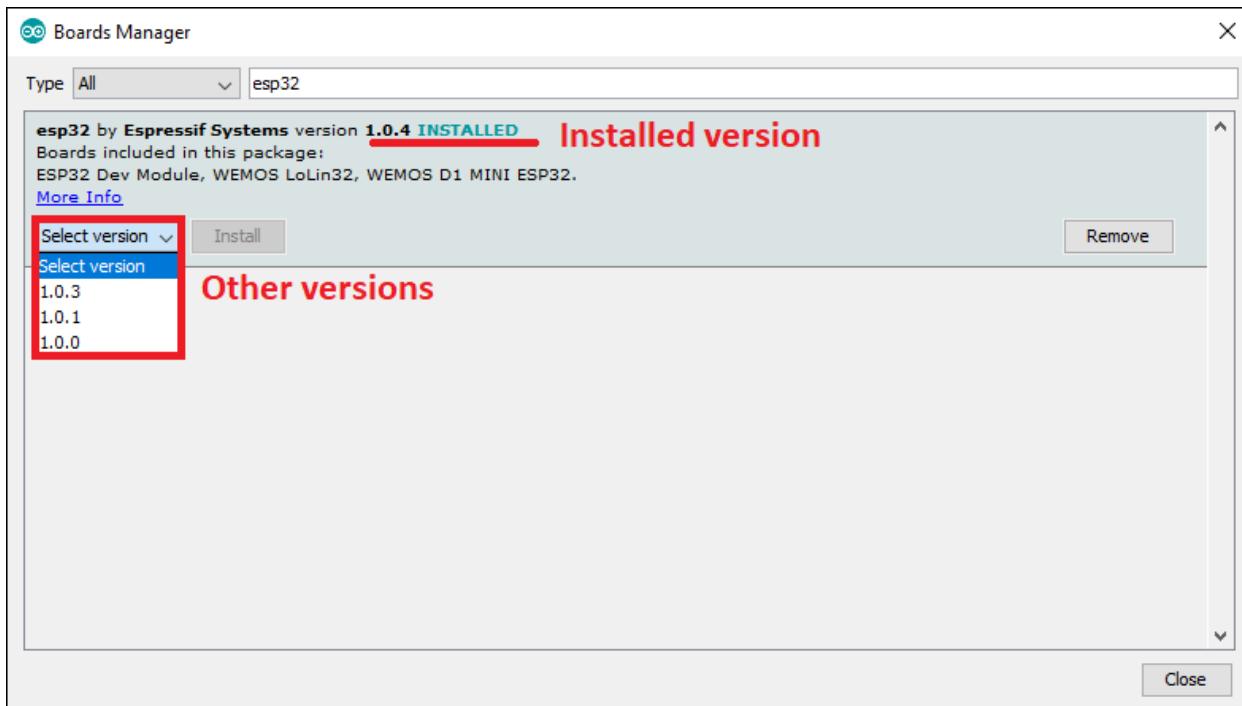
If you have several options of ESP32 boards available, it means the ESP32 add-on was successfully installed. You can proceed to the next Unit.

Update the ESP32 Boards Add-On

Once in a while, it's a good idea to check if you have the latest version of the ESP32 Boards add-on installed.

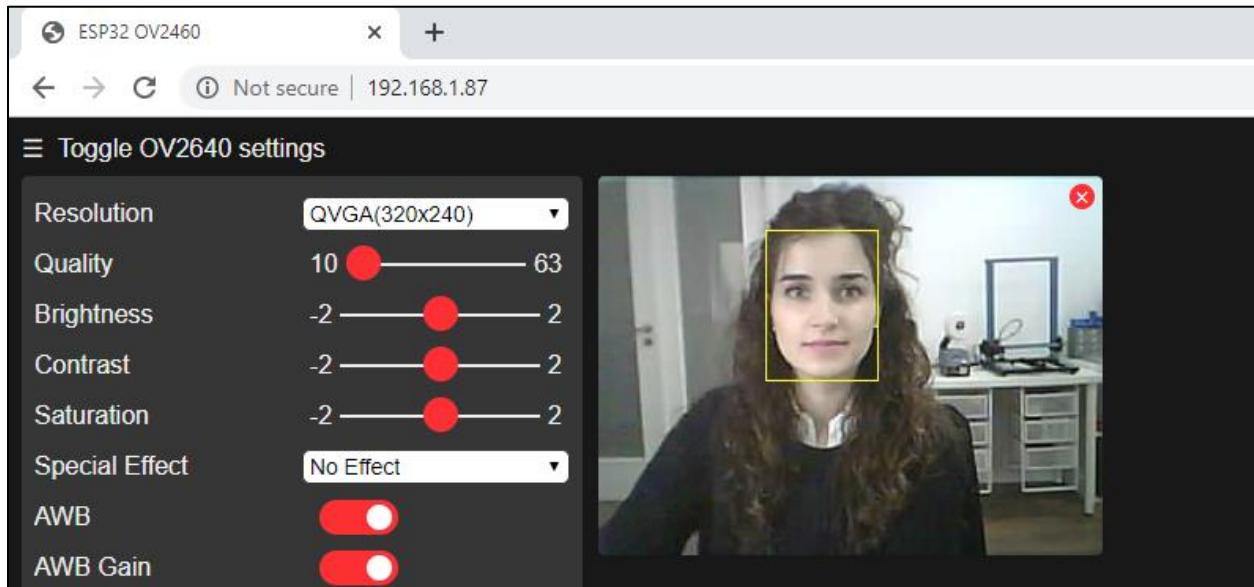
You just need to go to **Tools > Board > Boards Manager**

Search for **ESP32**. Check the version that you have installed and the versions available.



If there is a more recent version available, select that version to install. In the previous image, we already have the latest version installed (1.0.4) at the time.

Unit 3: Camera Web Server (Video Streaming and Face Recognition)



In this Unit we'll test an example for the ESP32-CAM that comes with the Arduino IDE. This example shows many of the ESP32-CAM functionalities in just one project: video streaming, take photo, face recognition and face detection, as well as changing the camera settings. It doesn't show how to use the microSD card.

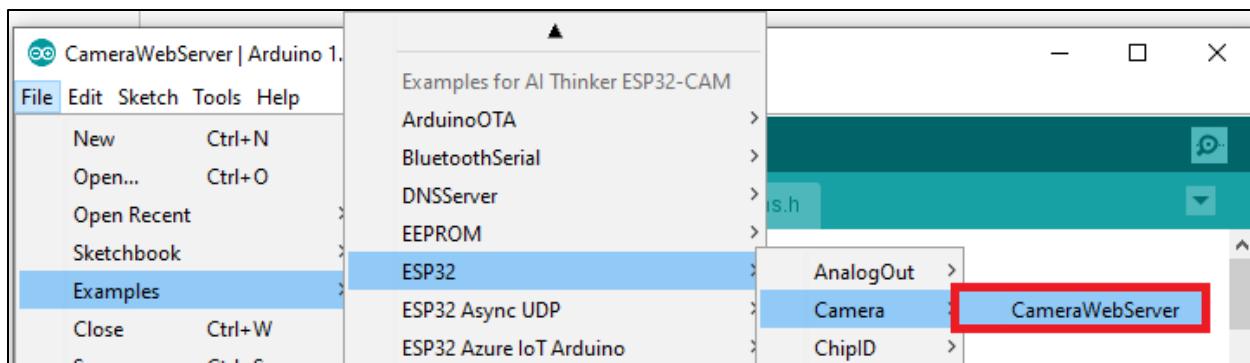
Before proceeding with this project make sure you have the latest version of Arduino IDE installed as well as the latest version of the ESP32 add-on. If you don't, get back to the previous Unit.

Compatibility: this project is fully compatible with ESP32 camera boards that have PSRAM. It also works with camera boards without PSRAM but face recognition and detection don't work neither streaming video in higher resolutions.

Camera Web Server Example

Open your Arduino IDE. Go to **Tools** ▶ **Board** and select the **AI Thinker ESP32-CAM**.

Then, go to **File** ▶ **Examples** ▶ **ESP32** ▶ **Camera** ▶ **CameraWebServer**.



The following code should load. You should have four different tabs in the IDE as shown in the following figure.

A screenshot of the Arduino IDE showing the "CameraWebServer" sketch. The title bar says "CameraWebServer | Arduino 1.8.10". The menu bar includes File, Edit, Sketch, Tools, and Help. Below the menu is a toolbar with icons for upload, refresh, and other functions. The code editor has four tabs open: "CameraWebServer" (highlighted with a red box), "app_httpd.cpp", "camera_index.h", and "camera_pins.h". The code itself is as follows:

```
1 #include "esp_camera.h"
2 #include <WiFi.h>
3
4 /**
5 // WARNING!!! Make sure that you have either selected ESP32 Wrover M
6 // or another board which has PSRAM enabled
7 //
8
9 // Select camera model
10 // #define CAMERA_MODEL_WROVER_KIT
11 // #define CAMERA_MODEL_ESP_EYE
12 // #define CAMERA_MODEL_M5STACK_PSRAM
13 // #define CAMERA_MODEL_M5STACK_WIDE
14 #define CAMERA_MODEL_AI_THINKER
```

Now, you need to make a few modifications to make the code work for you.

Alternatively, you can download the *CameraWebServer.zip* folder in the link below.

CODE

https://github.com/RuiSantosdotme/ESP32-CAM-eBook/raw/master/Code/Module_1/CameraWebServer/CameraWebServer.zip

Select Your Camera Model

First, select your camera model. If you're using a camera like ours, you should select the `CAMERA_MODEL_AI_THINKER`. Comment the `CAMERA_MODEL_WROVER_KIT` and uncomment the `CAMERA_MODEL_AI_THINKER` as shown below.

```
// Select camera model
//#define CAMERA_MODEL_WROVER_KIT
//#define CAMERA_MODEL_ESP_EYE
//#define CAMERA_MODEL_M5STACK_PSRAM
//#define CAMERA_MODEL_M5STACK_WIDE
#define CAMERA_MODEL_AI_THINKER
```

If none of these correspond to the camera you're using, you need to add the pin assignment for your specific board in the **camera_pins.h** tab. [Go to this section](#) to learn how to add or change the camera pin assignment for the board you're using.

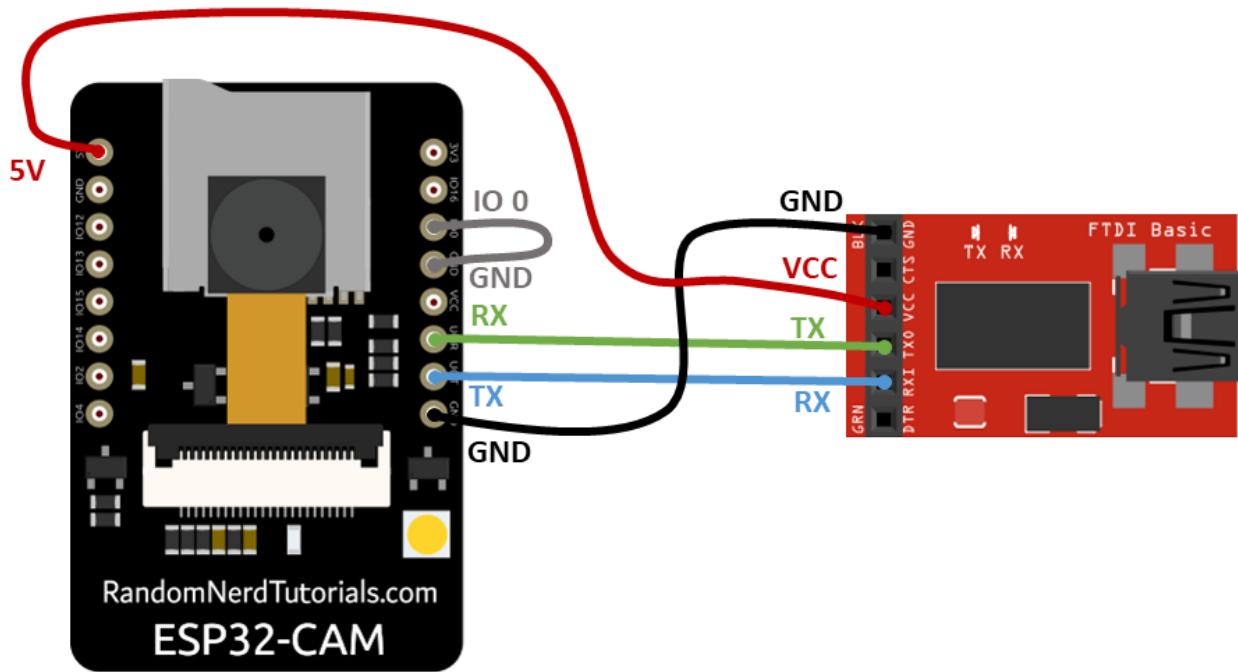
Your Network Credentials

Then, type your network credentials, SSID and password in the following variables.

```
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
```

ESP32-CAM Upload Code

After making the necessary changes to the code, you are ready to upload the sketch. As mentioned previously, you need to connect your ESP32-CAM AI-Thinker board to an FTDI programmer to upload code. You can follow the next schematic diagram.



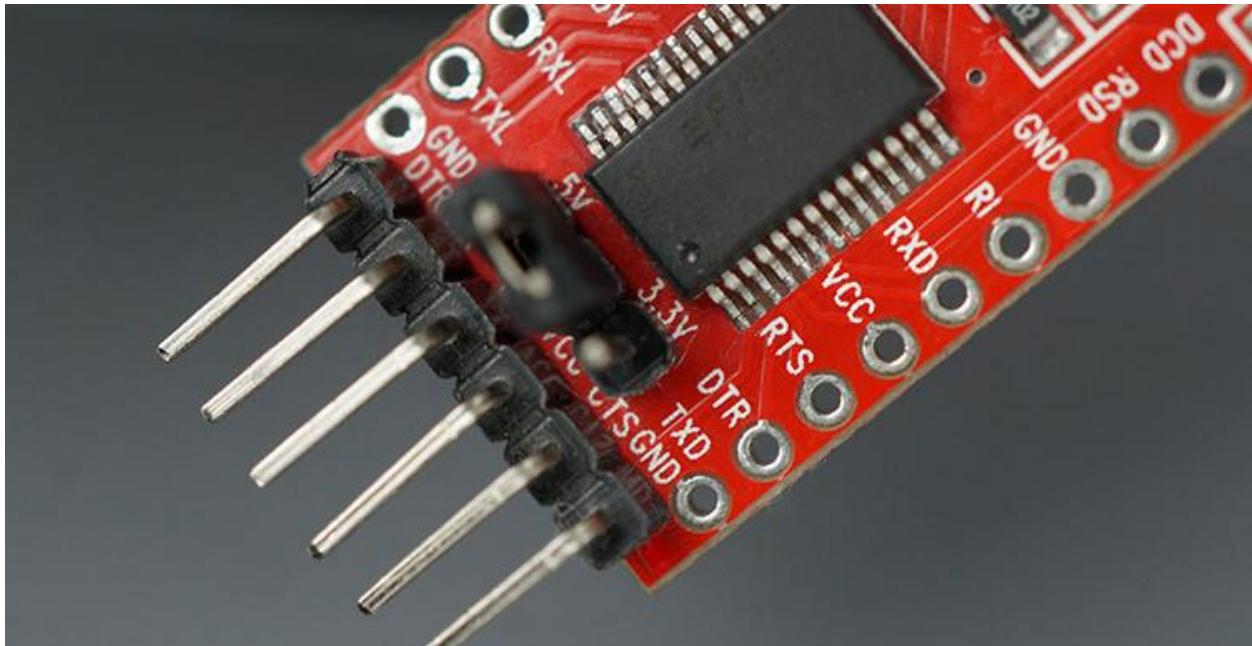
WARNING: the order of the FTDI pins on the diagram may not match yours. Make sure you check the silkscreen label next to each pin.

You can also use the following table as a reference:

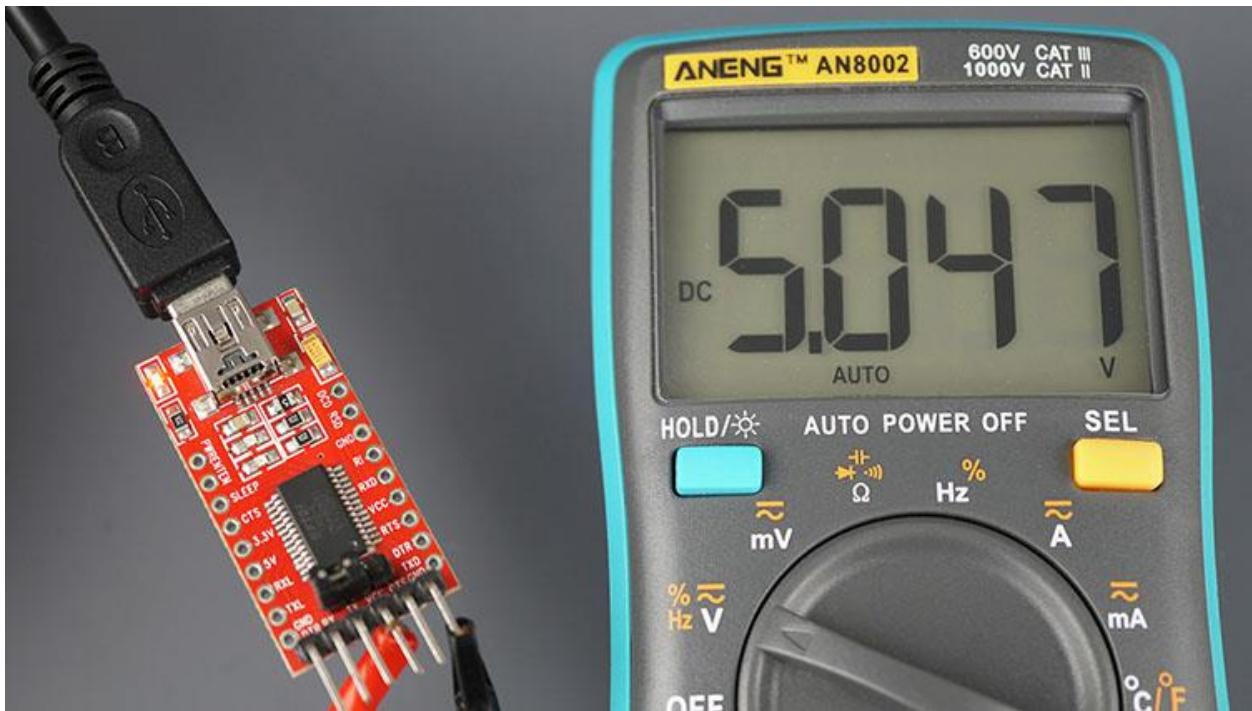
| ESP32-CAM | FTDI Programmer |
|-----------|-----------------|
| GND | GND |
| 5V | VCC (5V) |
| U0R | TX |
| T0R | RX |
| GPIO 0 | GND |

Note that GPIO 0 needs to be connected to GND to upload code – you can use a female-to-female jumper wire to connect the pins. Having GPIO 0 connected to GND puts the ESP32 in flashing mode. This means the ESP32 is ready to receive new code.

Many FTDI programmers have a jumper that allows you to select 3.3V or 5V. Make sure the jumper is in the right position to select 5V.



Warning: some FTDI programmers output 3.3V even when selecting the 5V option. So, check the output of the FTDI programmer VCC pin with a multimeter.



Once you made all the necessary connections, in your Arduino IDE, make sure you have your ESP32-CAM board selected in **Tools** ▶ **Board**. Then, go to **Tools** ▶ **Port** and select the COM port the ESP32-CAM is connected to.

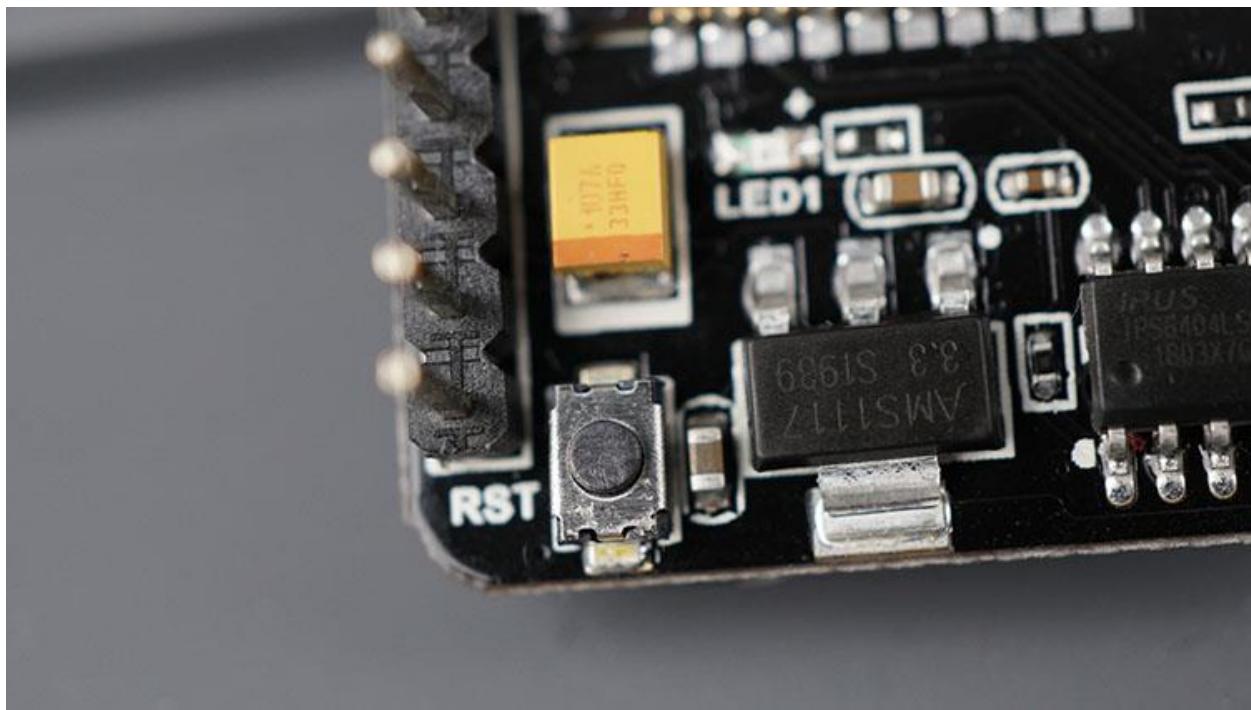
Then, click the upload button.



When you start to see these dots on the debugging window as shown below:

```
esptool.py v2.6-beta1
Serial port COM10
Connecting.....
```

Press the ESP32-CAM on-board RST button. After a few seconds, the code should be successfully uploaded to your board.



If you're using an M5-stack camera module or any board with built-in programmer, you just need to connect it to your computer through the USB cable and click the

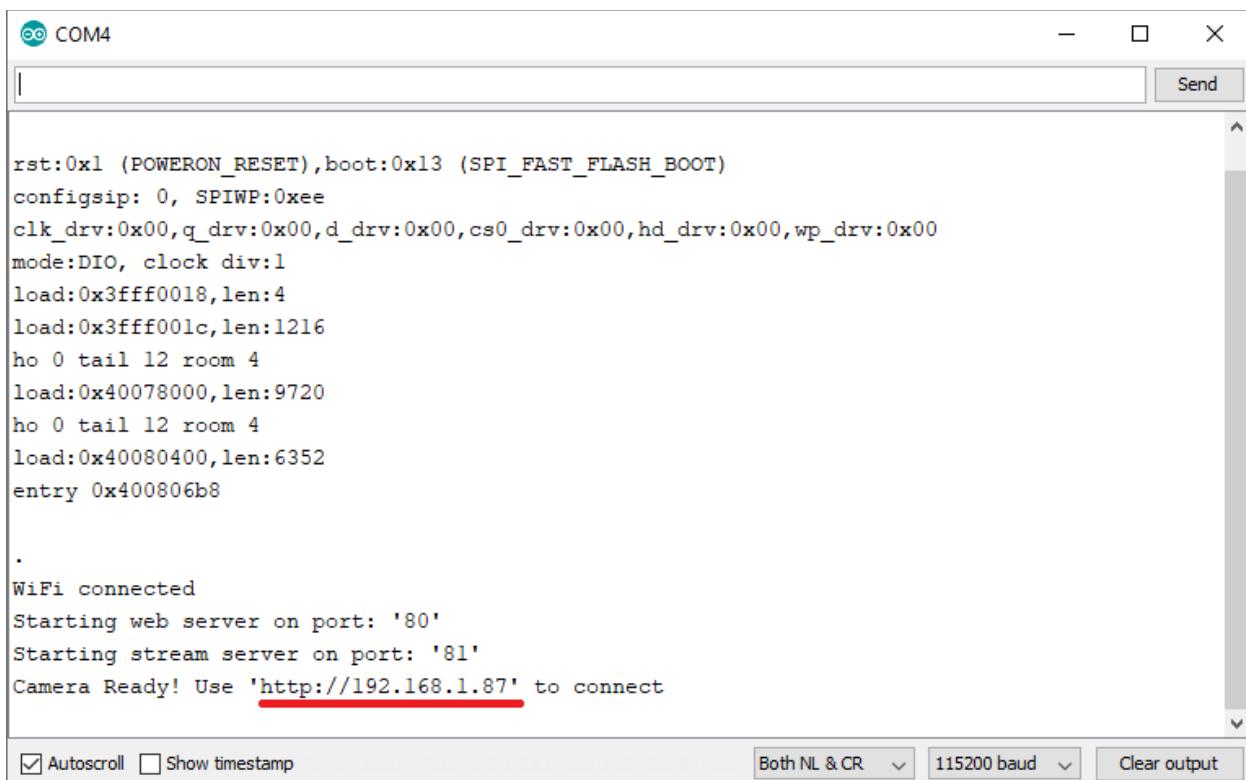
upload button. You can select the **AI-Thinker** module in the **Boards** menu – at the time of writing this Unit, there isn't an option for M5-stack camera boards.

Getting the IP Address

After uploading the code, **disconnect GPIO 0 from GND** (because the ESP32 no longer needs to be in flashing mode).

Open the Serial Monitor at a baud rate of 115200. **Press the ESP32-CAM on-board Reset button.**

The ESP32 IP address should be printed in the Serial Monitor. In our case, it is 192.168.1.87.



The screenshot shows a Windows-style serial monitor window titled "COM4". The text area displays the following log output:

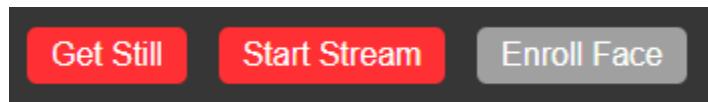
```
rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:1216
ho 0 tail 12 room 4
load:0x40078000,len:9720
ho 0 tail 12 room 4
load:0x40080400,len:6352
entry 0x400806b8

.
WiFi connected
Starting web server on port: '80'
Starting stream server on port: '81'
Camera Ready! Use 'http://192.168.1.87' to connect
```

At the bottom of the window, there are several control buttons: "Autoscroll" (checked), "Show timestamp" (unchecked), "Both NL & CR" (dropdown), "115200 baud" (dropdown), and "Clear output".

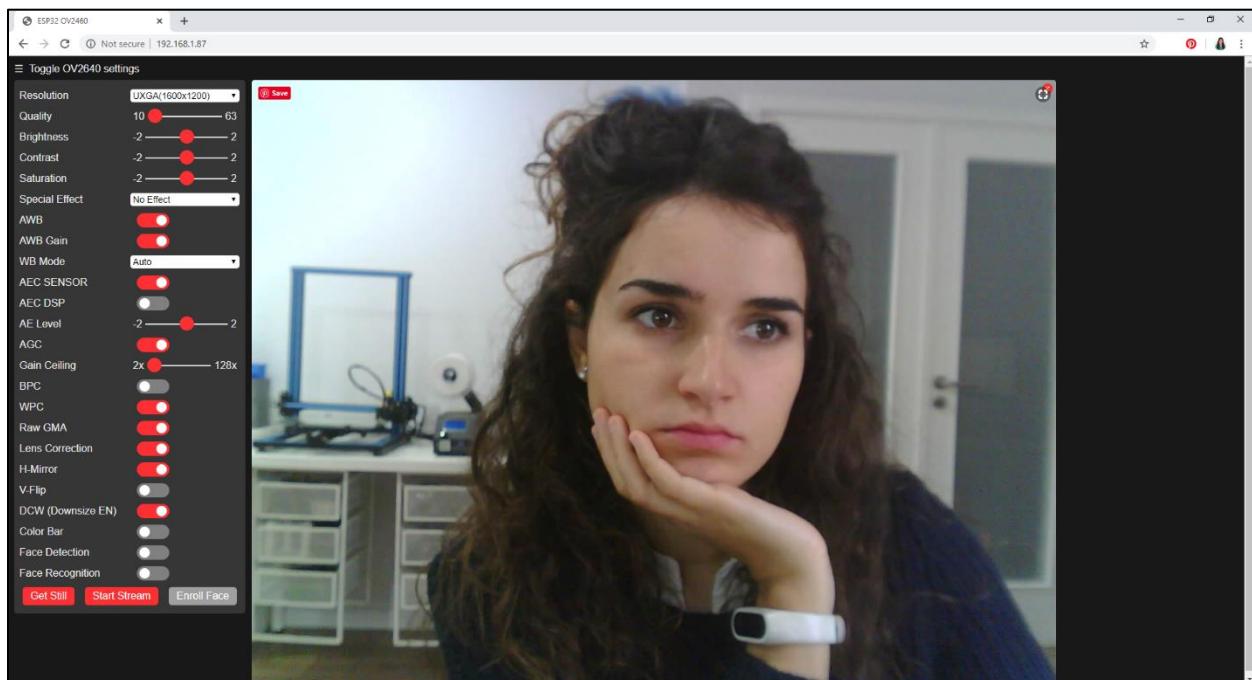
Accessing the Video Streaming Server

Now, you can access your camera streaming server on your local network. Open a browser and type the ESP32-CAM IP address. Press the **Start Stream** button to start video streaming. You can also take photos by clicking the **Get Still** button.



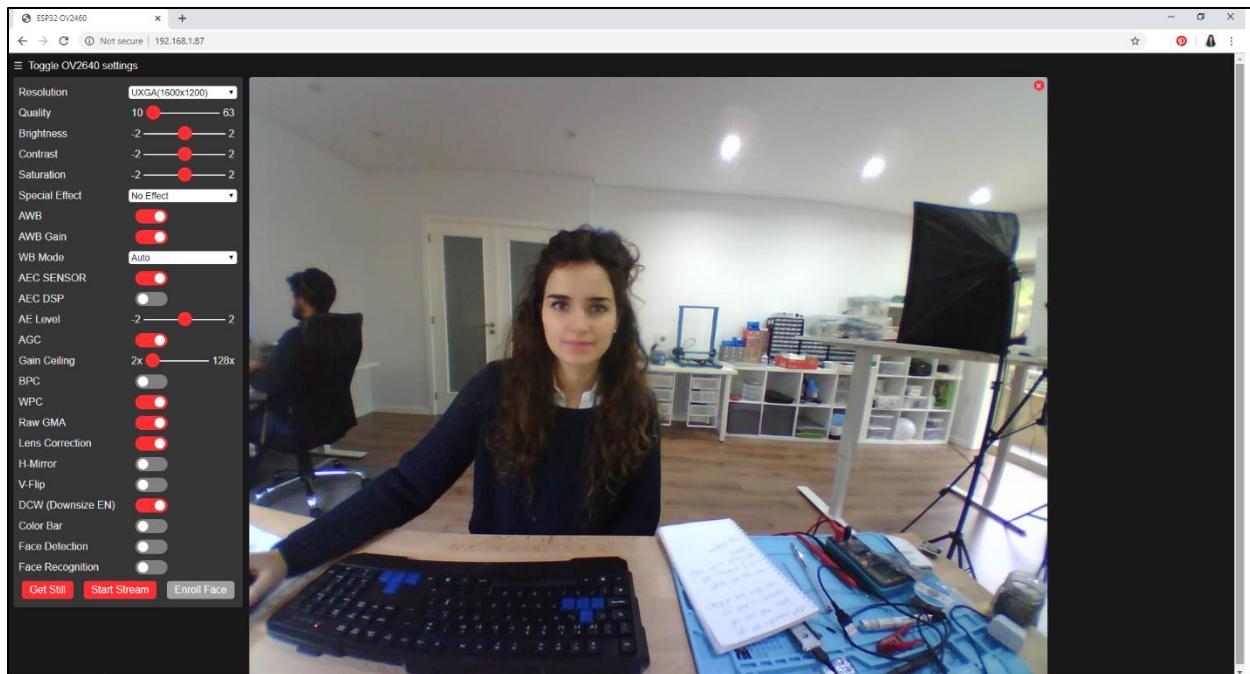
Note: the ESP32-CAM only supports streaming to one client at a time. So, you can have just one web browser tab opened with the web server. You can't have video streaming on your computer and in your smartphone at the same time. If you're on your computer and then, you want to access video streaming in your smartphone, you need to close the tab in your computer and then, open a new in your smartphone.

In our opinion, the image is pretty good taking into account such a small and inexpensive camera. The following picture was taken with UXGA (1600x1200) resolution.



Note: unfortunately, this example doesn't save the photos, but you can modify it to use the on board microSD card to store the captured photos. We'll see how to save photos on the microSD card in the next sections.

To give you an idea of the differences between the normal camera and the camera with fish-eye lens, you can take a look at the following photo (taken with fish-eye lens). The camera was in the same place as in the previous photo.



As you can see, the camera with [fish-eye lens](#) captures a wider area, so it is a better option if you want to use it for surveillance.

This video streaming interface, allows you to select different resolutions:

- **UXGA:** 320x240
- **SXGA:** 1280x1204
- **XGA:** 1024x768
- **SVGA:** 800x600
- **VGA:** 640x480
- **CIF:** 400x296

- **QVGA:** 320x240
- **HQVGA:** 240x176
- **QQVGA:** 160x120

There are also many other camera settings that you can play with like brightness, contrast, saturation, several effects like grayscale, sepia, and many others. We recommend exploring all the options to see what you can do with the ESP32-CAM.

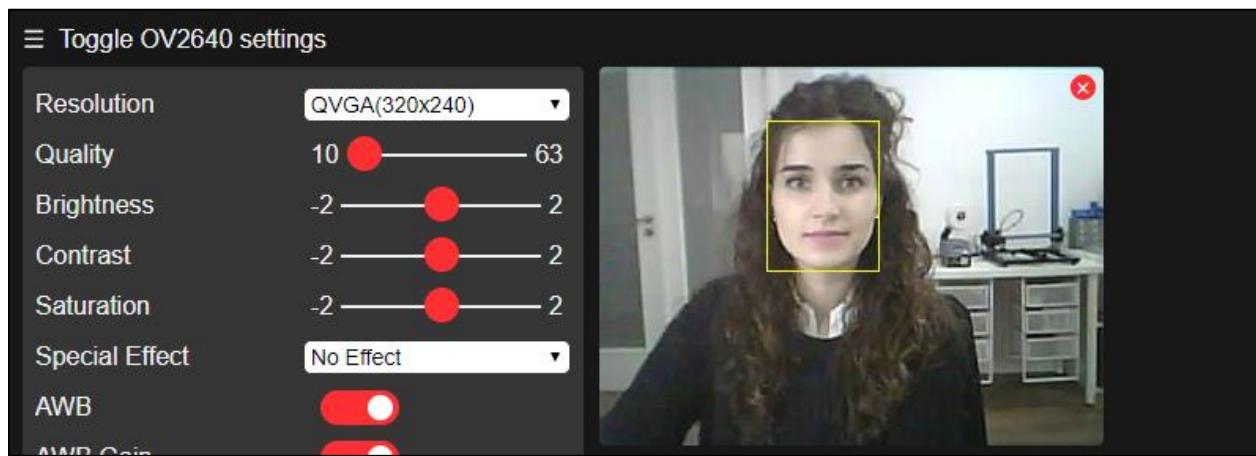
[Module 2, Unit 3](#) shows how to implement all those settings in your code.

Face Detection and Face Recognition

This example also includes face recognition and face detection. Face recognition and detection only works on CIF resolution or lower.

Note: face recognition and detection doesn't work with boards without PSRAM.

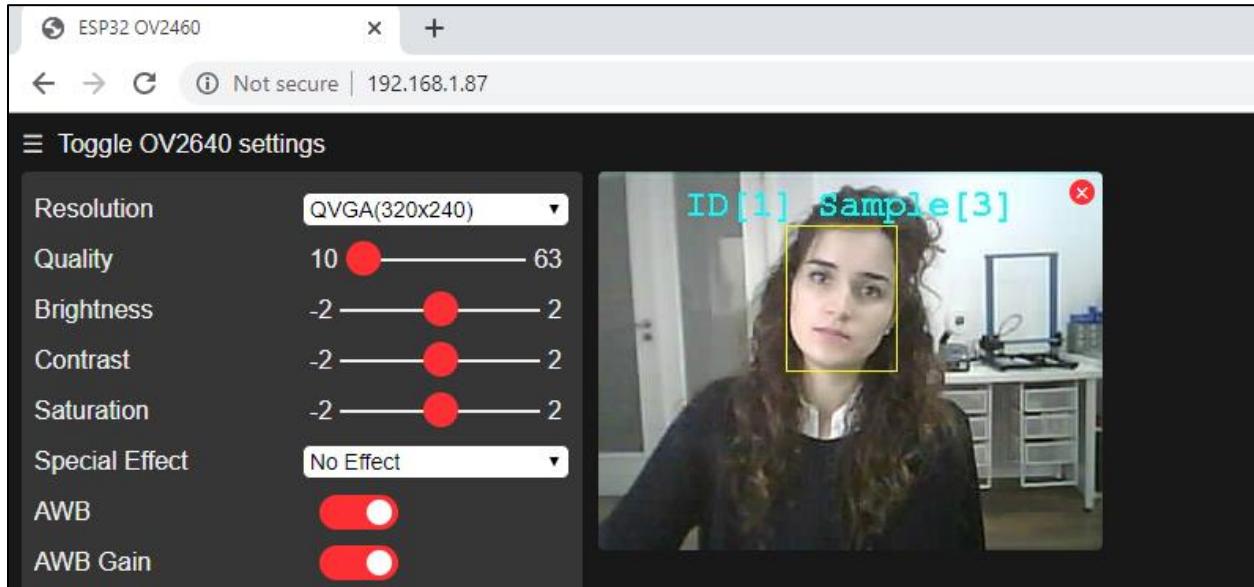
If you toggle the **Face Detection** button, and if you point the camera at your face, it should draw a yellow rectangle around your face. In this example, we're using QVGA (320x240) resolution.



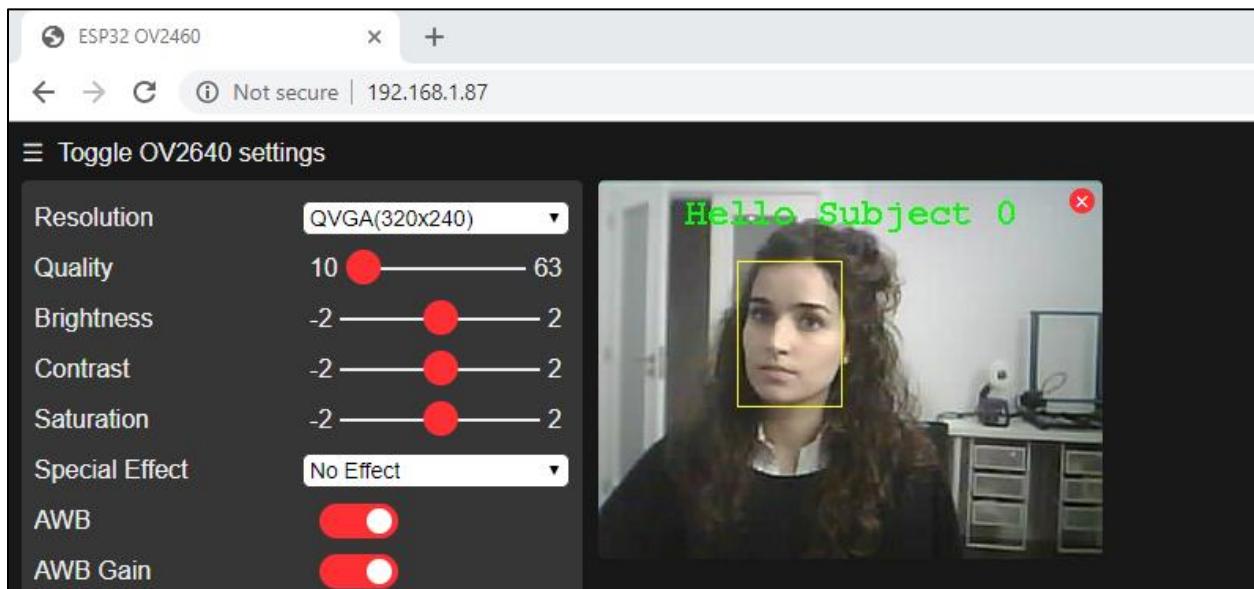
If you get a yellow rectangle around your face, it means that your ESP32-CAM was able to detect your face, and it will probably be able to detect other faces too. Face

detection works pretty well, it detects the face almost immediately, and it doesn't struggle following the face as it moves.

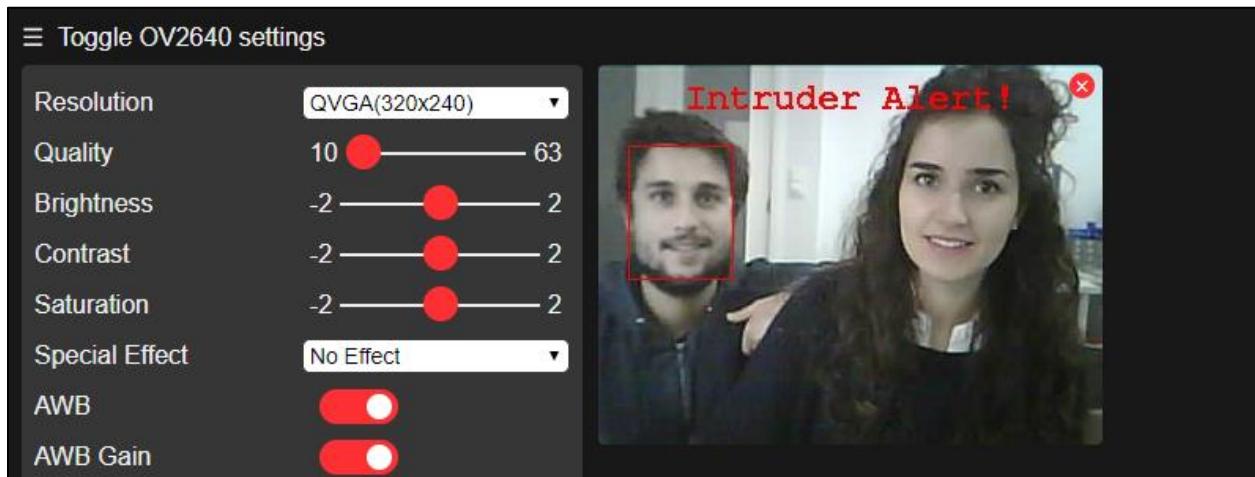
To experiment face recognition, you need to enroll a new face first. Toggle the **Face Recognition** option and press the **Enroll Face** button. It will make 5 attempts to save your face.



After enrolling your face, it should be able to recognize your face as subject 0 (if this is the first face you enrolled) later on.



If it doesn't recognize the face, it and will display a red rectangle and an "Intruder Alert!" message.



Face detection works pretty well, but face recognition is a bit slow, and you'll notice that the streaming will lag a bit when trying to identify faces.

Add/Change Camera Pin Assignment

If your camera is not included in the code, open the **camera_pins.h** tab.

A screenshot of the Arduino IDE showing the CameraWebServer sketch. The tabs at the top are "File", "Edit", "Sketch", "Tools", and "Help". Below the tabs, there are icons for saving, loading, and navigating. The tabs in the code editor are "CameraWebServer", "app_httpd.cpp", "camera_index.h", and "camera_pins.h", with "camera_pins.h" highlighted by a red box. The code in the editor is as follows:

```
1 |
2 #if defined(CAMERA_MODEL_WROVER_KIT)
3 #define PWDN_GPIO_NUM      -1
4 #define RESET_GPIO_NUM     -1
5 #define XCLK_GPIO_NUM       21
6 #define SIOD_GPIO_NUM       26
7 #define SIOC_GPIO_NUM       27
8
9 #define Y9_GPIO_NUM         35
10 #define Y8_GPIO_NUM        34
```

Create a new camera model with your pin definition. For example, the M5-Camera B from M5stack pin assignment is not included by default. From the M5-Camera B datasheet, we know it has the following pin assignment:

| M5-CAMERA | ESP32 | Variable name on code |
|-----------|---------|-----------------------|
| D0 | GPIO 32 | Y2_GPIO_NUM |
| D1 | GPIO 35 | Y3_GPIO_NUM |
| D2 | GPIO 34 | Y4_GPIO_NUM |
| D3 | GPIO 5 | Y5_GPIO_NUM |
| D4 | GPIO 39 | Y6_GPIO_NUM |
| D5 | GPIO 18 | Y7_GPIO_NUM |
| D6 | GPIO 36 | Y8_GPIO_NUM |
| D7 | GPIO 19 | Y9_GPIO_NUM |
| XCLK | GPIO 27 | XCLK_GPIO_NUM |
| PCLK | GPIO 21 | PCLK_GPIO_NUM |
| VSYNC | GPIO 25 | VSYNC_GPIO_NUM |
| HREF | GPIO 26 | HREF_GPIO_NUM |
| SDA | GPIO 22 | SIOD_GPIO_NUM |

| | | |
|-----------|---------|----------------|
| SCL | GPIO 23 | SIOC_GPIO_NUM |
| POWER PIN | GPIO 15 | RESET_GPIO_NUM |

So, we can create a new camera, in the **camera_pins.h** tab with that pin definition.

In this case, we're calling it **CAMERA_MODEL_M5STACK_PSRAM_B**.

```
#elif defined(CAMERA_MODEL_M5STACK_PSRAM_B)
#define PWDN_GPIO_NUM      -1
#define RESET_GPIO_NUM     15
#define XCLK_GPIO_NUM      27
#define SIOD_GPIO_NUM      22
#define SIOC_GPIO_NUM      23

#define Y9_GPIO_NUM         19
#define Y8_GPIO_NUM         36
#define Y7_GPIO_NUM         18
#define Y6_GPIO_NUM         39
#define Y5_GPIO_NUM         5
#define Y4_GPIO_NUM         34
#define Y3_GPIO_NUM         35
#define Y2_GPIO_NUM         32
#define VSYNC_GPIO_NUM      25
#define HREF_GPIO_NUM       26
#define PCLK_GPIO_NUM       21
```

Copy the previous pinout to the **camera_pins.h** tab after the first camera model.

Then, in the main code tab **CameraWebServer**, define your own camera model as follows:

```
// Select camera model
//#define CAMERA_MODEL_WROVER_KIT
//#define CAMERA_MODEL_ESP_EYE
///#define CAMERA_MODEL_M5STACK_PSRAM
///#define CAMERA_MODEL_M5STACK_WIDE
///#define CAMERA_MODEL_AI_THINKER
#define CAMERA_MODEL_M5STACK_PSRAM_B
```

Note: go to the [Appendix](#) to check the pinout for other ESP32 camera development boards.

Wrapping Up

Face detection works pretty well, but face recognition is a bit slow, and sometimes it won't recognize you if you are making a different facial expression than the one it enrolled. Also, streaming with face recognition enabled is extremely slow. So, you need to have some realistic expectations about this camera when it comes to face recognition.

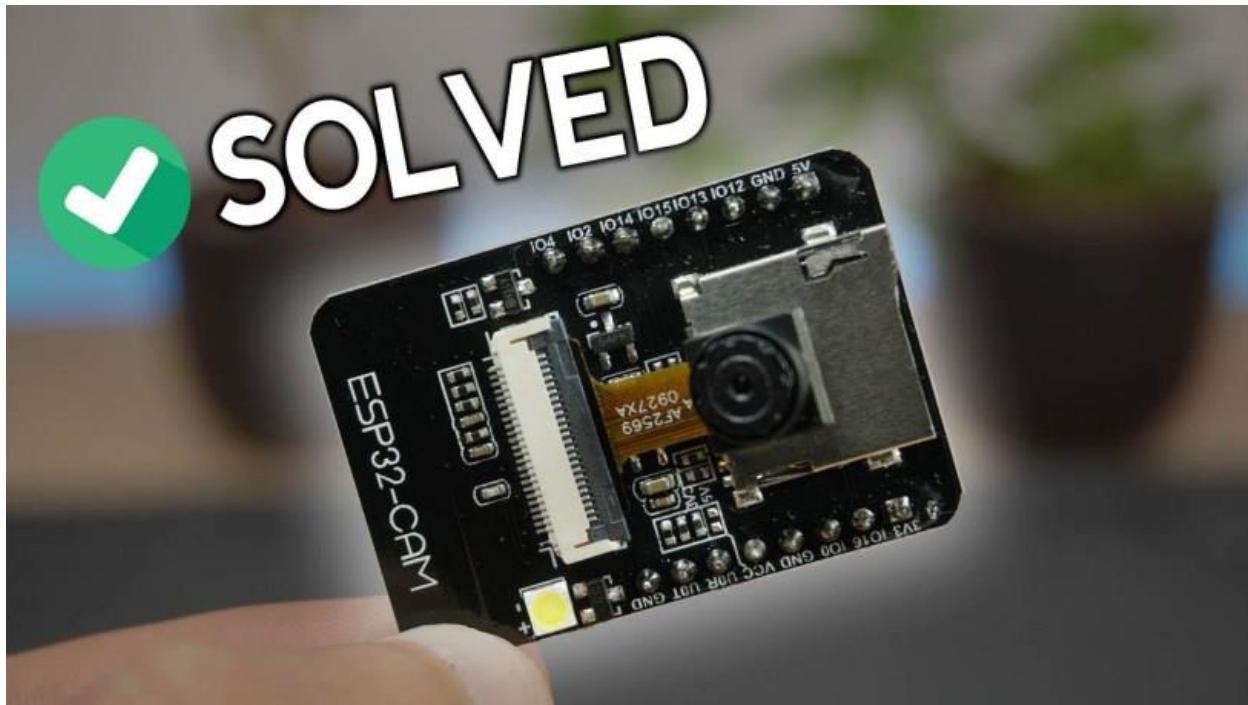
Nonetheless, the results are very impressive for such a small and cheap camera development board. Video streaming and taking photos works flawlessly, and the image is pretty good if you select the highest resolution options.

We highly recommend using an external antenna with your ESP32-CAM, otherwise this example will not work as expected – very slow streaming and lag.

That's it for this project. Now you have your video streaming web server up and running with face detection and recognition with the example from the library.

If you made it until here, congratulations! If your project is not working, or it is not working as expected, or if you're stuck at some of the parts, take a look at the troubleshooting guide in the next Unit.

Unit 4: Troubleshooting Most Common Problems



If you've followed the previous project without any issues, you can proceed to the next Unit. You can come back to this Troubleshooting Guide whenever you have a problem and see if you can find a solution for your issue.

If you had trouble following the previous Unit, we recommend taking a look at the following list of the most common errors. Most of the problems you'll find with the ESP32-CAM are related with the issues referred in this guide.

1. Failed to connect to ESP32: Timed out waiting for packet header

This is one of the most common problems. You are not able to upload code and you get the following message.

```

A fatal error occurred: Failed to connect to ESP32: Timed out waiting for packet header
Sketch uses 2233518 bytes (71%) of program storage space. Maximum is 3145728 bytes.
Global variables use 50692 bytes (15%) of dynamic memory, leaving 276988 bytes for local
esptool.py v2.6-beta1
Serial port COM10
Connecting.....  

A fatal error occurred: Failed to connect to ESP32: Timed out waiting for packet header

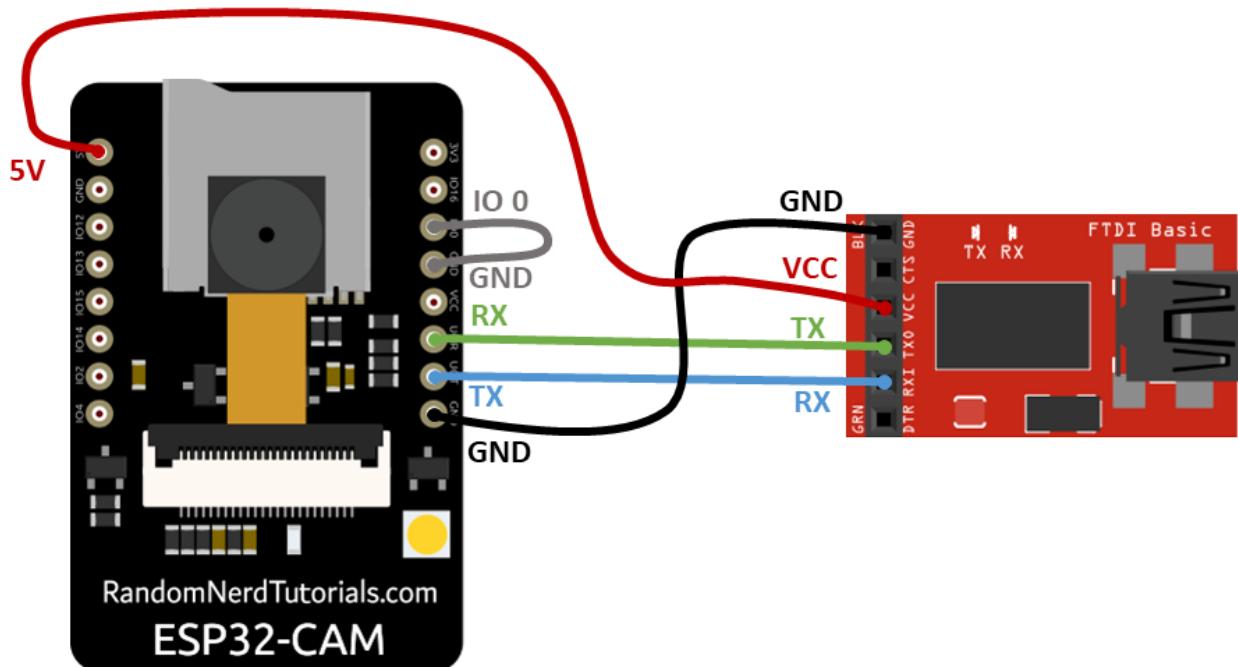
```

This error means that the ESP32-CAM is not in flashing mode or it is not connected properly to the FTDI programmer.

Double-check the steps to upload code

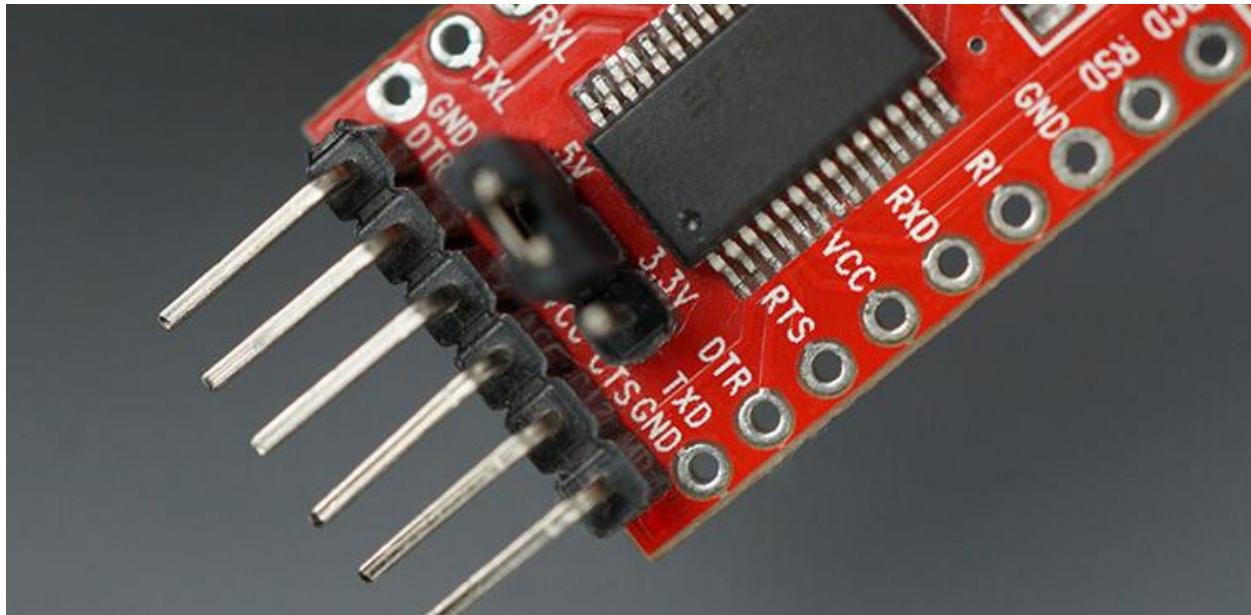
Double-check that you've followed the exact steps to put your ESP32-CAM in flashing mode. Failing to complete one of the steps may result in that error. Follow exactly the following steps to upload code.

Connect the ESP32-CAM board to your computer using an FTDI programmer. Follow the next schematic diagram:



Note: the order of the FTDI pins on the diagram may not match yours. Make sure you check the silkscreen label next to each pin.

Many FTDI programmers have a jumper that allows you to select 3.3V or 5V. **Make sure the jumper is in the right position to select 5V.**



Warning: some FTDI programmers only output 3.3V even when selecting the 5V option. So, **check the output of the FTDI programmer VCC pin with a multimeter** to make sure you're getting the right voltage. Many people are unable to upload code when powering the ESP32-CAM with 3.3V.



You can also use the following table as a reference:

| ESP32-CAM | FTDI Programmer |
|-----------|-----------------|
| GND | GND |
| 5V | VCC (5V) |
| U0R | TX |
| T0R | RX |
| GPIO 0 | GND |

Note that GPIO 0 needs to be connected to GND to upload code – you can use a female-to-female jumper wire to connect the pins. Having GPIO 0 connected to GND puts the ESP32 in flashing mode. This means the ESP32 is ready to receive new code.

The ESP32-CAM Receiver pin (U0R) connects to the FTDI programmer transmitter pin (TX), the ESP32-CAM Transmitter pin (T0R) connects to the FTDI programmer receiver pin (RX). Many people swap these connections, and you won't be able to establish a communication with the ESP32-CAM.

Once you made all the necessary connections, in your Arduino IDE, make sure you have your ESP32-CAM board selected in **Tools > Board**. Then, go to **Tools > Port** and select the COM port the ESP32-CAM is connected to.

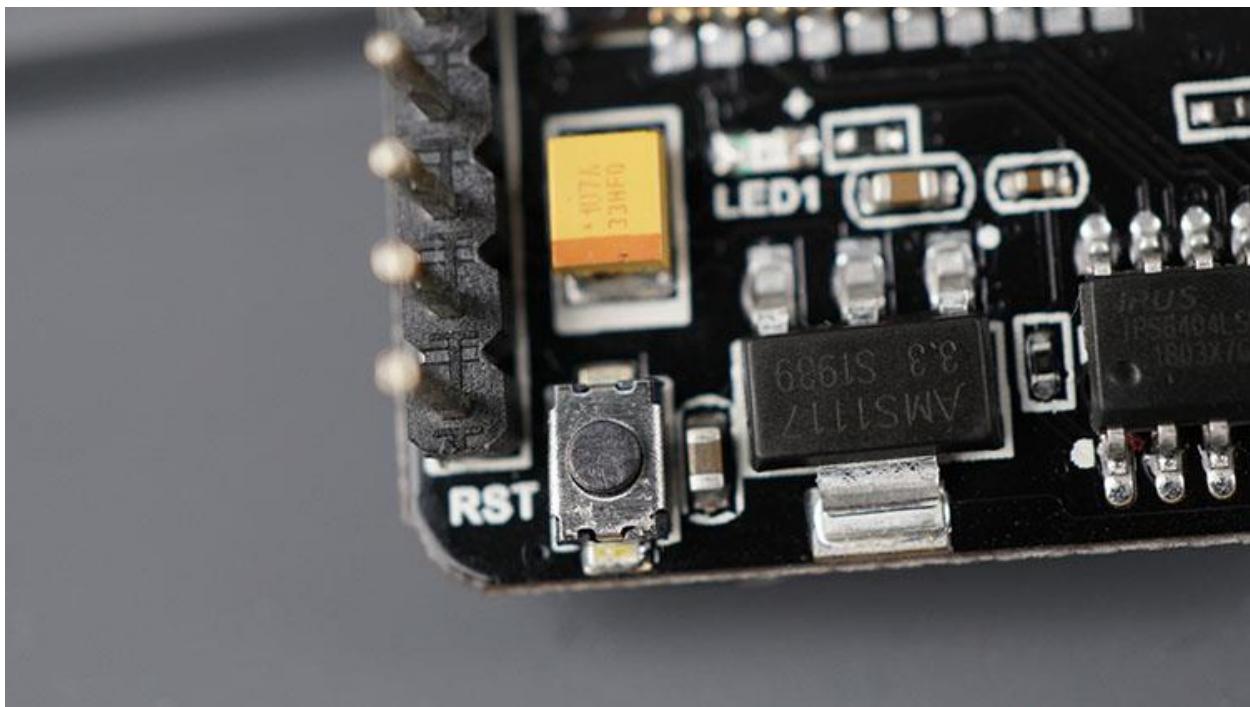
Then, click the upload button.



When you start to see these dots on the debugging window as shown below:

```
esptool.py v2.6-beta1
Serial port COM10
Connecting.....
```

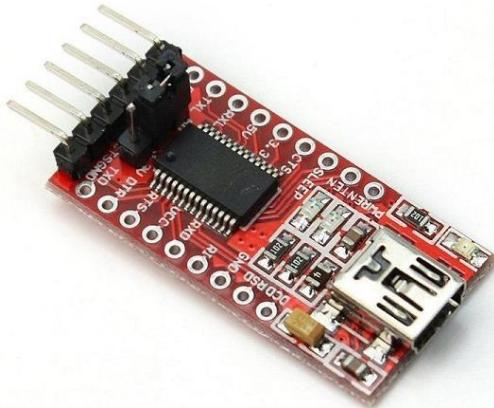
Press the ESP32-CAM on-board RST button. After a few seconds, the code should be successfully uploaded to your board.



If you're using an M5-stack camera module or any board with built-in programmer, you just need to connect it to your computer through the USB cable and click the upload button. You can select the **AI-Thinker** module in the **Boards** menu – at the time of writing this Unit, there isn't an option for M5-stack camera boards.

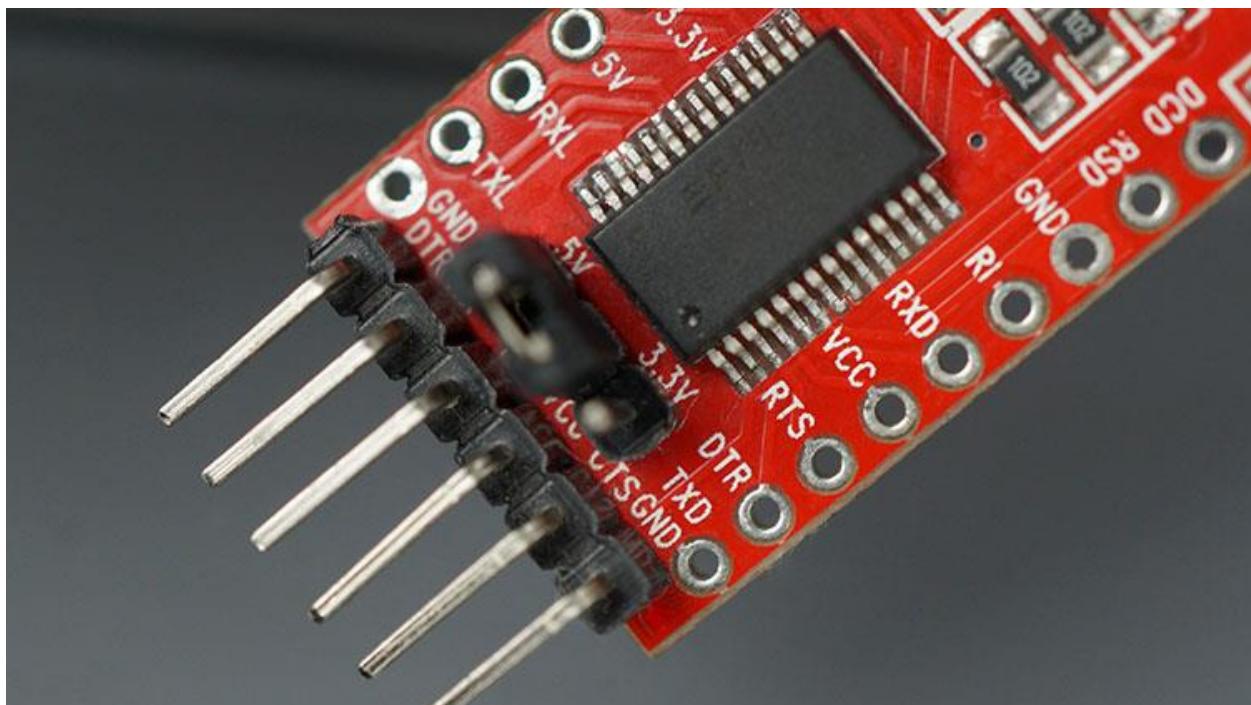
Check the FTDI programmer you are using

One of our readers reported the following: "found out that you can program the board with a USB-to-TTL module model CP2102 and that the CH340 model does NOT work" – [recommended FTDI programmer](#).



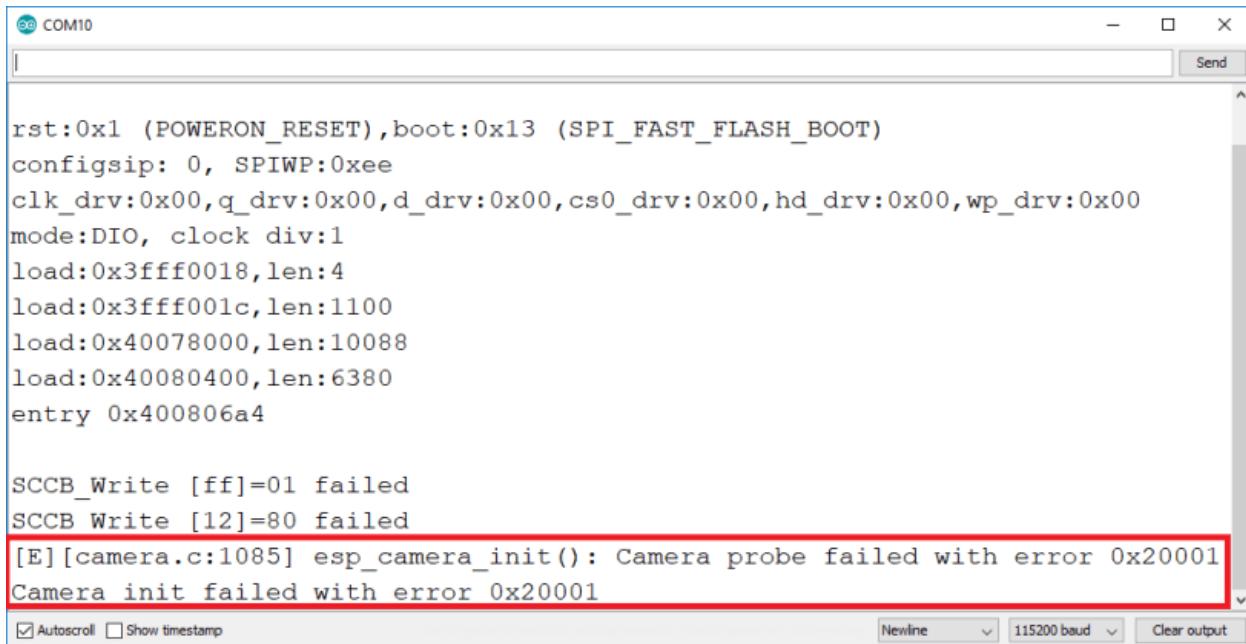
Power the ESP32-CAM with 5V

Double-check that your FTDI programmer has the jumper in the right place to select 5V as shown in the figure below.



Some FTDI programmers only output 3.3V even when selecting the 5V option. So, **check the output of the FTDI programmer VCC pin with a multimeter** to make sure you're getting the right voltage. Many people are unable to upload code when powering the ESP32-CAM with 3.3V.

2. Camera init failed with error 0x20001 or similar



```
rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:1100
load:0x40078000,len:10088
load:0x40080400,len:6380
entry 0x400806a4

SCCB_Write [ff]=01 failed
SCCB Write [12]=80 failed
[E] [camera.c:1085] esp_camera_init(): Camera probe failed with error 0x20001
Camera init failed with error 0x20001
```

- **0x20001:** camera not detected
- **0x2002:** failed to set frame size
- **0x2003:** failed to set output format
- **0x2004:** camera not supported

If you get the 0x2001 or any of the other errors above, it means that your camera OV2640 is not connected properly to your ESP32 board or you have the wrong pin assignment in the code.

Sometimes, unplugging and plugging the FTDI programmer multiple times or restart the board multiple times, might solve the issue.

Camera not connected properly

The camera has a tiny connector and you must ensure it's connected in the right way and with a secure fit, otherwise it will fail to establish a connection.

Wrong pin assignment in the code

When you get this error, it might also mean that you didn't select the right board in the define section or the pin definition is wrong for your board. Make sure you select the right camera module in your projects. You just need to uncomment the right camera module and comment all the others.

If you're using a camera like ours, you should select the `CAMERA_MODEL_AI_THINKER`. Comment the `CAMERA_MODEL_WROVER_KIT` and uncomment the `CAMERA_MODEL_AI_THINKER` as shown below.

```
// Select camera model
//#define CAMERA_MODEL_WROVER_KIT
//#define CAMERA_MODEL_ESP_EYE
//#define CAMERA_MODEL_M5STACK_PSRAM
//#define CAMERA_MODEL_M5STACK_WIDE
#define CAMERA_MODEL_AI_THINKER
```

Your camera has a different pin assignment

If you're using a different board, it probably has a different pin assignment. Make sure you get the right pin assignment for your board. If you're following the CameraWebServer example, you can change the pin assignment on the **camera_pins.h** tab.

You can also take a look at the [Appendix](#) to check the pinout for several ESP32 camera boards.

“Clone/Fake” ESP32-CAM boards

There are many ESP32-CAM boards being released (“clone/fake boards”) that the wiring between the ESP32 and the OV camera might be different, so selecting the camera module, might not be enough. You might need to check each GPIO definition with your board pinout. To be honest, this never happened to us, but some of our readers reported this kind of issue.

Not enough power through USB source

Some USB ports can't provide enough power to operate the ESP32-CAM. You might need to get an external 5V power supply.

Faulty FTDI programmer

Some readers also reported this problem was solved by replacing their actual FTDI programmer with a new one.

The camera/connector is broken

If you get this error, it might also mean that your camera or the camera flex cable is broken. If that is the case, you may get a new OV2640 camera probe.

3. Brownout Detector or Guru Meditation Error

When you open your Arduino IDE Serial monitor and the error message "Brownout detector was triggered" is constantly being printed over and over again. It means that there's some sort of hardware problem.

It's often related to one of the following issues:

- Poor quality USB cable;
- USB cable is too long;
- Board with some defect (bad solder joints);
- Bad computer USB port;
- Or not enough power provided by the computer USB port.

Solution:

- try a different shorter USB cable (with data wires);

- use a different computer USB port or use a USB hub with an external power supply;
- Also, follow the suggestions described in issue 2.

4. Board at COMX is not available – COM Port Not Selected

```
the selected serial port Failed to execute script esptool does not exist or your board is not connected
File "site-packages\serial\__init__.py", line 88, in serial_for_url
  File "site-packages\serial\serialwin32.py", line 62, in open
serial.serialutil.SerialException: could not open port 'COM8': WindowsError(2, 'The system
Failed to execute script esptool
the selected serial port Failed to execute script esptool
  does not exist or your board is not connected

ESP32 Wrover Module, Huge APP (3MB No OTA), QIO, 80MHz, 921600, None on COM8
```

If you get the following error or similar:

```
serial.serialutil.SerialException:    could    not    open    port    'COM8':
WindowsError(2, 'The system cannot find the file specified.')
Failed to execute script esptool
the selected serial port Failed to execute script esptool
  does not exist or your board is not connected
Board at COM8 is not available
```

It means that you haven't selected the COM port in the **Tools** menu. In your Arduino IDE, go to **Tools** ▶ **Port** and select the COM port the ESP32 is connected to.

It might also mean that the ESP32-CAM is not establishing a serial connection with your computer or it is not properly connected to the USB connector.

5. Psram error: GPIO isr service is not installed

```
ets Jun  8 2016 00:22:57

rst:0x1 (POWERON_RESET),boot:0x12 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:
mode:DIO, clock div:1
load:0x3fff0018,len:4
load:0x3fff001c,len:1100
load:0x40078000,len:10088
load:0x40080400,len:6380
entry 0x400806a4
E (161) gpio: gpio_isr_handler_remove(380): GPIO isr service is not
Camera init failed with error 0x101
```

You are using a board without PSRAM and you get the following error or similar:

```
E (161) gpio: gpio_isr_handler_remove(380): GPIO isr service is not
installed, call gpio_install_isr_service() first
Camera init failed with error 0x101
```

when the board was initialized with the following settings:

```
config.frame_size = FRAMESIZE_UXGA;
config.jpeg_quality = 10;
config.fb_count = 2;
```

Adding the following will fix the errors (it lowers the image resolution so it won't need so much space to store images. However, as a result, you cannot get higher resolution formats due to the limited memory):

```
if(psramFound()) {
    config.frame_size = FRAMESIZE_UXGA;
    config.jpeg_quality = 10;
    config.fb_count = 2;
} else {
    config.frame_size = FRAMESIZE_SVGA;
    config.jpeg_quality = 12;
    config.fb_count = 1;
}
```

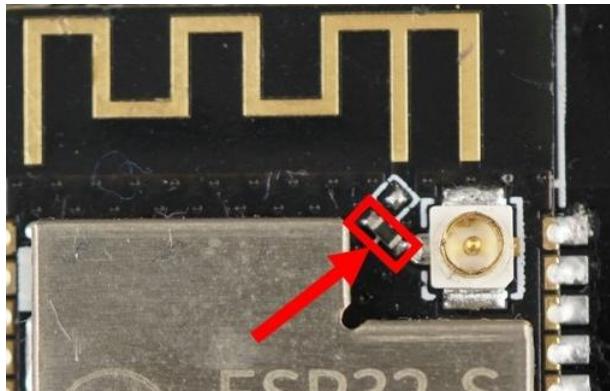
Note: face recognition and detection doesn't work with boards without PSRAM.

6. Weak Wi-Fi Signal

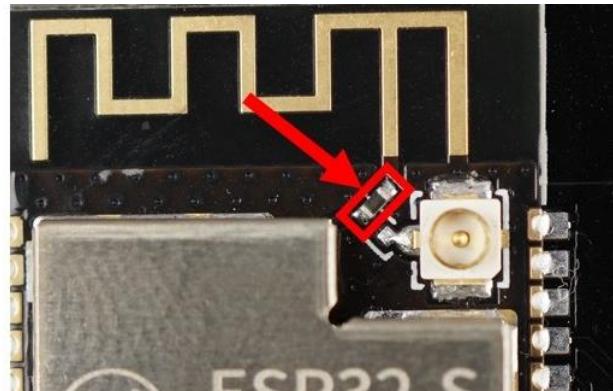
The ESP32-CAM has the option to use either the built-in antenna or an external antenna. If your ESP32-CAM AI-Thinker has no Wi-Fi connection or poor connection, it might have the wrong antenna enabled. The ESP32-CAM has the option to use either the built-in antenna or an external antenna.

Next to the IPEX connector there are three little white squares laid out like a “<” with the middle position being common. There is a resistor selecting the desired antenna.

To use the PCB antenna (on-board antenna), the resistor must be on the top position, like this “/”. To use the IPEX connector with an external antenna, the resistor must be on the bottom position, like this “\”. See illustration below.



External Antenna



On-board Antenna

Take a look at your board to see if it is set to use the on-board antenna or the IPEX connector. Using the on-board antenna works well if you are close to your router. We recommend using the IPEX connector with an external antenna for better results.

To enable or disable the on-board antenna, you just need to unsolder that resistor and solder it in the desired configuration. You can also drop some solder to connect those points (you don't necessarily need to add the resistor as long as the pads are connected). **You can't use the two antennas at the same time**, so you can only have one connection for the antenna.

7. No IP Address in Arduino IDE Serial Monitor

If you just see dots printed in the serial monitor (.....), it means that your ESP32-CAM is not establishing a Wi-Fi connection with your router.

Double-check your network credentials

You need to make sure that you've typed your exact network credentials (SSID and password) in the following variables:

```
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
```

Select the right baud rate

If you don't select the right baud rate in the Arduino IDE Serial Monitor, you won't get your board IP address or you'll just get garbage on the screen.

Make sure you select the right baud rate. In our examples with the ESP32-CAM, we use 115200 baud rate.

Reset the board multiple times

You might also need to press the ESP32-CAM on-board RESET button multiple times to restart your ESP and print the IP address during boot. At this point, **GPIO 0 must be disconnected from GND**.

RX and TX swapped

Double-check the connections between your ESP32 board and the FTDI programmer. RX goes to TX and TX goes to RX. If these connections are swapped, the ESP32-CAM is not able to establish a serial communication with your computer.

Wi-Fi Range

If the router is far away from your ESP32 board, it might not be able to catch the Wi-Fi signal. Ensure that your ESP32-CAM is fairly close to your router or that you use an external antenna.

Check the Antenna

Check the antenna connection. Check whether the board is selected to use the on-board antenna or the external antenna. Read [issue 6](#).

8. Can't open web server

If the ESP32-CAM is printing the IP address in your Arduino IDE Serial Monitor, but when you try to open the web server in your web browser you see a blank screen, it usually means that you are trying to access the ESP32-CAM web server with multiple web browser tabs.

At the moment, these ESP32-CAM sketches only work with one client connected at a time.

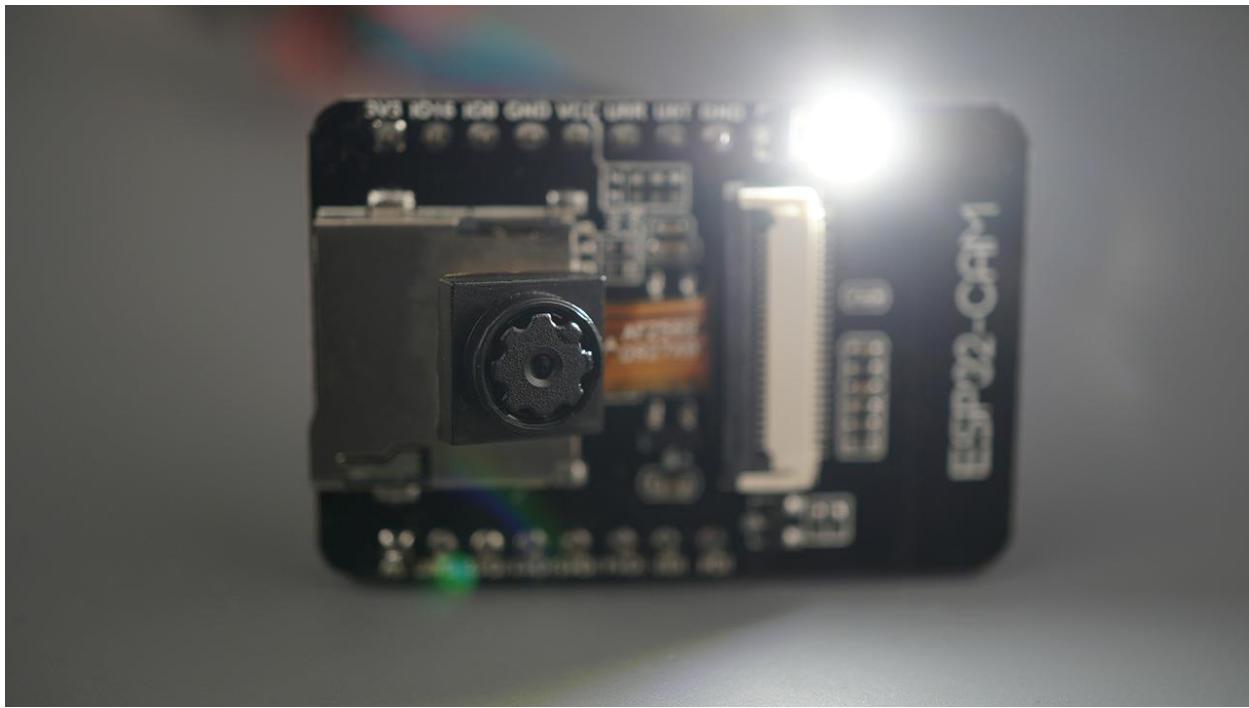
Also, the Wi-Fi signal can be too weak to show video streaming. Check the previous issues related to weak Wi-Fi signal.

Wrapping Up

These are the most common problems with the ESP32-CAM. We also have a Troubleshooting Guide on our website that is often updated with suggestions from our readers. To keep up to date with the most common issues and possible solutions, you can follow our article in the following link:

- [ESP32-CAM Troubleshooting Guide: Most Common Problems Fixed](#)

Unit 5: ESP32-CAM Flashlight and External Pushbutton



The ESP32-CAM works like any other ESP32 development board. However, you need to keep in mind that a bunch of GPIOs are connected to the OV2640 camera and to the microSD card interface. If you're not using the camera or the microSD card functionalities, you can use the ESP32-CAM GPIOs the way you want.

Compatibility: this Unit is compatible with **ESP32-CAM AI-Thinker**. The examples in this Unit show how to control the on-board LED connected to GIO 4. This is the only ESP32-CAM dev board with this feature. If you're using a different board, you can still follow along to learn how to control outputs and read inputs with the ESP32, but you won't be able to see the final demonstration.

The commands to read inputs and control outputs are exactly the same as in a normal ESP32 development board. In this Unit, we'll show you some simple examples.

1. Control the ESP32-CAM flashlight;
2. Control the flashlight with a pushbutton;
3. Control the flashlight brightness.

The ESP32-CAM has a very bright built-in LED that can work as a flash when taking photos. That LED is internally connected to GPIO 4. That GPIO is also connected to the microSD card slot, so you may have troubles when trying to use both at the same time. In this section, we'll just show you some simple examples to control the LED, without using anything else (camera or microSD card).

If you're already quite familiar with the ESP32, you can skip this section or take a quick look to refresh your memory about some simple concepts.

Control the Flashlight

The following code blinks the ESP32-CAM flashlight every 2 seconds.

CODE

https://github.com/RuiSantosdotme/ESP32-CAM-eBook/blob/master/Code/Module_1/Blink_Flashlight/Blink_Flashlight.ino

```
// ledPin refers to ESP32-CAM GPIO 4 (flashlight)
const int ledPin = 4;

void setup() {
    // initialize digital pin ledPin as an output.
    pinMode(ledPin, OUTPUT);
}

void loop() {
    digitalWrite(ledPin, HIGH);
    delay(2000);
    digitalWrite(ledPin, LOW);
    delay(2000);
}
```

Note: we recommend that you copy the code from the link provided in the **CODE** box to prevent errors related with copy/paste from the eBook PDF. Additionally, the code provided in the link is the most up-to-date code (in case the code needs some changes after the eBook publication).

How the Code Works

Create a variable `ledPin` that refers to GPIO 4 (the built-in flashlight is connected to).

```
const int ledPin = 4;
```

Then, in the `setup()`, define the `ledPin` as an `OUTPUT`.

```
pinMode(ledPin, OUTPUT);
```

In the `loop()`, turn the LED on using the `digitalWrite()` function and passing as argument the GPIO you want to control and the state:

```
digitalWrite(ledPin, HIGH);
```

Wait for 2 seconds (2000 milliseconds)

```
delay(2000);
```

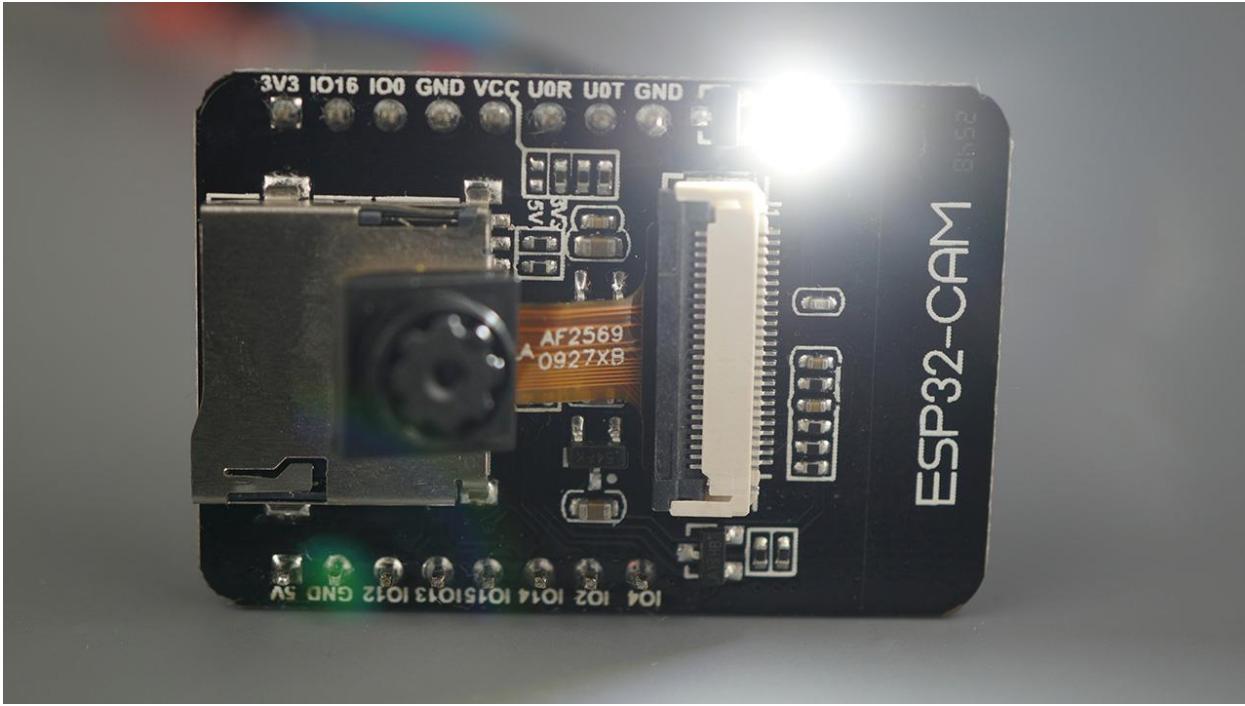
Turn the LED off for another 2 seconds.

```
digitalWrite(ledPin, LOW);
delay(2000);
```

The `loop()` repeats over and over again blinking the LED.

Demonstration

Upload the code to your ESP32-CAM. After uploading, disconnect GPIO 0 from GND, press the on-board RST button, and the flash LED should be blinking every 2 seconds.



Warning: the flashlight is very bright, avoid looking at the LED directly.

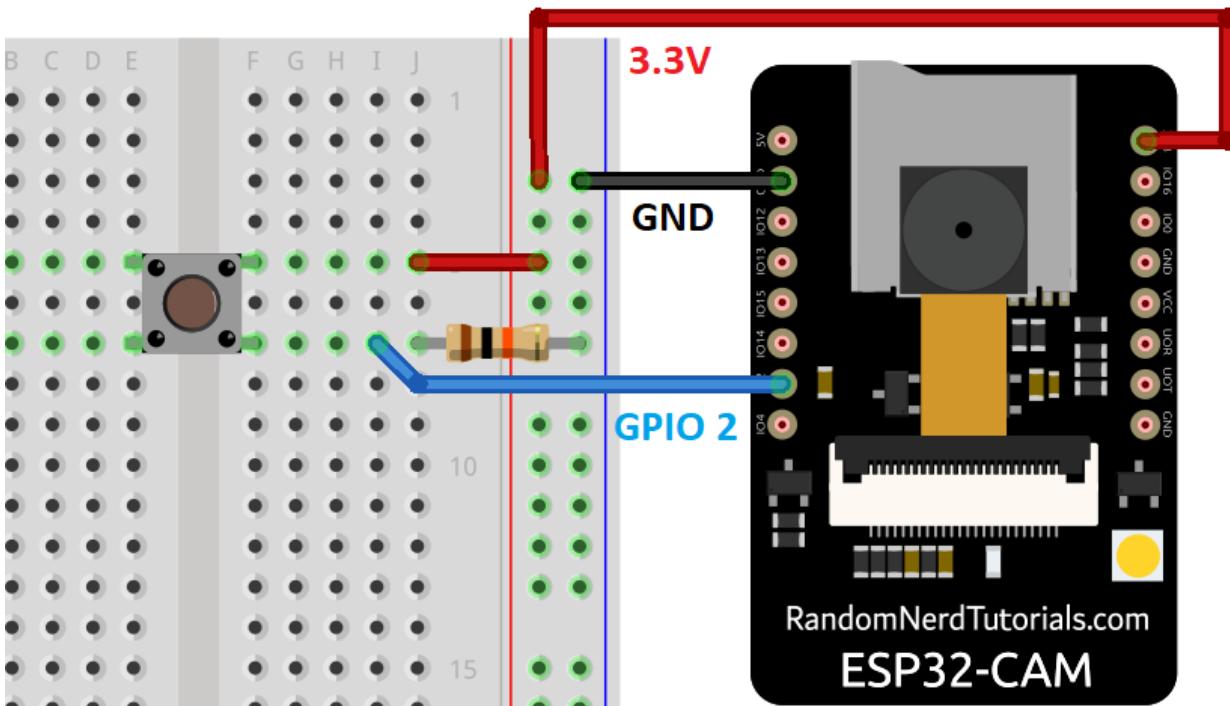
Controlling the Flashlight with a Pushbutton

In this section, we'll control the on-board LED with an external pushbutton. When you press the button, the flash turns on. When you release the pushbutton, the LED remains turned off.

Parts Required:

- [ESP32-CAM](#)
- [Pushbutton](#)
- [Breadboard](#)
- [10k Ohm resistor](#)
- [Jumper wires](#)

Wire a pushbutton to the ESP32-CAM as shown in the following schematic diagram.



Upload the following code to your board.

CODE

[https://github.com/RuiSantosdotme/ESP32-CAM-eBook/blob/master/Code/Module 1/Flashlight Pushbutton/Flashlight Pushbutton.ino](https://github.com/RuiSantosdotme/ESP32-CAM-eBook/blob/master/Code/Module%201/Flashlight%20PushButton/Flashlight%20PushButton.ino)

```
// GPIO 2 connected to pushbutton with 10k Ohm pull-down resistor
const int buttonPin = 2;
const int ledPin = 4;

int buttonState = 0;

void setup() {
  Serial.begin(115200);
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT);
}

void loop() {
  buttonState = digitalRead(buttonPin);
  Serial.println(buttonState);

  if (buttonState == HIGH) {
    digitalWrite(ledPin, HIGH);
  } else {
    digitalWrite(ledPin, LOW);
  }
}
```

How the Code Works

Create variables for the pushbutton and for the LED. In this case, we've connected the pushbutton to GPIO 2. The LED is internally connected to GPIO 4.

```
const int buttonPin = 2;  
const int ledPin = 4;
```

Create a variable to hold the button state:

```
buttonState = digitalRead(buttonPin);
```

In the `setup()`, set the button as an INPUT, and the LED as an OUTPUT.

```
pinMode(ledPin, OUTPUT);  
pinMode(buttonPin, INPUT);
```

In the `loop()`, read the pushbutton state and save it in the `buttonState` variable.

```
buttonState = digitalRead(buttonPin);
```

To read the state of a GPIO, you just need to use the `digitalRead()` function and pass as argument the GPIO you want to read the state.

When you press the pushbutton, the button state goes HIGH. When that happens, turn on the flashlight.

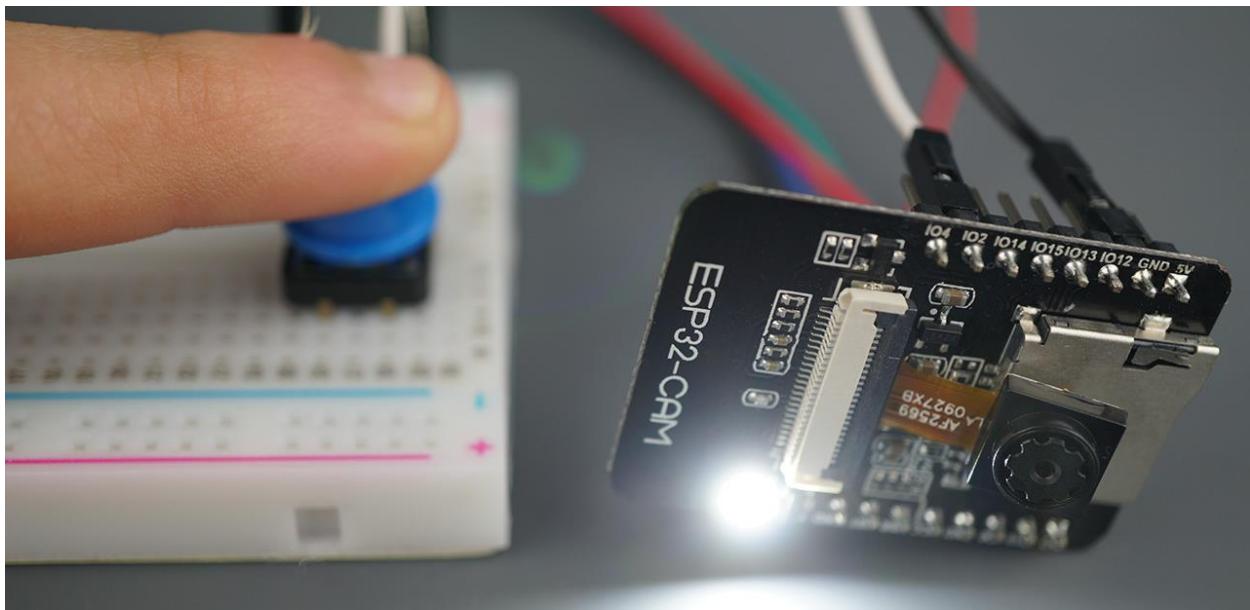
```
if (buttonState == HIGH) {  
    digitalWrite(ledPin, HIGH);
```

When you're not pressing the pushbutton, the flashlight remains off.

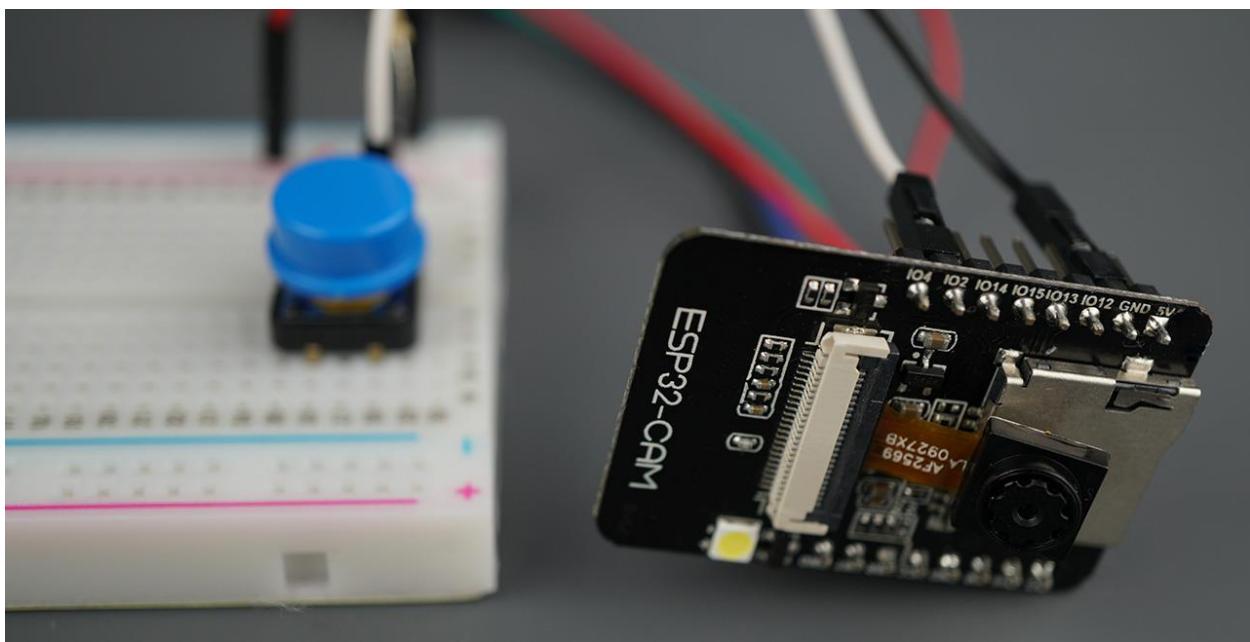
```
} else {  
    digitalWrite(ledPin, LOW);  
}
```

Demonstration

When you press the pushbutton, the flash LED lights up.



When you release the button, the LED remains off.



Controlling the Flashlight Brightness

Controlling the intensity of the flashlight is as easy as controlling the brightness of an LED - you need to use PWM.

The following code increases and decreases the LED brightness over time.

CODE

https://github.com/RuiSantosdotme/ESP32-CAM-eBook/blob/master/Code/Module_1/Flashlight_Brightness_PWM/Flashlight_Brightness_PWM.ino

```
// flashlight is connected to GPIO 4
const int ledPin = 4;

// setting PWM properties
const int freq = 5000;
const int ledChannel = 0;
const int resolution = 8;

void setup() {
    // configure LED PWM functionalities
    ledcSetup(ledChannel, freq, resolution);
    // attach the channel to the GPIO to be controlled
    ledcAttachPin(ledPin, ledChannel);
}

void loop() {
    // increase the LED brightness
    for(int dutyCycle = 0; dutyCycle <= 255; dutyCycle++) {
        // changing the LED brightness with PWM
        ledcWrite(ledChannel, dutyCycle);
        delay(15);
    }
    // decrease the LED brightness
    for(int dutyCycle = 255; dutyCycle >= 0; dutyCycle--) {
        // changing the LED brightness with PWM
        ledcWrite(ledChannel, dutyCycle);
        delay(15);
    }
}
```

How the Code Works

Start by defining the pin the LED is attached to. In case of the ESP32-CAM, the LED is attached to GPIO 4.

```
const int ledPin = 4;
```

Then, set the PWM signal properties. Define a frequency of 5000 Hz, choose channel 0 to generate the signal, and set a resolution of 8 bits. You can choose other properties, different than these, to generate different PWM signals.

```
const int freq = 5000;
const int ledChannel = 0;
const int resolution = 8;
```

Important: the OV2640 camera uses PWM channel 0 when working with the ESP32-CAM. So, if you're using PWM with the ESP32 and the camera at the same time, select a different channel for the PWM that is controlling the flashlight.

In the `setup()`, you need to configure LED PWM with the properties you've defined earlier by using the `ledcSetup()` function that accepts as arguments, the `ledChannel`, the `frequency`, and the `resolution`, as follows:

```
ledcSetup(ledChannel, freq, resolution);
```

Next, you need to choose the GPIO you'll get the signal from. For that use the `ledcAttachPin()` function that accepts as arguments the GPIO where you want to get the signal, and the channel that is generating the signal. In this example, we'll get the signal in the `ledPin` GPIO, that corresponds to GPIO 4. The channel that generates the signal is the `ledChannel`, that corresponds to channel 0.

```
ledcAttachPin(ledPin, ledChannel);
```

In the `loop()`, you vary the duty cycle between 0 and 255 to increase the LED brightness.

```
for(int dutyCycle = 0; dutyCycle <= 255; dutyCycle++) {
    // changing the LED brightness with PWM
    ledcWrite(ledChannel, dutyCycle);
    delay(15);
}
```

And then, between 255 and 0 to decrease the brightness.

```
for(int dutyCycle = 255; dutyCycle >= 0; dutyCycle--) {
    // changing the LED brightness with PWM
    ledcWrite(ledChannel, dutyCycle);
    delay(15);
}
```

To set the brightness of the LED, use the `ledcWrite()` function that accepts as arguments the channel that is generating the signal, and duty cycle.

```
ledcWrite(ledChannel, dutyCycle);
```

As we're using 8-bit resolution, the duty cycle will be controlled using a value from 0 to 255. Note that the `ledcWrite()` function accepts as argument the channel that is generating the signal, and not the GPIO.

Demonstration

After uploading the code to your ESP32-CAM, you should see the LED brightness increasing and decreasing.



Wrapping Up

In this section we've created some simple examples to demonstrate how to control an ESP32-CAM like a "regular" ESP32 development board. If you're already familiar with the ESP32, these examples are not new for you.

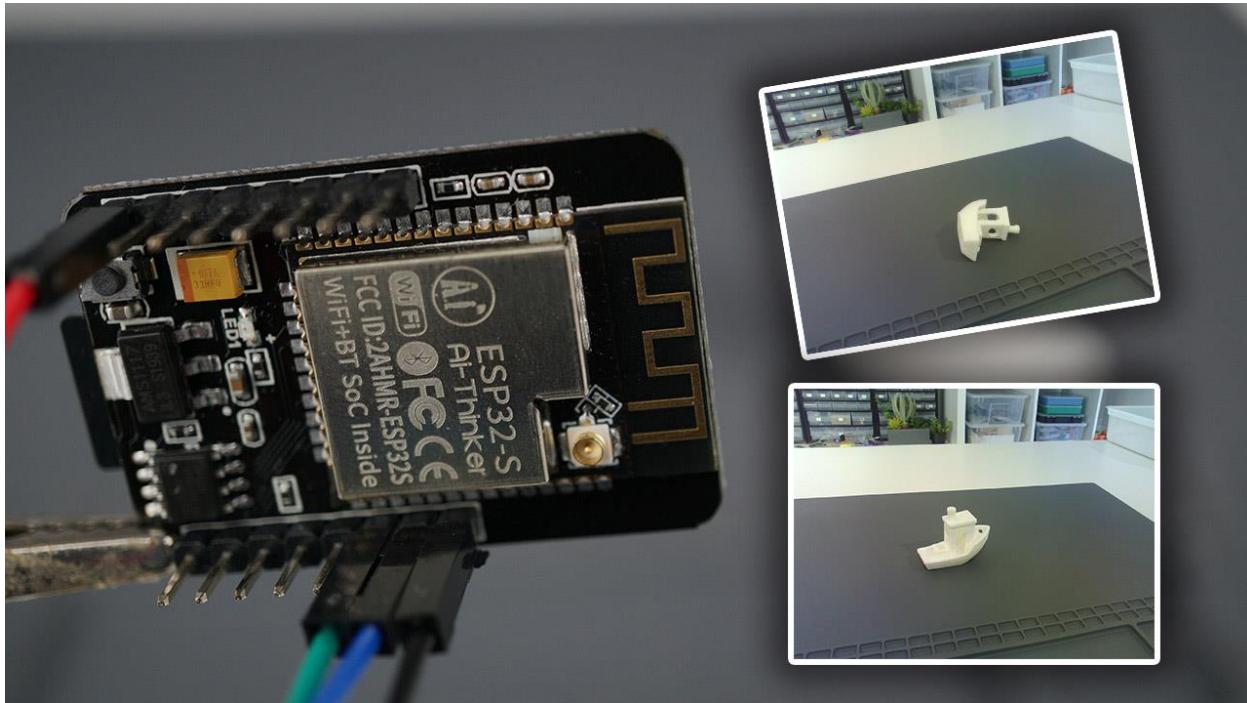
If you're a complete beginner to the ESP32, these were just quick getting started tutorials. To learn more about the ESP32 we have an in-depth course: [Learn ESP32 with Arduino IDE](#).

Now, you can go to the next Modules to start building interesting projects with the camera.

MODULE 2

**Take Photos: Time-lapse,
Camera Settings, Web
Server, SD Card Manager**

Unit 1: Take Photos and Save to MicroSD Card (Time-lapse)



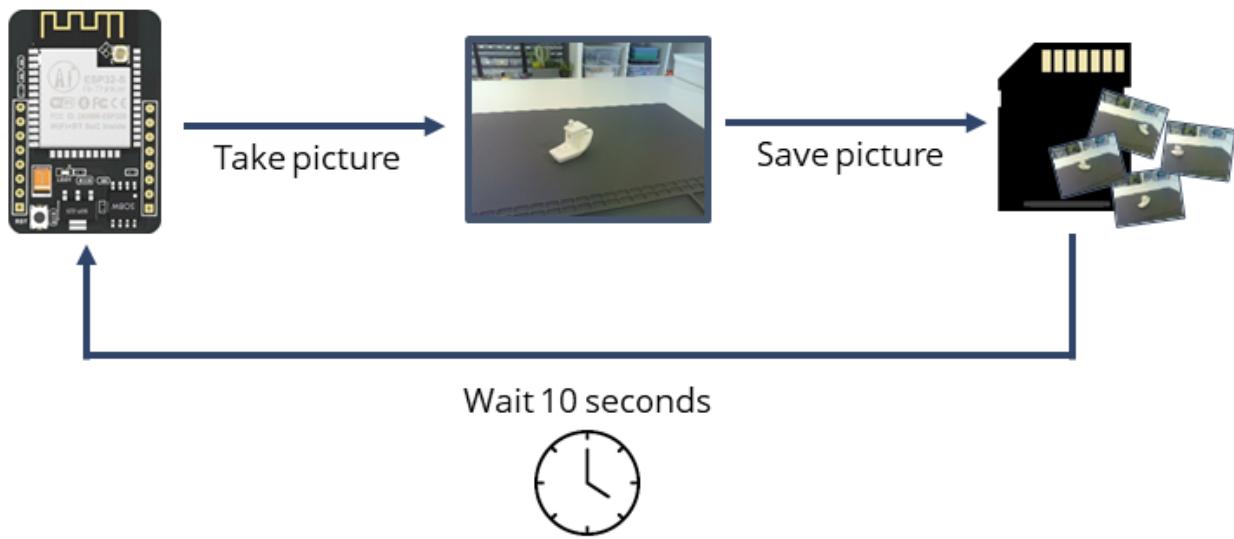
In this Unit you'll learn how to take photos with the ESP32-CAM and save them to the microSD card.

Compatibility: for this project you need a camera board with microSD card support:
ESP32-CAM AI-Thinker. This project is not compatible with any of the M5-Stack camera boards, ESP-EYE, or TTGO T-Journal because these don't have microSD card support.

To keep this project as simple as possible, we'll take a photo with the ESP32-CAM and save it in the microSD card every predetermined number of seconds in a loop. You'll capture a sequence of photos that will allow you to make a time-lapse video by combining all the images.

Project Overview

In this project, the ESP32-CAM takes a photo every 10 seconds and saves it on the microSD card. The photos are saved under the name pictureX.jpg, where X corresponds to the picture number.



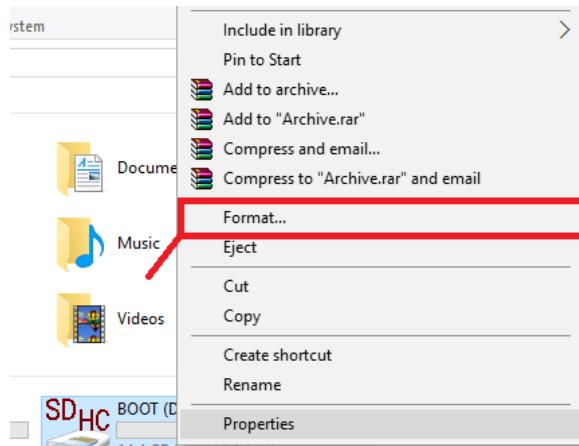
Here's a list of the steps to take into account when writing our code:

1. Configure and initialize the camera;
2. Initialize the microSD card;
3. Take a photo;
4. Save the photo in the microSD card;
5. Increment the picture number;
6. Repeat steps 3 to 6.

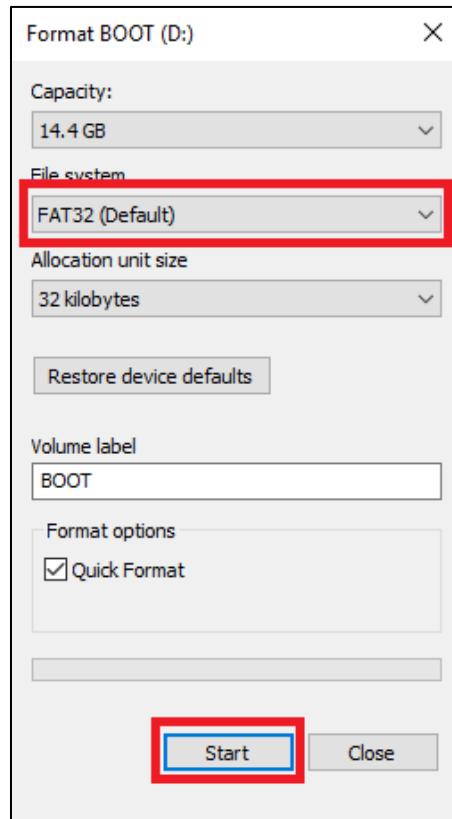
Formatting MicroSD Card

The first thing we recommend doing is formatting your microSD card. You can use the Windows formatter tool or any other microSD formatter software.

1. Insert the microSD card in your computer. Go to **My Computer** and right click in the SD card. Select **Format...** as shown in figure below.



2. A new window pops up. Select FAT32, press **Start** to initialize the formatting process and follow the onscreen instructions.



Note: accordingly, to the ESP32-CAM datasheet, it only supports 4 GB SD cards. However, we've tested with 16 GB SD card and it worked without any problems.

Code

Copy the following code to your Arduino IDE.

CODE

<https://github.com/RuiSantosdotme/ESP32-CAM-eBook/blob/master/Code/Module%202/Take%20Photos%20MicroSD%20Card%20Timelapse/Take%20Photos%20MicroSD%20Card%20Timelapse.ino>

```
#include "esp_camera.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"

// Pin definition for CAMERA_MODEL_AI_THINKER
// Change pin definition if you're using another ESP32 with camera module
#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

// Keep track of number of pictures
unsigned int pictureNumber = 0;

// Stores the camera configuration parameters
camera_config_t config;

void setup() {
    WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0); // disable brownout detector
    Serial.begin(115200);

    // Initialize the camera
    Serial.print("Initializing the camera module... ");
    configInitCamera();
    Serial.println("Ok!");
}
```

```

// Initialize MicroSD
Serial.print("Initializing the MicroSD card module... ");
initMicroSDCard();
}

void loop() {
    // Path where new picture will be saved in SD Card
    String path = "/picture" + String(pictureNumber) + ".jpg";
    Serial.printf("Picture file name: %s\n", path.c_str());

    takeSavePhoto(path);
    pictureNumber++;
    delay(10000);
}

void configInitCamera() {
    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; //YUV422,GRAYSCALE,RGB565,JPEG

    // Select lower framesize if the camera doesn't support PSRAM
    if(psramFound()) {
        config.frame_size=FRAMESIZE_UXGA;//FRAMESIZE_+QVGA|CIF|VGA|SVGA|XGA|SXGA|UXGA
        config.jpeg_quality = 1; //0-63 lower number means higher quality
        config.fb_count = 2;
    } else {
        config.frame_size = FRAMESIZE_SVGA;
        config.jpeg_quality = 12;
        config.fb_count = 1;
    }

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

```

```

        return;
    }
}

void initMicroSDCard() {
    // Start Micro SD card
    Serial.println("Starting SD Card");
    if(!SD_MMC.begin()){
        Serial.println("SD Card Mount Failed");
        return;
    }
    uint8_t cardType = SD_MMC.cardType();
    if(cardType == CARD_NONE){
        Serial.println("No SD Card attached");
        return;
    }
}

void takeSavePhoto(String path) {
    // Take Picture with Camera
    camera_fb_t * fb = esp_camera_fb_get();
    if(!fb) {
        Serial.println("Camera capture failed");
        return;
    }

    // Save picture to microSD card
    fs::FS &fs = SD_MMC;
    File file = fs.open(path.c_str(), FILE_WRITE);
    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());
    }
    file.close();

    // return the frame buffer back to the driver for reuse
    esp_camera_fb_return(fb);
}

```

How the Code Works

Let's take a quick look at the code to see how it works.

Start by importing the required libraries. Note that we need to include the `FS.h` and `SD_MMC.h` libraries to interface with the microSD card.

```
#include "esp_camera.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"
```

Then, include the pin definition for the camera model we're using. For the ESP32-CAM AI-Thinker module, this is the pin definition:

```
#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
```

Then, create a variable called `pictureNumber` that allows us to keep track of the number of pictures taken as well as to give a name to the picture.

```
unsigned int pictureNumber = 0;
```

The following line is needed to save the camera configurations. We'll add the pin definition and the image settings later in the code.

```
camera_config_t config;
```

In the `setup()`, initialize the camera module and the microSD card. To keep everything simpler, we've created a function to initialize the camera and another function to initialize the microSD card: `configInitCamera()` and `initMicroSDCard()`, respectively.

```
// Initialize the camera
Serial.print("Initializing the camera module...");
configInitCamera();
Serial.println("Ok!");
```

```
// Initialize MicroSD
Serial.print("Initializing the MicroSD card module... ");
initMicroSDCard();
```

Initialize the Camera

The `configInitCamera()` function, starts by assigning the GPIOs you've defined earlier as well as the picture format.

```
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; //YUV422, GRayscale, RGB565, JPEG
```

The picture format is defined in the following line

```
config.pixel_format = PIXFORMAT_JPEG; //YUV422, GRAYSCALE, RGB565, JPEG
```

It can be set to one the following formats:

- PIXFORMAT_JPEG
- PIXFORMAT_YUV422
- PIXFORMAT_GRAYSCALE
- PIXFORMAT_RGB565
- PIXFORMAT_JPEG (format that we're using)

Define the frame size:

```
if(psramFound()) {
```

```

    config.frame_size=FRAMESIZE_UXGA; //FRAMESIZE_+VGA|CIF|VGA|SVGA|XGA|SXGA|UXGA
    config.jpeg_quality = 1; //0-63 lower number means higher quality
    config.fb_count = 2;
} else {
    config.frame_size = FRAMESIZE_SVGA;
    config.jpeg_quality = 12;
    config.fb_count = 1;
}

```

In case the camera model supports PSRAM, we set the frame size to UXGA (1600x1200) and image quality to 1.

The frame size can be set to one of these options:

- FRAMESIZE_UXGA (1600 x 1200)
- FRAMESIZE_QVGA (320 x 240)
- FRAMESIZE_CIF (352 x 288)
- FRAMESIZE_VGA (640 x 480)
- FRAMESIZE_SVGA (800 x 600)
- FRAMESIZE_XGA (1024 x 768)
- FRAMESIZE_SXGA (1280 x 1024)

The image quality can be a number between 0 and 63. A lower number means a higher quality.

Important: very low numbers for image quality, specially at higher resolution can make the ESP32-CAM to crash or it may not be able take the photos properly. So, if you notice that the images taken with the ESP32-CAM are cut in half, or with strange colors, that's probably a sign that you need to lower the quality (select a higher number like 10).

The following lines initialize the camera with the configurations you've set up previously. If the camera doesn't initialize successfully, this will return an error message.

```
// Initialize the Camera
esp_err_t err = esp_camera_init(&config);
```

```
if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
}
```

You need to use the preceding snippet in all your sketches to initialize the camera.

Initialize the MicroSD Card

The `initMicroSDCard()` function initializes the microSD card. It returns an error message in case the initialization is not successful.

```
// Start Micro SD card
Serial.println("Starting SD Card");
if(!SD_MMC.begin()){
    Serial.println("SD Card Mount Failed");
    return;
}
```

The next lines return an error in case there isn't a microSD card attached.

```
uint8_t cardType = SD_MMC.cardType();
if(cardType == CARD_NONE) {
    Serial.println("No SD Card attached");
    return;
}
```

loop()

In the `loop()`, create a path to where the picture will be saved. In this case, the picture is saved on the root directory of the microSD card, so the path corresponds to the picture name.

```
void loop() {
    // Path where new picture will be saved in SD Card
    String path = "/picture" + String(pictureNumber) + ".jpg";
    Serial.printf("Picture file name: %s\n", path.c_str());
```

For this example, the path is `/pictureX.jpg` where **X** corresponds to the current picture number.

To take and save a photo, call the `takeSavePhoto()` function that accepts the picture path as an argument.

```
takeSavePhoto(path);  
pictureNumber++;  
delay(10000);
```

After taking and saving the photo, the picture number is incremented and the process is repeated every 10 seconds. You can change the delay time depending on desired application.

Take and Save a Photo

To take and save a picture, use the `takeSavePhoto()` function that accepts as argument the path where you want to save the photo.

```
void takeSavePhoto(String path) {
```

To take a picture, use the next snippet:

```
camera_fb_t * fb = esp_camera_fb_get();
```

This creates a frame buffer `fb` that contains the image. In case the camera is not able to get a picture, print an error message in the Serial Monitor:

```
if(!fb) {  
    Serial.println("Camera capture failed");  
    return;  
}
```

Save to MicroSD Card

Now that we have the picture (frame buffer `fb`), we can save it on the micro SD card.

Open the microSD card in writing mode:

```
fs::FS &fs = SD_MMC;  
File file = fs.open(path.c_str(), FILE_WRITE);
```

Save the image by passing as arguments the frame buffer and its length to the `write()` method.

```
file.write(fb->buf, fb->len); // payload (image), payload length
```

After that, close the file:

```
file.close();
```

Finally, call `esp_camera_fb_return(fb)` to clear the `fb` buffer, so that it's available to use for the next photo.

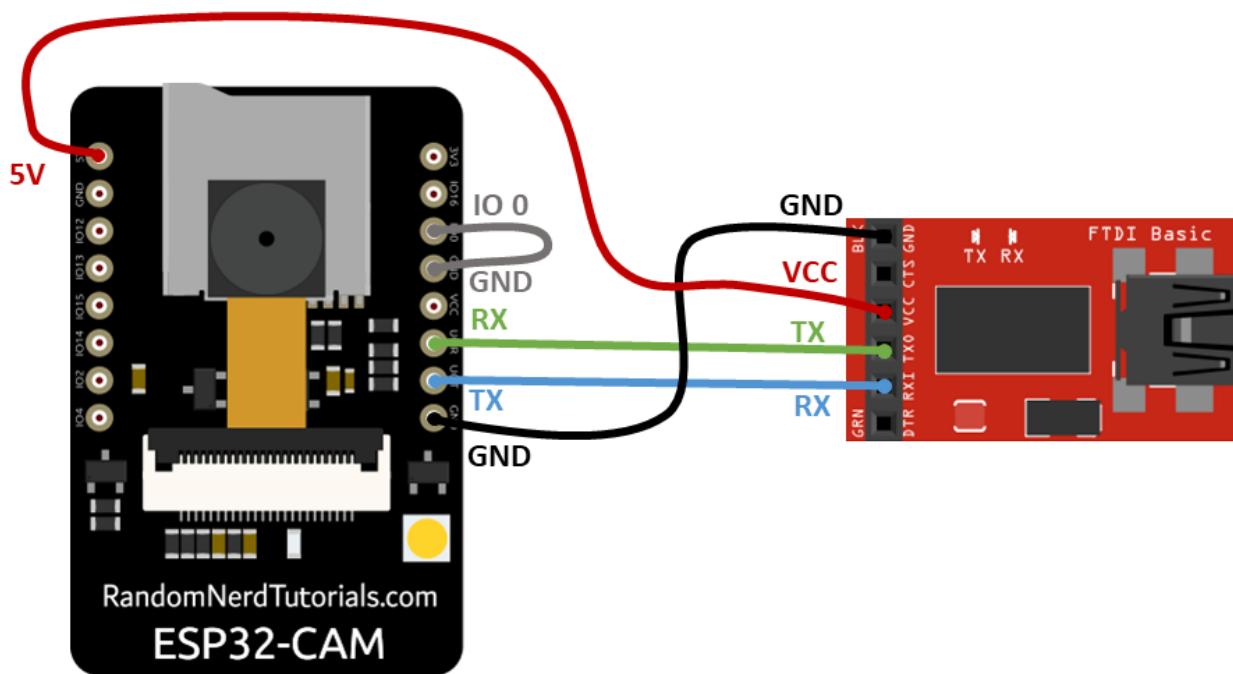
```
esp_camera_fb_return(fb);
```

That's pretty much how the code works.

Uploading the Code

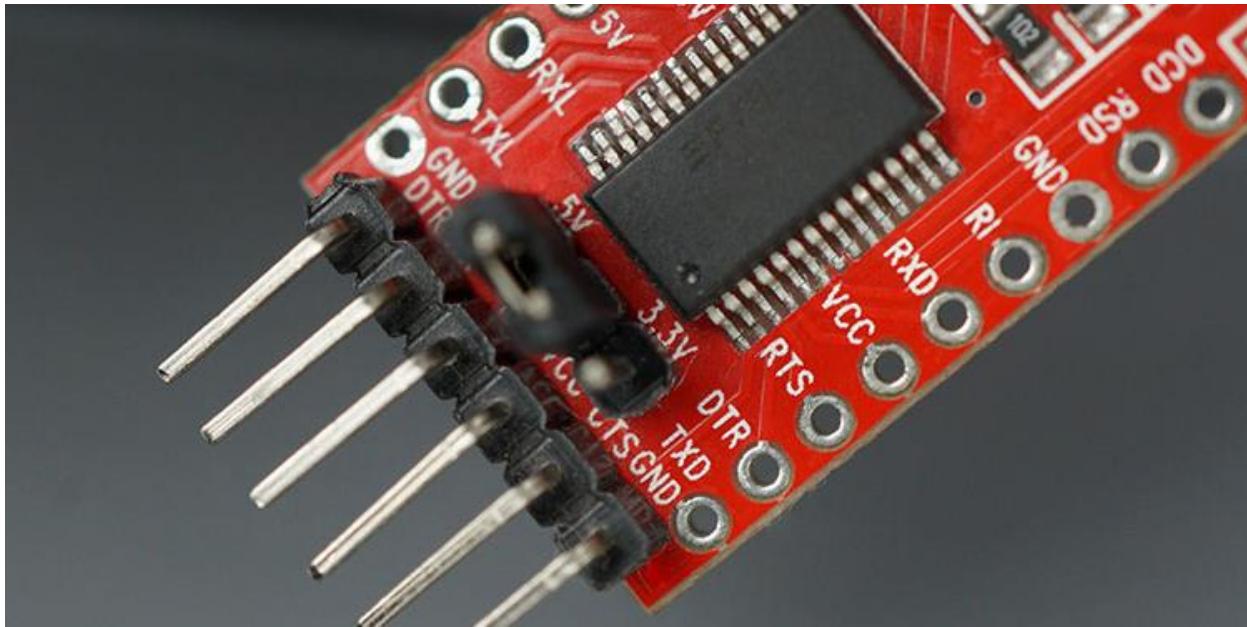
Upload the code to your ESP32-CAM board. Follow the next steps to upload code.

Connect your ESP32-CAM board to an FTDI programmer to upload code. You can follow the next schematic diagram.



Note: the order of the FTDI pins on the diagram may not match yours. Make sure you check the silkscreen label next to each pin.

Many FTDI programmers have a jumper that allows you to select 3.3V or 5V. Make sure the jumper is in the right position to select 5V.



Warning: some FTDI programmers only output 3.3V even when selecting the 5V option. So, check the output of the FTDI programmer VCC pin with a multimeter to make sure you're getting the right voltage. Many people are unable to upload code when powering the ESP32-CAM with 3.3V.

You can also use the following table as a reference:

| ESP32-CAM | FTDI Programmer |
|-----------|-----------------|
| GND | GND |
| 5V | VCC (5V) |
| U0R | TX |
| T0R | RX |
| GPIO 0 | GND |

Note that GPIO 0 needs to be connected to GND to upload code – you can use a female-to-female jumper wire to connect the pins. Having GPIO 0 connected to GND puts the ESP32 in flashing mode. This means the ESP32 is ready to receive new code.

Once you made all the necessary connections, in your Arduino IDE, make sure you have your ESP32-CAM board selected in **Tools > Board**. Then, go to **Tools > Port** and select the COM port the ESP32-CAM is connected to.

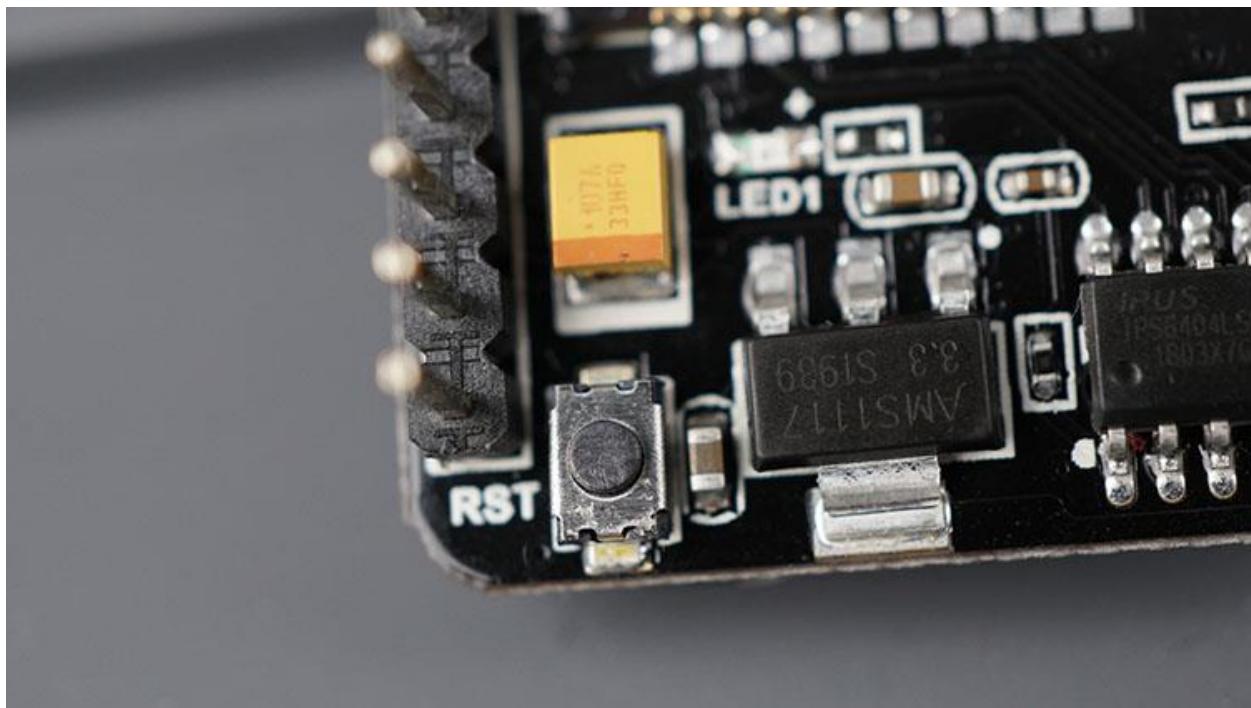
Then, click the upload button.



When you start to see these dots on the debugging window as shown below:

```
esptool.py v2.6-beta1
Serial port COM10
Connecting....._____....._____....._____
```

Press the ESP32-CAM on-board RST button. After a few seconds, the code should be successfully uploaded to your board.



Testing the Project

After uploading the code, disconnect GPIO 0 from GND. Then, make sure you have the microSD card inserted into the microSD card slot.

Press the on-board RST button so that the ESP32-CAM starts running the code. At the same time, you can open your Serial Monitor to see if everything is working as expected.



```
ho 0 tail 12 room 4
load:0x40080400,len:6352
entry 0x400806b8
Initializing the camera module...ok!
Initializing the MicroSD card module... Starting SD Card
Picture file name: /picture0.jpg
Saved file to path: /picture0.jpg
Picture file name: /picture1.jpg
Saved file to path: /picture1.jpg
Picture file name: /picture2.jpg
Saved file to path: /picture2.jpg
Picture file name: /picture3.jpg
Saved file to path: /picture3.jpg
```

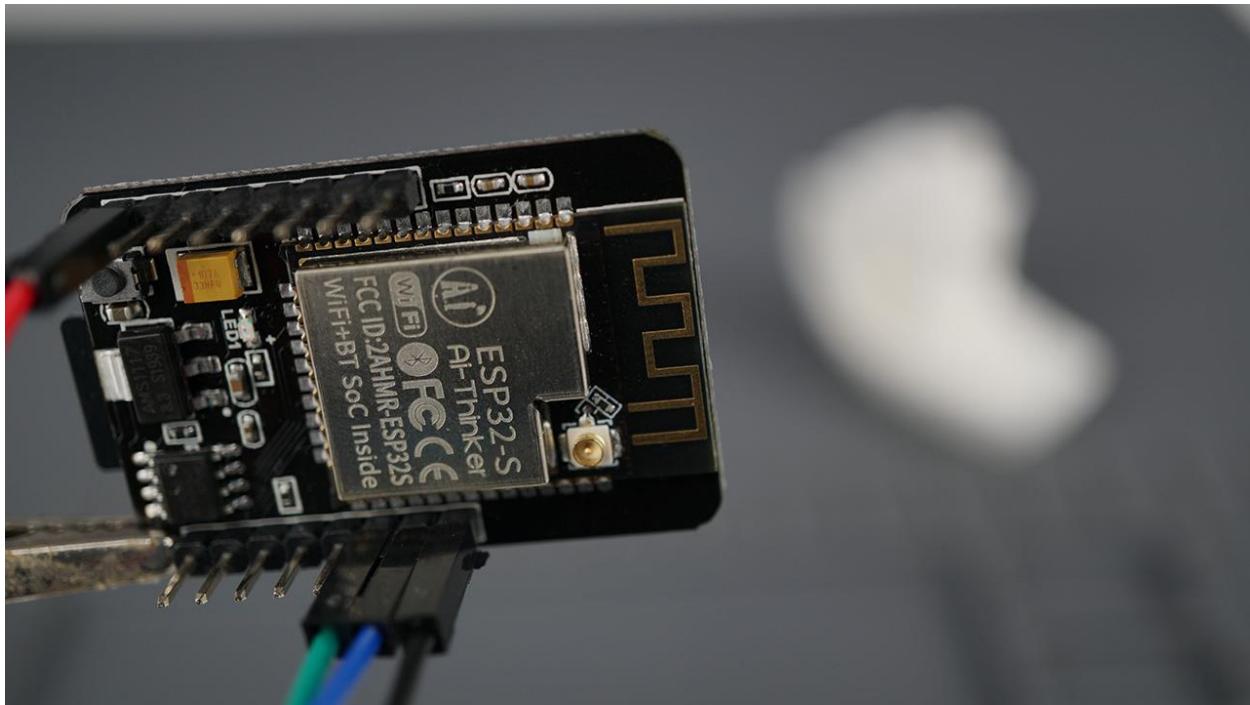
The screenshot shows a Windows-style serial monitor window titled "COM4". The main text area displays the output of the ESP32-CAM's boot and camera initialization process. It includes messages like "Initializing the camera module...ok!" and "Initializing the MicroSD card module... Starting SD Card". Below this, it shows the camera taking four pictures, naming them "picture0.jpg" through "picture3.jpg" and saving them to the root directory. At the bottom of the window, there are checkboxes for "Autoscroll" and "Show timestamp", and dropdown menus for "Both NL & CR" and "115200 baud". A "Clear output" button is also present.

As you can see in the preceding screenshot, the camera and the microSD card initialized OK. After that, it starts taking and saving pictures to the microSD card.

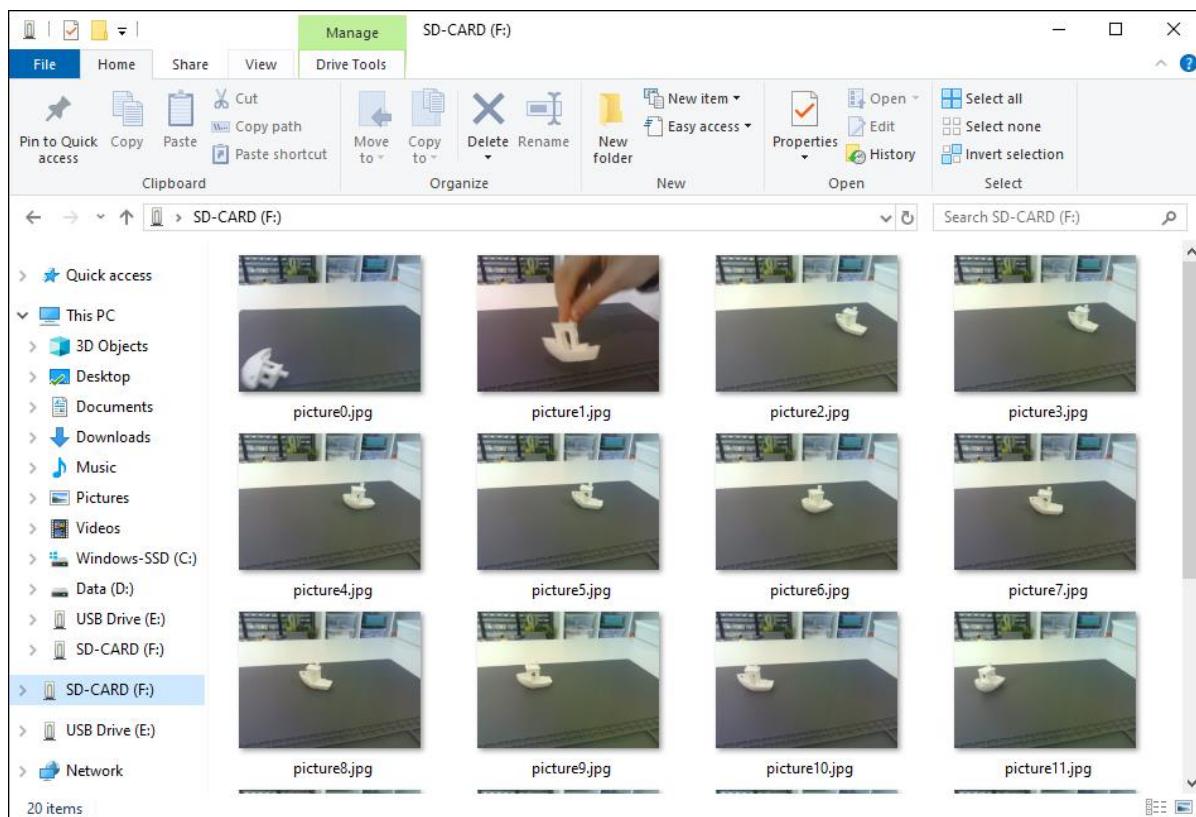
Every time it takes a new picture, it turns on the flashlight because it shares its GPIO with the microSD card. Additionally, you may also notice that the camera turns on the flash when the microSD card is initialized and that's normal.

As we've covered in a Previous Unit, the microSD card DATA3 pin is also the pin that controls the on-board LED – that's why that happens. Additionally, after initializing the microSD card, the LED remains on (in lower brightness) even when it isn't taking photos.

Let your camera run for a while so that it captures some photos.



Then, remove the microSD card and insert it into your computer. You should be able to access all the photos taken with the camera.



This sequence of photos can give a good stop motion video – basically, we've moved the boat a little between each photo. This example can also be applied to create a time-lapse (like sunset, sunrise, clouds moving, people walking on the street, etc.). You can also change the delay time, depending on your project requirements.

Wrapping Up

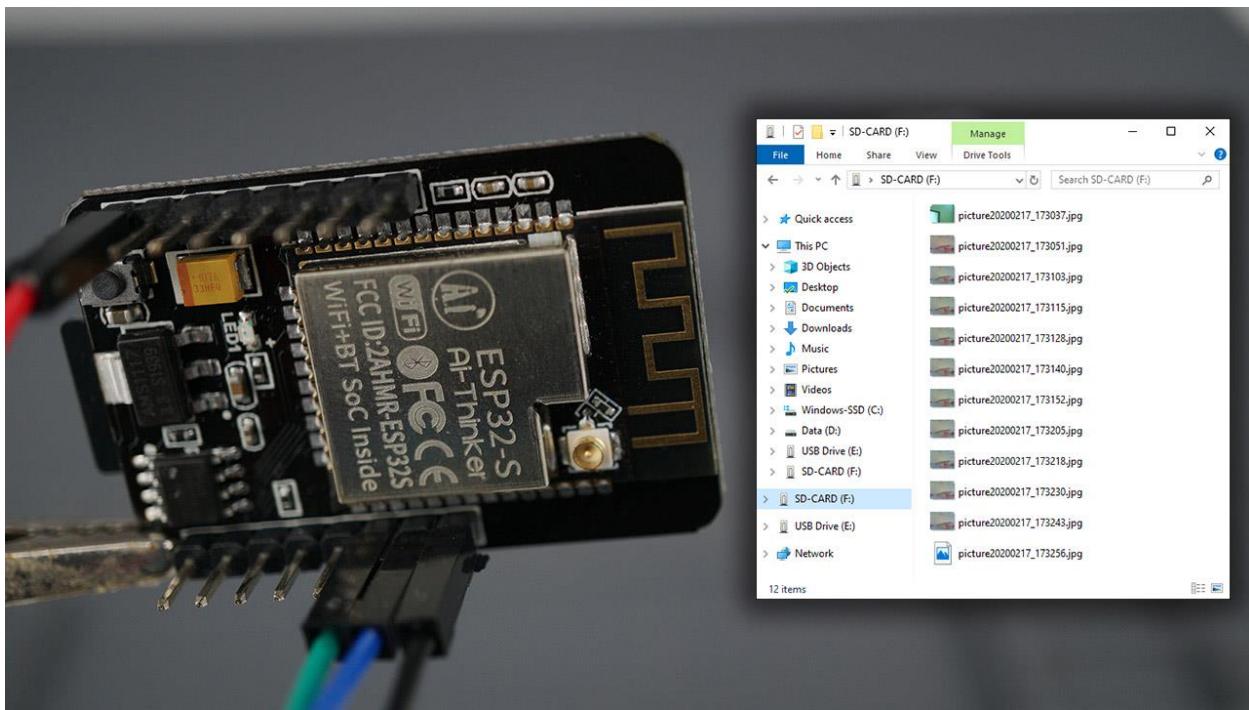
In this Unit, you've learned how to initialize the camera, how to initialize the microSD card, how to take a photo and how to save it in the microSD card.

You can use the functions we've created to initialize the camera and save photos in other project applications.

In this project, every time you reset the ESP32-CAM, it overwrites the previous photos. In the next Unit, you'll learn how to give each photo a unique filename based on date and time.

As always, we've tried to keep this project as simple as possible, so that it's easier to understand and modify.

Unit 2: Photo Filename with Date and Time Saved to MicroSD Card

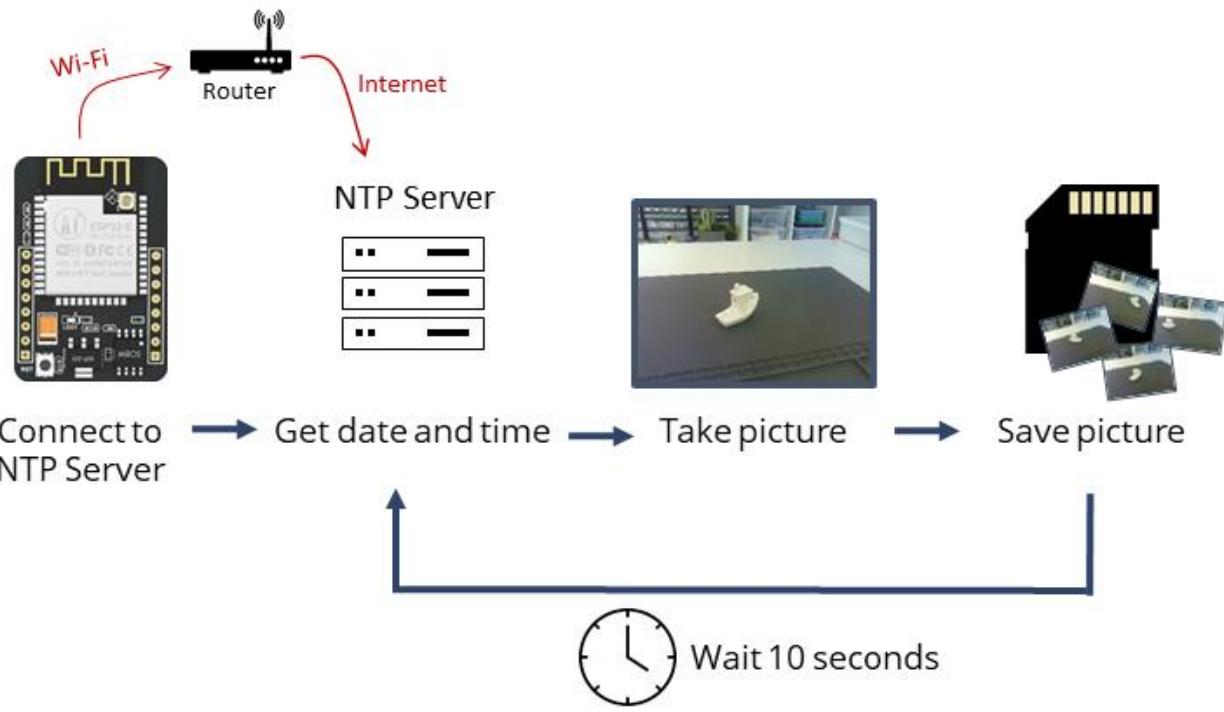


In this Unit, we'll modify the previous project to include date and time in the photo filename – every photo has a unique filename. We'll request time from an NTP (Network Time Protocol) Server, so you need to have access to the Internet.

Compatibility: for this project you need a camera board with microSD card support: **ESP32-CAM AI-Thinker**. This project is not compatible with any of the M5-Stack camera boards neither the ESP-EYE, or TTGO T-Journal because these don't have microSD card support.

Project Overview

The following figure shows a high-level overview on how the project works.



Here's a list with all the steps to describe this project.

1. ESP32-CAM connects to your router via Wi-Fi;
2. Connects to an NTP server to initialize date and time;
3. Initialize the camera and the microSD card;
4. Get current date and time;
5. Take a new photo;
6. Save the photo to the microSD card – its filename contains the date and time it was taken (it's a unique filename);
7. Repeat steps 4 to 6 every 10 seconds (you can change this interval).

Code

Copy the following code to your Arduino IDE.

CODE

https://github.com/RuiSantosdotme/ESP32-CAM-eBook/blob/master/Code/Module_2/Take_Photos_Date_Time/Take_Photos_Date_Time.ino

```
#include "esp_camera.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 <WiFi.h>
#include "time.h"
#include <WiFiUdp.h>

// REPLACE WITH YOUR NETWORK CREDENTIALS
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "RAPLCE_WITH_YOUR_PASSWORD";

// NTP Server
const char* ntpServer = "pool.ntp.org";
const long gmtOffset_sec = 0;
const int daylightOffset_sec = 3600;

// Pin definition for CAMERA_MODEL_AI_THINKER
// Change pin definition if you're using another ESP32 with camera module
#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

// Stores the camera configuration parameters
camera_config_t config;
```

```

void setup() {
    WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0); // disable brownout detector

    Serial.begin(115200);
    delay(2000);

    // connect to WiFi
    Serial.println("Initializing...");
    Serial.printf("Connecting to %s ", ssid);
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.printf("\nConnected to %s IP address: ", ssid);

    // Initialize and get the time
    configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
    Serial.print("Initializing Date and Time from NTP Server: ");
    Serial.println("");

    // disconnect WiFi as it's no longer needed
    /*WiFi.disconnect(true);
    WiFi.mode(WIFI_OFF);*/

    // Initialize the camera
    Serial.print("Initializing the camera module... ");
    configInitCamera();
    Serial.println("Ok!");

    // Initialize MicroSD
    Serial.print("Initializing the MicroSD card module... ");
    initMicroSDCard();
}

void loop() {
    // Get date and time
    struct tm timeinfo;
    char now[20];
    getLocalTime(&timeinfo);
    strftime(now, 20, "%Y%m%d_%H%M%S", &timeinfo);

    // Path where new picture will be saved in SD Card
    String path = "/picture" + String(now) + ".jpg";
    Serial.printf("Picture file name: %s\n", path.c_str());

    // Take and Save Photo
    takeSavePhoto(path);

    delay(10000);
}

void configInitCamera() {

```

```

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; //YUV422,GRAYSCALE,RGB565,JPEG

// Select lower framesize if the camera doesn't support PSRAM
if(psramFound()){
    config.frame_size=FRAMESIZE_UXGA;//FRAMESIZE_QVGA|CIF|VGA|SVGA|XGA|SXGA|UXGA
    config.jpeg_quality = 1; //0-63 lower number means higher quality
    config.fb_count = 2;
} else {
    config.frame_size = FRAMESIZE_SVGA;
    config.jpeg_quality = 12;
    config.fb_count = 1;
}

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

void initMicroSDCard(){
// Start Micro SD card
Serial.println("Starting SD Card");
if(!SD_MMC.begin()){
    Serial.println("SD Card Mount Failed");
    return;
}
uint8_t cardType = SD_MMC.cardType();
if(cardType == CARD_NONE){
    Serial.println("No SD Card attached");
    return;
}
}

```

```

void takeSavePhoto(String path) {
    // Take Picture with Camera
    camera_fb_t * fb = esp_camera_fb_get();
    if(!fb) {
        Serial.println("Camera capture failed");
        return;
    }
    // Save picture to microSD card
    fs::FS &fs = SD_MMC;
    File file = fs.open(path.c_str(), FILE_WRITE);
    if(!file) {
        Serial.printf("Failed to open file in writing mode");
    }
    else {
        file.write(fb->buf, fb->len); // payload (image), payload length
        Serial.printf(" Saved: %s\n", path.c_str());
    }
    file.close();
    esp_camera_fb_return(fb);
}

```

How the Code Works

This code is very similar to the previous project, but adds the necessary lines to get date and time.

You need to include the following libraries to connect to an NTP server and handle the time.

```
#include <WiFi.h>
#include "time.h"
#include <WiFiUdp.h>
```

Insert your network credentials so that the ESP32-CAM is able to connect to your router.

```
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
```

Then, you need to define the following variables to configure and get time from an NTP server.

```
const char* ntpServer = "pool.ntp.org";
const long gmtOffset_sec = 0;
const int daylightOffset_sec = 3600;
```

We'll request the time from pool.ntp.org, which is a cluster of timeservers that anyone can use to request the time.

The `gmtOffset_sec` variable defines the offset in seconds between your time zone and GMT. We live in Portugal, so the time offset is 0. Change the time `gmtOffset_sec` variable to match your time zone.

```
const long gmtOffset_sec = 0;
```

The `daylightOffset_sec` variable defines the offset in seconds for daylight saving time. It is generally one hour, that corresponds to 3600 seconds.

```
const int daylightOffset_sec = 3600;
```

In the `setup()`, connect to your router via Wi-Fi:

```
Serial.println("Initializing...");  
Serial.printf("Connecting to %s ", ssid);  
WiFi.begin(ssid, password);  
while (WiFi.status() != WL_CONNECTED) {  
    delay(500);  
    Serial.print(".");
```

Configure the time with the settings you've defined earlier:

```
configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
```

As with the previous example, you configure and initialize the camera as well as the microSD card.

```
// Initialize the camera  
Serial.print("Initializing the camera module...");  
configInitCamera();  
Serial.println("Ok!");  
  
// Initialize MicroSD  
Serial.print("Initializing the MicroSD card module... ");  
initMicroSDCard();
```

In the `loop()`, get the current date and time before taking a new photo. The date and time are saved in the `now` char variable.

```
// Get date and time  
struct tm timeinfo;
```

```
char now[20];
getLocalTime(&timeinfo);
strftime(now,20,"%Y%m%d %H%M%S",&timeinfo);
```

Create a path for the picture (in this case corresponds to the picture name) with the date and time:

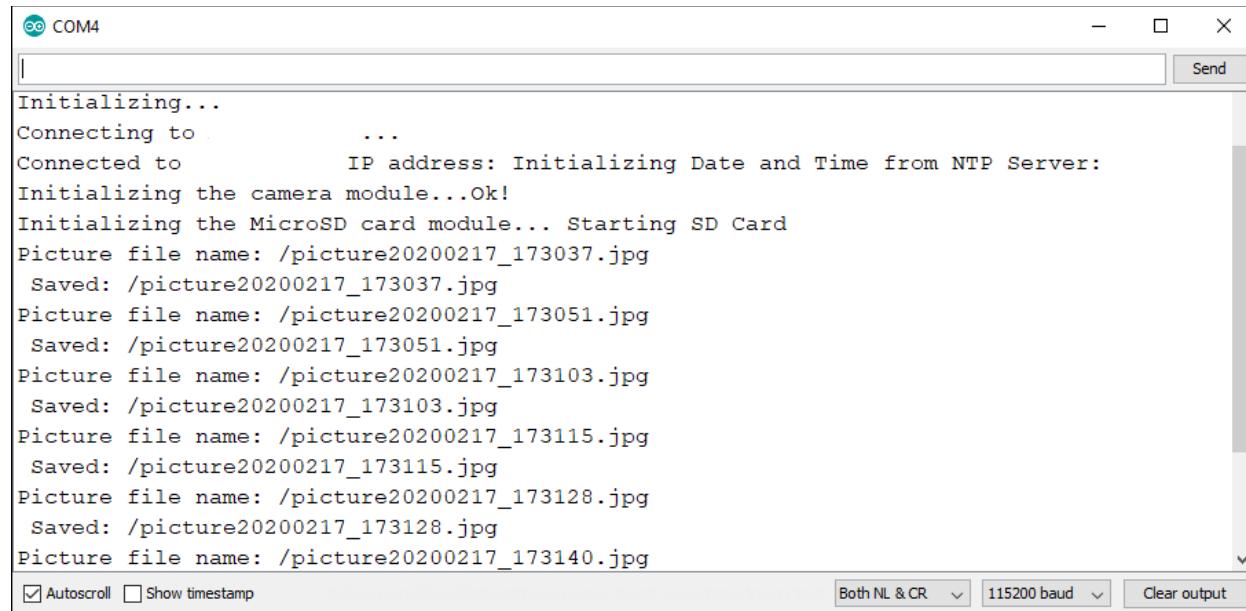
```
String path = "/picture" + String(now) + ".jpg";
Serial.printf("Picture file name: %s\n", path.c_str());
```

Finally, take and save the photo to that path:

```
takeSavePhoto(path);
```

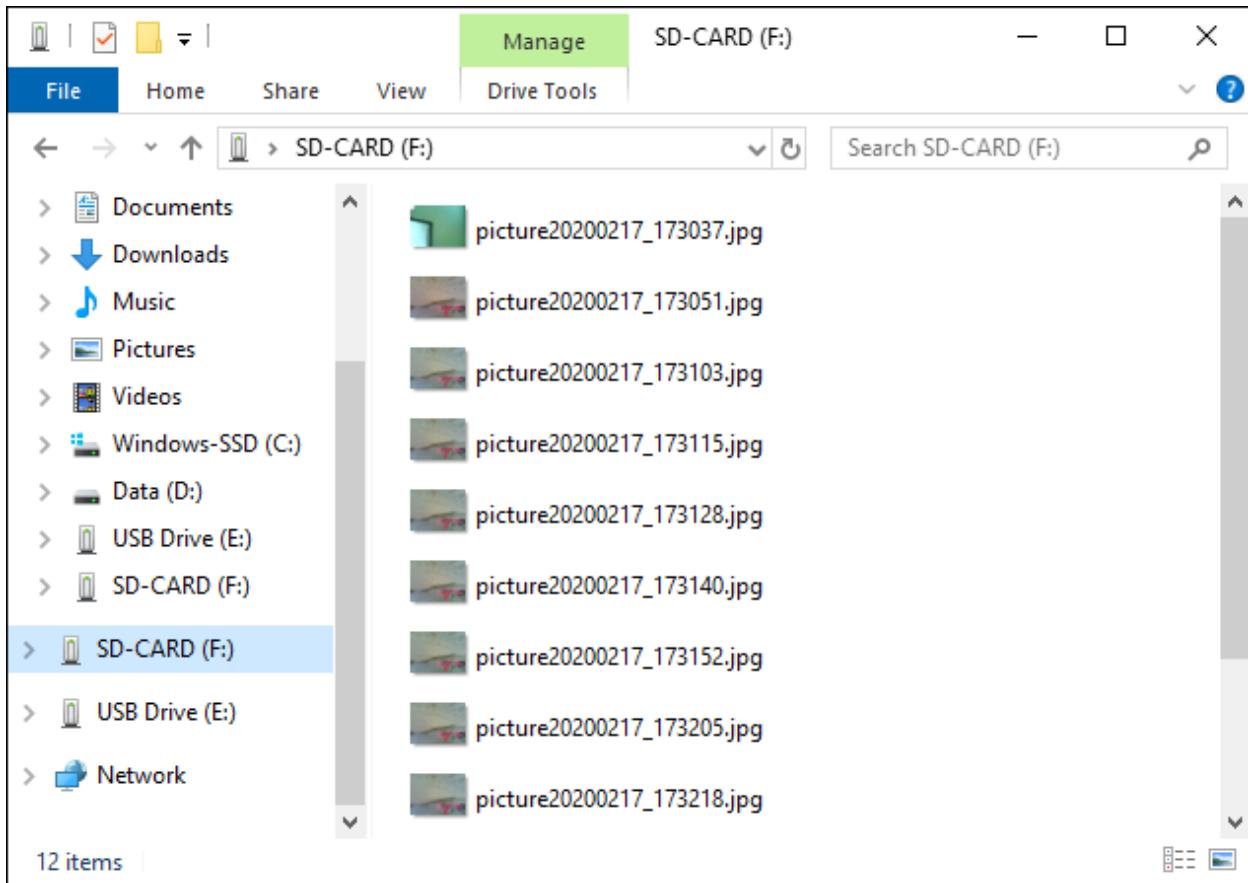
Demonstration

Upload the code to your ESP32-CAM. Let the code run for a while, so that it captures some pictures. You can open the Serial Monitor at a baud rate of 115200 to see if everything is working as expected.



```
Initializing...
Connecting to ...
Connected to IP address: Initializing Date and Time from NTP Server.
Initializing the camera module...Ok!
Initializing the MicroSD card module... Starting SD Card
Picture file name: /picture20200217_173037.jpg
Saved: /picture20200217_173037.jpg
Picture file name: /picture20200217_173051.jpg
Saved: /picture20200217_173051.jpg
Picture file name: /picture20200217_173103.jpg
Saved: /picture20200217_173103.jpg
Picture file name: /picture20200217_173115.jpg
Saved: /picture20200217_173115.jpg
Picture file name: /picture20200217_173128.jpg
Saved: /picture20200217_173128.jpg
Picture file name: /picture20200217_173140.jpg
```

Then, insert the microSD card into your computer to see the saved photos.



The photos should have the date and time on its name. For example:

picture20200217_173140.jpg

This means that this picture was taken on:

- Year: 2020
- Month: 02 (February)
- Day: 17th
- Time: 17:31:40 (5:31:40 PM)

Wrapping Up

In this project you learned how to request date and time from the internet using the ESP32 and use that information to name your pictures. This way you know what time

and date a certain photo was taken and you don't have to worry overwriting photos on the microSD card because all have a different name.

For a simple example sketch with the ESP32 to get the time (only), check the example on the following link:

- [ESP32 SimpleTime Example Sketch](#)

Unit 3: Change Camera Settings



- Contrast
- Brightness
- Resolution
- Quality
- Saturation
- Special Effects
- White Balance
- Mirror and Flip
- Exposure
- Etc...

In this Unit, you'll learn how to change the camera settings such as contrast, brightness, resolution, saturation, and more.

Compatibility: the concepts you'll learn in this Unit can be applied to any ESP32 camera model.

In [Module 1, Unit 3: Video Streaming Web Server and Face Recognition](#), the web server provided a lot of options to change the image settings. We recommend that you follow that project and play with the image settings to see what each setting does.

Depending on where your camera is located, you may want to change some settings to get a better picture. Playing with that web server gives you an idea of what you need to change and what values you need to set to get a better picture. Once you know the best settings for your camera, you may want to apply them in your other projects.

In this Unit, we'll show you the code to change the camera settings. As an example, we'll use the project from [Module 2 – Unit 1](#).

Code

The following code takes a photo every 10 seconds and saves it to the microSD card.

CODE

https://github.com/RuiSantosdotme/ESP32-CAM-eBook/blob/master/Code/Module_2/Take_Photo_Change_Camera_Settings/Take_Photo_Change_Camera_Settings.ino

There's a section in the code that allows you to change the camera settings inside the `configInitCamera()` function.

Basically, after initializing the camera you can change the settings using the following lines.

```
sensor_t * s = esp_camera_sensor_get()

s->set_brightness(s, 0);      // -2 to 2
s->set_contrast(s, 0);        // -2 to 2
s->set_saturation(s, 0);      // -2 to 2
s->set_special_effect(s, 0);  // 0 to 6 (0 - No Effect, 1 - Negative, 2 - Grayscale, 3 - Red Tint, 4 - Green Tint, 5 - Blue Tint, 6 - Sepia)
s->set_whitebal(s, 1);        // 0 = disable , 1 = enable
s->set_awb_gain(s, 1);        // 0 = disable , 1 = enable
s->set_wb_mode(s, 0);         // 0 to 4 - if awb_gain enabled (0 - Auto, 1 - Sunny, 2 - Cloudy, 3 - Office, 4 - Home)
s->set_exposure_ctrl(s, 1);   // 0 = disable , 1 = enable
s->set_aec2(s, 0);            // 0 = disable , 1 = enable
s->set_ae_level(s, 0);        // -2 to 2
s->set_aec_value(s, 300);     // 0 to 1200
s->set_gain_ctrl(s, 1);       // 0 = disable , 1 = enable
s->set_agc_gain(s, 0);        // 0 to 30
s->set_gainceiling(s, (gainceiling_t)0); // 0 to 6
s->set_bpc(s, 0);             // 0 = disable , 1 = enable
s->set_wpc(s, 1);             // 0 = disable , 1 = enable
s->set_raw_gma(s, 1);          // 0 = disable , 1 = enable
s->set_lenc(s, 1);             // 0 = disable , 1 = enable
s->set_hmirror(s, 0);          // 0 = disable , 1 = enable
s->set_vflip(s, 0);            // 0 = disable , 1 = enable
s->set_dcw(s, 1);              // 0 = disable , 1 = enable
s->set_colorbar(s, 0);          // 0 = disable , 1 = enable
```

The following table shows what each function means and the values accepted:

| Function | Meaning | Values |
|----------------------|------------------------|--|
| set_brightness() | Set brightness | -2 to 2 |
| set_contrast() | Set contrast | -2 to 2 |
| set_saturation() | Set saturation | -2 to 2 |
| set_special_effect() | Set a special effect | 0 – No effect 1 – Negative 2 – Grayscale 3 – Red Tint 4 – Green Tint 5 – Blue Tint 6 – Sepia |
| set_whitebal() | Set white balance | 0 – disable 1 – enable |
| set_awb_gain() | Set white balance gain | 0 – disable 1 – enable |
| set_wb_mode() | Set white balance mode | 0 – Auto 1 – Sunny 2 – Cloudy 3 – Office 4 – Home |
| set_exposure_ctrl() | | 0 – disable 1 – enable |

| | | |
|-------------------|---------------------|---------------------------|
| set_aec2() | | 0 – disable 1 – enable |
| set_ae_level() | | -2 to 2 |
| set_aec_value() | | 0 to 1200 |
| set_gain_ctrl() | | 0 – disable 1 – enable |
| set_agc_gain() | | 0 to 30 |
| set_gainceiling() | | 0 to 6 |
| set_bpc() | | 0 – disable 1 – enable |
| set_wpc() | | 0 – disable 1 – enable |
| set_raw_gma() | | 0 – disable 1 – enable |
| set_lenc() | Set lens correction | 0 – disable 1 – enable |
| set_hmirror() | Horizontal mirror | 0 – disable 1 – enable |
| set_vflip() | Vertical flip | 0 – disable 1 – enable |
| set_dcw() | | 0 – disable 1 – enable |

| | | |
|----------------|----------------|---------------------------|
| set_colorbar() | Set a colorbar | 0 – disable 1 – enable |
|----------------|----------------|---------------------------|

As you can see, changing the camera settings is pretty straightforward. You just need to use those lines of code after initializing the camera. After that, you can use the usual functions and code to control the camera.

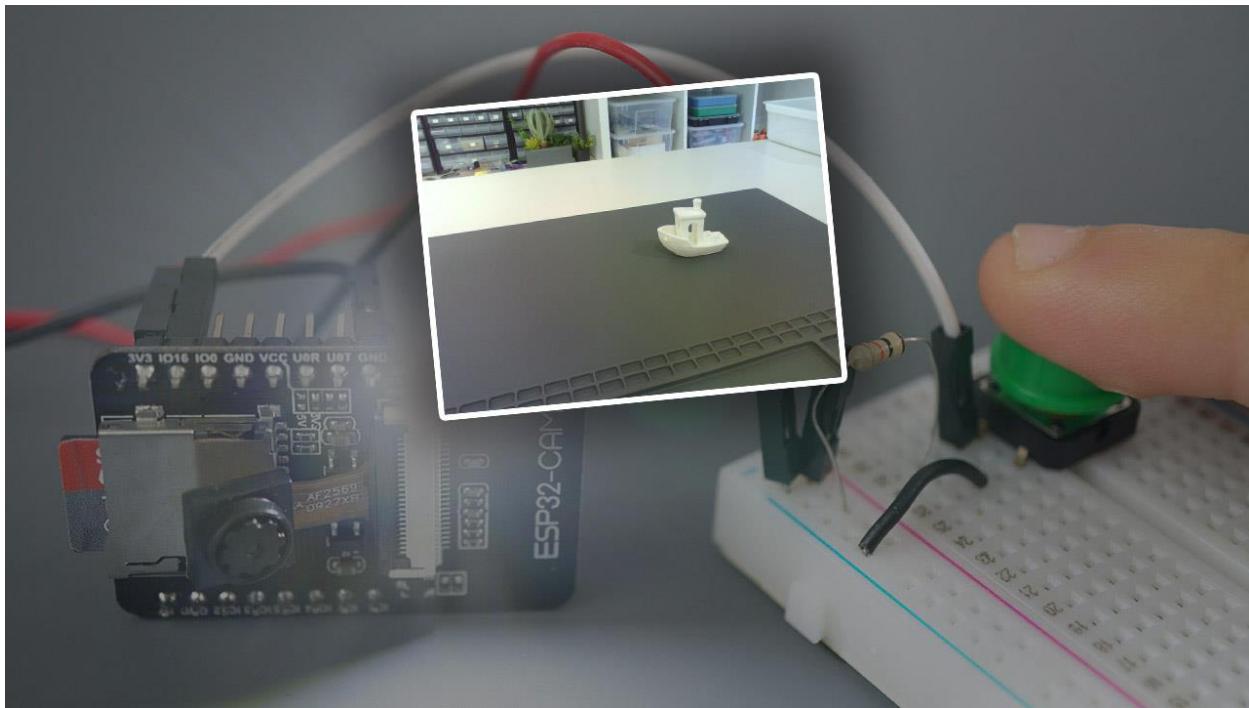
The functions in the table appear in the same order as in the [CameraWebServer example in Module 1, Unit 3](#), so that it is easier to identify which functions and values you should use to get a better image in your scenario.

Wrapping Up

In this Unit, you've learned how to change the camera settings to adjust the image you get with the camera. This can be useful because depending on where you place your camera you may need to change the settings to get a better image.

You can use the functions we've shown you here in any of your projects with the ESP32-CAM to adjust the settings.

Unit 4: Take Photo and Save to MicroSD Card with Pushbutton



In this example, we'll connect a pushbutton to the ESP32-CAM that when pressed takes a photo and saves it in the microSD card. The photos are saved under the name pictureX.jpg, where X corresponds to the picture number.

Compatibility: for this project you need a camera board with microSD card support: **ESP32-CAM AI-Thinker**. This project is not compatible with any of the M5-Stack camera boards neither the ESP-EYE, or TTGO T-Journal because these don't have microSD card support.

Project Overview

The following image shows a high-level overview of the project we'll build.



Press pushbutton → Take picture → Save picture

Here's a list of the steps that we need to include in the code:

- Configure and initialize the camera;
- Initialize the microSD card;
- Check the pushbutton state: when pressed takes a photo;
- Take a photo;
- Save the photo in the microSD card;
- Increment the picture number.

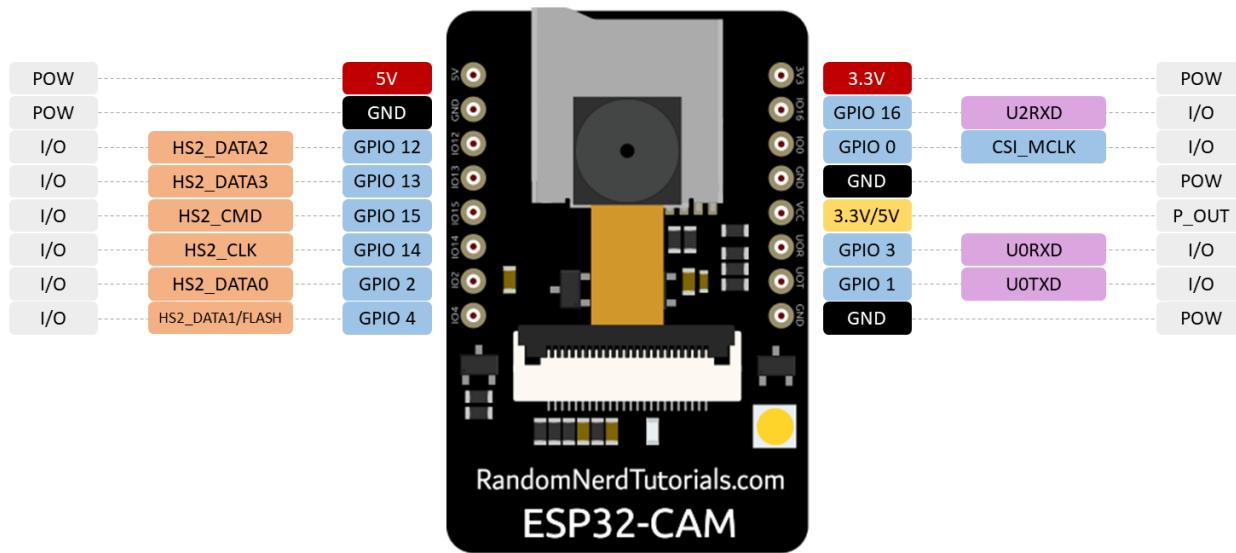
Every time you reset the ESP32, it will overwrite the first pictures because the picture number is reset to 0. To prevent this, you can add date and time to the photo filename as shown in [Module 2, Unit 2](#). However, to keep things as simple as possible, we just numbered the photos.

Connect a Pushbutton to the ESP32-CAM

Connecting a pushbutton to the ESP32-CAM is not as straightforward as it may seem.

But it is not difficult either. You just need to choose wisely the pins to wire your button.

Let's quickly recap the ESP32-CAM pinout.



GPIO 4, **GPIO 2**, **GPIO 14**, **GPIO 15**, **GPIO 13** and **GPIO 12** are all being used by the microSD card. So, if we want to use the microSD card in a project, using these pins to connect other peripherals is not a good idea.

GPIO 0 is used to put the ESP32 in flashing mode, so you should not use it to connect a pushbutton.

GPIO 1 and **GPIO 3** are the UART pins. So, if they are not being used to establish a serial communication, these should be OK to use after uploading the code. However, you'll not be able to establish a serial communication with the Serial Monitor to debug your project.

GPIO 16 is also a UART pin. If you're not using it for serial communication (which you usually don't), you can use that pin to connect peripherals without any problems.

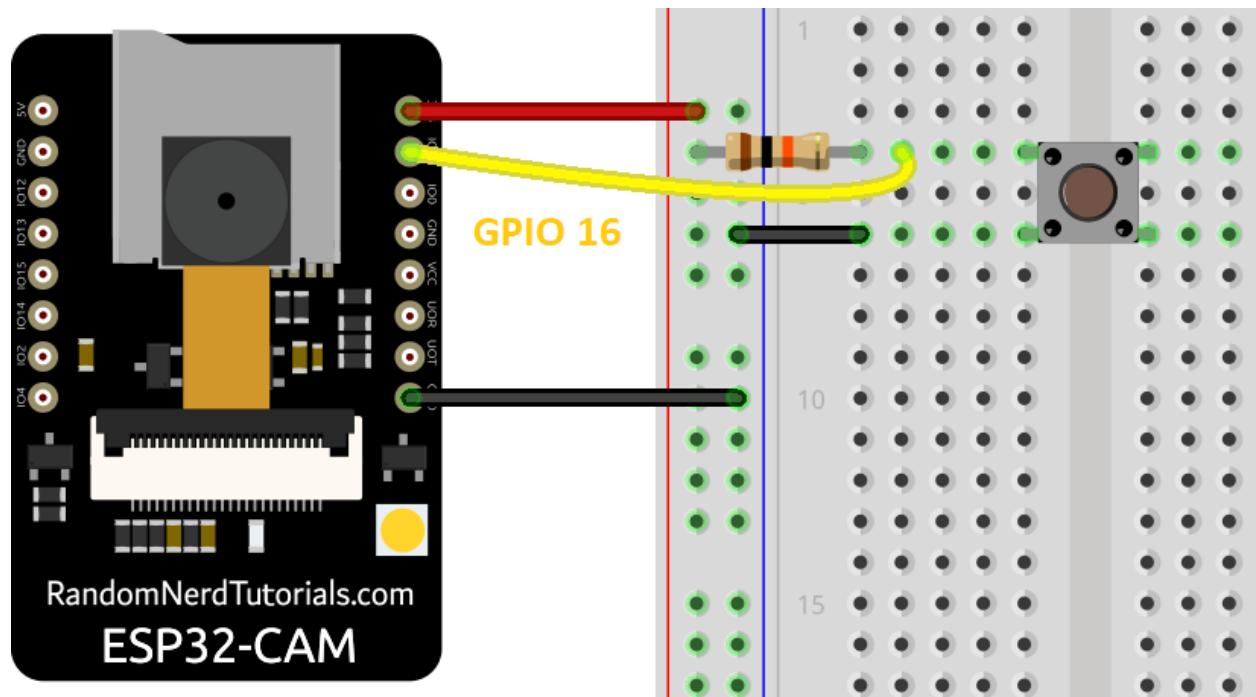
Note: the best pin to use to connect a pushbutton without crashing the microSD card or the camera is **GPIO 16**.

Schematic Diagram

Parts required for this circuit:

- [ESP32-CAM](#)
- [Pushbutton](#)
- [10k Ohm resistor](#)
- [Jumper wires](#)
- [Breadboard](#)

Wire a pushbutton to the ESP32-CAM GPIO 16 as shown in the following schematic diagram.



Code

The following code takes a photo and saves it to the microSD card when you press the pushbutton connected to GPIO 16.

CODE

https://github.com/RuiSantosdotme/ESP32-CAM-eBook/blob/master/Code/Module_2/Take_Photo_Pushbutton/Take_Photo_Pushbutton.ino

```
#include "esp_camera.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"

// Pin definition for CAMERA_MODEL_AI_THINKER
// Change pin definition if you're using another ESP32 with camera module
#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

// Keep track of number of pictures
int pictureNumber = 0;

// Stores the camera configuration parameters
camera_config_t config;

// Button pin
const int buttonPin = 16;
int buttonState = HIGH;
int lastButtonState = LOW; // the previous reading from the input pin

unsigned long lastDebounceTime = 0; // last time output pin was toggled
```

```

unsigned long debounceDelay = 50; // the debounce time; increase if the
output flickers

void setup() {
    pinMode(buttonPin, INPUT);

    WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0); // disable brownout detector

    Serial.begin(115200);

    // Initialize the camera
    Serial.print("Initializing the camera module...");
    configInitCamera();
    Serial.println("Ok!");
    // Initialize MicroSD
    Serial.print("Initializing the MicroSD card module... ");
    initMicroSDCard();
}

void loop() {
    int reading = digitalRead(buttonPin);

    // If the switch changed, due to noise or pressing:
    if (reading != lastButtonState) {
        // reset the debouncing timer
        lastDebounceTime = millis();
    }

    if ((millis() - lastDebounceTime) > debounceDelay) {
        // if the button state has changed
        if (reading != buttonState) {
            buttonState = reading;
            if (buttonState==LOW){
                Serial.println("Button Pressed");
                // Path where new picture will be saved in SD Card
                String path = "/picture" + String(pictureNumber) + ".jpg";
                Serial.printf("Picture file name: %s\n", path.c_str());
                takeSavePhoto(path);
                pictureNumber++;
            }
        }
    }
    lastButtonState = reading;
}

void configInitCamera() {
    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; //YUV422,GRAYSCALE,RGB565,JPEG

// Select lower framesize if the camera doesn't support PSRAM
if(psramFound()){
    config.frame_size=FRAMESIZE_UXGA;//FRAMESIZE_+QVGA|CIF|VGA|SVGA|XGA|SXGA|UXGA
    config.jpeg_quality = 1; //0-63 lower number means higher quality
    config.fb_count = 2;
} else {
    config.frame_size = FRAMESIZE_SVGA;
    config.jpeg_quality = 12;
    config.fb_count = 1;
}

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

void initMicroSDCard(){
// Start Micro SD card
Serial.println("Starting SD Card");
if(!SD_MMC.begin()){
    Serial.println("SD Card Mount Failed");
    return;
}
uint8_t cardType = SD_MMC.cardType();
if(cardType == CARD_NONE){
    Serial.println("No SD Card attached");
    return;
}
}

void takeSavePhoto(String path){
// Take Picture with Camera
camera_fb_t * fb = esp_camera_fb_get();
if(!fb) {
    Serial.println("Camera capture failed");
    return;
}
}

```

```

// Save picture to microSD card
fs::FS &fs = SD_MMC;
File file = fs.open(path.c_str(), FILE_WRITE);
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());
}
file.close();

delay(50);
//return the frame buffer back to the driver for reuse
esp_camera_fb_return(fb);
}

```

How the Code Works

We need to use a pushbutton debounce sketch to prevent the ESP32-CAM from crashing in case of “false” positive button presses.

The debounce sketch checks twice in a short period of time the state of the button to make sure it is actually pressed. Without debouncing, pressing the button once may cause the ESP32-CAM trying to take multiple photos in a short period of time. This will eventually crash the ESP32 and you’ll have to manually restart the board.

For debouncing, we need to use the `millis()` function to keep track of the time that has passed since the button was pressed.

This code is very similar to the previous projects presented this Module. It adds the code to debounce the pushbutton state. The pushbutton is wired with a pull-up resistor. This means that:

- Button not pressed → GPIO 16 state: HIGH
- Button pressed → GPIO 16 state: LOW

Taking and saving photos was already addressed previously, so we’ll just take a look at the relevant parts for this project.

Defining Variables

Create a variable to hold the GPIO pin. In this case, it's connected to GPIO 16.

```
const int buttonPin = 16;
```

The `buttonState` variable holds the current pushbutton state. Because it is not being pressed, its state is HIGH.

```
int buttonState = HIGH;
```

The `lastButtonState` variable holds the previous state of the pushbutton. At start, let's assume the previous state is LOW.

```
int lastButtonState = LOW;
```

The `lastDebounceTime` variable saves the last time the ESP32 took a photo.

```
unsigned long lastDebounceTime = 0;
```

The `debounceDelay` corresponds to the debounce time – period of time to check if the pushbutton has actually changed its state.

```
unsigned long debounceDelay = 50;
```

setup()

In the `setup()`, set the pushbutton as an input.

```
pinMode(buttonPin, INPUT);
```

loop()

In the `loop()`, start by getting the button state and save it in the `reading` variable.

```
int reading = digitalRead(buttonPin);
```

If the current reading is different from the last button state, it means that the button state changed due to noise or pressing. So, reset the debounce timer.

```
if (reading != lastButtonState) {  
    // reset the debouncing timer
```

```
    lastDebounceTime = millis();
}
```

Then, if the debounce timer has passed and the pushbutton has changed its state, check if it was pressed (when it is pressed the state is `LOW`).

```
if ((millis() - lastDebounceTime) > debounceDelay) {
    // if the button state has changed:
    if (reading != buttonState) {
        buttonState = reading;
        if (buttonState==LOW) {
```

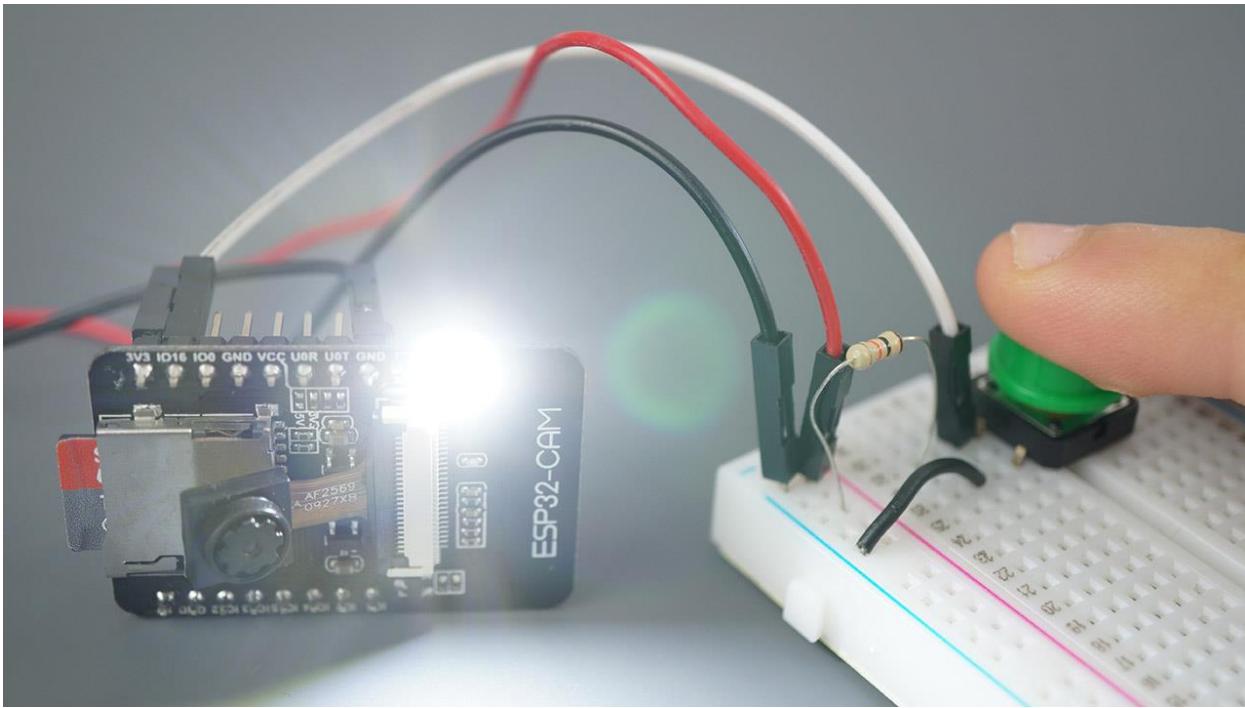
If the button was pressed, create the path to save the picture, take a photo and save it in the microSD card. Update the `pictureNumber` and the `lastButtonState` variables.

```
if ((millis() - lastDebounceTime) > debounceDelay) {
    // if the button state has changed:
    if (reading != buttonState) {
        buttonState = reading;
        if (buttonState==LOW) {
            Serial.println("Button Pressed");
            //Path where new picture will be saved in SD Card
            String path = "/picture" + String(pictureNumber) + ".jpg";
            Serial.printf("Picture file name: %s\n", path.c_str());
            takeSavePhoto(path);
            pictureNumber++;
        }
    }
}
lastButtonState = reading;
```

Demonstration

After assembling the circuit with a pushbutton and uploading the code to your board, you can test your setup.

When you press the pushbutton, it turns on the flash, takes a picture and saves it to the microSD card.

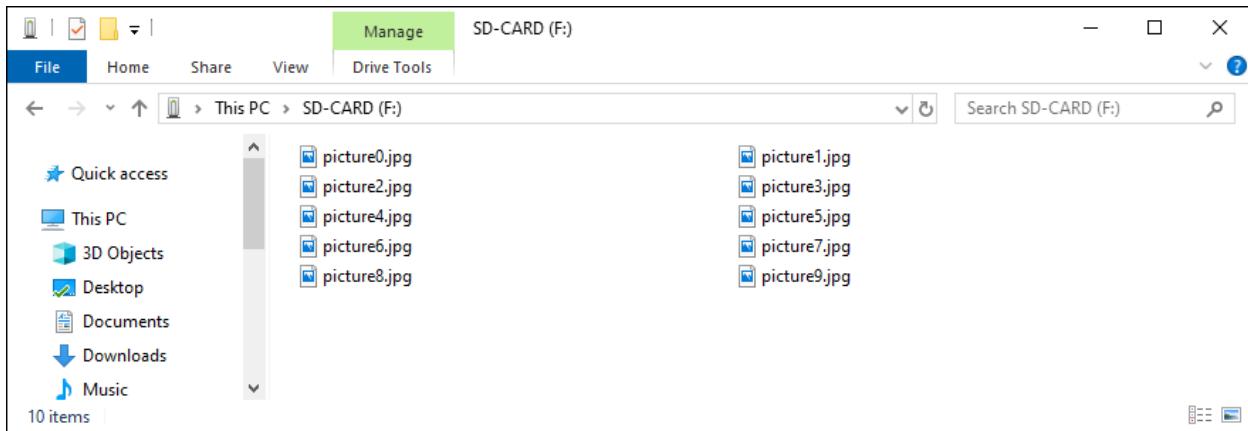


In the Serial Monitor, you should get a similar message as shown below.

```
entry 0x400806b8
Initializing the camera module...Ok!
Initializing the MicroSD card module... Starting SD Card
Button Pressed
Picture file name: /picture0.jpg
Saved file to path: /picture0.jpg
Button Pressed
Picture file name: /picture1.jpg
Saved file to path: /picture1.jpg
Button Pressed
Picture file name: /picture2.jpg
Saved file to path: /picture2.jpg
Button Pressed
Picture file name: /picture3.jpg
Saved file to path: /picture3.jpg
```

Press the button to take several pictures to test if everything is working properly.

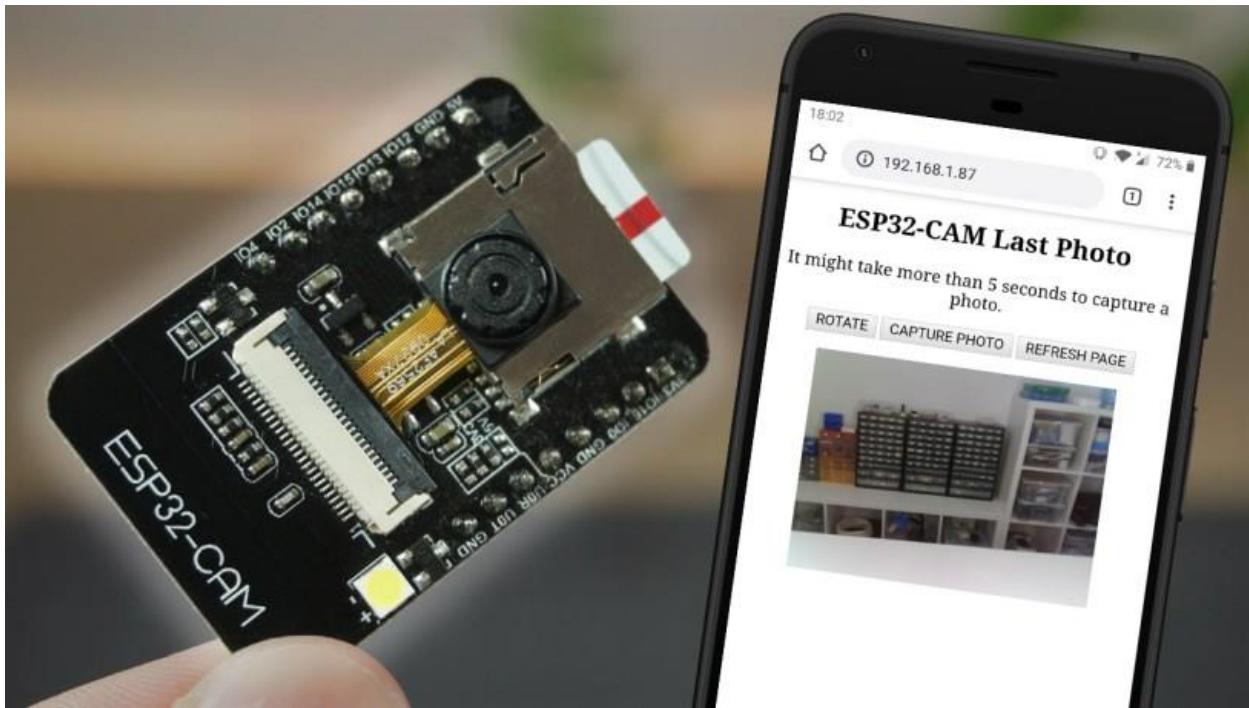
Then, you should be able to access all your photos saved on the microSD card.



Wrapping Up

In this Unit, you've learned how to wire a pushbutton to the ESP32-CAM. The best pin to connect a pushbutton is GPIO 16 because it is not being used by the microSD card neither by the camera. Instead of a pushbutton, you can connect a digital sensor or a magnetic reed switch.

Unit 5: Take Photo and Display in Web Server

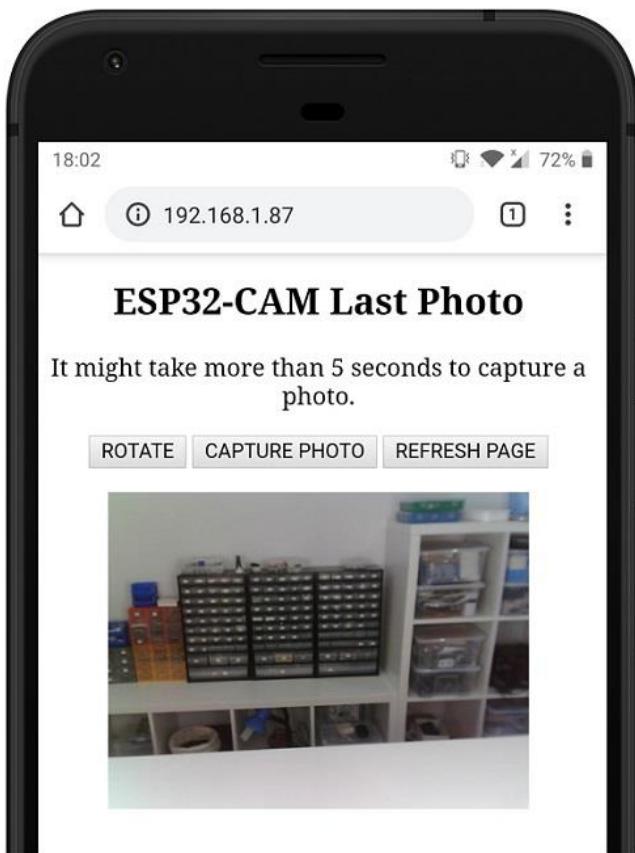


Learn how to build a web server that allows you to send a command to take a photo and visualize the latest captured photo in your browser saved in SPIFFS. We also add the option to rotate the image if necessary.

Compatibility: this project is compatible with any ESP32 camera boards with the OV2640 camera. You just need to make sure you use the right pinout for the board you're using.

Project Overview

The following image shows the web server we'll build in this tutorial.



When you access the web server, you'll see three buttons:

- **ROTATE**: depending on your ESP32-CAM orientation, you might need to rotate the photo;
- **CAPTURE PHOTO**: when you click this button, the ESP32-CAM takes a new photo and saves it in the ESP32 SPIFFS. Please wait at least 5 seconds before refreshing the web page to ensure the ESP32-CAM takes and stores the photo;
- **REFRESH PAGE**: when you click this button, the web page refreshes and it's updated with the latest photo.

Note: as mentioned previously, the latest photo captured is stored in the ESP32 SPIFFS, so even if you restart your board, you can always access the last saved photo.

Installing Libraries

To build the web server, we'll use the [ESPAsyncWebServer library](#). This library also requires the [AsyncTCP library](#) to work properly. Follow the next steps to install those libraries.

Installing the ESPAsyncWebServer library

Follow the next steps to install the ESPAsyncWebServer library:

- 1) [Click here to download the ESPAsyncWebServer library](#). You should have a .zip folder in your *Downloads* folder.
- 2) Unzip the .zip folder and you should get *ESPAsyncWebServer-master* folder.
- 3) Rename your folder from *ESPAsyncWebServer-master* to *ESPAsyncWebServer*.
- 4) Move the *ESPAsyncWebServer* folder to your Arduino IDE installation libraries folder.
- 5) Finally, re-open your Arduino IDE.

Alternatively, after downloading the library, you can go to **Sketch** ▶ **Include Library** ▶ **Add .ZIP library...** and select the library you've just downloaded.

Installing the AsyncTCP library

The ESPAsyncWebServer library requires the AsyncTCP library to work. Follow the next steps to install the AsyncTCP library:

- 1) [Click here to download the AsyncTCP library](#). You should have a .zip folder in your Downloads folder.
- 2) Unzip the .zip folder and you should get *AsyncTCP-master* folder.
- 3) Rename your folder from *AsyncTCP-master* to *AsyncTCP*.
- 4) Move the *AsyncTCP* folder to your Arduino IDE installation libraries folder.

- 5) Finally, re-open your Arduino IDE.

Alternatively, after downloading the library, you can go to **Sketch ▶ Include Library ▶ Add .ZIP library...** and select the library you've just downloaded.

Code

Copy the following code to your Arduino IDE. This code builds a web server that allows you to take a photo with your ESP32-CAM and display the last photo taken. Depending on the orientation of your ESP32-CAM, you may want to rotate the picture, so we also included that feature.

Before uploading the code to your board, you need to insert your network credentials: SSID and password. The code is well commented on where you should insert them.

CODE

https://github.com/RuiSantosdotme/ESP32-CAM-eBook/blob/master/Code/Module_2/Take_Photo_Web_Server/Take_Photo_Web_Server.ino

```
#include "WiFi.h"
#include "esp_camera.h"
#include "esp_timer.h"
#include "img_converters.h"
#include "Arduino.h"
#include "soc/soc.h"          // Disable brownout problems
#include "soc/rtc_cntl_reg.h"  // Disable brownout problems
#include "driver/rtc_io.h"
#include <ESPAsyncWebServer.h>
#include <StringArray.h>
#include <SPIFFS.h>
#include <FS.h>

// Replace with your network credentials
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

// Create AsyncWebServer object on port 80
AsyncWebServer server(80);

boolean takeNewPhoto = false;
```

```

// Photo File Name to save in SPIFFS
#define FILE_PHOTO "/photo.jpg"

// OV2640 camera module pins (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

const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE HTML><html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<style>
    body { text-align:center; }
    .vert { margin-bottom: 10%; }
    .hori{ margin-bottom: 0%; }
</style>
</head>
<body>
<div id="container">
    <h2>ESP32-CAM Last Photo</h2>
    <p>It might take more than 5 seconds to capture a photo.</p>
    <p>
        <button onclick="rotatePhoto()">ROTATE</button>
        <button onclick="capturePhoto()">CAPTURE PHOTO</button>
        <button onclick="location.reload()">REFRESH PAGE</button>
    </p>
</div>
<div></div>
</body>
<script>
    var deg = 0;
    function capturePhoto() {
        var xhr = new XMLHttpRequest();
        xhr.open('GET', '/capture', true);
        xhr.send();
    }
    function rotatePhoto() {
        var img = document.getElementById("photo");
        deg += 90;

```

```

        if(isOdd(deg/90)){ document.getElementById("container").className =
"vert"; }
        else{ document.getElementById("container").className = "hori"; }
        img.style.transform = "rotate(" + deg + "deg)";
    }
    function isOdd(n) { return Math.abs(n % 2) == 1; }
</script>
</html>)rawliteral";

void setup() {
    // Serial port for debugging purposes
    Serial.begin(115200);

    // Connect to Wi-Fi
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.println("Connecting to WiFi...");
    }
    if (!SPIFFS.begin(true)) {
        Serial.println("An Error has occurred while mounting SPIFFS");
        ESP.restart();
    }
    else {
        delay(500);
        Serial.println("SPIFFS mounted successfully");
    }

    // Print ESP32 Local IP Address
    Serial.print("IP Address: http://");
    Serial.println(WiFi.localIP());

    // Turn-off the 'brownout detector'
    WRITE_PERI_REG RTC_CNTL_BROWN_OUT_REG, 0;

    // OV2640 camera module
    camera_config_t config;
    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;

if (psramFound()) {
    config.frame_size = FRAMESIZE_UXGA;
    config.jpeg_quality = 10;
    config.fb_count = 2;
} else {
    config.frame_size = FRAMESIZE_SVGA;
    config.jpeg_quality = 12;
    config.fb_count = 1;
}
// Camera init
esp_err_t err = esp_camera_init(&config);
if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    ESP.restart();
}

// Route for root / web page
server.on("/", HTTP_GET, [] (AsyncWebServerRequest * request) {
    request->send_P(200, "text/html", index_html);
});

server.on("/capture", HTTP_GET, [] (AsyncWebServerRequest * request) {
    takeNewPhoto = true;
    request->send_P(200, "text/plain", "Taking Photo");
});

server.on("/saved-photo", HTTP_GET, [] (AsyncWebServerRequest * request) {
    request->send(SPIFFS, FILE_PHOTO, "image/jpg", false);
});

// Start server
server.begin();

}

void loop() {
    if (takeNewPhoto) {
        capturePhotoSaveSpiffs();
        takeNewPhoto = false;
    }
    delay(1);
}

// Check if photo capture was successful
bool checkPhoto( fs::FS &fs ) {
    File f_pic = fs.open( FILE_PHOTO );
    unsigned int pic_sz = f_pic.size();
    return ( pic_sz > 100 );
}

```

```

// Capture Photo and Save it to SPIFFS
void capturePhotoSaveSpiffs( void ) {
    camera_fb_t * fb = NULL; // pointer
    bool ok = 0; // Boolean indicating if the picture was captured correctly

    do {
        // Take a photo with the camera
        Serial.println("Taking a photo...");

        fb = esp_camera_fb_get();
        if (!fb) {
            Serial.println("Camera capture failed");
            return;
        }

        // Photo file name
        Serial.printf("Picture file name: %s\n", FILE_PHOTO);
        File file = SPIFFS.open(FILE_PHOTO, FILE_WRITE);

        // Insert the data in the photo file
        if (!file) {
            Serial.println("Failed to open file in writing mode");
        }
        else {
            file.write(fb->buf, fb->len); // payload (image), payload length
            Serial.print("The picture has been saved in ");
            Serial.print(FILE_PHOTO);
            Serial.print(" - Size: ");
            Serial.print(file.size());
            Serial.println(" bytes");
        }
        // Close the file
        file.close();
        esp_camera_fb_return(fb);

        // check if file has been correctly saved in SPIFFS
        ok = checkPhoto(SPIFFS);
    } while ( !ok );
}

```

How the Code Works

First, include the required libraries to work with the camera, to build the web server and to use SPIFFS.

```

#include "WiFi.h"
#include "esp_camera.h"
#include "esp_timer.h"
#include "img_converters.h"

```

```
#include "Arduino.h"
#include "soc/soc.h"           // Disable brownout problems
#include "soc/rtc_CNTL_REG.h"   // Disable brownout problems
#include "driver/rtc_io.h"
#include <ESPAsyncWebServer.h>
#include <StringArray.h>
#include <SPIFFS.h>
#include <FS.h>
```

Next, write your network credentials in the following variables, so that the ESP32-CAM can connect to your local network.

```
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = " REPLACE_WITH_YOUR_PASSWORD";
```

Create an AsyncWebServer object on port 80.

```
AsyncWebServer server(80);
```

The `takeNewPhoto` boolean variable indicates when it's time to take a new photo.

```
boolean takeNewPhoto = false;
```

Then, define the path and name of the photo to be saved in SPIFFS.

```
#define FILE_PHOTO "/photo.jpg"
```

Next, define the camera pins for your ESP32 camera development board. This is the pinout for the AI-THINKER. Check the [Appendix](#) if you're using a different camera model.

```
// OV2640 camera module pins (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
```

Building the Web Page

Next, we have the HTML text to build the web page:

```
const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE HTML><html>
<head>
<meta name="viewport" content="width=device-width, initial-scale=1">
<style>
    body { text-align:center; }
    .vert { margin-bottom: 10%; }
    .hori{ margin-bottom: 0%; }
</style>
</head>
<body>
<div id="container">
    <h2>ESP32-CAM Last Photo</h2>
    <p>It might take more than 5 seconds to capture a photo.</p>
    <p>
        <button onclick="rotatePhoto()">ROTATE</button>
        <button onclick="capturePhoto()">CAPTURE PHOTO</button>
        <button onclick="location.reload()">REFRESH PAGE</button>
    </p>
</div>
<div></div>
</body>
<script>
    var deg = 0;
    function capturePhoto() {
        var xhr = new XMLHttpRequest();
        xhr.open('GET', '/capture', true);
        xhr.send();
    }
    function rotatePhoto() {
        var img = document.getElementById("photo");
        deg += 90;
        if(isOdd(deg/90)){ document.getElementById("container").className =
"vert"; }
        else{ document.getElementById("container").className = "hori"; }
        img.style.transform = "rotate(" + deg + "deg)";
    }
    function isOdd(n) { return Math.abs(n % 2) == 1; }
</script>
</html>)rawliteral";
```

We won't go into much detail on how this HTML works. We'll just take a quick overview.

Basically, create three buttons: **ROTATE**; **CAPTURE PHOTO** and **REFRESH PAGE**. Each photo calls a different JavaScript function: `rotatePhoto()`, `capturePhoto()` and `reload()`.

```
<button onclick="rotatePhoto()">ROTATE</button>
<button onclick="capturePhoto()">CAPTURE PHOTO</button>
<button onclick="location.reload()">REFRESH PAGE</button>
```

The `capturePhoto()` function sends a request on the **/capture** URL to the ESP32, so it takes a new photo.

```
function capturePhoto() {
  var xhr = new XMLHttpRequest();
  xhr.open('GET', "/capture", true);
  xhr.send();
}
```

The `rotatePhoto()` function rotates the photo.

```
function rotatePhoto() {
  var img = document.getElementById("photo");
  deg += 90;
  if(isOdd(deg/90)){ document.getElementById("container").className = "vert"; }
  else{ document.getElementById("container").className = "hori"; }
  img.style.transform = "rotate(" + deg + "deg)";
}
function isOdd(n) { return Math.abs(n % 2) == 1; }
```

We're not sure what's the "best" way to rotate a photo with JavaScript. This method works perfectly, but there may be better ways to do this.

Finally, the following section displays the photo.

```
<div></div>
```

When, you click the **REFRESH** button, it will load the latest image.

setup()

In the `setup()`, initialize a Serial communication:

```
Serial.begin(115200);
```

Connect the ESP32-CAM to your local network:

```
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
}
```

Initialize SPIFFS:

```
if (!SPIFFS.begin(true)) {
    Serial.println("An Error has occurred while mounting SPIFFS");
    ESP.restart();
}
else {
    delay(500);
    Serial.println("SPIFFS mounted successfully");
}
```

Print the ESP32-CAM local IP address:

```
Serial.print("IP Address: http://");
Serial.println(WiFi.localIP());
```

The lines that follow, configure and initialize the camera with the right settings.

```
camera_config_t config;
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;

if (psramFound()) {
    config.frame_size = FRAMESIZE_UXGA;
    config.jpeg_quality = 10;
    config.fb_count = 2;
```

```

} else {
    config.frame_size = FRAMESIZE_SVGA;
    config.jpeg_quality = 12;
    config.fb_count = 1;
}
// Camera init
esp_err_t err = esp_camera_init(&config);
if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    ESP.restart();
}

```

Handle the Web Server

Next, we need to handle what happens when the ESP32-CAM receives a request on a certain URL.

When the ESP32-CAM receives a request on the root **/** URL, we send the HTML text to build the web page.

```

server.on("/", HTTP_GET, [] (AsyncWebServerRequest * request) {
    request->send_P(200, "text/html", index_html);
});

```

When we press the “**CAPTURE**” button on the web server, we send a request to the ESP32 **/capture** URL. When that happens, we set the `takeNewPhoto` variable to `true`, so that we know it is time to take a new photo.

```

server.on("/capture", HTTP_GET, [] (AsyncWebServerRequest * request) {
    takeNewPhoto = true;
    request->send_P(200, "text/plain", "Taking Photo");
});

```

In case there's a request on the **/saved-photo** URL, send the photo saved in SPIFFS to a connected client:

```

server.on("/saved-photo", HTTP_GET, [] (AsyncWebServerRequest * request)
{
    request->send(SPIFFS, FILE_PHOTO, "image/jpg", false);
});

```

Finally, start the web server.

```

server.begin();

```

loop()

In the `loop()`, if the `takeNewPhoto` variable is true, we call the `capturePhotoSaveSpiffs()` to take a new photo and save it to SPIFFS. Then, set the `takeNewPhoto` variable to false.

```
void loop() {
    if (takeNewPhoto) {
        capturePhotoSaveSpiffs();
        takeNewPhoto = false;
    }
    delay(1);
}
```

Take a Photo

There are two other functions in the sketch: `checkPhoto()` and `capturePhotoSaveSpiffs()`.

The `checkPhoto()` function checks if the photo was successfully saved to SPIFFS.

```
// Check if photo capture was successful
bool checkPhoto( fs::FS &fs ) {
    File f_pic = fs.open( FILE_PHOTO );
    unsigned int pic_sz = f_pic.size();
    return ( pic_sz > 100 );
}
```

The `capturePhotoSaveSpiffs()` function takes a photo and saves it to SPIFFS.

```
void capturePhotoSaveSpiffs( void ) {
    camera_fb_t * fb = NULL; // pointer
    bool ok = 0; // Boolean indicating if the picture has been taken
    correctly

    do {
        // Take a photo with the camera
        Serial.println("Taking a photo...");

        fb = esp_camera_fb_get();
        if (!fb) {
            Serial.println("Camera capture failed");
            return;
        }

        // Photo file name
```

```

Serial.printf("Picture file name: %s\n", FILE_PHOTO);
File file = SPIFFS.open(FILE_PHOTO, FILE_WRITE);

// Insert the data in the photo file
if (!file) {
    Serial.println("Failed to open file in writing mode");
}
else {
    file.write(fb->buf, fb->len); // payload (image), payload length
    Serial.print("The picture has been saved in ");
    Serial.print(FILE_PHOTO);
    Serial.print(" - Size: ");
    Serial.print(file.size());
    Serial.println(" bytes");
}
// Close the file
file.close();
esp_camera_fb_return(fb);

// check if file has been correctly saved in SPIFFS
ok = checkPhoto(SPIFFS);
} while ( !ok );
}

```

This function was based on this [sketch by dualvim](#).

Demonstration

Upload the previous code to your board. Don't forget to insert your network credentials so that the ESP32 can connect to your local network. Also, don't forget to change the camera pins definition if you're not using an ESP32-CAM AI-Thinker board.

After uploading the code, disconnect GPIO 0 from GND. Open the Serial Monitor at a baud rate of 115200 and press the ESP32 camera on-board RESET button.

The ESP32 IP address will be printed on the Serial Monitor.

Open your browser and type the ESP32-CAM IP Address. Then, click the "CAPTURE PHOTO" to take a new photo and wait a few seconds for the photo to be saved in SPIFFS.

Then, if you press the “REFRESH PAGE” button, the page will update with the latest saved photo. If you need to adjust the image orientation, you can always use the “ROTATE” button to do it so.



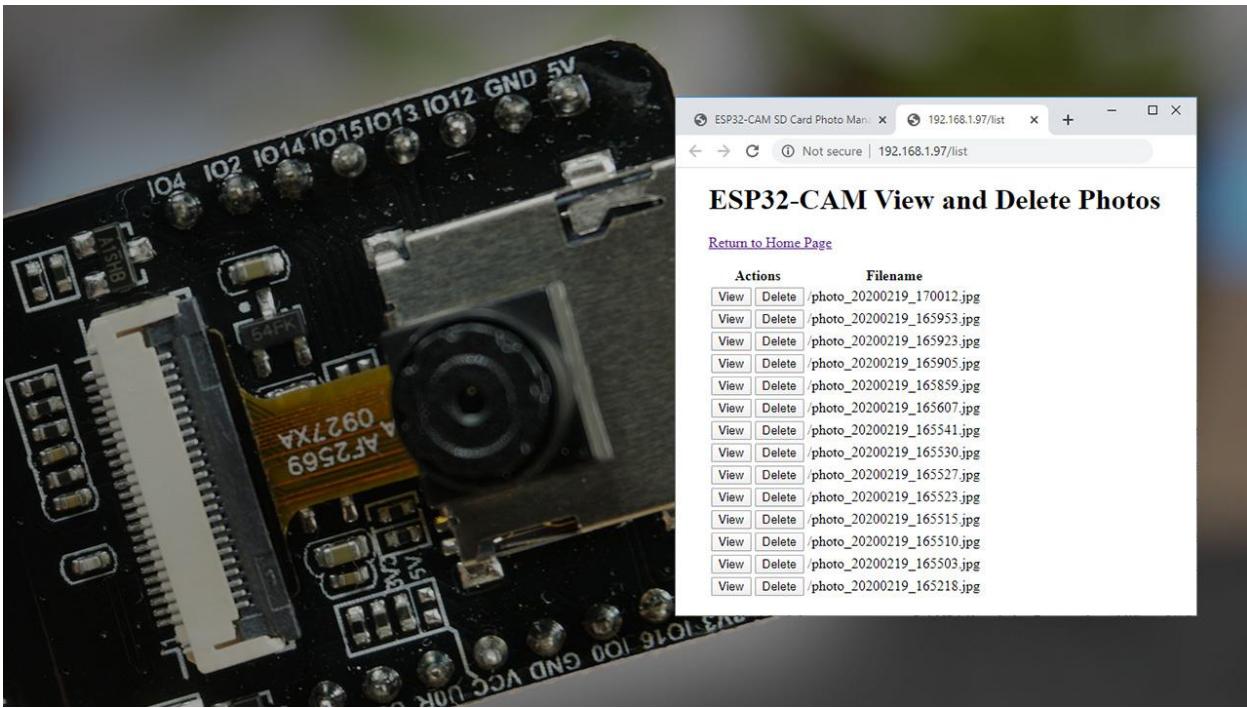
In your Arduino IDE Serial Monitor window, you should get similar messages when you press the buttons:

```
∞ COM4
|
ho 0 tail 12 room 4
load:0x40080400,len:6352
entry 0x400806b8
Connecting to WiFi..
Connecting to WiFi..
SPIFFS mounted successfully
IP Address: http://192.168.1.87
Taking a picture...
Picture file name: /photo.jpg
The picture has been saved in /photo.jpg - Size: 0 bytes
Taking a picture...
Picture file name: /photo.jpg
The picture has been saved in /photo.jpg - Size: 86528 bytes
```

Wrapping Up

This project allowed you to take a photo with your ESP32-CAM and display it in a web server. The photo taken is temporarily saved in the ESP32 SPIFFS. You can modify this project to save all the photos in a microSD card instead of using SPIFFS.

Unit 6: Web Server SD Card Photo Manager: Capture, View and Delete



In this Unit, you'll build an ESP32-CAM SD Card Photo Manager Web Server. This web server allows you to take a photo, save it in the microSD card and manage all the photos saved previously. You can view previous captured photos or delete them when needed. The photo filename includes date and time. You can skip to the Demonstration section to see how the project works in practical terms.

Compatibility: for this project you need a camera board with microSD card support: **ESP32-CAM AI-Thinker**. This project is not compatible with any of the M5-Stack camera boards neither the ESP-EYE, or TTGO T-Journal because these don't have microSD card support.

Project Overview

When you access the ESP32-CAM IP address, you'll get a similar web page.



The web page displays four buttons:

- **ROTATE**: rotate the image 90 degrees.
- **CAPTURE PHOTO**: captures a new photo and saves it on the microSD card.
- **REFRESH PAGE**: refreshes the page to show the last captured photo.
- **VIEW AND DELETE PHOTOS**: opens a new tab on the `/list` URL that shows the filename of all photos saved on the microSD card. There are **View** and **Delete** buttons to view or delete saved photos.

Installing Libraries

To build the web server, we'll use the [ESPAsyncWebServer library](#). This library also requires the [AsyncTCP library](#) to work properly. Follow the next steps to install those libraries. If you have followed the previous project, you should already have these libraries installed, so you can skip this section.

Installing the ESPAsyncWebServer library

Follow the next steps to install the ESPAsyncWebServer library:

- 1) [Click here to download the ESPAsyncWebServer library](#). You should have a .zip folder in your *Downloads* folder.
- 2) Unzip the .zip folder and you should get *ESPAsyncWebServer-master* folder.
- 3) Rename your folder from *ESPAsyncWebServer-master* to *ESPAsyncWebServer*.
- 4) Move the *ESPAsyncWebServer* folder to your Arduino IDE installation libraries folder.
- 5) Finally, re-open your Arduino IDE.

Alternatively, after downloading the library, you can go to **Sketch** ▶ **Include Library** ▶ **Add .ZIP library...** and select the library you've just downloaded.

Installing the AsyncTCP library

The ESPAsyncWebServer library requires the AsyncTCP library to work. Follow the next steps to install the AsyncTCP library:

- 1) [Click here to download the AsyncTCP library](#). You should have a .zip folder in your *Downloads* folder.
- 2) Unzip the .zip folder and you should get *AsyncTCP-master* folder.
- 3) Rename your folder from *AsyncTCP-master* to *AsyncTCP*.

- 4) Move the *AsyncTCP* folder to your Arduino IDE installation libraries folder.
- 5) Finally, re-open your Arduino IDE.

Alternatively, after downloading the library, you can go to **Sketch ▶ Include Library ▶ Add .ZIP library...** and select the library you've just downloaded.

Code

Here's the code to build the SD Card Photo Manager Web Server. To make the project work for you, you need to insert your network credentials and change the NTP server settings if you're on a different time zone.

CODE

https://github.com/RuiSantosdotme/ESP32-CAM-eBook/blob/master/Code/Module_2/Web_Server_Photo_Manager/Web_Server_Photo_Manager.ino

```
#include "WiFi.h"
#include "esp_camera.h"
#include "esp_timer.h"
#include "img_converters.h"
#include "Arduino.h"
#include "soc/soc.h"          // Disable brownout problems
#include "soc/rtc_cntl_reg.h"  // Disable brownout problems
#include "driver/rtc_io.h"
#include <ESPAsyncWebServer.h>
#include <StringArray.h>
#include "FS.h"                // SD Card ESP32
#include "SD_MMC.h"            // SD Card ESP32
#include "time.h"
#include <WiFiUdp.h>

// Replace with your network credentials
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_PASSWORD";

// NTP Server
const char* ntpServer = "pool.ntp.org";
const long gmtOffset_sec = 0;
const int daylightOffset_sec = 3600;

// Create AsyncWebServer object on port 80
AsyncWebServer server(80);

boolean takeNewPhoto = false;
```

```

String lastPhoto = "";
String list = "";

// HTTP GET parameter
const char* PARAM_INPUT_1 = "photo";

// OV2640 camera module pins (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

//Stores the camera configuration parameters
camera_config_t config;

File root;

const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE HTML><html>
<head>
    <title>ESP32-CAM SD Card Photo Manager</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <style>
        body { text-align:center; }
        .vert { margin-bottom: 25%; }
        .horiz{ margin-bottom: 0%; }
        img { height:auto; width:100%; }
    </style>
</head>
<body>
    <div id="container">
        <h1>ESP32-CAM SD Card Photo Manager</h1>
        <p>It might take more than 5 seconds to capture a photo.</p>
        <p>
            <button onclick="rotatePhoto();">ROTATE</button>
            <button onclick="capturePhoto()">CAPTURE PHOTO</button>
            <button onclick="location.reload();">REFRESH PAGE</button>
            <button onclick="window.open('/list','_blank')">VIEW AND DELETE
PHOTOS</button>
        </p>
    </div>
</body>
</html>
)rawliteral";

```

```

</div>

</body>
<script>
var deg = 0;
function capturePhoto() {
    var xhr = new XMLHttpRequest();
    xhr.open('GET', '/capture', true);
    xhr.send();
}
function rotatePhoto() {
    var img = document.getElementById("photo");
    deg += 90;
    if(isOdd(deg/90)){ document.getElementById("container").className =
"vert"; }
    else{ document.getElementById("container").className = "hori"; }
    img.style.transform = "rotate(" + deg + "deg)";
}
function isOdd(n){return Math.abs(n % 2)==1;}
</script>
</html>) rawliteral;

void setup() {
    // Turn-off the brownout detector
    WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0);

    // Serial port for debugging purposes
    Serial.begin(115200);

    // Connect to Wi-Fi
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.println("Connecting to WiFi...");
    }

    // Print ESP32 Local IP Address
    Serial.print("IP Address: http://");
    Serial.println(WiFi.localIP());

    configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);

    Serial.println("Initializing the camera module...");
    configInitCamera();

    Serial.println("Initializing the MicroSD card module... ");
    initMicroSDCard();

    server.on("/", HTTP_GET, [] (AsyncWebServerRequest * request) {
        request->send_P(200, "text/html", index_html);
    });
}

```

```

server.on("/capture", HTTP_GET, [] (AsyncWebServerRequest * request) {
    takeNewPhoto = true;
    request->send_P(200, "text/plain", "Taking Photo");
});

server.on("/saved-photo", HTTP_GET, [] (AsyncWebServerRequest * request) {
    request->send(SD_MMC, lastPhoto, "image/jpg", false);
});

server.on("/list", HTTP_GET, [] (AsyncWebServerRequest * request) {
    request->send_P(200, "text/html", list.c_str());
});

server.on("/view", HTTP_GET, [] (AsyncWebServerRequest * request) {
    String inputMessage;
    String inputParam;
    // GET input1 value on <ESP_IP>/view?photo=<inputMessage>
    if (request->hasParam(PARAM_INPUT_1)) {
        inputMessage = request->getParam(PARAM_INPUT_1)->value();
        inputParam = PARAM_INPUT_1;
    }
    else {
        inputMessage = "No message sent";
        inputParam = "none";
    }
    Serial.println(inputMessage);
    request->send(SD_MMC, inputMessage, "image/jpg", false);
});

// Send a GET request to <ESP_IP>/delete?photo=<inputMessage>
server.on("/delete", HTTP_GET, [] (AsyncWebServerRequest *request) {
    String inputMessage;
    String inputParam;
    // GET input1 value on <ESP_IP>/delete?photo=<inputMessage>
    if (request->hasParam(PARAM_INPUT_1)) {
        inputMessage = request->getParam(PARAM_INPUT_1)->value();
        inputParam = PARAM_INPUT_1;
    }
    else {
        inputMessage = "No message sent";
        inputParam = "none";
    }
    Serial.println(inputMessage);
    deleteFile(SD_MMC, inputMessage.c_str());
    request->send(200, "text/html", "Done. Your photo named " +
inputMessage + " was removed.<br><a href=\"/\">Return to Home Page</a>
or <a href=\"/list\">view/delete other photos</a>.");
});
// Start server
server.begin();

root = SD_MMC.open("/");
listDirectory(SD_MMC);

```

```

}

void loop() {
    if (takeNewPhoto) {
        takeSavePhoto();
        takeNewPhoto = false;
    }
    delay(1);
}

void configInitCamera() {
    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; //YUV422,GRAYSCALE,RGB565,JPEG

    // Select lower framesize if the camera doesn't support PSRAM
    if(psramFound()){
        config.frame_size=FRAMESIZE_UXGA;//FRAMESIZE_QVGA|CIF|VGA|SVGA|XGA|SXGA|UXGA
        config.jpeg_quality = 1; //0-63 lower number means higher quality
        config.fb_count = 2;
    }
    else {
        config.frame_size = FRAMESIZE_SVGA;
        config.jpeg_quality = 12;
        config.fb_count = 1;
    }

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

void initMicroSDCard() {

```

```

// Start Micro SD card
Serial.println("Starting SD Card");
if(!SD_MMC.begin()){
    Serial.println("SD Card Mount Failed");
    return;
}
uint8_t cardType = SD_MMC.cardType();
if(cardType == CARD_NONE){
    Serial.println("No SD Card attached");
    return;
}
}

void takeSavePhoto(){
    struct tm timeinfo;
    char now[20];
    // Take Picture with Camera
    camera_fb_t * fb = esp_camera_fb_get();
    if(!fb) {
        Serial.println("Camera capture failed");
        return;
    }
    // Path where new picture will be saved in SD Card
    getLocalTime(&timeinfo);
    strftime(now, 20, "%Y%m%d_%H%M%S", &timeinfo); // Format Date & Time
    String path = "/photo_" + String(now) + ".jpg";
    lastPhoto = path;
    Serial.printf("Picture file name: %s\n", path.c_str());
    // Save picture to microSD card
    fs::FS &fs = SD_MMC;
    File file = fs.open(path.c_str(), FILE_WRITE);
    if(!file){
        Serial.printf("Failed to open file in writing mode");
    }
    else {
        file.write(fb->buf, fb->len); // payload (image), payload length
        Serial.printf(" Saved: %s\n", path.c_str());
        listDirectory(SD_MMC);
    }
    file.close();
    esp_camera_fb_return(fb);
}

void listDirectory(fs::FS &fs) {
    File root = fs.open("/");
    list = "";
    if(!root){
        Serial.println("Failed to open directory");
        return;
    }
    if(!root.isDirectory()){
        Serial.println("Not a directory");
        return;
    }
}

```

```

File file = root.openNextFile();
while(file){
    if(!file.isDirectory()){
        String filename=String(file.name());
        filename.toLowerCase();
        if (filename.indexOf(".jpg")!= -1) {
            list = "<tr><td><button onclick=\"window.open('/view?photo=" +String(file.name())+"', '_blank')\">View</button></td><td><button onclick=\"window.location.href='/delete?photo=" +String(file.name())+"'">Delete</button></td><td>" +String(file.name()) + "</td><td></td></tr>" +list;
        }
    }
    lastPhoto = file.name();
    file = root.openNextFile();
}

if (list=="") {
    list=<tr>No photos Stored</tr>;
}
else {
    list=<h1>ESP32-CAM View and Delete Photos</h1><p><a href="/">Return to Home Page</a></p><table><th colspan="2">Actions</th><th>Filename</th>" +list+"</table>";
}
}

void deleteFile(fs::FS &fs, const char * path) {
    Serial.printf("Deleting file: %s\n", path);
    if(fs.remove(path)){
        Serial.println("File deleted");
        listDirectory(SD_MMC);
    }
    else {
        Serial.println("Delete failed");
    }
}

```

How the Code Works

The code for this project is quite long and a bit complex. Explaining exactly what each line of code does, goes beyond the scope of this eBook. We'll just explain the sections of code that are relevant for this project. Alternatively, you can skip to the Demonstration section.

Import Libraries

Import all the necessary libraries. You need to import the `ESPASyncWebServer` library to build the web server. The `FS.h` and `SD_MMC.h` are used to interact with the microSD card and manage files. The `time.h` and `WiFiUdp.h` are used to get date and time from the NTP server. We explain about the NTP server in [Module 2, Unit 2](#).

```
#include "WiFi.h"
#include "esp_camera.h"
#include "esp_timer.h"
#include "img_converters.h"
#include "Arduino.h"
#include "soc/soc.h"           // Disable brownout problems
#include "soc/rtc_CNTL_Reg.h"  // Disable brownout problems
#include "driver/rtc_io.h"
#include <ESPASyncWebServer.h>
#include <StringArray.h>
#include "FS.h"                 // SD Card ESP32
#include "SD_MMC.h"             // SD Card ESP32
#include "time.h"
#include <WiFiUdp.h>
```

Network Credentials

Insert your network credentials in the following variables.

```
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_PASSWORD";
```

NTP Server Settings

Then, you need to define the following variables to configure and get time from an NTP server.

```
const char* ntpServer = "pool.ntp.org";
const long gmtOffset_sec = 0;
const int daylightOffset_sec = 3600;
```

We'll request the time from pool.ntp.org, which is a cluster of timeservers that anyone can use to request the time.

The `gmtOffset_sec` variable defines the offset in seconds between your time zone and GMT. We live in Portugal, so the time offset is 0. Change the time `gmtOffset_sec` variable to match your time zone.

```
const long gmtOffset_sec = 0;
```

The `daylightOffset_sec` variable defines the offset in seconds for daylight saving time. It is generally one hour, that corresponds to 3600 seconds.

```
const int daylightOffset_sec = 3600;
```

Web Page

The following lines create a variable called `index_html` that contains the HTML text to build the web page.

```
const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE HTML><html>
<head>
  <title>ESP32-CAM SD Card Photo Manager</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <style>
    body { text-align:center; }
    .vert { margin-bottom: 25%; }
    .horiz { margin-bottom: 0%; }
    img { height:auto; width:100%; }
  </style>
</head>
<body>
  <div id="container">
    <h1>ESP32-CAM SD Card Photo Manager</h1>
    <p>It might take more than 5 seconds to capture a photo.</p>
    <p>
      <button onclick="rotatePhoto();">ROTATE</button>
      <button onclick="capturePhoto()">CAPTURE PHOTO</button>
      <button onclick="location.reload();">REFRESH PAGE</button>
      <button onclick="window.open('/list','_blank')">VIEW AND DELETE
PHOTOS</button>
    </p>
  </div>
  
</body>
<script>
  var deg = 0;
```

```

function capturePhoto() {
    var xhr = new XMLHttpRequest();
    xhr.open('GET', '/capture', true);
    xhr.send();
}
function rotatePhoto() {
    var img = document.getElementById("photo");
    deg += 90;
    if(isOdd(deg/90)){ document.getElementById("container").className = "vert"; }
    else{ document.getElementById("container").className = "hori"; }
    img.style.transform = "rotate(" + deg + "deg)";
}
function isOdd(n){return Math.abs(n % 2)==1;}
</script>
</html>) rawliteral";

```

Between the `<head>` and `</head>` tags, we define the title of our page. The title is what shows up on the web browser tab. You can change the title if you want.

```
<title>ESP32-CAM SD Card Photo Manager</title>
```

The following line makes the web server responsive in every device.

```
<meta name="viewport" content="width=device-width, initial-scale=1">
```

Between the `<style></style>` tags, we add some CSS to align the buttons and the photos.

```

<head>
    <style>
        body { text-align:center; }
        .vert { margin-bottom: 25%; }
        .hori{ margin-bottom: 0%; }
        img { height:auto; width:100%; }
    </style>
</head>

```

Next, inside the `<body></body>` tags, and between the `<h1></h1>` tags you can write the first heading of your web page. In this case, its “ESP32-CAM SD Card Photo Manager”, but you can change it if you want.

```
<h1>ESP32-CAM SD Card Photo Manager</h1>
```

There's a paragraph indicating that taking a photo might take a few seconds. You can remove this line if you don't want this paragraph to show up in your web page.

```
<p>It might take more than 5 seconds to capture a photo.</p>
```

Next, we create four buttons.

```
<button onclick="rotatePhoto()">ROTATE</button>
<button onclick="capturePhoto()">CAPTURE PHOTO</button>
<button onclick="location.reload()">REFRESH PAGE</button>
<button onclick="window.open('/list','_blank')">VIEW AND DELETE
PHOTOS</button>
```

The first button contains the “ROTATE” text and when clicked calls the `rotatePhoto()` JavaScript function. The CAPTURE PHOTO button calls the `capturePhoto()` function. The REFRESH PAGE button calls `location.reload()` that refreshes the web page. Finally, the VIEW AND DELETE PHOTOS button opens a new window on the `/list` URL.

The last image taken is displayed after the buttons. To get the image, a request is made on the `/saved-photo` URL. In case there isn’t a photo available, the alt text is displayed instead.

```

```

Finally, there’s a section between the `<script></script>` tags that contains the JavaScript functions.

```
<script>
  var deg = 0;
  function capturePhoto() {
    var xhr = new XMLHttpRequest();
    xhr.open('GET', '/capture', true);
    xhr.send();
  }
  function rotatePhoto() {
    var img = document.getElementById("photo");
    deg += 90;
    if(isOdd(deg/90)){ document.getElementById("container").className =
"vert"; }
    else{ document.getElementById("container").className = "hori"; }
    img.style.transform = "rotate(" + deg + "deg)";
  }
  function isOdd(n){return Math.abs(n % 2)==1;}
</script>
```

The `capturePhoto()` function makes an HTTP GET request on the `/capture` URL.

```
xhr.open('GET', "/capture", true);
```

The `rotatePhoto()` rotates the photo 90 degrees every time you click on the button.

```
function rotatePhoto() {
  var img = document.getElementById("photo");
  deg += 90;
  if(isOdd(deg/90)){ document.getElementById("container").className = "vert"; }
  else{ document.getElementById("container").className = "hori"; }
  img.style.transform = "rotate(" + deg + "deg)";
}
```

setup()

In the `setup()`, initialize the Serial Monitor.

```
// Serial port for debugging purposes
Serial.begin(115200);
```

Connect to Wi-Fi and print the ESP32 IP address.

```
// Connect to Wi-Fi
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
  delay(1000);
  Serial.println("Connecting to WiFi...");
}
// Print ESP32 Local IP Address
Serial.print("IP Address: http://");
Serial.println(WiFi.localIP());
```

Configure the time with the settings you've defined earlier.

```
configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
```

Call the `configInitCamera()` and `initMicroSDCard()` functions to initialize the camera and the microSD card.

```
configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
```

Handle the Web Server

Then, we need to handle what happens when the ESP32 gets requests. When you access the root URL / (the ESP32 IP address), send the HTML page as follows.

```
server.on("/", HTTP_GET, [] (AsyncWebServerRequest * request) {  
    request->send_P(200, "text/html", index_html);  
});
```

As mentioned previously, when you click on the CAPTURE PHOTO button, a request is made on the */capture* URL. When that happens we set the `takeNewPhoto` variable to true, so that we know it's time to take a new photo.

```
server.on("/capture", HTTP_GET, [] (AsyncWebServerRequest * request) {  
    takeNewPhoto = true;  
    request->send_P(200, "text/plain", "Taking Photo");  
});
```

The following lines send the last photo taken to be displayed on the web server. Take a look at the arguments of the `send()` function. `SD_MMC` indicates that the file we want to send is saved on the microSD card, the `lastPhoto` is the name of the last photo taken, “`image/jpg`” is the file type.

```
server.on("/saved-photo", HTTP_GET, [] (AsyncWebServerRequest * request) {  
    request->send(SD_MMC, lastPhoto, "image/jpg", false);  
});
```

When a request is received on the */list* URL, we send the `list` variable that contains the HTML text to display a list with all the photos saved on the microSD card. The value of the `list` variable is defined on the `listDirectory()` function declared and called later on.

```
server.on("/list", HTTP_GET, [] (AsyncWebServerRequest * request) {  
    request->send_P(200, "text/html", list.c_str());  
});
```

When you are on the */list* tab, you can click on the **View** or **Delete** buttons to view a photo or delete it.

The following lines handle what happens when you click the **View** buttons.

```
server.on("/view", HTTP_GET, [] (AsyncWebServerRequest * request) {
    String inputMessage;
    String inputParam;
    // GET input1 value on <ESP_IP>/view?photo=<inputMessage>
    if (request->hasParam(PARAM_INPUT_1)) {
        inputMessage = request->getParam(PARAM_INPUT_1)->value();
        inputParam = PARAM_INPUT_1;
    }
    else {
        inputMessage = "No message sent";
        inputParam = "none";
    }
    Serial.println(inputMessage);
    request->send(SD_MMC, inputMessage, "image/jpg", false);
});
```

For example, when you click the **View** button for a photo with the `photo_20200219_170012.jpg` name, you are making a request on the following URL.

```
<ESP_IP_address>/view?photo=/photo_20200219_170012.jpg
```

The URL contains a parameter with the photo name. When a request is received, the photo name is saved on the `inputMessage` variable.

```
inputMessage = request->getParam(PARAM_INPUT_1)->value();
```

So, now we know which photo we need to send, because its name is saved on the `inputMessage` variable.

```
request->send(SD_MMC, inputMessage, "image/jpg", false);
```

The process to delete a photo is similar, but you receive a request on the `/delete` URL instead.

```
// Send a GET request to <ESP_IP>/delete?photo=<inputMessage>
server.on("/delete", HTTP_GET, [] (AsyncWebServerRequest *request) {
    String inputMessage;
    String inputParam;
    // GET input1 value on <ESP_IP>/delete?photo=<inputMessage>
    if (request->hasParam(PARAM_INPUT_1)) {
        inputMessage = request->getParam(PARAM_INPUT_1)->value();
        inputParam = PARAM_INPUT_1;
    }
});
```

```

    else {
        inputMessage = "No message sent";
        inputParam = "none";
    }
    Serial.println(inputMessage);
    deleteFile(SD_MMC, inputMessage.c_str());
    request->send(200, "text/html", "Done. Your photo named " +
inputMessage + " was removed.<br><a href=\"/\">Return to Home Page</a>
or <a href=\"/list\">view/delete other photos</a>.");
}

```

To delete the photo, we call the `deleteFile()` function and pass the photo filename (saved on the `inputMessage` variable) as argument.

```
deleteFile(SD_MMC, inputMessage.c_str());
```

Finally, start the server.

```
server.begin();
```

And that's pretty much how we handle the requests from the web server.

loop()

In the `loop()`, we check whether is it time to take a new photo. We should take a new photo when the `takeNewPhoto` variable is true. When that happens, we call the `takeSavePhoto()` function and set the `takeNewPhoto` variable to false.

takeSavePhoto()

The `takeSavePhoto()` function gets date and time, captures a new phot and saves it on the microSD card. The photo filename contains the date and time as shown in a previous project ([Module 2, Unit 2](#)).

```

void takeSavePhoto() {
    struct tm timeinfo;
    char now[20];
    // Take Picture with Camera
    camera_fb_t * fb = esp_camera_fb_get();
    if(!fb) {
        Serial.println("Camera capture failed");
        return;
    }
}

```

```

// Path where new picture will be saved in SD Card
getLocalTime(&timeinfo);
strftime(now, 20, "%Y%m%d %H%M%S", &timeinfo); // Format Date & Time
String path = "/photo_" + String(now) + ".jpg";
lastPhoto = path;
Serial.printf("Picture file name: %s\n", path.c_str());
// Save picture to microSD card
fs::FS &fs = SD_MMC;
File file = fs.open(path.c_str(), FILE_WRITE);
if(!file){
    Serial.printf("Failed to open file in writing mode");
}
else {
    file.write(fb->buf, fb->len); // payload (image), payload length
    Serial.printf(" Saved: %s\n", path.c_str());
    listDirectory(SD_MMC);
}
file.close();
esp_camera_fb_return(fb);
}

```

listDirectory()

The `listDirectory()` function declared after the `loop()`, opens the microSD card and lists all the files. The `list` variable contains HTML text to show all the photo's filenames as well as the buttons to delete and view the photos.

```

void listDirectory(fs::FS &fs) {
    File root = fs.open("/");
    list = "";
    if(!root){
        Serial.println("Failed to open directory");
        return;
    }
    if(!root.isDirectory()){
        Serial.println("Not a directory");
        return;
    }

    File file = root.openNextFile();
    while(file){
        if(!file.isDirectory()){
            String filename=String(file.name());
            filename.toLowerCase();
            if (filename.indexOf(".jpg")!= -1){
                list = "<tr><td><button";
onclicl="window.open('/view?photo="+String(file.name())+"', '_blank')\"
">View</button></td><td><button
onclicl="window.location.href='/delete?photo="+String(file.name())+"'";

```

```

        \">Delete</button></td><td>"+String(file.name())+"</td><td></td></tr>" +list;
    }
}
lastPhoto = file.name();
file = root.openNextFile();
}

if (list=="") {
    list=<tr>No photos Stored</tr>;
}
else {
    list=<h1>ESP32-CAM View and Delete Photos</h1><p><a href=\"/\">Return to Home Page</a></p><table><thead><tr><th colspan=\"2\">Actions</th><th>Filename</th></tr></thead><tbody>"+list+"</tbody></table>";
}
}

```

deleteFile()

Finally, the `deleteFile()` function deletes a photo from the microSD card.

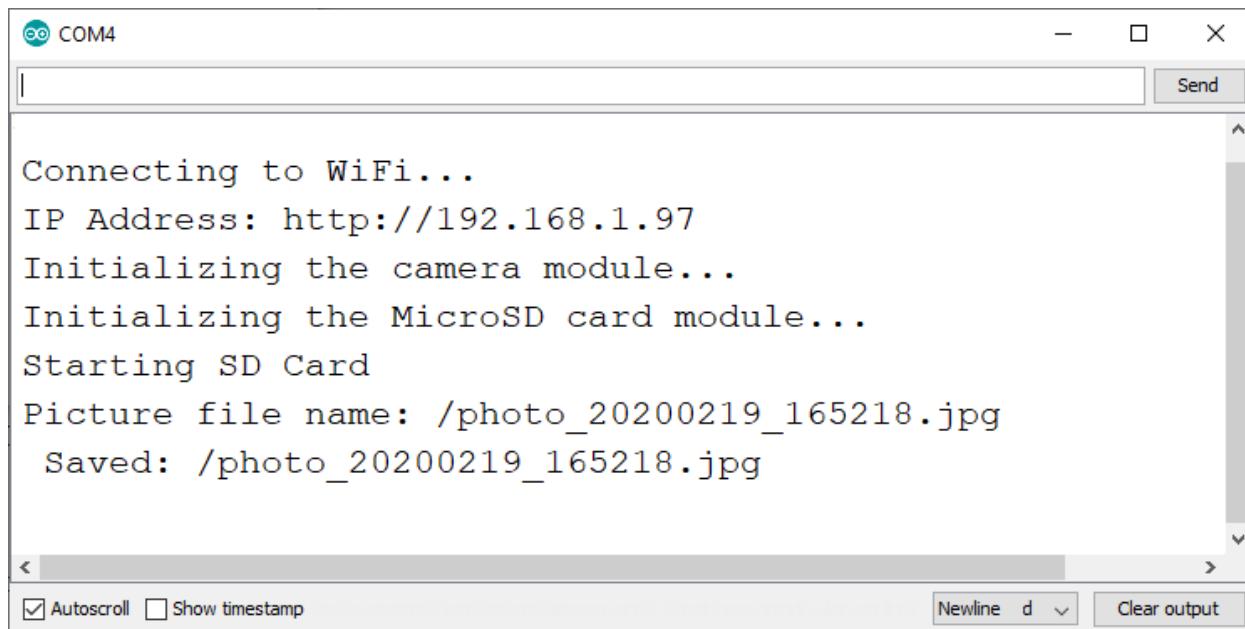
```

void deleteFile(fs::FS &fs, const char * path){
    Serial.printf("Deleting file: %s\n", path);
    if(fs.remove(path)){
        Serial.println("File deleted");
        listDirectory(SD_MMC);
    }
    else {
        Serial.println("Delete failed");
    }
}

```

Demonstration

After making any necessary changes, upload the code to your ESP32-CAM board. After uploading, open the Serial Monitor at a baud rate of 115200 and press the on-board RESET button. It should connect to Wi-Fi, initialize the camera, and microSD card and print the ESP32 IP address. The **ESP32-CAM should have a microSD card inserted.**

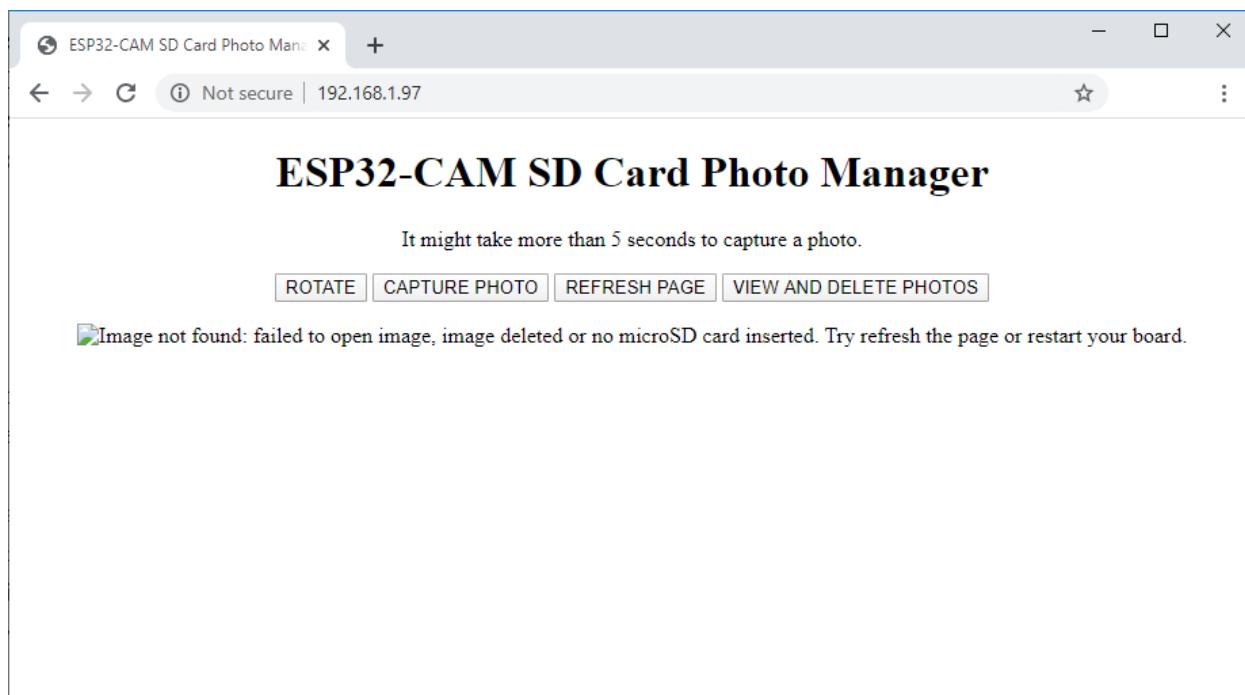


The screenshot shows a terminal window titled "COM4". The text output is as follows:

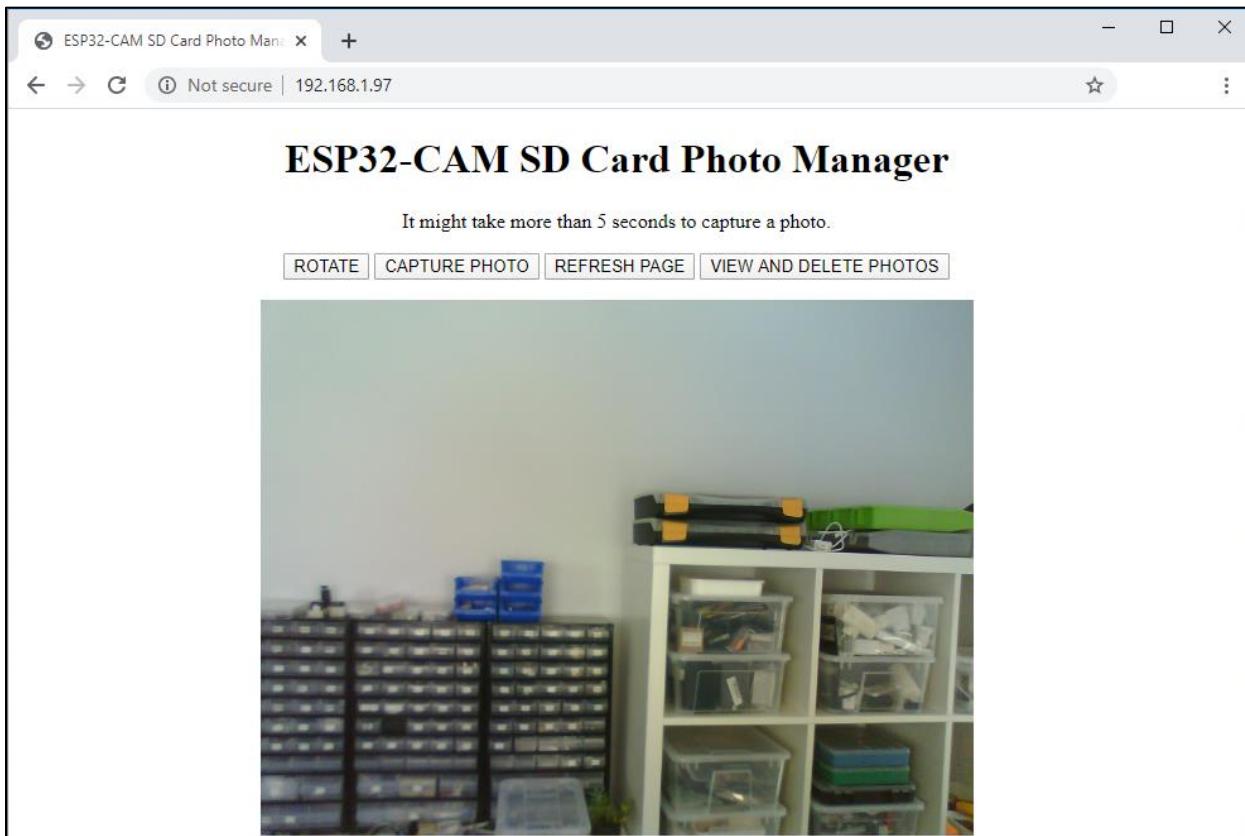
```
Connecting to WiFi...
IP Address: http://192.168.1.97
Initializing the camera module...
Initializing the MicroSD card module...
Starting SD Card
Picture file name: /photo_20200219_165218.jpg
Saved: /photo_20200219_165218.jpg
```

At the bottom of the terminal window, there are checkboxes for "Autoscroll" and "Show timestamp", and buttons for "Newline", "Clear output", and a dropdown menu.

Open a browser, and type the ESP32 IP address. This is how the web server should look like if you don't have any photos saved on the microSD card.



Press the **CAPTURE PHOTO** button to capture a new photo. Wait approximately 5 seconds and press the **REFRESH PAGE** button – you should see a new photo. You can press the **ROTATE PHOTO** button if needed.

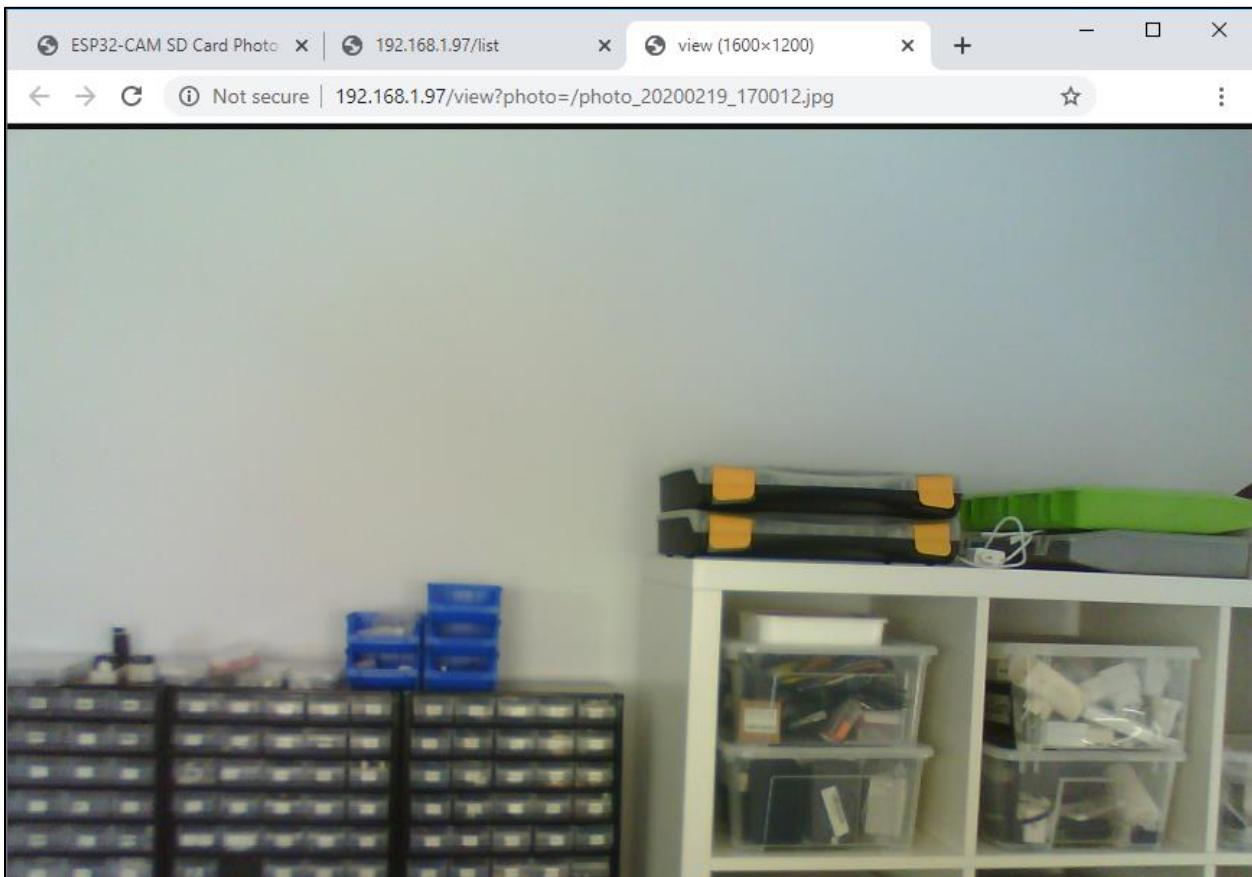


Take several photos to test this project. Then, click the **VIEW AND DELETE PHOTOS** button. You'll be redirected to a page with a list that shows all the photos saved on the microSD card.

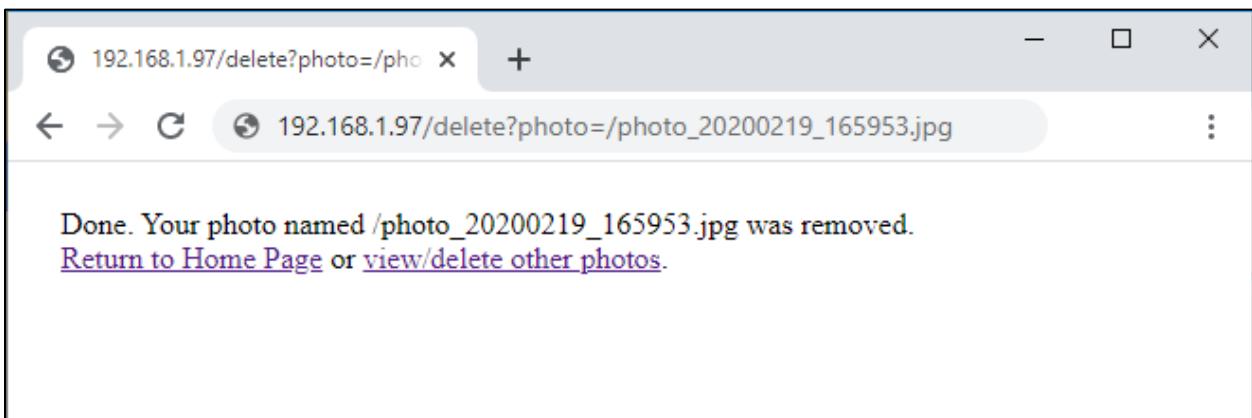
A screenshot of a web browser window titled "ESP32-CAM SD Card Photo Manager" with the URL "192.168.1.97/list". The main content area has a heading "ESP32-CAM View and Delete Photos" and a link "Return to Home Page". Below this is a table with two columns: "Actions" and "Filename". The "Actions" column contains "View" and "Delete" buttons for each file. The "Filename" column lists eight image files: /photo_20200219_170012.jpg, /photo_20200219_165953.jpg, /photo_20200219_165923.jpg, /photo_20200219_165905.jpg, /photo_20200219_165859.jpg, /photo_20200219_165607.jpg, and /photo_20200219_165541.jpg.

Next to each photo filename, there are two buttons – one to view the photo and another to delete the photo.

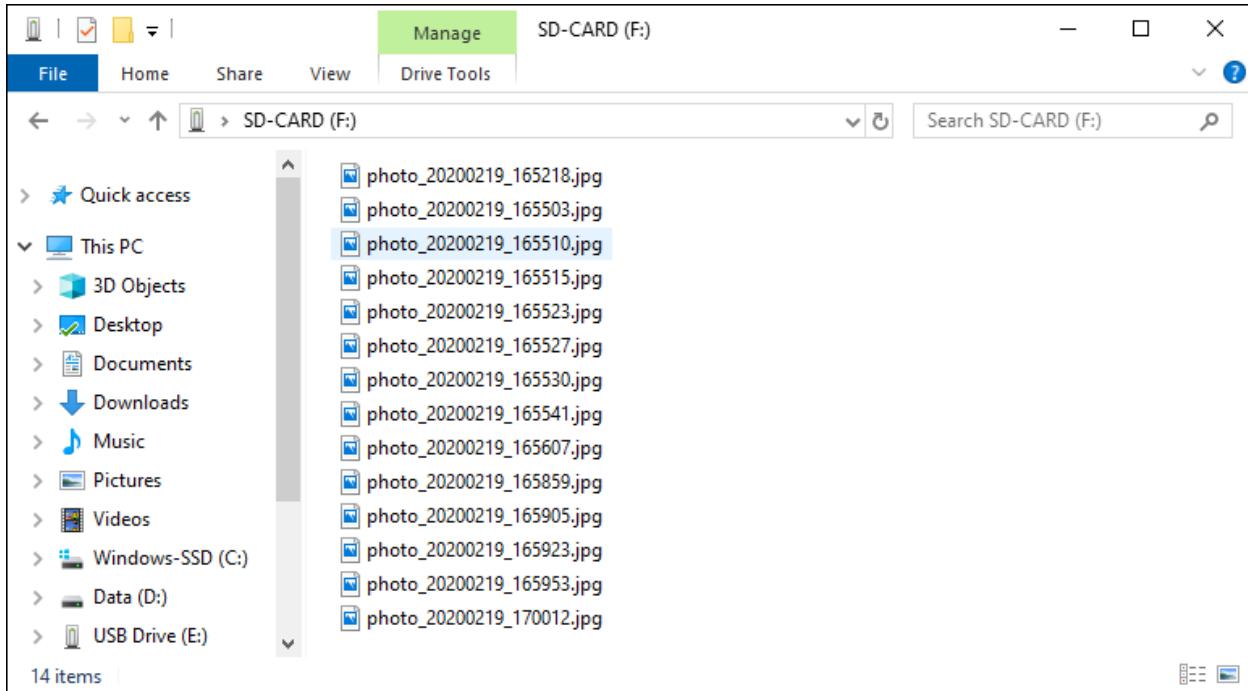
Click on one of the **View** buttons. A new tab opens with the photo. You can download the photo, if needed.



You can click the **Delete** button to delete any photos.



Finally, you can also insert the microSD card in your computer to copy and save all the photos.



Wrapping Up

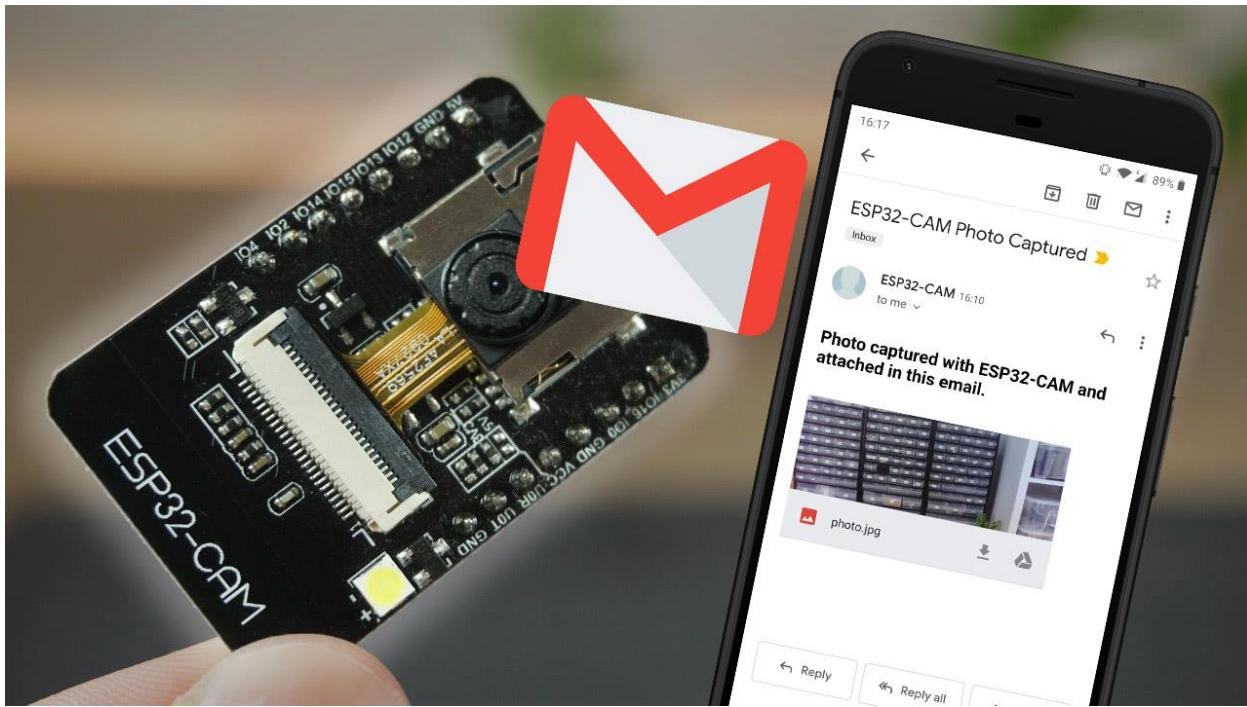
This project allows you to take photos with the ESP32-CAM by accessing a Web Server and manage all the photos saved on the microSD card. Each photo is saved with date and time, so that you know when each photo was taken.

We hope you found this example useful and you can modify it to use in your own projects.

MODULE 3

**Take Photos and Send
Notifications**

Unit 1: Send Photos via Email



In this Unit, you'll learn how to send captured photos from the ESP32 camera to your email account. We'll show you a simple example that takes a photo when the ESP32 boots and sends it via email. The last photo taken is temporarily saved in the ESP32 SPIFFS.

To make this project work, the ESP32-CAM needs to be connected to a router with access to the internet – needs to be connected to your local network.

Compatibility: this project is compatible with any ESP32 camera board with the OV2640 camera. You just need to make sure you use the right pinout for the board you're using.

ESP32 Mail Client Library

To send emails with the ESP32, we'll use the **ESP32 Mail Client** library. This library allows the ESP32 to send and receive emails with or without attachment via SMTP and IMAP servers. In this tutorial we'll use SMTP to send an email with an attachment. The attachment is a photo taken with the ESP32-CAM.

SMTP means Simple Mail Transfer Protocol and it is an internet standard for email transmission. To send emails through code, you need to know your SMTP server details. Each email provider has a different SMTP server.

Installing the ESP32 Mail Client library

Before proceeding with this tutorial, you need to install the [ESP32 Mail Client library](#). This library can be installed through the Arduino IDE Library Manager.

In your Arduino IDE go to **Sketch > Include Library > Manage Libraries...**

The Library Manager should open. Search for **ESP32 Mail Client** by Mobitz and install the library as shown below.



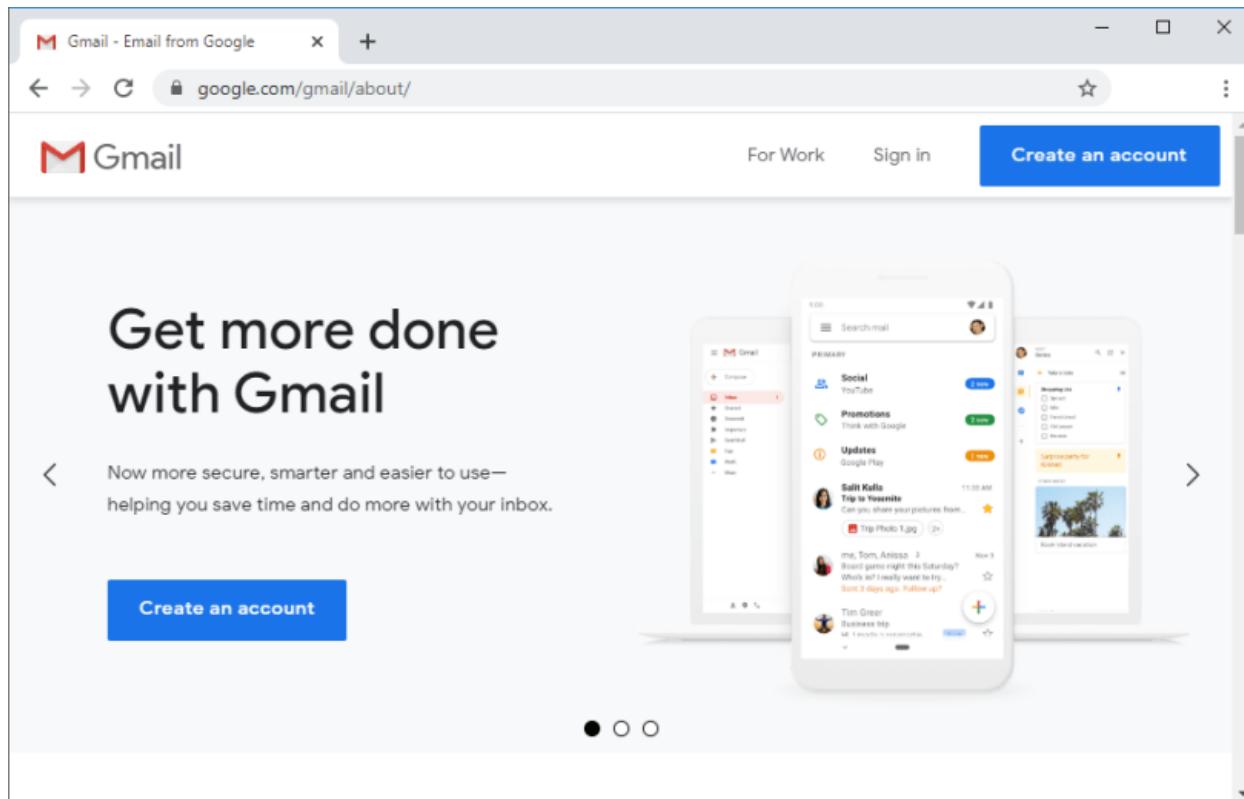
Sender Email (New Account)

We recommend creating a new email account to send the emails to your main personal email address. **Do not use your main personal email to send emails via ESP32.** If something goes wrong in your code or if by mistake you make too many requests, you can be banned or have your account temporary disabled.

We'll use a newly created Gmail.com account to send the emails, but you can use any other email provider. The receiver email can be your personal email without any problem.

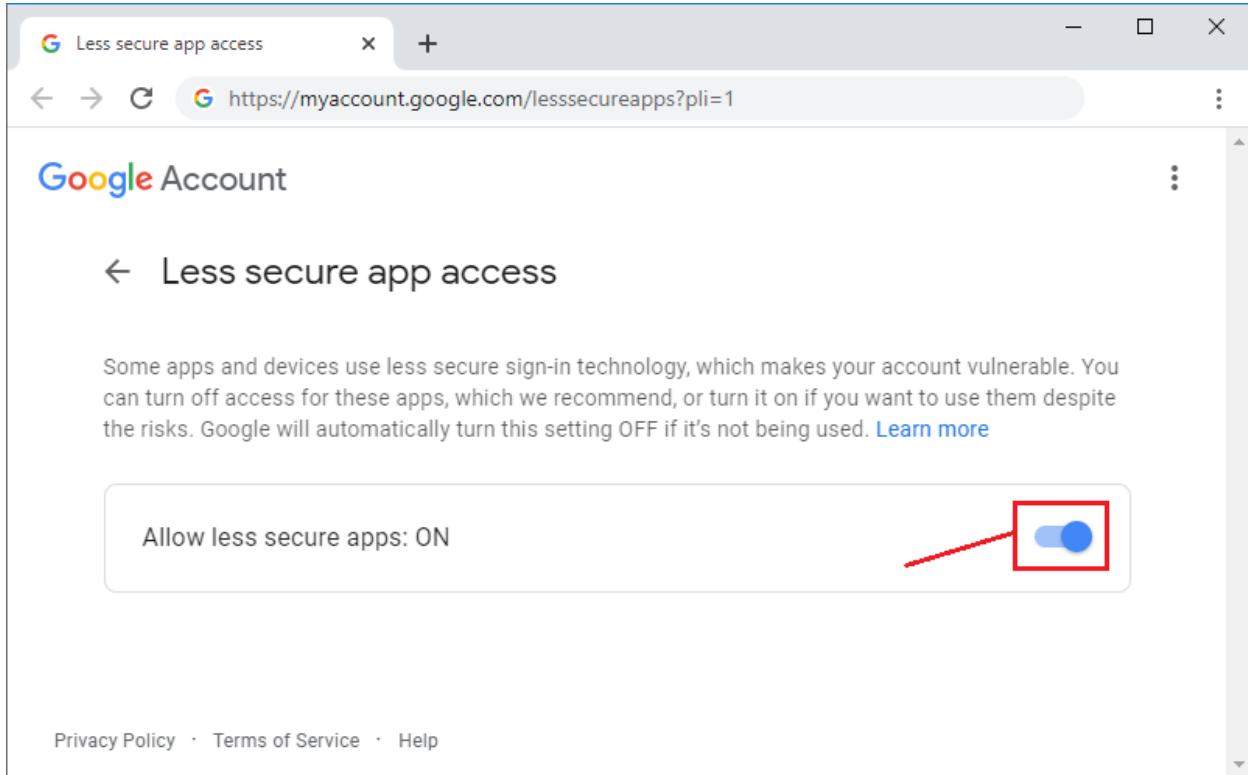
Create a Sender Email Account

Create a new email account for sending emails with the ESP32. If you want to use a Gmail account, [go to this link](#) to create a new one.



Allow less secure apps

Allow less secure apps to get access to this new Gmail account, so that you're able to send emails. You can [open this link](#) to go to that menu.



Gmail SMTP Server Settings

If you're using a Gmail account, these are the SMTP Server details:

- SMTP Server: **smtp.gmail.com**
- SMTP username: Complete Gmail address
- SMTP password: Your Gmail password
- SMTP port (TLS): **587**
- SMTP port (SSL): **465**
- SMTP TLS/SSL required: **yes**

Outlook SMTP Server Settings

For Outlook accounts, these are the SMTP Server settings:

- SMTP Server: **smtp.office365.com**
- SMTP Username: Complete Outlook email address
- SMTP Password: Your Outlook password
- SMTP Port: **587**
- SMTP TLS/SSL Required: **Yes**

Live or Hotmail SMTP Server Settings

For Live or Hotmail accounts, these are the SMTP Server settings:

- SMTP Server: **smtp.live.com**
- SMTP Username: Complete Live/Hotmail email address
- SMTP Password: Your Windows Live Hotmail password
- SMTP Port: **587**
- SMTP TLS/SSL Required: **Yes**

If you're using another email provider, you need to search for its SMTP Server settings.

Now, you have everything ready to start sending emails with photos using your ESP32-CAM.

Code

The following code takes a photo when the ESP32-CAM first boots and sends it to your email account. Before uploading the code, make sure you insert your sender email settings as well as your recipient email.

CODE

https://github.com/RuiSantosdotme/ESP32-CAM-eBook/blob/master/Code/Module_3/Send_Photos_Email/Send_Photos_Email.ino

```
#include "esp_camera.h"
#include "SPI.h"
#include "driver/rtc_io.h"
#include "ESP32_MailClient.h"
#include <FS.h>
#include <SPIFFS.h>
#include <WiFi.h>

//REPLACE WITH YOUR NETWORK CREDENTIALS
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

// To send Email using Gmail use port 465 (SSL) and SMTP Server
smtp.gmail.com
// YOU MUST ENABLE less secure app option
https://myaccount.google.com/lesssecureapps?pli=1
#define emailSenderAccount      "EXAMPLE_EMAIL@gmail.com"
#define emailSenderPassword     "YOUR_EXAMPLE_EMAIL_PASSWORD"
#define smtpServer              "smtp.gmail.com"
#define smtpServerPort          465
#define emailSubject            "ESP32-CAM Photo Captured"
#define emailRecipient          "YOUR_EMAIL_RECIPIENT@example.com"

// This project was tested with the AI Thinker Model
#define CAMERA_MODEL_AI_THINKER

#if defined(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
#else
#error "Camera model not selected"
#endif

// The Email Sending data object contains config and data to send
SMTPData smtpData;

// Photo File Name to save in SPIFFS
#define FILE_PHOTO "/photo.jpg"

void setup() {
    WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0); // disable brownout detector

    Serial.begin(115200);
    Serial.println();

    // Connect to Wi-Fi
    WiFi.begin(ssid, password);
    Serial.print("Connecting to WiFi...");
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println();

    if (!SPIFFS.begin(true)) {
        Serial.println("An Error has occurred while mounting SPIFFS");
        ESP.restart();
    }
    else {
        delay(500);
        Serial.println("SPIFFS mounted successfully");
    }

    // Print ESP32 Local IP Address
    Serial.print("IP Address: http://");
    Serial.println(WiFi.localIP());

    camera_config_t config;
    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;

if(psramFound()) {
    config.frame_size = FRAMESIZE_UXGA;
    config.jpeg_quality = 10;
    config.fb_count = 2;
} else {
    config.frame_size = FRAMESIZE_SVGA;
    config.jpeg_quality = 12;
    config.fb_count = 1;
}

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

capturePhotoSaveSpiffs();
sendPhoto();
}

void loop() {

}

// Check if photo capture was successful
bool checkPhoto( fs::FS &fs ) {
    File f_pic = fs.open( FILE_PHOTO );
    unsigned int pic_sz = f_pic.size();
    return ( pic_sz > 100 );
}

// Capture Photo and Save it to SPIFFS
void capturePhotoSaveSpiffs( void ) {
    camera_fb_t * fb = NULL; // pointer
    bool ok = 0; // Boolean indicating if the picture was captured correctly

    do {
        // Take a photo with the camera
        Serial.println("Taking a photo...");

```

```

fb = esp_camera_fb_get();
if (!fb) {
    Serial.println("Camera capture failed");
    return;
}

// Photo file name
Serial.printf("Picture file name: %s\n", FILE_PHOTO);
File file = SPIFFS.open(FILE_PHOTO, FILE_WRITE);

// Insert the data in the photo file
if (!file) {
    Serial.println("Failed to open file in writing mode");
}
else {
    file.write(fb->buf, fb->len); // payload (image), payload length
    Serial.print("The picture has been saved in ");
    Serial.print(FILE_PHOTO);
    Serial.print(" - Size: ");
    Serial.print(file.size());
    Serial.println(" bytes");
}
// Close the file
file.close();
esp_camera_fb_return(fb);

// check if file has been correctly saved in SPIFFS
ok = checkPhoto(SPIFFS);
} while ( !ok );
}

void sendPhoto( void ) {
// Preparing email
Serial.println("Sending email...");
// Set the SMTP Server Email host, port, account and password
smtpData.setLogin(smtpServer, smtpServerPort, emailSenderAccount,
emailSenderPassword);

// Set the sender name and Email
smtpData.setSender("ESP32-CAM", emailSenderAccount);

// Set Email priority or importance High, Normal, Low or 1 to 5 (1 is
highest)
smtpData.setPriority("High");

// Set the subject
smtpData.setSubject(emailSubject);

// Set the email message in HTML format
smtpData.setMessage("<h2>Photo captured with ESP32-CAM and attached in
this email.</h2>", true);
// Set the email message in text format
//smtpData.setMessage("Photo captured with ESP32-CAM and attached in
this email.", false);
}

```

```

// Add recipients, can add more than one recipient
smtpData.addRecipient(emailRecipient);
//smtpData.addRecipient(emailRecipient2);

// Add attach files from SPIFFS
smtpData.addAttachFile(FILE_PHOTO, "image/jpg");
// Set the storage type to attach files in your email (SPIFFS)
smtpData.setFileStorageType(MailClientStorageType::SPIFFS);

smtpData.setSendCallback(sendCallback);

// Start sending Email, can be set callback function to track the
status
if (!MailClient.sendMail(smtpData))
    Serial.println("Error sending Email,"+MailClient.smtpErrorReason());

// Clear all data from Email object to free memory
smtpData.empty();
SD.end();
}

// Callback function to get the Email sending status
void sendCallback(SendStatus msg) {
    // Print the current status
    Serial.println(msg.info());
}

```

How the Code Works

Continue reading to learn how the code works, or skip to the Demonstration section. Don't forget to insert in the code your network credentials and email settings. Also, if you're using a camera model other than an ESP32-CAM AI-Thinker, don't forget to change the pin assignment.

We've already covered how to initialize and take photos with the camera in previous Units, so we'll just take a look at the relevant parts for this project.

Importing Libraries

Import the required libraries. The `ESP32_MailClient.h` is used to send emails, the `FS.h` and `SPIFFS.h` is used to access and save files to SPIFFS and the `WiFi.h` library is used to initialize Wi-Fi and connect your ESP32-CAM to your local network.

```
#include "esp_camera.h"
#include "SPI.h"
#include "driver/rtc_io.h"
#include "ESP32_MailClient.h"
#include <FS.h>
#include <SPIFFS.h>
#include <WiFi.h>
```

Network Credentials

Insert your network credentials in the following variables

```
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
```

Email Settings

Type the email sender account on the `emailSenderAccount` variable and its password on the `emailSenderPassword` variable.

```
#define emailSenderAccount "example_sender_account@gmail.com"
#define emailSenderPassword "email_sender_password"
```

Insert the recipient's email. This is the email that will receive the emails sent by the ESP32:

```
#define emailRecipient "your_email_recipient@gmail.com"
```

Insert your email provider SMTP settings on the following lines. We're using the settings for a Gmail account. If you're using a different email provider, replace with the corresponding SMTP settings.

```
#define smtpServer "smtp.gmail.com"
#define smtpServerPort 465
```

Write the email subject on the `emailSubject` variable.

```
#define emailSubject "ESP32-CAM Photo Captured"
```

Create an `SMTPData` object called `smtpData` that contains the data to send via email and all the other configurations.

```
SMTPData smtpData;
```

The photo taken with the ESP32 camera will be temporarily saved in SPIFFS under the name `photo.jpg`.

```
#define FILE_PHOTO "/photo.jpg"
```

setup()

In the `setup()`, connect the ESP32 to Wi-Fi.

```
WiFi.begin(ssid, password);
Serial.print("Connecting to WiFi...");
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}
Serial.println();
```

Initialize SPIFFS (SPI Flash File System) to save the last photo taken with the ESP32-CAM.

```
if (!SPIFFS.begin(true)) {
    Serial.println("An Error has occurred while mounting SPIFFS");
    ESP.restart();
}
else {
    delay(500);
    Serial.println("SPIFFS mounted successfully");
}
```

After initializing the camera, call the `capturePhotoSaveSpiffs()` and the `sendPhoto()` functions. These functions are defined at the end of the code.

capturePhotoSaveSpiffs() function

The `capturePhotoSaveSpiffs()` function captures a photo and saves it in the ESP32 SPIFFS. In the following lines, you take a photo and save it in the framebuffer `fb`:

```
fb = esp_camera_fb_get();
if (!fb) {
    Serial.println("Camera capture failed");
    return;
}
```

Then, create a new file in SPIFFS where the photo will be saved.

```
Serial.printf("Picture file name: %s\n", FILE_PHOTO);
File file = SPIFFS.open(FILE_PHOTO, FILE_WRITE);
```

Check if the file was successfully created. If not, print an error message.

```
if (!file) {
    Serial.println("Failed to open file in writing mode");
}
```

If a new file was successfully created, “copy” the image from the buffer to that newly created file.

```
file.write(fb->buf, fb->len); // payload (image), payload length
Serial.print("The picture has been saved in ");
Serial.print(FILE_PHOTO);
Serial.print(" - Size: ");
Serial.print(file.size());
Serial.println(" bytes");
}
```

Close the file and clear the buffer for future use.

```
file.close();
esp_camera_fb_return(fb);
```

Finally, check whether the photo was successfully taken and saved. We can do that by checking the photo file size with the `checkPhoto()` function.

```
ok = checkPhoto(SPIFFS);
```

The `checkPhoto()` function, checks the file picture size. If the picture size is bigger than 100 bytes, it's almost certain the photo was taken and saved successfully. In that case it returns 1.

```
bool checkPhoto( fs::FS &fs ) {
    File f_pic = fs.open( FILE_PHOTO );
    unsigned int pic_sz = f_pic.size();
    return ( pic_sz > 100 );
}
```

So, the `ok` variable is equal to 1.

```
ok = checkPhoto(SPIFFS);
```

In case the photo size is less than 100 bytes, the `ok` variable will continue to be 0, and it will keep trying to take a new photo and save it.

sendPhoto() function

After having the photo successfully saved in SPIFFS, we'll send it via email by calling the `sendPhoto()` function. Let's take a look at that function.

```
void sendPhoto(void) {
```

The following line sets the SMTP Server host, SMTP port, account email address and password used to login:

```
smtpData.setLogin(smtpServer, smtpServerPort, emailSenderAccount,
emailSenderPassword);
```

Set the sender name and sender email. In this case, we're setting the sender name to ESP32-CAM.

```
smtpData.setSender("ESP32-CAM", emailSenderAccount);
```

Set the email priority.

```
smtpData.setPriority("High");
```

Set the email subject.

```
smtpData.setSubject(emailSubject);
```

The following line sets the message. You can send an HTML text or raw text. In this case, we're sending a message with some HTML.

```
smtpData.setMessage("<h2>Photo captured with ESP32-CAM and attached in  
this email.</h2>", true);
```

When sending a message in HTML format, you should pass `true` as a second parameter to the `setMessage()` method. If you want to send raw text, set `false`.

```
//smtpData.setMessage("Photo captured with ESP32-CAM and attached in  
this email.", false);
```

Finally, set the recipient email. This is the email that will receive the messages from the ESP32.

```
smtpData.addRecipient(emailRecipient);
```

Then, to attach a file, you just need to call the `addAtatachFile()` on the `smtpData` object and pass as argument the file path.

```
smtpData.addAttachFile(FILE_PHOTO, "image/jpg");
```

Finally, you need to set where your files are saved (SPIFFS or SD card). We're using SPIFFS:

```
smtpData.setFileStorageType(MailClientStorageType::SPIFFS);
```

Set a callback function that will be called upon sending an email.

```
smtpData.setSendCallback(sendCallback);
```

The `sendCallback()` function returns whether the email was successfully sent or not.

Now that we've set all the details of the `smtpData`, we're ready to send the email.

```
if (!MailClient.sendMail(smtpData))  
    Serial.println("Error sending Email, " + MailClient.smtpErrorReason());
```

After sending the email, you can clear all the data from the `smtpData` object.

```
smtpData.empty();
```

In this example, the email is sent once when the ESP32 boots, that's why the `loop()` is empty.

```
void loop() {  
}
```

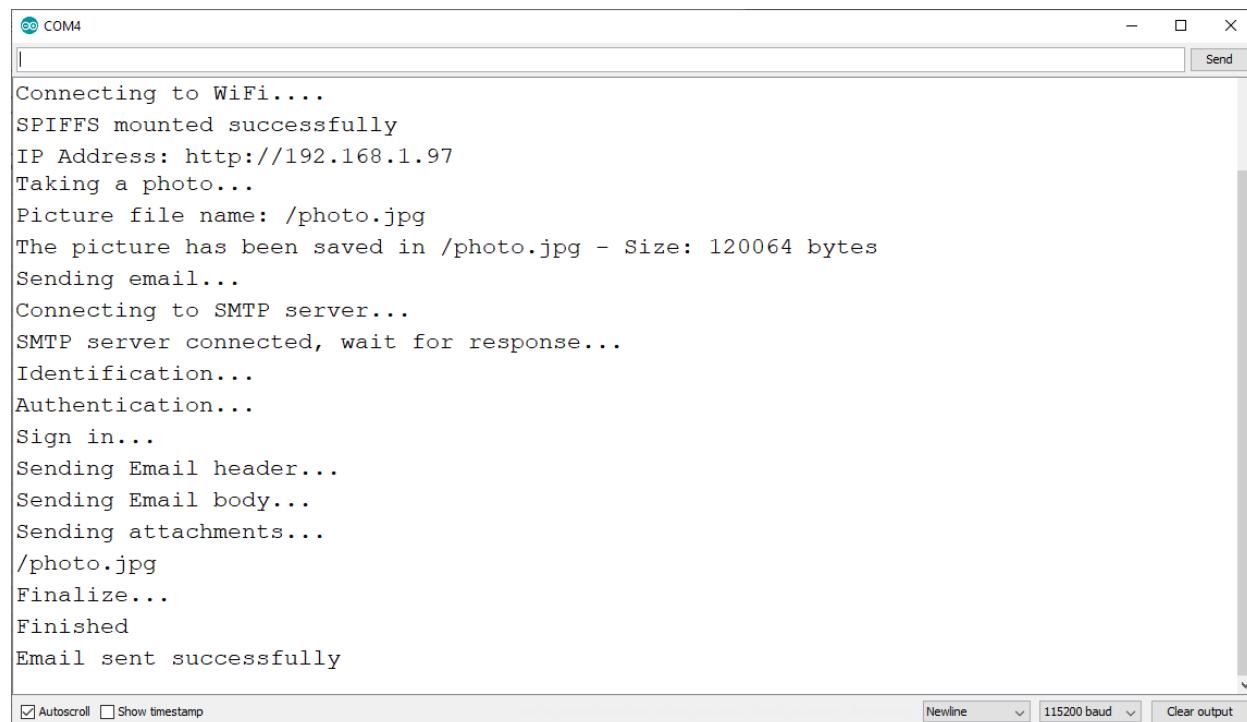
To send a new email. You just need to reset your board (press the on-board RESET button), so that it restarts and runs the code again.

Demonstration

After making the necessary changes to the code: camera pinout, sender's email address, sender's email password, recipient's email address and network credentials, you can upload the code to your board.

After uploading, open the Serial Monitor and press the ESP32-CAM RESET button.

The ESP32 should connect to Wi-Fi, take a photo, save it in SPIFFS, connect to the SMTP server and send the email as shown below.

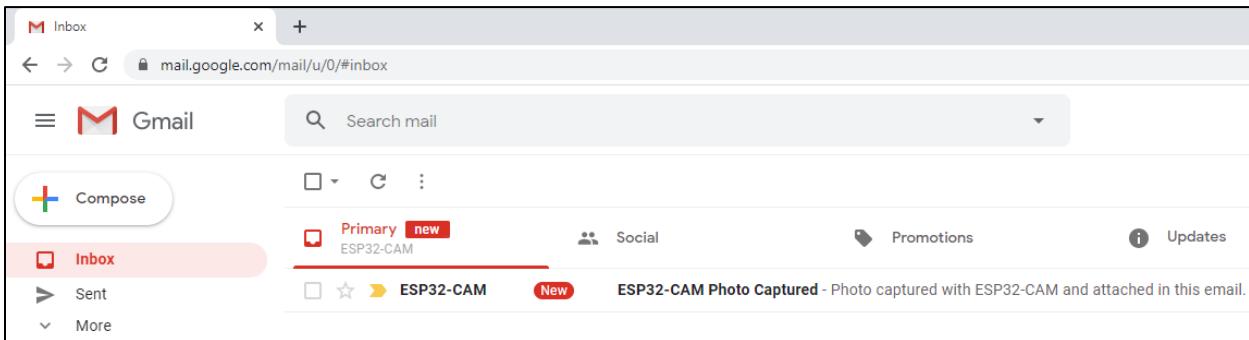


The screenshot shows a Windows-style terminal window titled "COM4". The text output is as follows:

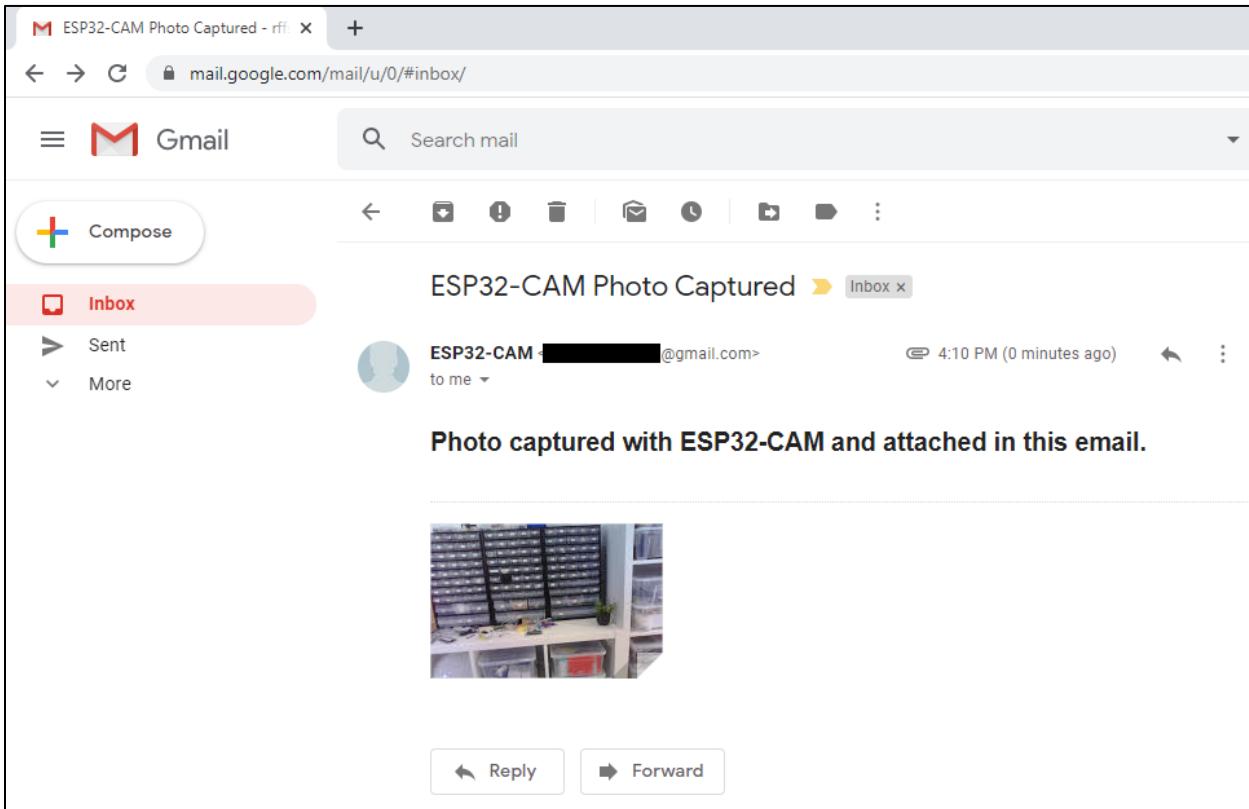
```
Connecting to WiFi....  
SPIFFS mounted successfully  
IP Address: http://192.168.1.97  
Taking a photo...  
Picture file name: /photo.jpg  
The picture has been saved in /photo.jpg - Size: 120064 bytes  
Sending email...  
Connecting to SMTP server...  
SMTP server connected, wait for response...  
Identification...  
Authentication...  
Sign in...  
Sending Email header...  
Sending Email body...  
Sending attachments...  
/photo.jpg  
Finalize...  
Finished  
Email sent successfully
```

At the bottom of the window, there are checkboxes for "Autoscroll" and "Show timestamp", and dropdown menus for "Newline", "115200 baud", and "Clear output".

After a few seconds, you should have a new email from the ESP32-CAM in your inbox. As you can see in the image below, the sender's email name is "ESP32-CAM" as we've defined in the code and the subject "ESP32-CAM Photo Captured".



Open the email and you should see a photo captured by the ESP32-CAM.



You can open or download the photo to see it in full size.



Wrapping Up

In this Unit you've learned how to send emails with photos taken with the ESP32-CAM. The example presented is as simple as possible: it takes a photo and sends it via email when the ESP32-CAM first boots. To send another email, you need to reset the board.

This project doesn't have a practical application, but it is useful to understand how to send an email with an attachment. After this, it should be fairly easy to include this feature in your own ESP32-CAM projects.

Continue to the next Unit to send a photo to your email when motion is detected.

Unit 2: Motion Detector with Photo Capture and Email Notifications

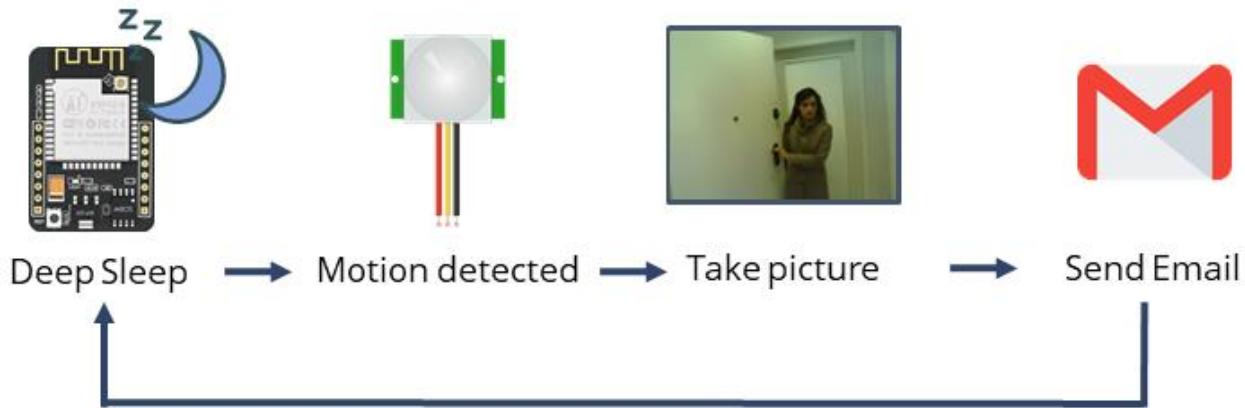


In this project, we're going to make a motion sensor detector with photo capture and email notifications using an ESP32-CAM. The ESP32-CAM is in deep sleep mode and when the PIR sensor detects motion, the ESP32-CAM wakes up, takes a photo and sends it via email. Then, it goes back to sleep until motion is detected.

Compatibility: in this project we'll wire a PIR motion sensor to the ESP32-CAM, so your camera model needs to have at least one GPIO exposed. That GPIO should be an RTC GPIO so that it is able to wake up the ESP32. If you don't know what RTC GPIOs are, [check this article about the ESP32 pins](#). The ESP32 RTC GPIOs are: 0, 2, 4, 12, 13, 14, 15, 25, 26, 27, 32, 33 ,34, 35, 36 and 39. This project doesn't require a microSD card.

Project Overview

The following diagram shows a high-level overview on how the project works.



- The ESP32-CAM is in deep sleep mode with external wake up enabled;
- When motion is detected, the PIR motion sensor sends a signal to wake up the ESP32;
- The ESP32-CAM takes a photo, saves it temporarily on SPIFFS and sends an email alert with the photo captured;
- It goes back to deep sleep mode until a new signal from the PIR motion sensor is received.

Prerequisites

Before proceeding with this project, you need to:

- Install the `ESP32_MailClient` Library;
- Set up a sender email address (this **shouldn't** be your personal email).

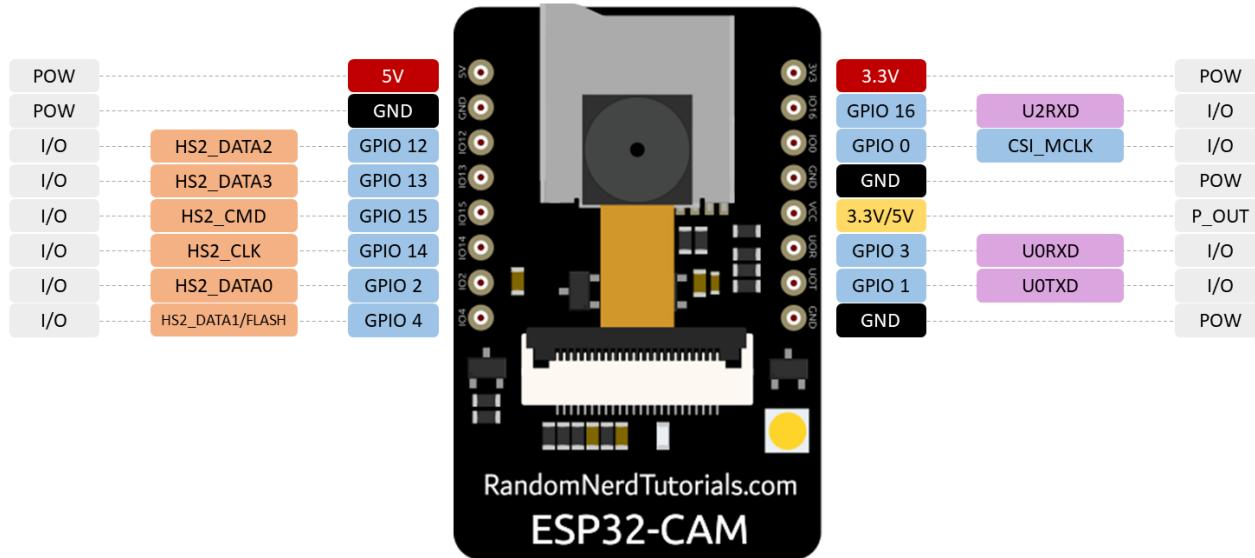
We cover these topics in the previous Unit. So, follow that Unit first if you haven't already. After that, you can proceed with this project.

Deep Sleep with External Wake Up

In this project, the ESP32-CAM will be in deep sleep mode and it will wake up with an external wake-up source (the PIR motion sensor sends a HIGH signal when it detects motion).

Only RCT GPIOs can be used as a wake up source. The ESP32 RTC GPIOs are: 0, 2, 4, 12, 13, 14, 15, 25, 26, 27, 32, 33, 34, 35, 36 and 39.

Let's recap the ESP32-CAM pinout.



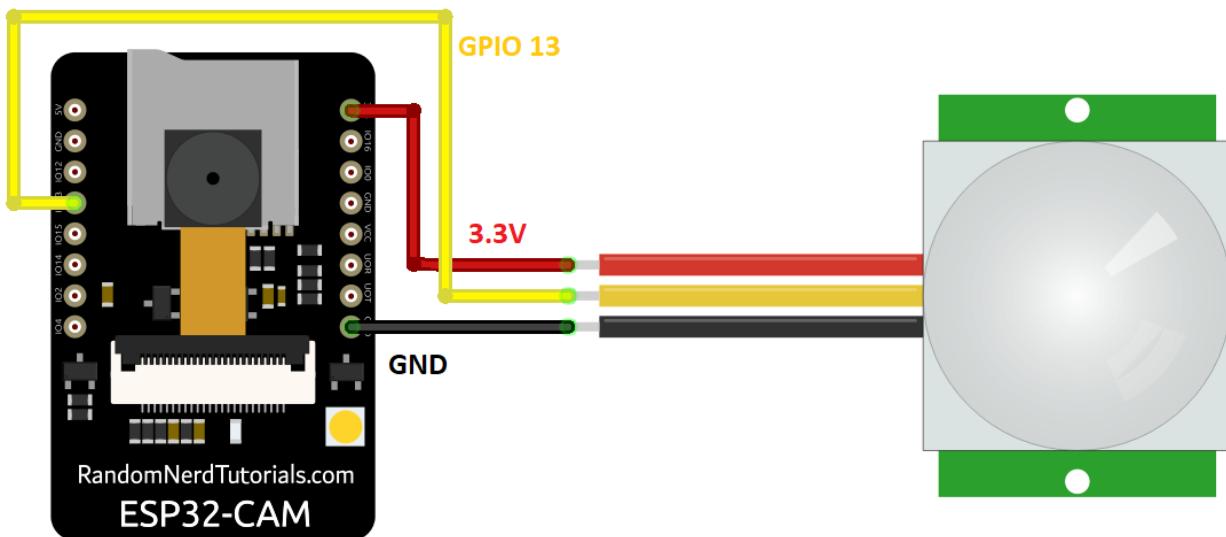
Because this project doesn't use the microSD card, we have the following GPIOs available in the ESP32-CAM AI-Thinker: **GPIO 4**, **GPIO 2**, **GPIO 14**, **GPIO 15**, **GPIO 13** and **GPIO 12**. So, you can use any of these GPIOs as an external wake-up source (all of them are RTC GPIOs). We'll connect the PIR motion sensor to GPIO 13.

Parts Required

For this project you need the following parts:

- [ESP32-CAM](#)
- [PIR motion sensor](#)
- [Jumper wires](#)
- [Breadboard](#) (optional)

Assemble the circuit as shown in the following schematic diagram with the PIR motion sensor data pin connected to GPIO 13.



Note: some PIR motion sensors operate at 5V. So, you need to provide 5V to the VCC pin instead of 3.3V. Please check the datasheet for the PIR motion sensor you're using. This example was tested using a [AM312 mini PIR](#).

Code

This code is very similar with the previous project, but it adds some lines of code to put the ESP32-CAM in deep sleep and wake it up when motion is detected.

Like in the previous project, you need to insert your network credentials as well as the sender email settings and the recipient's email.

CODE

https://github.com/RuiSantosdotme/ESP32-CAM-eBook/blob/master/Code/Module_3/Motion_Detector_Photo_Email/Motion_Detector_Photo_Email.ino

```
#include "esp_camera.h"
#include "SPI.h"
#include "driver/rtc_io.h"
#include "ESP32_MailClient.h"
#include <FS.h>
#include <SPIFFS.h>
#include <WiFi.h>

//REPLACE WITH YOUR NETWORK CREDENTIALS
const char* ssid = "REPLACE_WIT_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

// To send Email using Gmail use port 465 (SSL) and SMTP Server
smtp.gmail.com
// YOU MUST ENABLE less secure app option
https://myaccount.google.com/lesssecureapps?pli=1
#define emailSenderAccount      "EXAMPLE_EMAIL@gmail.com"
#define emailSenderPassword     "YOUR_EXAMPLE_EMAIL_PASSWORD"
#define smtpServer              "smtp.gmail.com"
#define smtpServerPort          465
#define emailSubject            "[WARNING] ESP32-CAM Motion Detected"
#define emailRecipient          "YOUR_EMAIL_RECIPIENT@example.com"

// This project was tested with the AI Thinker Model
#define CAMERA_MODEL_AI_THINKER

#if defined(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
#else
    #error "Camera model not selected"
#endif

// The Email Sending data object contains config and data to send
SMTPData smtpData;

// Photo File Name to save in SPIFFS
#define FILE_PHOTO "/photo.jpg"

void setup() {
    WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0); // disable brownout detector

    Serial.begin(115200);
    Serial.println();

    // Connect to Wi-Fi
    WiFi.begin(ssid, password);
    Serial.print("Connecting to WiFi...");
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println();

    if (!SPIFFS.begin(true)) {
        Serial.println("An Error has occurred while mounting SPIFFS");
        ESP.restart();
    }
    else {
        delay(500);
        Serial.println("SPIFFS mounted successfully");
    }

    // Print ESP32 Local IP Address
    Serial.print("IP Address: http://");
    Serial.println(WiFi.localIP());

    camera_config_t config;
    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;

if(psramFound()) {
    config.frame_size = FRAMESIZE_UXGA;
    config.jpeg_quality = 10;
    config.fb_count = 2;
} else {
    config.frame_size = FRAMESIZE_SVGA;
    config.jpeg_quality = 12;
    config.fb_count = 1;
}

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

capturePhotoSaveSpiffs();
sendPhoto();

esp_sleep_enable_ext0_wakeup(GPIO_NUM_13, 1);
Serial.println("Going to sleep now");
WiFi.disconnect();
delay(1000);
esp_deep_sleep_start();
}

void loop() {

}

// Check if photo capture was successful
bool checkPhoto( fs::FS &fs ) {
    File f_pic = fs.open( FILE_PHOTO );
    unsigned int pic_sz = f_pic.size();
    return ( pic_sz > 100 );
}

```

```

// Capture Photo and Save it to SPIFFS
void capturePhotoSaveSpiffs( void ) {
    camera_fb_t * fb = NULL; // pointer
    bool ok = 0; // Boolean indicating if the picture was captured correctly

    do {
        // Take a photo with the camera
        Serial.println("Taking a photo...");

        fb = esp_camera_fb_get();
        if (!fb) {
            Serial.println("Camera capture failed");
            return;
        }

        // Photo file name
        Serial.printf("Picture file name: %s\n", FILE_PHOTO);
        File file = SPIFFS.open(FILE_PHOTO, FILE_WRITE);

        // Insert the data in the photo file
        if (!file) {
            Serial.println("Failed to open file in writing mode");
        }
        else {
            file.write(fb->buf, fb->len); // payload (image), payload length
            Serial.print("The picture has been saved in ");
            Serial.print(FILE_PHOTO);
            Serial.print(" - Size: ");
            Serial.print(file.size());
            Serial.println(" bytes");
        }
        // Close the file
        file.close();
        esp_camera_fb_return(fb);

        // check if file has been correctly saved in SPIFFS
        ok = checkPhoto(SPIFFS);
    } while ( !ok );
}

boolean sendPhoto( void ) {
    // Preparing email
    Serial.println("Sending email...");
    // Set the SMTP Server Email host, port, account and password
    smtpData.setLogin(smtpServer, smtpServerPort, emailSenderAccount,
emailSenderPassword);

    // Set the sender name and Email
    smtpData.setSender("ESP32-CAM", emailSenderAccount);

    // Set Email priority or importance High, Normal, Low or 1 to 5 (1 is highest)
    smtpData.setPriority("High");
}

```

```

// Set the subject
smtpData.setSubject(emailSubject);

// Set the email message in HTML format
smtpData.setMessage("<h2>Motion Detected! Photo captured with ESP32-
CAM attached in this email.</h2>", true);
// Set the email message in text format
//smtpData.setMessage("Motion Detected! Photo captured with ESP32-CAM
attached in this email.", false);

// Add recipients, can add more than one recipient
smtpData.addRecipient(emailRecipient);
//smtpData.addRecipient(emailRecipient2);

// Add attach files from SPIFFS
smtpData.addAttachFile(FILE_PHOTO, "image/jpg");
// Set the storage type to attach files in your email (SPIFFS)
smtpData.setFileStorageType(MailClientStorageType::SPIFFS);

smtpData.setSendCallback(sendCallback);

// Start sending Email, can be set callback function to track the status
if (!MailClient.sendMail(smtpData))
    Serial.println("Error sending Email, " + MailClient.smtpErrorReason());

// Clear all data from Email object to free memory
smtpData.empty();

SD.end();
}

// Callback function to get the Email sending status
void sendCallback(SendStatus msg) {
    //Print the current status
    Serial.println(msg.info());
}

```

How the Code Works

This code is very similar with the previous Unit, so we'll just take a look at the relevant parts for this project.

Insert your network credentials in the following variables:

```
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
```

Insert your email settings.

```
#define emailSenderAccount      "EXAMPLE_EMAIL@gmail.com"
#define emailSenderPassword      "YOUR_EXAMPLE_EMAIL_PASSWORD"
#define smtpServer                "smtp.gmail.com"
#define smtpServerPort            465
#define emailSubject              "[WARNING] ESP32-CAM Motion Detected"
#define emailRecipient            "YOUR_EMAIL_RECIPIENT@example.com"
```

In the `setup()`, after capturing and sending a photo, enable external wake-up on GPIO 13 – the GPIO the PIR motion sensor is connected to.

```
esp_sleep_enable_ext0_wakeup(GPIO_NUM_13, 1);
```

The second argument of the `esp_sleep_enable_ext0_wakeup()` function can be 0 or 1. This represents the state of the GPIO that will trigger wake-up. We want to trigger the wake-up when the PIR sensor detects motion (when its state changes to HIGH). So, we should pass 1 as a second argument.

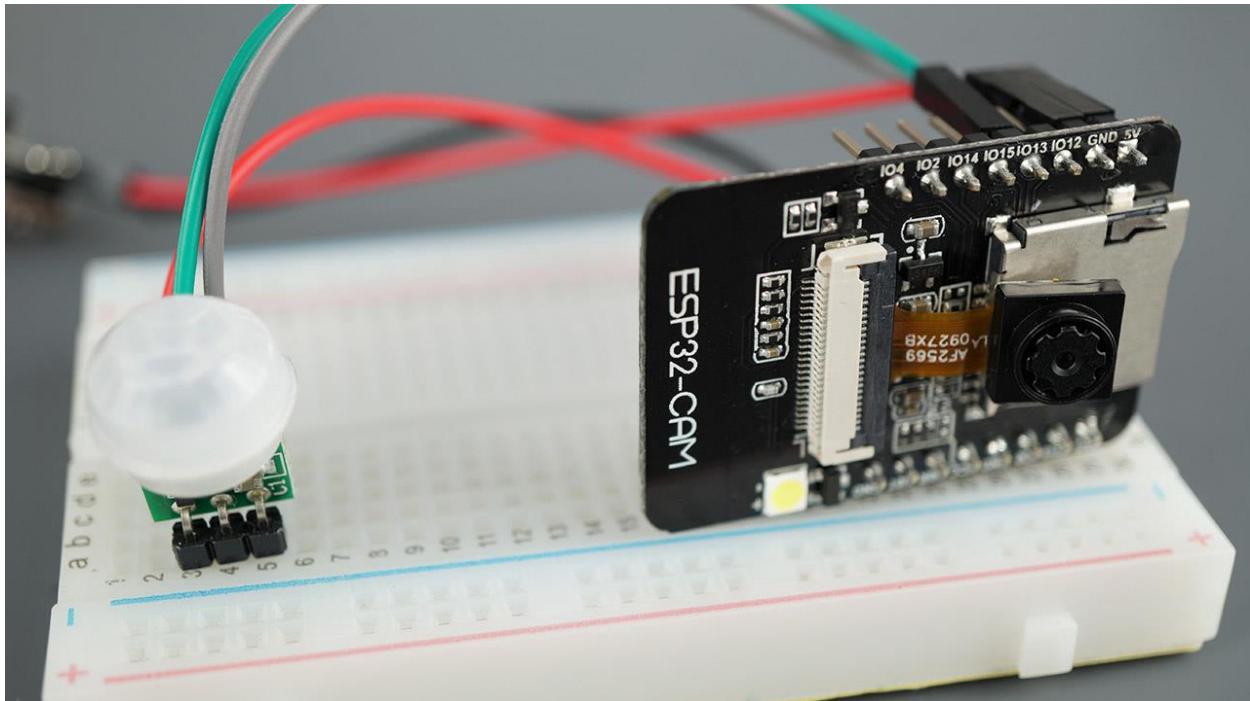
After setting the wake-up source, put the ESP32 in deep sleep mode.

```
esp_deep_sleep_start();
```

When the ESP32 wakes up, it will run the code from the start until going to sleep again.

Demonstration

After assembling the circuit and making all the necessary changes to the code, upload it to your board.



After uploading the code to your ESP32-CAM, test your setup. It will send an email when it first boots, and then it will go into deep sleep mode.

To wake up your board, the PIR sensor must detect motion. So, move your hand in front of the PIR motion sensor and check if the ESP32 wakes up, connects to Wi-Fi, takes a photo and sends it via email. On the Serial Monitor, you should receive several success messages as shown below.

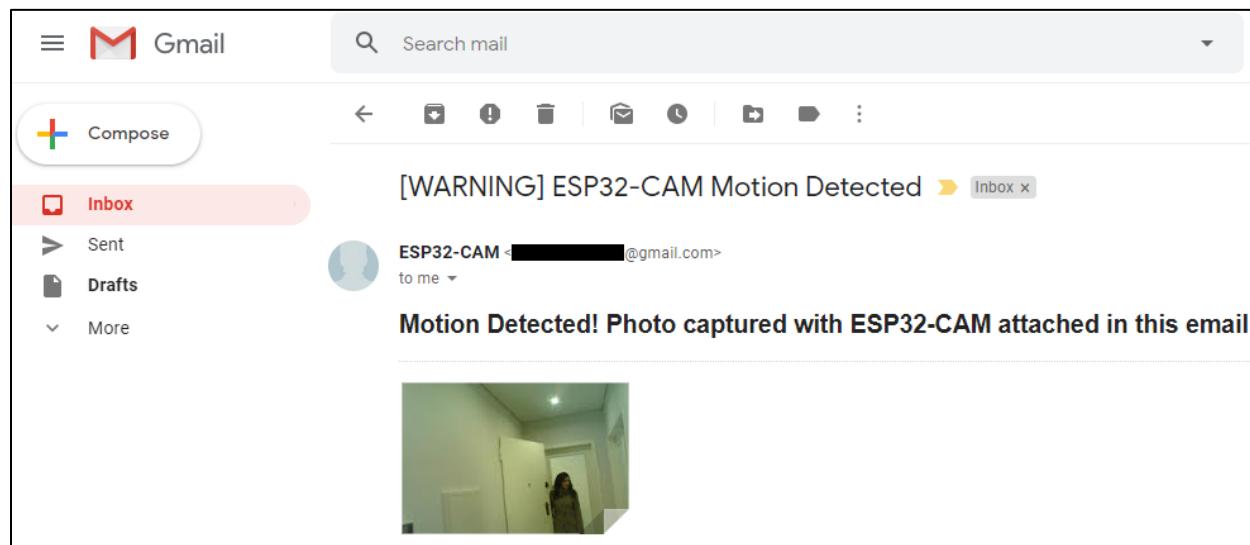
```
COM4
|
Connecting to WiFi....  
SPIFFS mounted successfully  
IP Address: http://192.168.1.97  
Taking a photo...  
Picture file name: /photo.jpg  
The picture has been saved in /photo.jpg - Size: 96512 bytes  
Sending email...  
Connecting to SMTP server...  
SMTP server connected, wait for response...  
Identification...  
Authentication...  
Sign in...  
Sending Email header...  
Sending Email body...  
Sending attachments...  
/photo.jpg  
Finalize...  
Finished  
Email sent successfully  
Going to sleep now
```

< >

Autoscroll Show timestamp

Newline 115200 baud Clear output

Check your email inbox. You should get an email or several emails with a message from the ESP32-CAM as well as the photo captured.



Warning: be careful when testing this project. If you activate the PIR motion sensor multiple times within a very short time interval, you'll receive many emails and your email sender account may be temporarily disabled or you may get banned from that email account.

Here's an example of one of the photos taken with the ESP32-CAM sent via email.

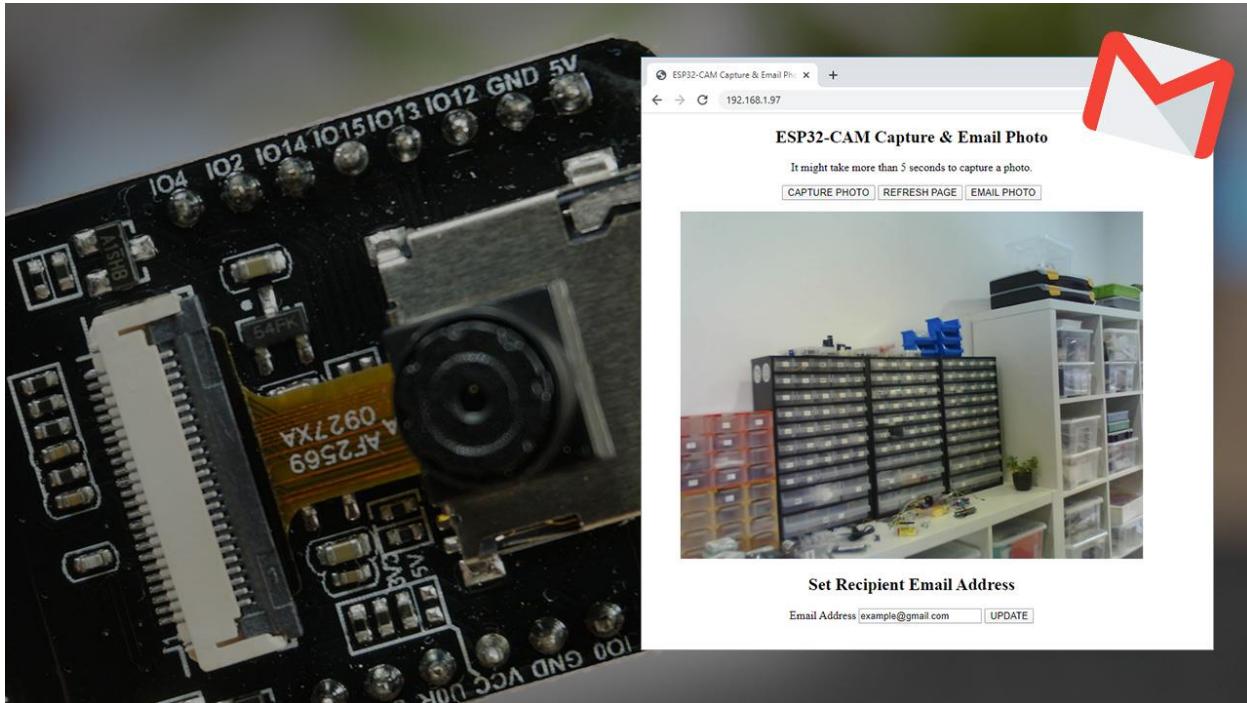


Wrapping Up

Now you can finish this project the way you want, you can either use a [dummy camera](#), or you can build your own enclosure and place it in a strategic place. When motion is detected in a certain part of the house, you'll be the first one to know what triggered the sensor.

Depending on where you place your camera, you may need to change the image settings like: brightness, saturation, exposure, etc... – check [Module 2, Unit 3](#).

Unit 3: Take and Email Photo with a Web Server



In this Unit, you'll setup a web server that allows you to take a photo with the ESP32-CAM and send it via email. In the web page, you can set the recipient's email address.

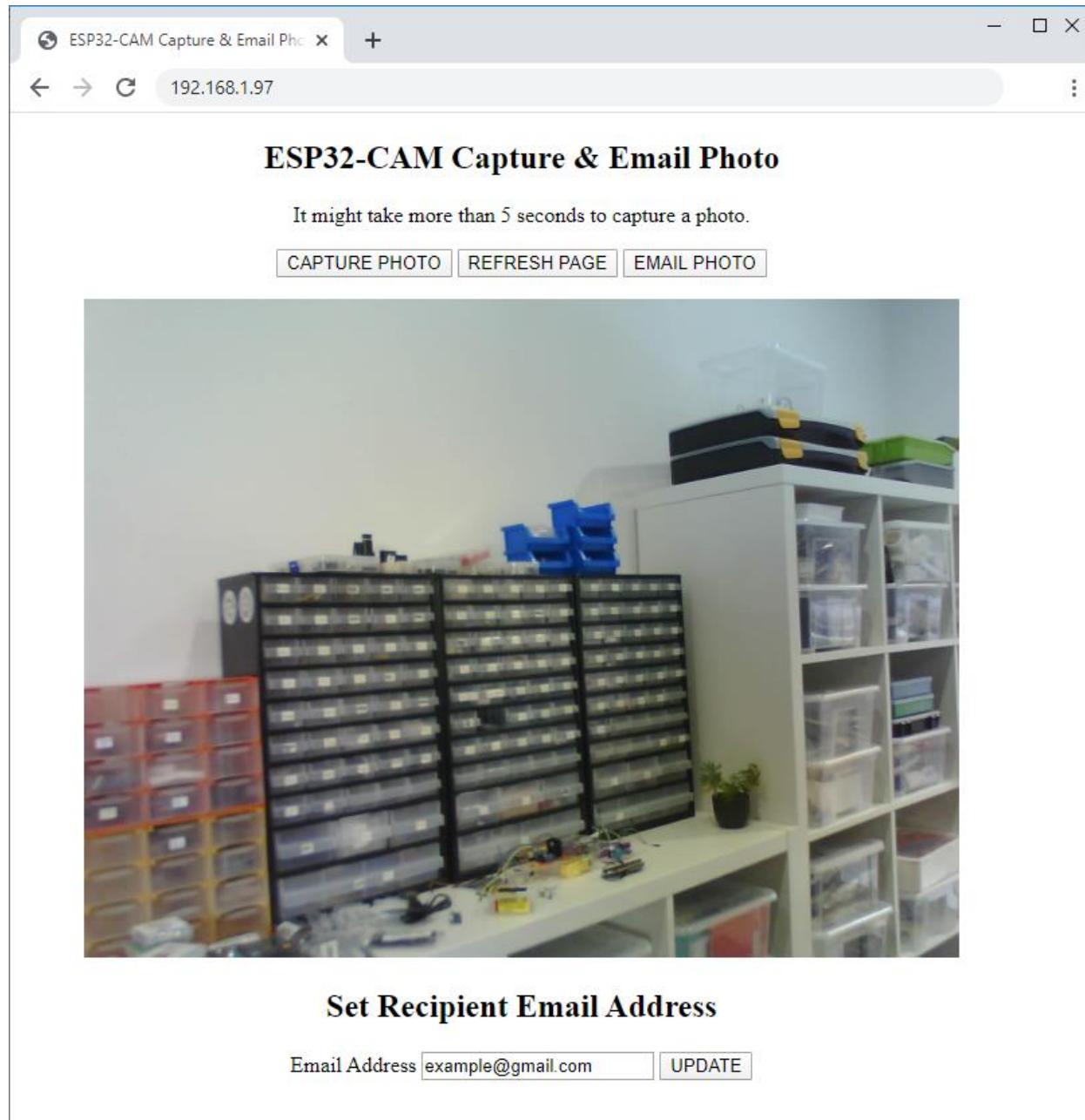
Compatibility: this project is compatible with any ESP32 Camera Board with the OV2640 camera. You just need to make sure you use the right pinout for the board you're using.

Project Overview

The web server has the following features (see image in the next page):

- **CAPTURE PHOTO** button: takes a photo with the ESP32-CAM and saves it in SPIFFS;

- **REFRESH PAGE** button: refreshes the page to show the most recent photo taken;
- **EMAIL PHOTO** button: emails the last photo taken via email;
- **Email Address Input Field**: allows you to set/change the recipient's email address. After inserting a new email address, click the **UPDATE** button for the changes to make effect.



Note: as mentioned previously the latest photo captured is stored in the ESP32 SPIFFS, so even if you restart your board, you can always access the last saved photo.

Prerequisites

Before proceeding with this project, make sure you check the following prerequisites:

Sending Emails

To send emails with the ESP32-CAM, you should:

- Install the [ESP32 Mail Client library](#);
- Set up a sender email address (this **shouldn't** be your personal email).

We cover these topics in one of the previous Units ([Module 3, Unit 1](#)). So, follow that Unit first if you haven't already.

Web Server Libraries

To build the web server, we'll use the [ESPAsyncWebServer library](#). This library also requires the [AsyncTCP library](#) to work properly. Follow the next steps to install those libraries if you haven't already.

Installing the ESPAsyncWebServer library

Follow the next steps to install the ESPAsyncWebServer library:

- 1) [Click here to download the ESPAsyncWebServer library](#). You should have a .zip folder in your *Downloads* folder.
- 2) Unzip the .zip folder and you should get *ESPAsyncWebServer-master* folder.
- 3) Rename your folder from *ESPAsyncWebServer-master* to *ESPAsyncWebServer*.
- 4) Move the *ESPAsyncWebServer* folder to your Arduino IDE installation libraries folder.

- 5) Finally, re-open your Arduino IDE.

Alternatively, after downloading the library, you can go to **Sketch ▶ Include Library** ▶ **Add .ZIP library...** and select the library you've just downloaded.

Installing the AsyncTCP library

The ESPAsyncWebServer library requires the AsyncTCP library to work. Follow the next steps to install that library:

- 1) [Click here to download the AsyncTCP library](#). You should have a .zip folder in your Downloads folder.
- 2) Unzip the .zip folder and you should get *AsyncTCP-master* folder.
- 3) Rename your folder from *AsyncTCP-master* to *AsyncTCP*.
- 4) Move the *AsyncTCP* folder to your Arduino IDE installation libraries folder.
- 5) Finally, re-open your Arduino IDE.

Alternatively, after downloading the library, you can go to **Sketch ▶ Include Library** ▶ **Add .ZIP library...** and select the library you've just downloaded.

Code

Copy the following code to your Arduino IDE. You need to make some minor changes to make it work for you. You need to insert your network credentials, as well as the sender email settings and the recipient's email.

CODE

[https://github.com/RuiSantosdotme/ESP32-CAM-eBook/blob/master/Code/Module 3/Take Photo Web Server Email/Take Photo Web Server Email.ino](https://github.com/RuiSantosdotme/ESP32-CAM-eBook/blob/master/Code/Module%203/Take%20Photo%20Web%20Server%20Email/Take%20Photo%20Web%20Server%20Email.ino)

```
#include "WiFi.h"
#include "esp_camera.h"
#include "esp_timer.h"
#include "img_converters.h"
#include "Arduino.h"
```

```

#include "soc/soc.h"                  // Disable brownout problems
#include "soc/rtc_cntl_reg.h"        // Disable brownout problems
#include "driver/rtc_io.h"
#include <ESPAsyncWebServer.h>
#include <StringArray.h>
#include <SPIFFS.h>
#include <FS.h>
#include "ESP32_MailClient.h"

// Replace with your network credentials
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = " REPLACE_WITH_YOUR_PASSWORD";

// To send Email using Gmail use port 465 (SSL) and SMTP Server
smtp.gmail.com
// YOU MUST ENABLE less secure app option
https://myaccount.google.com/lesssecureapps?pli=1
#define emailSenderAccount      "EXAMPLE_EMAIL@gmail.com"
#define emailSenderPassword    "YOUR_EXAMPLE_EMAIL_PASSWORD"
#define smtpServer              "smtp.gmail.com"
#define smtpServerPort          465
#define emailSubject            "ESP32-CAM Photo Captured"

// Default Recipient Email Address
String inputMessage = " YOUR_EMAIL_RECIPIENT@example.com ";
String inputMessage2 = "true";

// Create AsyncWebServer object on port 80
AsyncWebServer server(80);

boolean takeNewPhoto = false;

// Photo File Name to save in SPIFFS
#define FILE_PHOTO "/photo.jpg"

// OV2640 camera module pins (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

const char index_html[] PROGMEM = R"rawliteral(

```

```

<!DOCTYPE HTML><html>
<head>
    <title>ESP32-CAM Capture & Email Photo</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <style>
        body { text-align:center; }
    </style>
</head>
<body>
    <h2>ESP32-CAM Capture & Email Photo</h2>
    <p>It might take more than 5 seconds to capture a photo.</p>
    <p>
        <button onclick="capturePhoto()">CAPTURE PHOTO</button>
        <button onclick="location.reload();">REFRESH PAGE</button>
        <button onclick="emailPhoto();">EMAIL PHOTO</button>
    </p>
    <div></div>
    <h2>Set Recipient Email Address</h2>
    <form action="/get">
        Email Address <input type="email" name="email_input"
value="%EMAIL_INPUT%" required>
        <input type="submit" value="UPDATE">
    </form>
</body>
<script>
    var deg = 0;
    function capturePhoto() {
        var xhr = new XMLHttpRequest();
        xhr.open('GET', '/capture', true);
        xhr.send();
    }
    function emailPhoto() {
        var xhr = new XMLHttpRequest();
        xhr.open('GET', '/email-photo', true);
        xhr.send();
    }
</script>
</html>)rawliteral;

// Replaces placeholder with DS18B20 values
String processor(const String& var) {
    //Serial.println(var);
    if(var == "EMAIL_INPUT") {
        return inputMessage;
    }
    else if(var == "PER") {
        return "%";
    }
    return String();
}

// Flag variable to keep track if email notification was sent or not
bool emailSent = true;
const char* PARAM_INPUT_1 = "email_input";

```

```

// The Email Sending data object contains config and data to send
SMTPData smtpData;

void setup() {
    // Serial port for debugging purposes
    Serial.begin(115200);

    // Connect to Wi-Fi
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.println("Connecting to WiFi...");
    }
    if (!SPIFFS.begin(true)) {
        Serial.println("An Error has occurred while mounting SPIFFS");
        ESP.restart();
    }
    else {
        delay(500);
        Serial.println("SPIFFS mounted successfully");
    }

    // Print ESP32 Local IP Address
    Serial.print("IP Address: http://");
    Serial.println(WiFi.localIP());

    // Turn-off the 'brownout detector'
    WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0);

    // OV2640 camera module
    camera_config_t config;
    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;

    if (psramFound()) {

```

```

    config.frame_size = FRAMESIZE_UXGA;
    config.jpeg_quality = 10;
    config.fb_count = 2;
} else {
    config.frame_size = FRAMESIZE_SVGA;
    config.jpeg_quality = 12;
    config.fb_count = 1;
}
// Camera init
esp_err_t err = esp_camera_init(&config);
if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    ESP.restart();
}

// Route for root / web page
server.on("/", HTTP_GET, [] (AsyncWebServerRequest * request) {
    request->send_P(200, "text/html", index_html, processor);
});

server.on("/capture", HTTP_GET, [] (AsyncWebServerRequest * request) {
    takeNewPhoto = true;
    request->send_P(200, "text/plain", "Taking Photo");
});

server.on("/saved-photo", HTTP_GET, [] (AsyncWebServerRequest * request) {
    request->send(SPIFFS, FILE_PHOTO, "image/jpg", false);
});

server.on("/email-photo", HTTP_GET, [] (AsyncWebServerRequest * request) {
    emailSent = false;
    request->send_P(200, "text/plain", "Sending Photo");
});

// Receive an HTTP GET request at <ESP_IP>/get?email_input=<inputMessage>
server.on("/get", HTTP_GET, [] (AsyncWebServerRequest *request) {
    // GET email_input value on <ESP_IP>/get?email_input=<inputMessage>
    if (request->hasParam(PARAM_INPUT_1)) {
        inputMessage = request->getParam(PARAM_INPUT_1)->value();
    }
    else {
        inputMessage = "No message sent";
    }
    Serial.println(inputMessage);
    request->send(200, "text/html", "HTTP GET request sent to your
    ESP.<br><a href=\"/\">Return to Home Page</a>");
});

// Start server
server.begin();

}

void loop() {

```

```

if (takeNewPhoto) {
    capturePhotoSaveSpiffs();
    takeNewPhoto = false;
}
if (!emailSent) {
    String emailMessage = "Photo captured and emailed using an ESP32-CAM.";
    if(sendEmailNotification(emailMessage)) {
        Serial.println(emailMessage);
        emailSent = true;
    }
    else {
        Serial.println("Email failed to send");
    }
}
delay(1);
}

// Check if photo capture was successful
bool checkPhoto( fs::FS &fs ) {
    File f_pic = fs.open( FILE_PHOTO );
    unsigned int pic_sz = f_pic.size();
    return ( pic_sz > 100 );
}

// Capture Photo and Save it to SPIFFS
void capturePhotoSaveSpiffs( void ) {
    camera_fb_t * fb = NULL; // pointer
    bool ok = 0; // Boolean indicating if the picture was captured correctly

    do {
        // Take a photo with the camera
        Serial.println("Taking a photo...");

        fb = esp_camera_fb_get();
        if (!fb) {
            Serial.println("Camera capture failed");
            return;
        }

        // Photo file name
        Serial.printf("Picture file name: %s\n", FILE_PHOTO);
        File file = SPIFFS.open(FILE_PHOTO, FILE_WRITE);

        // Insert the data in the photo file
        if (!file) {
            Serial.println("Failed to open file in writing mode");
        }
        else {
            file.write(fb->buf, fb->len); // payload (image), payload length
            Serial.print("The picture has been saved in ");
            Serial.print(FILE_PHOTO);
            Serial.print(" - Size: ");
            Serial.print(file.size());
            Serial.println(" bytes");
        }
    }
}

```

```

    }
    // Close the file
    file.close();
    esp_camera_fb_return(fb);

    // check if file has been correctly saved in SPIFFS
    ok = checkPhoto(SPIFFS);
} while ( !ok );
}

bool sendEmailNotification(String emailMessage){
    // Set the SMTP Server Email host, port, account and password
    smtpData.setLogin(smtpServer, smtpServerPort, emailSenderAccount,
emailSenderPassword);

    // For library version 1.2.0 and later which STARTTLS protocol was
    // supported, the STARTTLS will be enabled automatically when port 587
    // was used, or enable it manually using setSTARTTLS function.
    //smtpData.setSTARTTLS(true);

    // Set the sender name and Email
    smtpData.setSender("ESP32", emailSenderAccount);

    // Set Email priority or importance High, Normal, Low or 1 to 5 (1 is highest)
    smtpData.setPriority("High");

    // Set the subject
    smtpData.setSubject(emailSubject);

    // Set the message with HTML format
    smtpData.setMessage(emailMessage, true);

    // Add recipients
    smtpData.addRecipient(inputMessage);

    // Add attach files from SPIFFS
    smtpData.addAttachFile(FILE_PHOTO, "image/jpg");
    // Set the storage type to attach files in your email (SPIFFS)
    smtpData.setFileType(MailClientStorageType::SPIFFS);

    smtpData.setSendCallback(sendCallback);

    // Start sending Email, can be set callback function to track the status
    if (!MailClient.sendMail(smtpData)) {
        Serial.println("Error sending Email, " + MailClient.smtpErrorReason());
        return false;
    }
    // Clear all data from Email object to free memory
    smtpData.empty();
    return true;
}

// Callback function to get the Email sending status
void sendCallback(SendStatus msg) {

```

```
//Print the current status  
Serial.println(msg.info());  
}
```

How the Code Works

Most of the code for this project was already explained in previous Units. So, we'll take a look at the relevant parts for this project.

Insert your network credentials in the following variables:

```
const char* ssid = "REPLACE_WITH_YOUR_SSID";  
const char* password = " REPLACE_WITH_YOUR_PASSWORD";
```

Insert the sender email account settings.

```
#define emailSenderAccount      "EXAMPLE_EMAIL@gmail.com"  
#define emailSenderPassword    "YOUR_EXAMPLE_EMAIL_PASSWORD"  
#define smtpServer             "smtp.gmail.com"  
#define smtpServerPort         465
```

In the `inputMessage` variable insert the default recipient's email address. You can change it later on the web server.

```
String inputMessage = " YOUR_EMAIL_RECIPIENT@example.com ";
```

The `index_html` variable contains the text to build the web page.

```
const char index_html[] PROGMEM = R"rawliteral(  
<!DOCTYPE HTML><html>  
<head>  
  <title>ESP32-CAM Capture & Email Photo</title>  
  <meta name="viewport" content="width=device-width, initial-scale=1">  
  <style>  
    body { text-align:center; }  
  </style>  
</head>  
<body>  
  <h2>ESP32-CAM Capture & Email Photo</h2>  
  <p>It might take more than 5 seconds to capture a photo.</p>  
  <p>  
    <button onclick="capturePhoto()">CAPTURE PHOTO</button>  
    <button onclick="location.reload();">REFRESH PAGE</button>  
    <button onclick="emailPhoto();">EMAIL PHOTO</button>  
  </p>  
  <div></div>  
<h2>Set Recipient Email Address</h2>
```

```

<form action="/get">
    Email Address <input type="email" name="email_input"
value="%EMAIL_INPUT%" required>
    <input type="submit" value="UPDATE">
</form>
</body>
<script>
    var deg = 0;
    function capturePhoto() {
        var xhr = new XMLHttpRequest();
        xhr.open('GET', '/capture', true);
        xhr.send();
    }
    function emailPhoto() {
        var xhr = new XMLHttpRequest();
        xhr.open('GET', '/email-photo', true);
        xhr.send();
    }
</script>
</html>) rawliteral";

```

The web page contains three buttons: CAPTURE PHOTO, REFRESH PAGE and EMAIL PHOTO.

```

<button onclick="capturePhoto()">CAPTURE PHOTO</button>
<button onclick="location.reload();">REFRESH PAGE</button>
<button onclick="emailPhoto();">EMAIL PHOTO</button>

```

When you click the CAPTURE PHOTO button, it calls the JavaScript function `capturePhoto()`. That function makes a request on the `/capture` URL.

```

function capturePhoto() {
    var xhr = new XMLHttpRequest();
    xhr.open('GET', '/capture', true);
    xhr.send();
}

```

The REFRESH PAGE button calls `location.reload()` that refreshes the web page.

Finally, the EMAIL PHOTO button calls the `emailPhoto()` function that makes a request on the `/email-photo` URL.

```

function emailPhoto() {
    var xhr = new XMLHttpRequest();
    xhr.open('GET', '/email-photo', true);
    xhr.send();
}

```

After the buttons there's a section to display the last photo taken. It requests the image on the `/saved-photo` URL.

```
<div></div>
```

There's also an HTML form to insert the recipient's email address

```
<form action="/get">
    Email Address <input type="email" name="email_input"
        value="%EMAIL_INPUT%" required>
    <input type="submit" value="UPDATE">
</form>
```

When you submit the form, it makes a request on the following URL.

```
get?email_input=<inputMessage>
```

The `%EMAIL_INPUT%` is a placeholder that will then be replaced by an actual value.

That is done in the `processor()` function.

```
// Replaces placeholder with DS18B20 values
String processor(const String& var) {
    //Serial.println(var);
    if(var == "EMAIL_INPUT") {
        return inputMessage;
    }
    else if(var == "PER") {
        return "%";
    }
    return String();
}
```

Basically, it replaces the placeholder with the value stored on the `inputMessage` variable. That variable contains the recipient's email address.

When you submit the form, you make a request on a URL that contains the recipient's email address. With the `AsyncWebServer` library we can handle that request and save the recipient's email address into a variable.

```
server.on("/get", HTTP_GET, [] (AsyncWebRequest *request) {
    // GET email_input value on <ESP_IP>/get?email_input=<inputMessage>
    if (request->hasParam(PARAM_INPUT_1)) {
        inputMessage = request->getParam(PARAM_INPUT_1)->value();
    }
    else {
        inputMessage = "No message sent";
```

```
    }
    Serial.println(inputMessage);
    request->send(200, "text/html", "HTTP GET request sent to your
ESP.<br><a href=\"/\">Return to Home Page</a>");
}
}
```

In the loop(), we check whether we should take and save a new photo and send it via email.

```
void loop() {
    if (takeNewPhoto) {
        capturePhotoSaveSpiffs();
        takeNewPhoto = false;
    }
    if (!emailSent) {
        String emailMessage = "Photo captured and emailed using an ESP32-CAM.";
        if (sendEmailNotification(emailMessage)) {
            Serial.println(emailMessage);
            emailSent = true;
        }
        else {
            Serial.println("Email failed to send");
        }
    }
    delay(1);
}
```

If you've followed previous projects, you should already be familiar with the rest of the code.

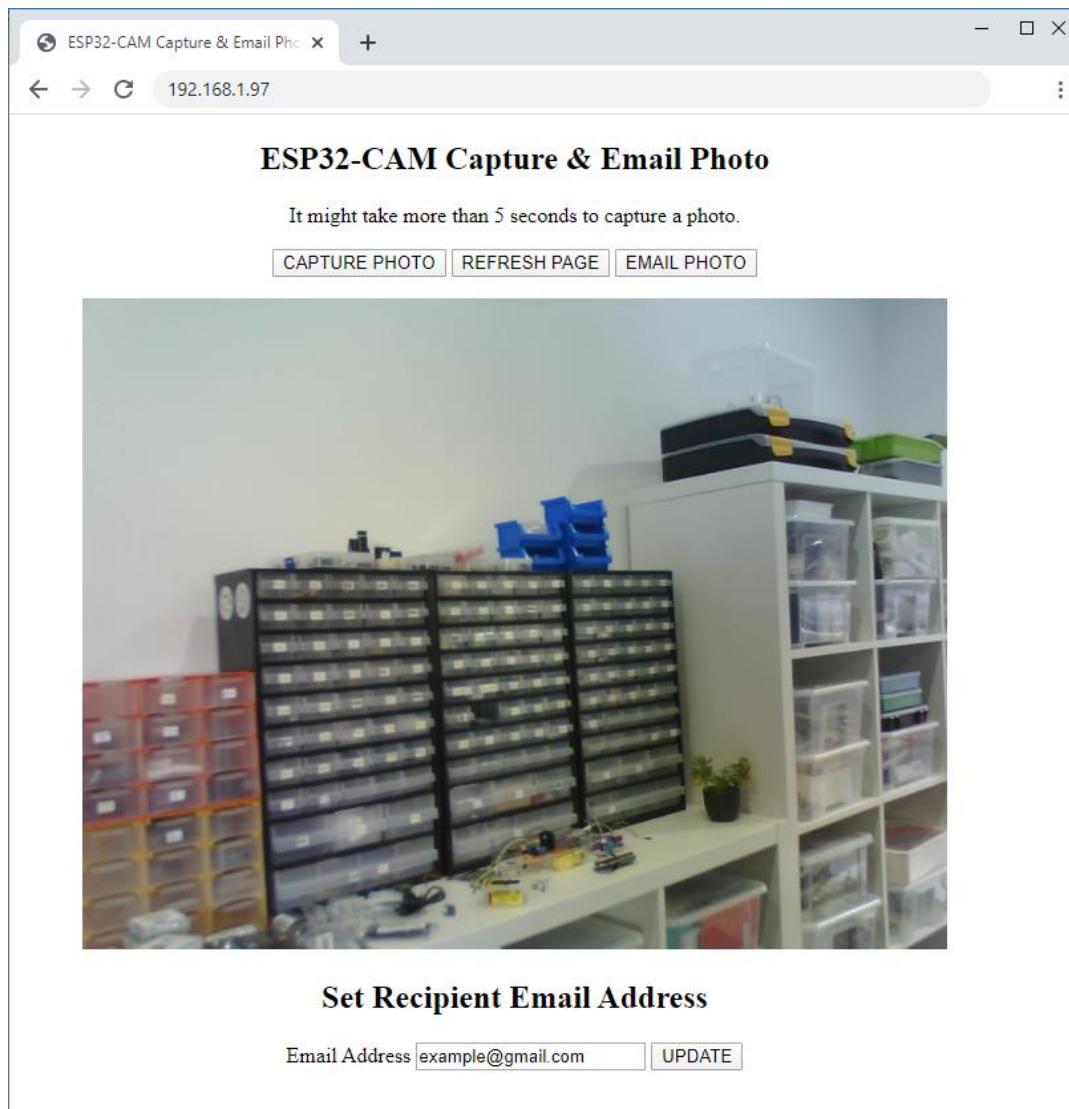
Demonstration

After making the necessary changes, upload the code to your ESP32-CAM. After uploading the code, open the Serial Monitor at a baud rate of 115200 and press the ESP32-CAM RESET button. The IP address should be printed on the Serial Monitor.

```
COM4
|
Connecting to WiFi...
IP Address: http://192.168.1.97
Initializing the camera module...
Initializing the MicroSD card module...
Starting SD Card
Picture file name: /photo_20200219_165218.jpg
Saved: /photo_20200219_165218.jpg

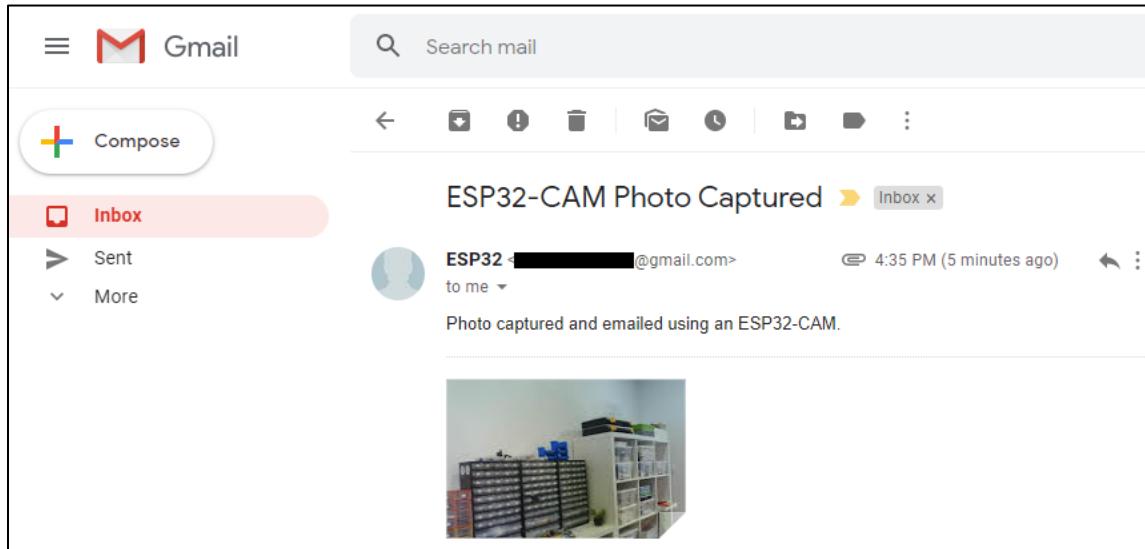
< >
 Autoscroll  Show timestamp  Newline  Clear output
```

Open a browser and type the ESP32 IP address, a web page as shown in the next image opens.



Click the CAPTURE PHOTO button to capture a new photo. Wait a few seconds and press the REFRESH PAGE button. The photo taken should be displayed. After that, you can email that photo by clicking the EMAIL PHOTO button. Before sending the email, you can set the recipient's email address on the Email Address input field.

After that, check your inbox. You should get an email from the ESP32-CAM with the last photo taken.



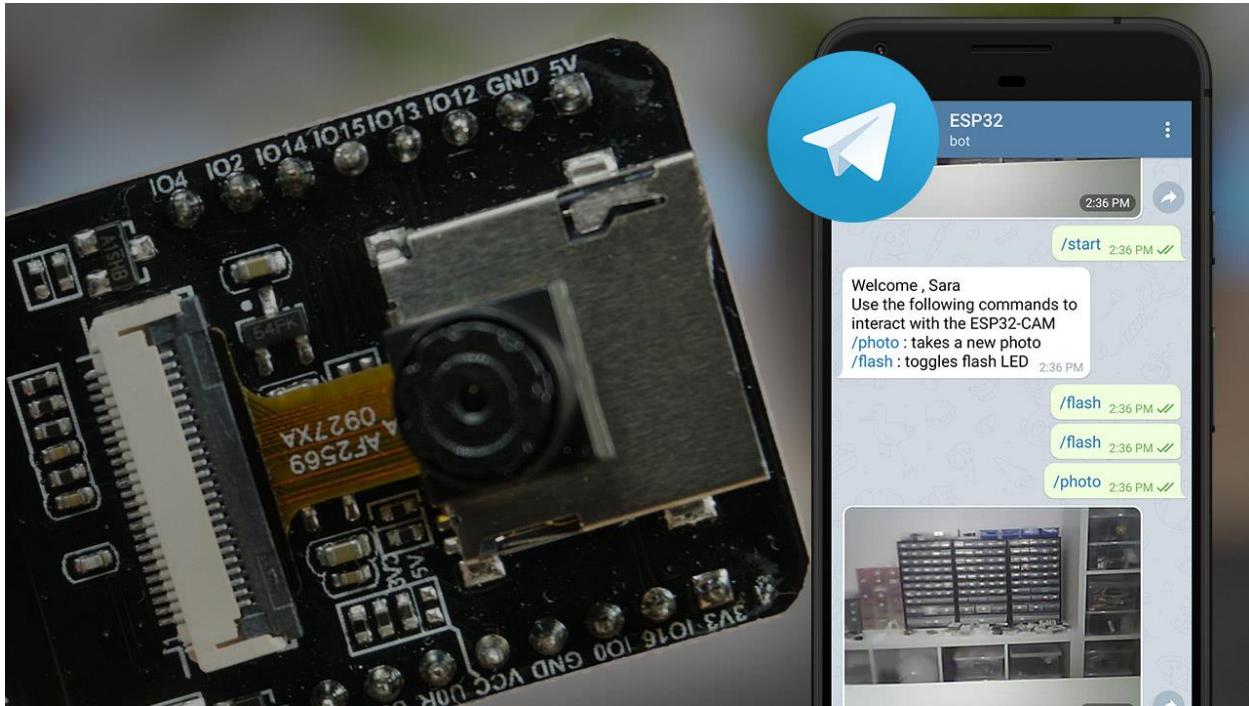
You can download and save the photo to your computer.



Wrapping Up

This project applied many of the concepts addressed in previous projects like: taking photos, building a web server and sending emails. As you can see the possibilities are endless. You can combine several sketches to build different projects with many applications. We hope you found this project interesting and you are able to modify it to include in your own projects.

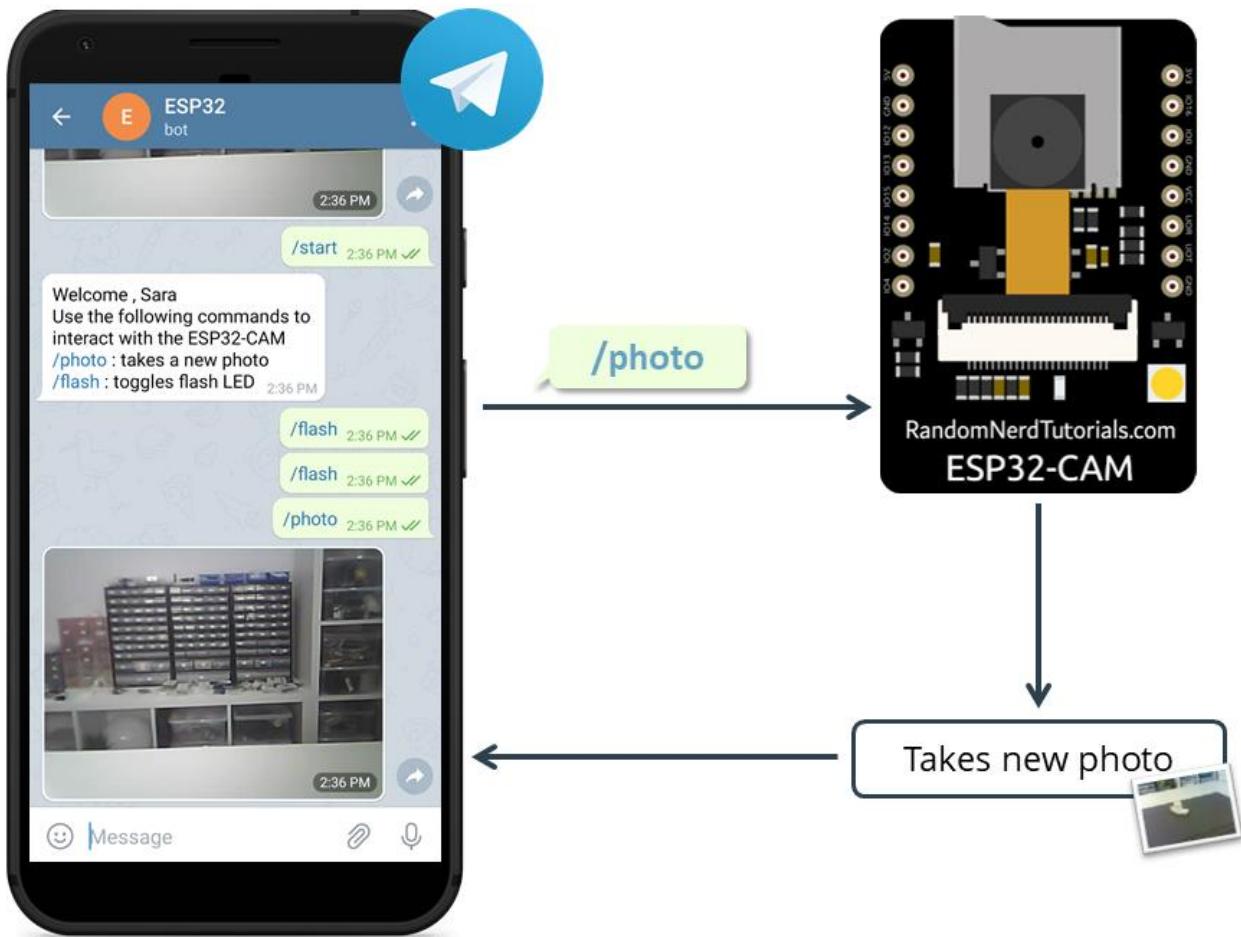
Unit 4: Take and Send Photo to Telegram App



In this Unit, you'll create a Telegram bot to interact with the ESP32-CAM to request a new photo. You can request a new photo using your Telegram account from anywhere. You just need to have access to the internet in your smartphone.

Compatibility: this project is compatible with any ESP32 Camera Board with the OV2640 camera. You just need to make sure you use the right pinout for the board you're using.

Project Overview



Here's an overview of the project you'll build:

- You'll create a Telegram bot for your ESP32-CAM;
- You can start a conversation with the ESP32-CAM bot;
- When you send the message **/photo** to the ESP32-CAM bot, the ESP32-CAM board receives the message, takes a new photo and responds with that photo;
- You can send the message **/flash** to toggle the ESP32-CAM's LED flash;
- You can send the **/start** message to receive a welcome message with the commands to control the board;
- The ESP32-CAM will only respond to messages coming from your Telegram account ID.

This is a simple project, but shows how you can use Telegram in your IoT and Home Automation projects. The idea is to apply the concepts learned in your own projects.

Introducing Telegram

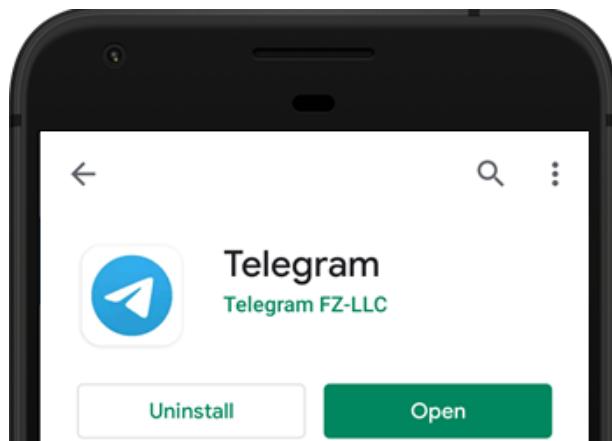
[Telegram](#) Messenger is a cloud-based instant messaging and voice over IP service. You can easily install it in your smartphone (Android and iPhone) or computer (PC, Mac and Linux). It's free and without any ads. Telegram allows you to create bots that you can interact with.

"Bots are third-party applications that run inside Telegram. Users can interact with bots by sending them messages, commands and inline requests. You control your bots using HTTPS requests to Telegram Bot API".

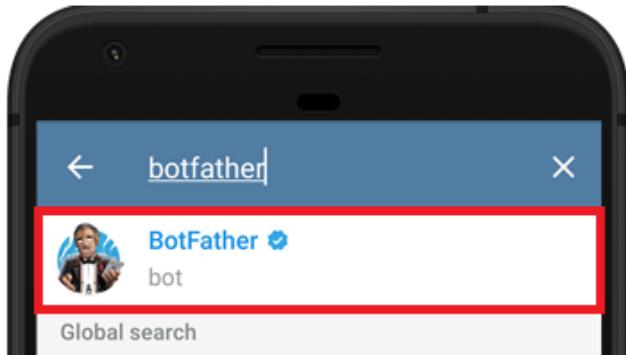
The ESP32-CAM will interact with the Telegram bot to receive and handle the messages, and send responses. In this tutorial, you'll learn how to use Telegram to send messages to your bot to request a new photo taken with the ESP32-CAM. You can receive the photo wherever you are (you just need Telegram and access to the internet).

Creating a Telegram Bot

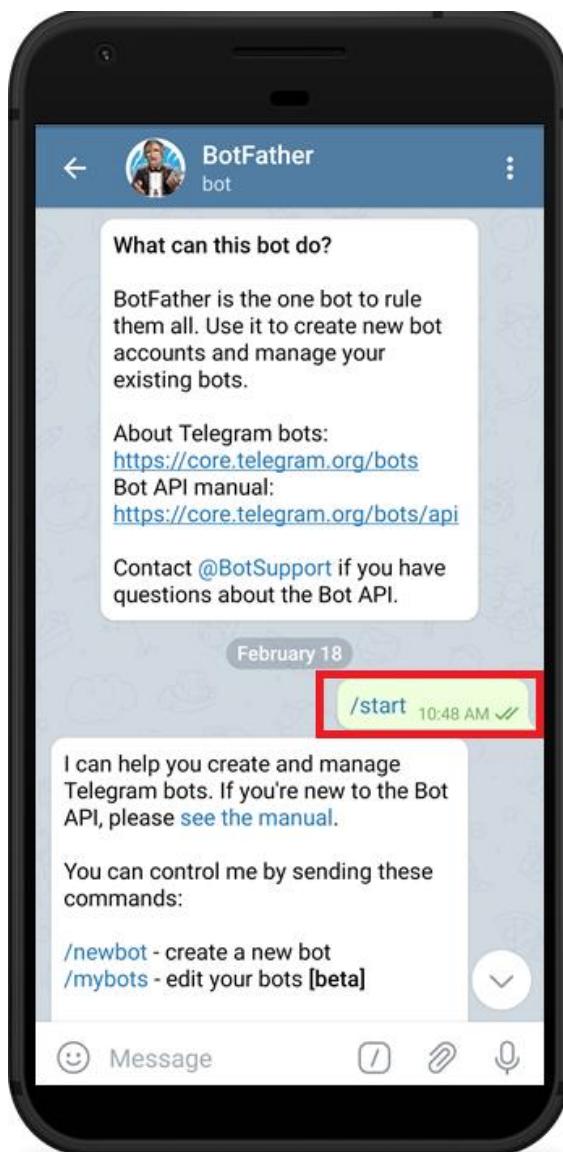
Go to Google Play or App Store, download and install **Telegram**.



Open Telegram and follow the next steps to create a Telegram Bot. First, search for "**botfather**" and click the BotFather as shown below. Or go to t.me/botfather.



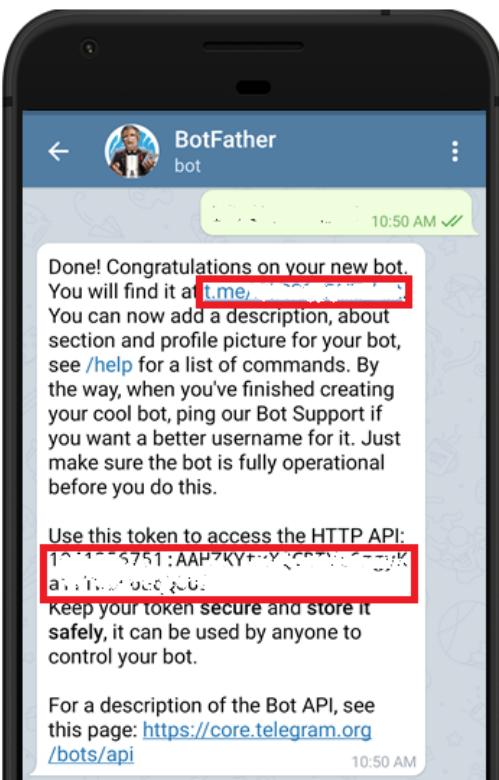
The following window should open and you'll be prompted to click the **start** button.



Type **/newbot** and follow the instructions to create your bot. Give it a name and username.



If your bot is successfully created, you'll receive a message with a link to access the bot and the bot token. Save the bot token because you'll need it later so that the ESP32-CAM can interact with the bot.

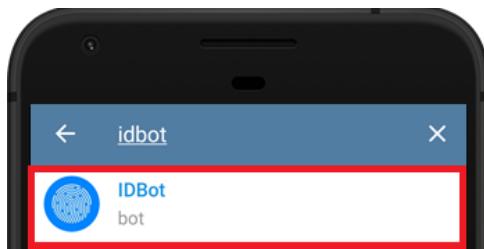


Note: the bot token is a very long string. To make sure you get it right, you can go to the [Telegram Web Interface](#) and copy your bot token from there.

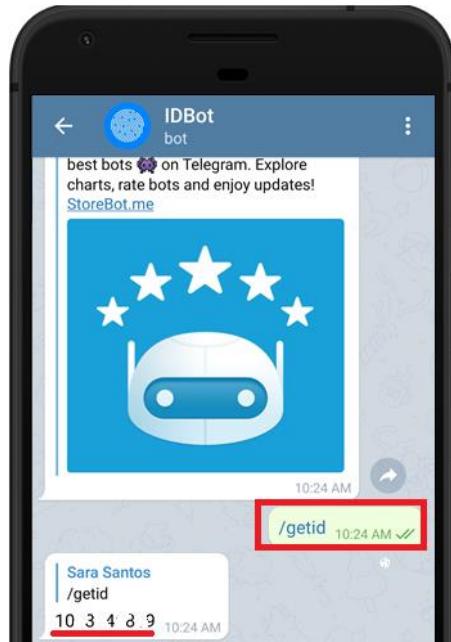
Get Your Telegram User ID

Anyone that knows your bot username can interact with it. To make sure that you ignore messages that are not from your Telegram account (or any authorized users), you can get your Telegram User ID. Then, when your telegram bot receives a message, the ESP32-CAM can check whether the sender ID corresponds to your user ID and handle the message or ignore it.

In your Telegram account, search for "**IDBot**".



Start a conversation with that bot and type **/getid**. You will get a reply back with your user ID. Save that user ID because you'll need it later in this tutorial.



Universal Telegram Bot Library

To interact with the Telegram bot, we'll use the [Universal Telegram Bot Library](#) created by Brian Lough that provides an easy interface for the Telegram Bot API.

Follow the next steps to install the latest release of the library.

1. [Click here to download the Universal Arduino Telegram Bot library.](#)
2. Go to **Sketch** ▶ **Include Library** ▶ **Add .ZIP Library...**
3. Add the library you've just downloaded.
4. And that's it. The library is installed.

Important: don't install the library through the Arduino Library Manager because it might install a deprecated version.

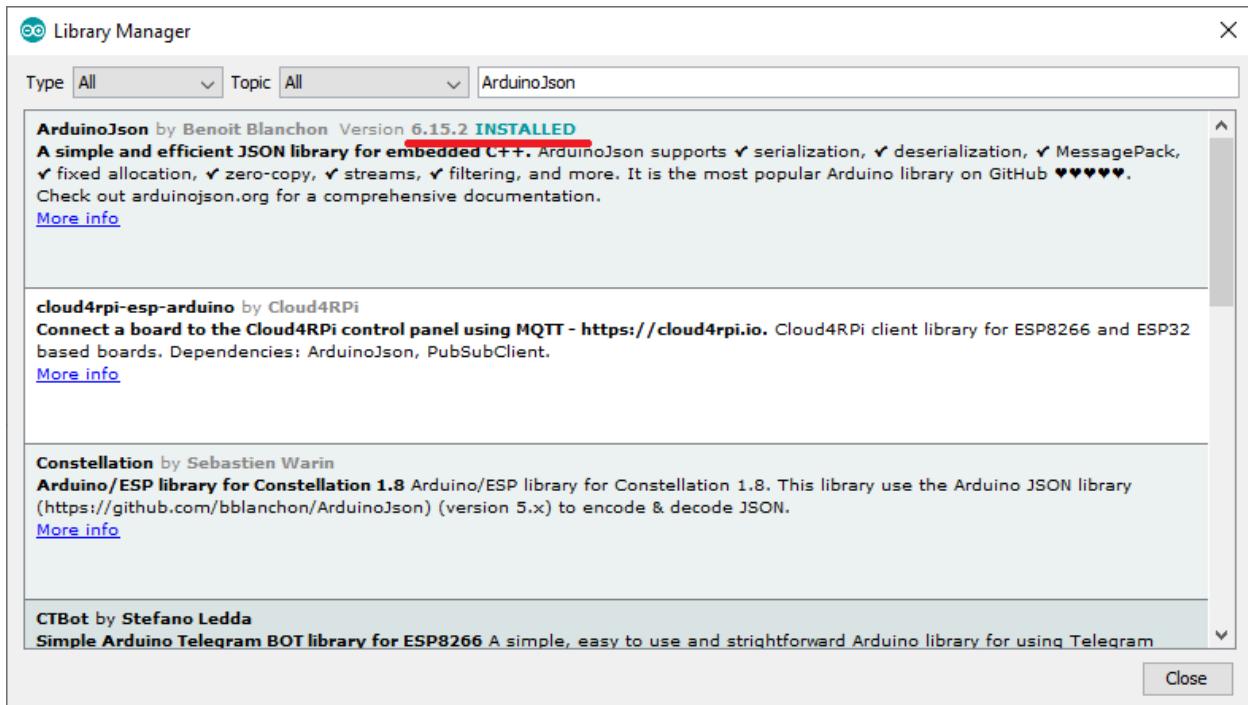
For all the details about the library, take a look at the [Universal Arduino Telegram Bot Library GitHub page](#).

ArduinoJson Library

You also have to install the [ArduinoJson](#) library. Follow the next steps to install the library.

1. In your Arduino IDE, go to **Sketch** ▶ **Include Library** ▶ **Manage Libraries**.
2. Search for "ArduinoJson".
3. Install the library.

We're using ArduinoJson library version 6.5.12.



Code

Copy the following code the Arduino IDE. To make this sketch work for you, you need to insert your network credentials (SSID and password), the Telegram Bot token and your Telegram user ID. Additionally, check the pin assignment for the camera board that you're using.

CODE

https://github.com/RuiSantosdotme/ESP32-CAM-eBook/blob/master/Code/Module_3/ESP32_CAM_Telegram_Send_Photo/ESP32_CAM_Telegram_Send_Photo.ino

```
#include <Arduino.h>
#include <WiFi.h>
#include <WiFiClientSecure.h>
#include "soc/soc.h"
#include "soc/rtc_cntl_reg.h"
#include "esp_camera.h"
#include <UniversalTelegramBot.h>
#include <ArduinoJson.h>

const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

// Initialize Telegram BOT
```

```

String BOTtoken = "XXXXXXXXXX:XXXXXXXXXXXXXXXXXXXXXXXXXXXX";
// your Bot Token (Get from Botfather)

// Use @myidbot to find out the chat ID of an individual or a group
// Also note that you need to click "start" on a bot before it can
// message you
String CHAT_ID = "XXXXXXXXXX";

bool sendPhoto = false;

WiFiClientSecure clientTCP;
UniversalTelegramBot bot(BOTtoken, clientTCP);

#define FLASH_LED_PIN 4
bool flashState = LOW;

//Checks for new messages every 1 second.
int botRequestDelay = 1000;
unsigned long lastTimeBotRan;

//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

void configInitCamera(){
    camera_config_t config;
    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;

//init with high specs to pre-allocate larger buffers
if(psramFound()){
    config.frame_size = FRAMESIZE_UXGA;
    config.jpeg_quality = 10; //0-63 lower number means higher quality
    config.fb_count = 2;
} else {
    config.frame_size = FRAMESIZE_SVGA;
    config.jpeg_quality = 12; //0-63 lower number means higher quality
    config.fb_count = 1;
}

// camera init
esp_err_t err = esp_camera_init(&config);
if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    delay(1000);
    ESP.restart();
}

// Drop down frame size for higher initial frame rate
sensor_t * s = esp_camera_sensor_get();
s->set_framesize(s, FRAMESIZE_CIF);
// UXGA|SXGA|XGA|SVGA|VGA|CIF|QVGA|HQVGA|QQVGA
}

void handleNewMessages(int numNewMessages) {
    Serial.print("Handle New Messages: ");
    Serial.println(numNewMessages);

    for (int i = 0; i < numNewMessages; i++) {
        String chat_id = String(bot.messages[i].chat_id);
        if (chat_id != CHAT_ID){
            bot.sendMessage(chat_id, "Unauthorized user", "");
            continue;
        }

        // Print the received message
        String text = bot.messages[i].text;
        Serial.println(text);

        String from_name = bot.messages[i].from_name;
        if (text == "/start") {
            String welcome = "Welcome , " + from_name + "\n";
            welcome += "Use the following commands to interact with the ESP32-CAM \n";
            welcome += "/photo : takes a new photo\n";
            welcome += "/flash : toggles flash LED \n";
            bot.sendMessage(CHAT_ID, welcome, "");
        }
        if (text == "/flash") {
            flashState = !flashState;
            digitalWrite(FLASH_LED_PIN, flashState);
            Serial.println("Change flash LED state");
        }
        if (text == "/photo") {
            sendPhoto = true;
        }
    }
}

```

```

        Serial.println("New photo request");
    }
}

String sendPhotoTelegram() {
    const char* myDomain = "api.telegram.org";
    String getAll = "";
    String getBody = "";

    camera_fb_t * fb = NULL;
    fb = esp_camera_fb_get();
    if(!fb) {
        Serial.println("Camera capture failed");
        delay(1000);
        ESP.restart();
        return "Camera capture failed";
    }

    Serial.println("Connect to " + String(myDomain));

    if (clientTCP.connect(myDomain, 443)) {
        Serial.println("Connection successful");

        String head = "--RandomNerdTutorials\r\nContent-Disposition: form-data; name=\\"chat_id\\"; \r\n\r\n" + CHAT_ID + "\r\n--RandomNerdTutorials\r\nContent-Disposition: form-data; name=\\"photo\\"; filename=\\"esp32-cam.jpg\\" \r\nContent-Type: image/jpeg\r\n\r\n";
        String tail = "\r\n--RandomNerdTutorials--\r\n";

        uint16_t imageLen = fb->len;
        uint16_t extraLen = head.length() + tail.length();
        uint16_t totalLen = imageLen + extraLen;

        clientTCP.println("POST /bot"+BOTtoken+"/sendPhoto HTTP/1.1");
        clientTCP.println("Host: " + String(myDomain));
        clientTCP.println("Content-Length: " + String(totalLen));
        clientTCP.println("Content-Type: multipart/form-data; boundary=RandomNerdTutorials");
        clientTCP.println();
        clientTCP.print(head);

        uint8_t *fbBuf = fb->buf;
        size_t fbLen = fb->len;
        for (size_t n=0;n<fbLen;n=n+1024) {
            if (n+1024<fbLen) {
                clientTCP.write(fbBuf, 1024);
                fbBuf += 1024;
            }
            else if (fbLen%1024>0) {
                size_t remainder = fbLen%1024;
                clientTCP.write(fbBuf, remainder);
            }
        }
        clientTCP.print(tail);
        esp_camera_fb_return(fb);
    }

    int waitTime = 10000; // timeout 10 seconds
}

```

```

long startTimer = millis();
boolean state = false;
while ((startTimer + waitTime) > millis()) {
    Serial.print(".");
    delay(100);
    while (clientTCP.available()) {
        char c = clientTCP.read();
        if (c == '\n') {
            if (getAll.length() == 0) state=true;
            getAll = "";
        }
        else if (c != '\r')
            getAll += String(c);
        if (state==true) getBody += String(c);
        startTimer = millis();
    }
    if (getBody.length()>0) break;
}
clientTCP.stop();
Serial.println(getBody);
}
else {
    getBody="Connected to api.telegram.org failed.";
    Serial.println("Connected to api.telegram.org failed.");
}
return getBody;
}

void setup(){
WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0);
// Init Serial Monitor
Serial.begin(115200);

// Set LED Flash as output
pinMode(FLASH_LED_PIN, OUTPUT);
digitalWrite(FLASH_LED_PIN, flashState);

// Config and init the camera
configInitCamera();

// Connect to Wi-Fi
WiFi.mode(WIFI_STA);
Serial.println();
Serial.print("Connecting to ");
Serial.println(ssid);
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(500);
}
Serial.println();
Serial.print("ESP32-CAM IP Address: ");
Serial.println(WiFi.localIP());
}
void loop() {
if (sendPhoto) {
    Serial.println("Preparing photo");
    sendPhotoTelegram();
}

```

```
    sendPhoto = false;
}
if (millis() > lastTimeBotRan + botRequestDelay) {
    int numNewMessages = bot.getUpdates(bot.last_message_received + 1);
    while (numNewMessages) {
        Serial.println("got response");
        handleNewMessages(numNewMessages);
        numNewMessages = bot.getUpdates(bot.last_message_received + 1);
    }
    lastTimeBotRan = millis();
}
}
```

How the Code Works

This section explains how the code works. Continue reading to learn how the code works or skip to the Demonstration section.

Importing Libraries

Start by importing the required libraries.

```
#include <Arduino.h>
#include <WiFi.h>
#include <WiFiClientSecure.h>
#include "soc/soc.h"
#include "soc/rtc_cntl_reg.h"
#include "esp_camera.h"
#include <UniversalTelegramBot.h>
#include <ArduinoJson.h>
```

Network Credentials

Insert your network credentials in the following variables.

```
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
```

Telegram User ID

Insert your chat ID. The one you've got from the IDBot.

```
String CHAT_ID = "XXXXXXXXXXXX";
```

Telegram Bot Token

Insert your Telegram Bot token you've got from **Botfather** on the `BOTtoken` variable.

```
String BOTtoken = "XXXXXXXXXX:XXXXXXXXXXXXXXXXXXXXXXXXXXXX";
```

The `sendPhoto` boolean variable indicates whether it is time to send a new photo to your telegram account. By default, it is set to `false`.

```
bool sendPhoto = false;
```

Create a new WiFi client with `WiFiClientSecure`.

```
WiFiClientSecure clientTCP;
```

Create a `bot` with the token and client defined earlier.

```
UniversalTelegramBot bot(BOTtoken, clientTCP);
```

Create a variable to hold the flash LED pin (`FLASH_LED_PIN`). In the ESP32-CAM AI-Thinker, the flash is connected to GPIO 4. By default, set it to `LOW`.

```
#define FLASH_LED_PIN 4
bool flashState = LOW;
```

The `botRequestDelay` and `lastTimeBotRan` variables are used to check for new Telegram messages every x number of seconds. In this case, the code will check for new messages every second (1000 milliseconds). You can change that delay time in the `botRequestDelay` variable.

```
int botRequestDelay = 1000;
unsigned long lastTimeBotRan;
```

ESP32-CAM Initialization

The following lines assign the ESP32-CAM AI-Thinker pins. If you're using a different ESP32 camera model, don't forget to change the pinout.

```
//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
```

The `configInitCamera()` function initializes the ESP32 camera.

```
void configInitCamera(){
    camera_config_t config;
    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;
    //init with high specs to pre-allocate larger buffers
    if(psramFound()) {
        config.frame_size = FRAMESIZE_UXGA;
        config.jpeg_quality = 10; //0-63 lower number means higher quality
        config.fb_count = 2;
    } else {
        config.frame_size = FRAMESIZE_SVGA;
        config.jpeg_quality = 12; //0-63 lower number means higher quality
        config.fb_count = 1;
    }
    // camera init
```

```

esp_err_t err = esp_camera_init(&config);
if (err != ESP_OK) {
    Serial.print("Camera init failed with error 0x%x", err);
    delay(1000);
    ESP.restart();
}
// Drop down frame size for higher initial frame rate
sensor_t * s = esp_camera_sensor_get();
s->set_framesize(s,           FRAME_SIZE_CIF); // UXGA|SXGA|XGA|SVGA|VGA|CIF|QVGA|HQVGA|QQVGA
}

```

handleNewMessages()

The `handleNewMessages()` function handles what happens when new messages arrive.

```

void handleNewMessages(int numNewMessages) {
    Serial.print("Handle New Messages: ");
    Serial.println(numNewMessages);
}

```

It checks the available messages:

```

for (int i = 0; i < numNewMessages; i++) {
}

```

Get the chat ID for a particular message and store it in the `chat_id` variable. The chat ID identifies who sent the message.

```

String chat_id = String(bot.messages[i].chat_id);

```

If the `chat_id` is different from your chat ID (`CHAT_ID`), it means that someone (that is not you) has sent a message to your bot. If that's the case, ignore the message and wait for the next message.

```

if (chat_id != CHAT_ID) {
    bot.sendMessage(chat_id, "Unauthorized user", "");
    continue;
}

```

Otherwise, it means that the message was sent from a valid user, so we'll save it in the `text` variable and check its content.

```

String text = bot.messages[i].text;
Serial.println(text);

```

The `from_name` variable saves the name of the sender.

```
String from_name = bot.messages[i].from_name;
```

If it receives the **/start** message, we'll send the valid commands to control the ESP.

This is useful if you happen to forget what are the commands to control your board.

```
if (text == "/start") {
    String welcome = "Welcome , " + from_name + "\n";
    welcome += "Use the following commands to interact with the ESP32-CAM
\n";
    welcome += "/photo : takes a new photo\n";
    welcome += "/flash : toggles flash LED \n";
    bot.sendMessage(CHAT_ID, welcome, "");
}
```

Sending a message to the bot is very simple. You just need to use the `sendMessage()` method on the `bot` object and pass as arguments the recipient's chat ID, the message, and the parse mode.

```
bool sendMessage(String chat_id, String text, String parse_mode = "");
```

In our example, we'll send the message to the ID stored on the `CHAT_ID` variable (that corresponds to your personal chat id) and send the message saved on the `welcome` variable.

```
bot.sendMessage(CHAT_ID, welcome, "");
```

If it receives the **/flash** message, invert the `flashState` variable and update the flash led state. If it was previously `LOW`, set it to `HIGH`. If it was previously `HIGH`, set it to `LOW`.

```
if (text == "/flash") {
    flashState = !flashState;
    digitalWrite(FLASH_LED_PIN, flashState);
    Serial.println("Change flash LED state");
}
```

Finally, if it receives the **/photo** message, set the `sendPhoto` variable to `true`. Then, in the `loop()`, check the value of the `sendPhoto` variable and proceed accordingly.

```
if (text == "/photo") {  
    sendPhoto = true;  
    Serial.println("New photo request");  
}
```

sendPhotoTelegram()

The `sendPhotoTelegram()` function takes a photo with the ESP32-CAM.

```
camera_fb_t * fb = NULL;  
fb = esp_camera_fb_get();  
if (!fb) {  
    Serial.println("Camera capture failed");  
    delay(1000);  
    ESP.restart();  
    return "Camera capture failed";  
}
```

Then, it makes an HTTP POST request to send the photo to your telegram bot.

```
clientTCP.println("POST /bot"+BOTtoken+"/sendPhoto HTTP/1.1");  
clientTCP.println("Host: " + String(myDomain));  
clientTCP.println("Content-Length: " + String(totalLen));  
clientTCP.println("Content-Type: multipart/form-data; boundary=RandomNerdTutorials");  
clientTCP.println();  
clientTCP.print(head);
```

setup()

In the `setup()`, initialize the Serial Monitor.

```
Serial.begin(115200);
```

Set the flash LED as an output and set it to its initial state.

```
pinMode(FLASH_LED_PIN, OUTPUT);  
digitalWrite(FLASH_LED_PIN, flashState);
```

Call the `configInitCamera()` function to configure and initialize the camera.

```
configInitCamera();
```

Connect your ESP32-CAM to your local network.

```
WiFi.mode(WIFI_STA);
Serial.println();
Serial.print("Connecting to ");
Serial.println(ssid);
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(500);
}
```

loop()

In the `loop()`, check the state of the `sendPhoto` variable. If it is `true`, call the `sendPhotoTelegram()` function to take and send a photo to your telegram account.

```
if (sendPhoto) {
    Serial.println("Preparing photo");
    sendPhotoTelegram();
```

When it's done, set the `sendPhoto` variable to `false`.

```
sendPhoto = false;
```

In the `loop()`, you also check for new messages every second.

```
if (millis() > lastTimeBotRan + botRequestDelay) {
    int numNewMessages = bot.getUpdates(bot.last_message_received + 1);
    while (numNewMessages) {
        Serial.println("got response");
        handleNewMessages(numNewMessages);
        numNewMessages = bot.getUpdates(bot.last_message_received + 1);
    }
    lastTimeBotRan = millis();
}
```

When a new message arrives, call the `handleNewMessages()` function.

```
while (numNewMessages) {
    Serial.println("got response");
    handleNewMessages(numNewMessages);
    numNewMessages = bot.getUpdates(bot.last_message_received + 1);
}
```

That's pretty much how the code works.

Demonstration

Upload the code to your ESP32-CAM board. Don't forget to go to **Tools** ▶ **Board** and select the board you're using. Go to **Tools** ▶ **Port** and select the COM port your board is connected to.

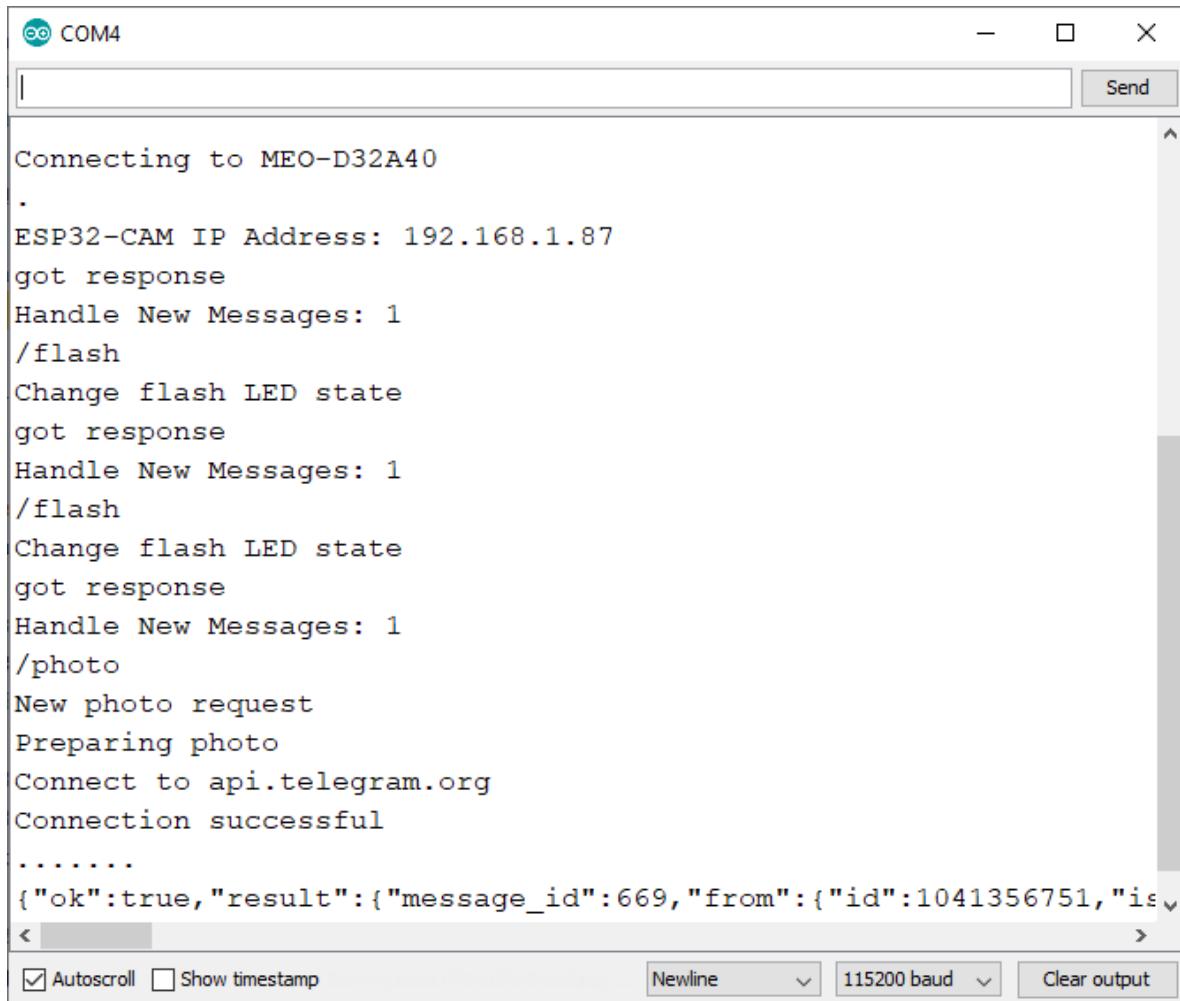
After uploading the code, press the ESP32-CAM on-board RST button so that it starts running the code. Then, you can open the Serial Monitor to check what's happening in the background.

Go to your Telegram account and open a conversation with your bot. Send the following commands and see the bot responding:

- **/start** shows the welcome message with the valid commands;
- **/flash** inverts the state of the LED flash;
- **/photo** takes a new photo and sends it to your Telegram account.



At the same time, on the Serial Monitor, you should see that the ESP32-CAM is receiving the messages.

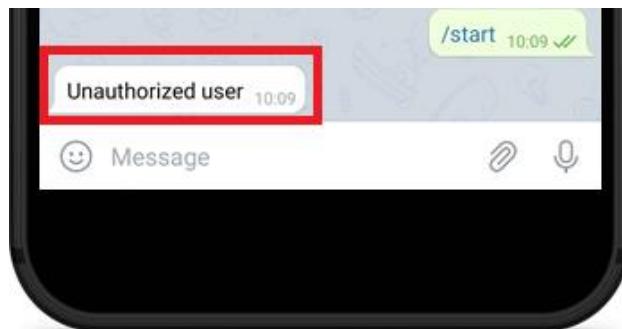


The screenshot shows a Windows-style serial monitor window titled "COM4". The text area displays the following log output from the ESP32-CAM:

```
Connecting to MEO-D32A40
.
ESP32-CAM IP Address: 192.168.1.87
got response
Handle New Messages: 1
/flash
Change flash LED state
got response
Handle New Messages: 1
/flash
Change flash LED state
got response
Handle New Messages: 1
/photo
New photo request
Preparing photo
Connect to api.telegram.org
Connection successful
.....
{"ok":true,"result":{"message_id":669,"from":{"id":1041356751,"is_v
< >
 Autoscroll  Show timestamp   
```

Note: sending a new photo might take some time.

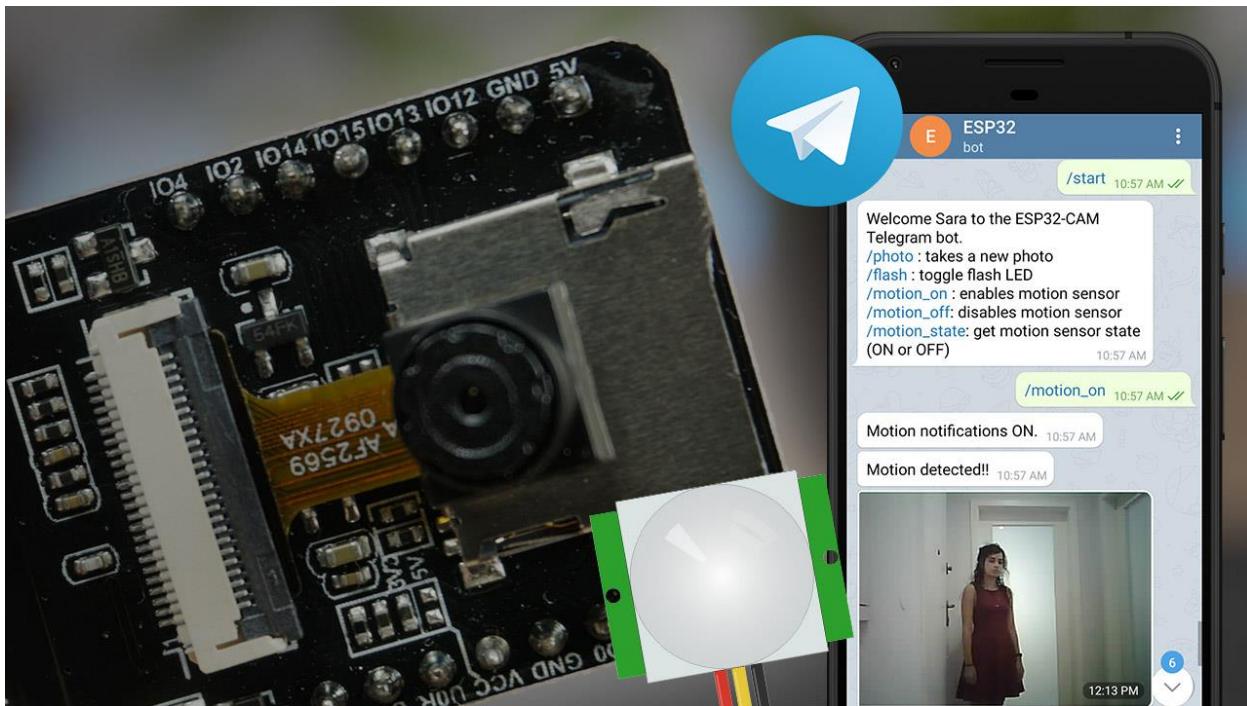
If you try to interact with your bot from another account, you'll get the "Unauthorized user" message



Wrapping Up

In this tutorial you've learned how to send a photo from the ESP32-CAM to your Telegram account. As long as you have access to the internet in your smartphone, you can request a new photo no matter where you are. This is great to monitor your ESP32-CAM from anywhere in the world.

Unit 5: Motion Detector with Photo Capture and Telegram Notifications



In this project, you're going to make a motion sensor detector with photo capture and Telegram notifications using the ESP32-CAM.

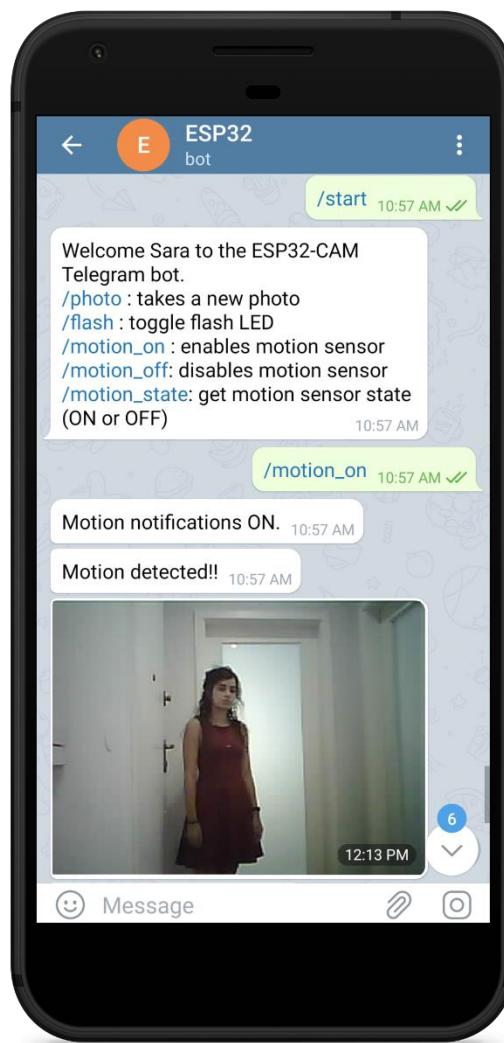
This is very similar to the previous project, but adds the PIR motion sensor. You'll receive a notification with a photo whenever motion is detected. You can disable or enable motion notifications through the app and much more.

Compatibility: this project is compatible with the ESP32-CAM AI-Thinker board or any other camera with an available GPIO to connect a PIR motion sensor.

Project Overview

Here's an overview of the project you'll build:

- You'll create a Telegram bot for your ESP32-CAM;
- You can start a conversation with the ESP32-CAM bot;
- You can send the message **/motion_on** to enable motion notifications;
- When motion is detected, you'll receive a notification with the photo;
- You can send the message **/motion_off** to disable motion notifications;
- The command **/motion_state** shows the current state of motion notifications (enabled or disabled).
- The commands of the previous project are also available:
 - **/photo**: the ESP32-CAM takes and sends a photo;
 - **/flash**: toggles the ESP32-CAM's LED flash;
 - **/start**: sends a welcome message with the commands to control the board;
- The ESP32-CAM will only respond to messages coming from your Telegram account ID.



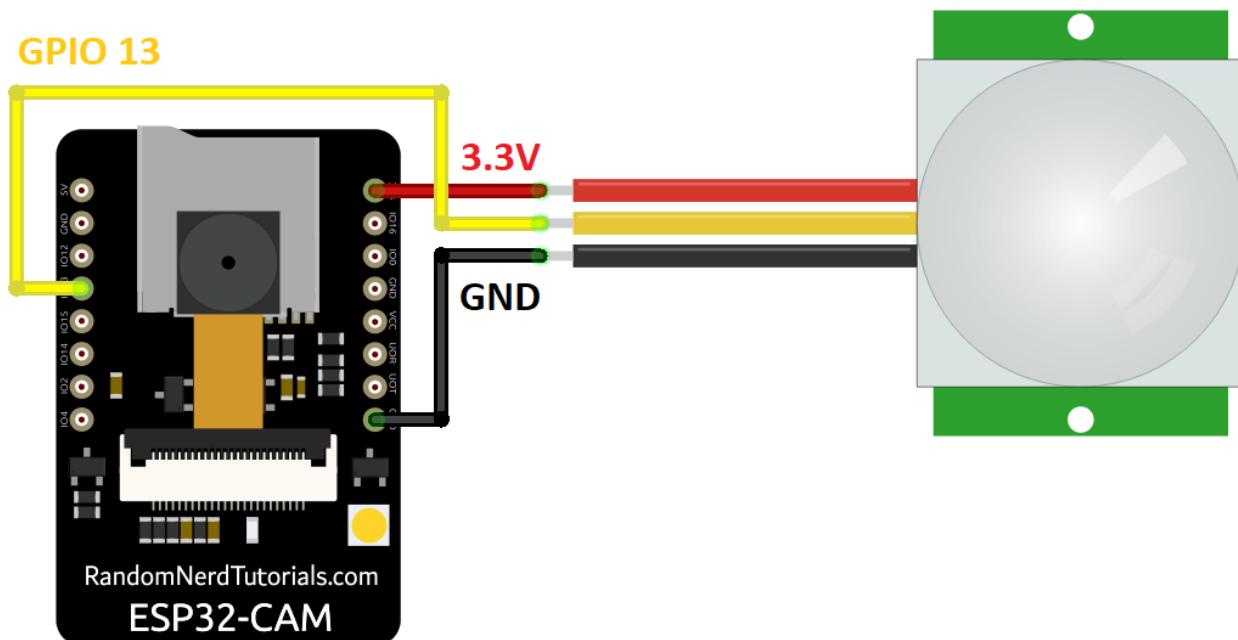
Parts Required

For this project, you need the following parts:

- [ESP32-CAM](#)
- [PIR Motion Sensor](#) (we're using the AM312)
- [Jumper wires](#)

Schematic Diagram

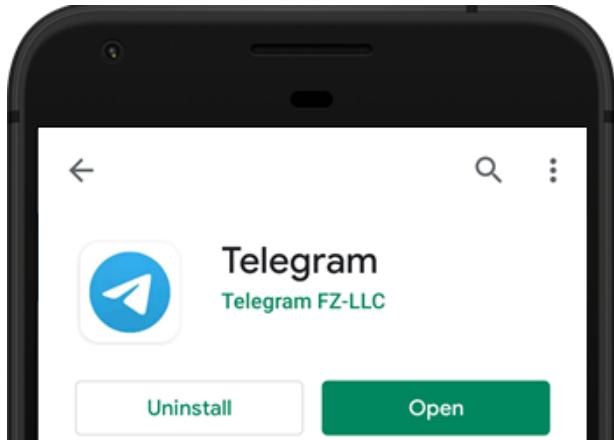
Wire the PIR motion sensor to the ESP32-CAM with the data pin connected to GPIO 13.



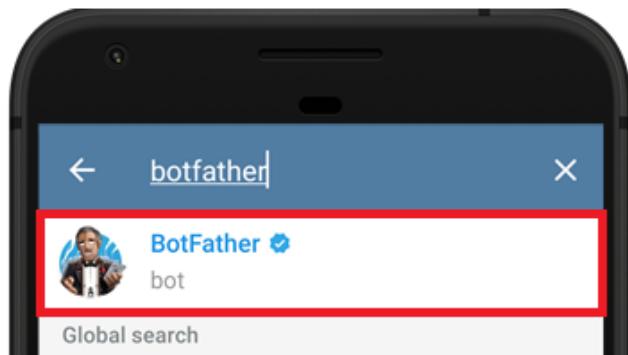
To follow this project, you need to create a Telegram Bot and install the Universal Telegram Bot library as well as the ArduinoJson library. If you've followed the previous project, you can skip the next steps.

Creating a Telegram Bot

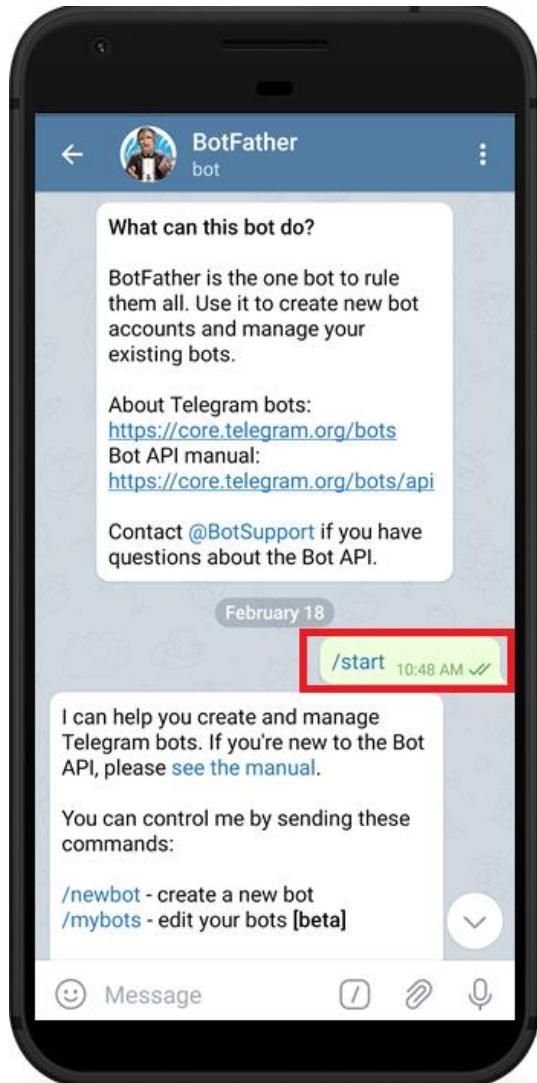
Go to Google Play or App Store, download and install **Telegram**.



Open Telegram and follow the next steps to create a Telegram Bot. First, search for "**botfather**" and click the BotFather as shown below. Or go to t.me/botfather.



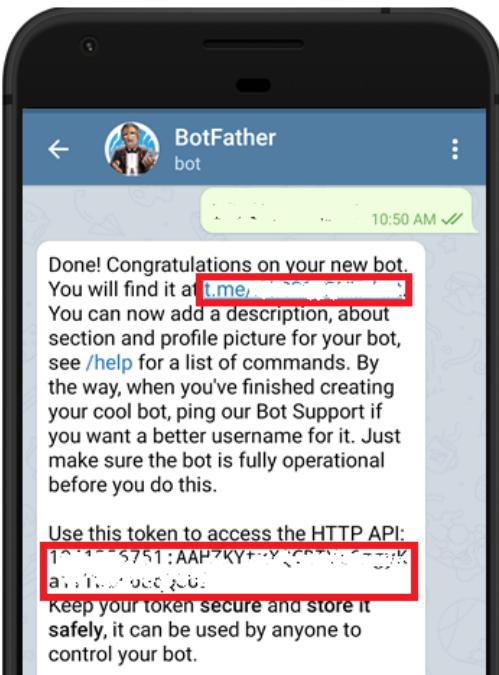
The following window should open and you'll be prompted to click the **start** button.



Type **/newbot** and follow the instructions to create your bot. Give it a name and username.



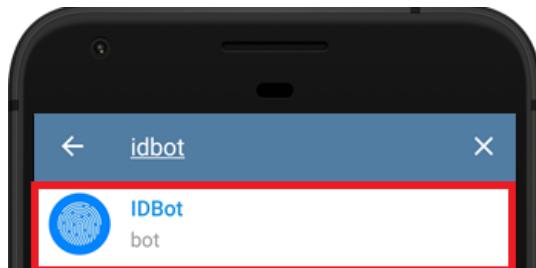
If your bot is successfully created, you'll receive a message with a link to access the bot and the bot token. Save the bot token because you'll need it later so that the ESP32-CAM can interact with the bot.



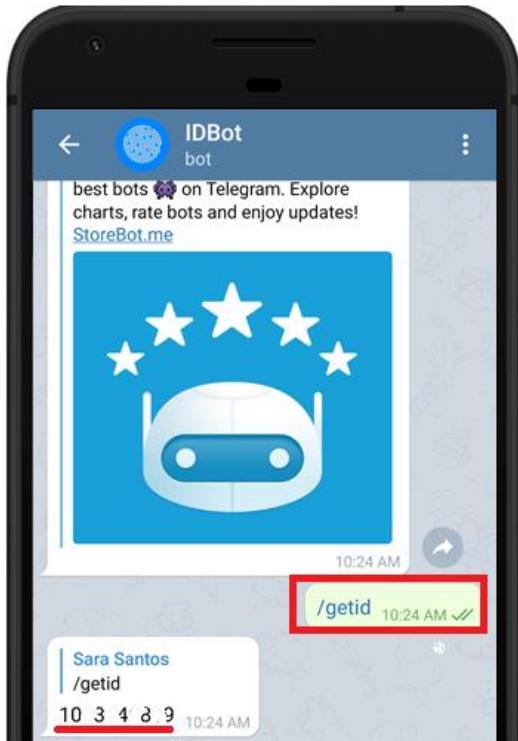
Get Your Telegram User ID

Anyone that knows your bot username can interact with it. To make sure that you ignore messages that are not from your Telegram account (or any authorized users), you can get your Telegram User ID. Then, when your telegram bot receives a message, the ESP32-CAM can check whether the sender ID corresponds to your User ID and handle the message or ignore it.

In your Telegram account, search for "**IDBot**".



Start a conversation with that bot and type **/getid**. You will get a reply back with your user ID. Save that user ID because you'll need it later in this tutorial.



Universal Telegram Bot Library

To interact with the Telegram bot, we'll use the [Universal Telegram Bot Library](#) created by Brian Lough that provides an easy interface for the Telegram Bot API.

Follow the next steps to install the latest release of the library.

1. [Click here to download the Universal Arduino Telegram Bot library](#).
2. Go to **Sketch** ▶ **Include Library** ▶ **Add .ZIP Library...**
3. Add the library you've just downloaded.
4. And that's it. The library is installed.

Important: don't install the library through the Arduino Library Manager because it might install a deprecated version.

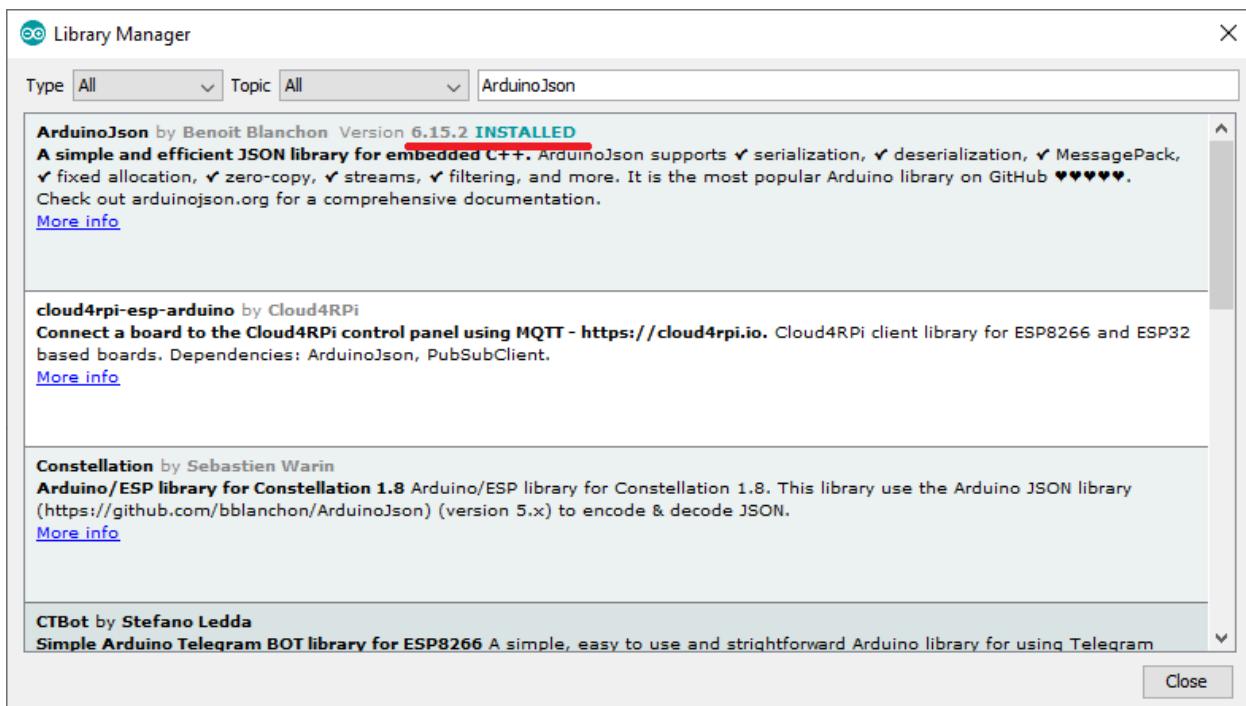
For all the details about the library, take a look at the [Universal Arduino Telegram Bot Library GitHub page](#).

ArduinoJson Library

You also have to install the [ArduinoJson](#) library. Follow the next steps to install the library.

1. In your Arduino IDE, go to **Sketch** ▶ **Include Library** ▶ **Manage Libraries**.
2. Search for "ArduinoJson".
3. Install the library.

We're using ArduinoJson library version 6.5.12.



Code

Copy the following code the Arduino IDE. To make this sketch work for you, you need to insert your network credentials (SSID and password), the Telegram Bot token and your Telegram user ID. Additionally, check the pin assignment for the camera board that you're using.

CODE

https://github.com/RuiSantosdotme/ESP32-CAM-eBook/blob/master/Code/Module_3/ESP32_CAM_Telegram_Send_Photo_Motion/ESP32_CAM_Telegram_Send_Photo_Motion.ino

```
#include <WiFi.h>
#include <WiFiClientSecure.h>
#include "soc/soc.h"
#include "soc/rtc_ctrl_reg.h"
#include "esp_camera.h"
#include <UniversalTelegramBot.h>
#include <ArduinoJson.h>

// Replace with your network credentials
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

// Use @myidbot to find out the chat ID of an individual or a group
// Also note that you need to click "start" on a bot before it can
// message you
String chatId = "XXXXXXXXXXXX";

// Initialize Telegram BOT
String BOTtoken = "XXXXXXXXXX:XXXXXXXXXXXXXXXXXXXXXXXXXXXXX";

bool sendPhoto = false;

WiFiClientSecure clientTCP;

UniversalTelegramBot bot(BOTtoken, clientTCP);

//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

#define FLASH_LED_PIN 4
bool flashState = LOW;

// Motion Sensor
bool motionEnabled = false;
bool motionDetected = false;

int botRequestDelay = 1000; // mean time between scan messages
long lastTimeBotRan; // last time messages' scan has been done

void handleNewMessages(int numNewMessages);
String sendPhotoTelegram();

// Indicates when motion is detected
static void IRAM_ATTR detectsMovement(void * arg) {
    //Serial.println("MOTION DETECTED!!!");
    motionDetected = true;
}

void setup() {
    WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0);
    Serial.begin(115200);

    pinMode(FLASH_LED_PIN, OUTPUT);
    digitalWrite(FLASH_LED_PIN, flashState);

    WiFi.mode(WIFI_STA);
    Serial.println();
    Serial.print("Connecting to ");
    Serial.println(ssid);
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        Serial.print(".");
        delay(500);
    }
    Serial.println();
    Serial.print("ESP32-CAM IP Address: ");
    Serial.println(WiFi.localIP());

    camera_config_t config;
    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;

//init with high specs to pre-allocate larger buffers
if(psramFound()){
    config.frame_size = FRAMESIZE_UXGA;
    config.jpeg_quality = 10; //0-63 lower number means higher quality
    config.fb_count = 2;
} else {
    config.frame_size = FRAMESIZE_SVGA;
    config.jpeg_quality = 12; //0-63 lower number means higher quality
    config.fb_count = 1;
}

// camera init
esp_err_t err = esp_camera_init(&config);
if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    delay(1000);
    ESP.restart();
}

// Drop down frame size for higher initial frame rate
sensor_t * s = esp_camera_sensor_get();
s->set_framesize(s, FRAMESIZE_CIF); // UXGA|SXGA|XGA|SVGA|VGA|CIF|QVGA|HQVGA|QQVGA

// PIR Motion Sensor mode INPUT_PULLUP
//err = gpio_install_isr_service(0);
err = gpio_isr_handler_add(GPIO_NUM_13, &detectsMovement, (void *) 13);
if (err != ESP_OK){
    Serial.printf("handler add failed with error 0x%x \r\n", err);
}
err = gpio_set_intr_type(GPIO_NUM_13, GPIO_INTR_POSEDGE);
if (err != ESP_OK){
    Serial.printf("set intr type failed with error 0x%x \r\n", err);
}
bot.sendMessage(chatId, "Bot started up", "");
}

```

```

void loop() {
    if (sendPhoto) {
        Serial.println("Preparing photo");
        sendPhotoTelegram();
        sendPhoto = false;
    }

    if(motionDetected && motionEnabled) {
        bot.sendMessage(chatId, "Motion detected!!", "");
        Serial.println("Motion Detected");
        sendPhotoTelegram();
        motionDetected = false;
    }

    if (millis() > lastTimeBotRan + botRequestDelay) {
        int numNewMessages = bot.getUpdates(bot.last_message_received + 1);
        while (numNewMessages) {
            Serial.println("got response");
            handleNewMessages(numNewMessages);
            numNewMessages = bot.getUpdates(bot.last_message_received + 1);
        }
        lastTimeBotRan = millis();
    }
}

String sendPhotoTelegram(){
    const char* myDomain = "api.telegram.org";
    String getAll = "";
    String getBody = "";

    camera_fb_t * fb = NULL;
    fb = esp_camera_fb_get();
    if(!fb) {
        Serial.println("Camera capture failed");
        delay(1000);
        ESP.restart();
        return "Camera capture failed";
    }

    Serial.println("Connect to " + String(myDomain));

    if (clientTCP.connect(myDomain, 443)) {
        Serial.println("Connection successful");

        String head = "--RandomNerdTutorials\r\nContent-Disposition: form-data; name=\"chat_id\"; \r\n\r\n" + chatId + "\r\n--RandomNerdTutorials\r\nContent-Disposition: form-data; name=\"photo\"; filename=\"esp32-cam.jpg\"\r\nContent-Type: image/jpeg\r\n\r\n";
        String tail = "\r\n--RandomNerdTutorials--\r\n";

        uint16_t imageLen = fb->len;
        uint16_t extraLen = head.length() + tail.length();
        uint16_t totalLen = imageLen + extraLen;
    }
}

```

```

clientTCP.println("POST /bot"+BOTtoken+"/sendPhoto HTTP/1.1");
clientTCP.println("Host: " + String(myDomain));
clientTCP.println("Content-Length: " + String(totalLen));
clientTCP.println("Content-Type: multipart/form-data;
boundary=RandomNerdTutorials");
clientTCP.println();
clientTCP.print(head);

uint8_t *fbBuf = fb->buf;
size_t fbLen = fb->len;
for (size_t n=0;n<fbLen;n=n+1024) {
    if (n+1024<fbLen) {
        clientTCP.write(fbBuf, 1024);
        fbBuf += 1024;
    }
    else if (fbLen%1024>0) {
        size_t remainder = fbLen%1024;
        clientTCP.write(fbBuf, remainder);
    }
}

clientTCP.print(tail);

esp_camera_fb_return(fb);

int waitTime = 10000; // timeout 10 seconds
long startTimer = millis();
boolean state = false;

while ((startTimer + waitTime) > millis()) {
    Serial.print(".");
    delay(100);
    while (clientTCP.available()) {
        char c = clientTCP.read();
        if (c == '\n') {
            if (getAll.length() == 0) state=true;
            getAll = "";
        }
        else if (c != '\r') {
            getAll += String(c);
        }
        if (state==true) {
           getBody += String(c);
        }
        startTimer = millis();
    }
    if (getBody.length()>0) break;
}
clientTCP.stop();
Serial.println(getBody);
}

else {
   getBody="Connected to api.telegram.org failed.";
}

```

```

        Serial.println("Connected to api.telegram.org failed.");
    }
    return getBody;
}

void handleNewMessages(int numNewMessages) {
    Serial.print("Handle New Messages: ");
    Serial.println(numNewMessages);

    for (int i = 0; i < numNewMessages; i++) {
        // Chat id of the requester
        String chat_id = String(bot.messages[i].chat_id);
        if (chat_id != chatId) {
            bot.sendMessage(chat_id, "Unauthorized user", "");
            continue;
        }

        // Print the received message
        String text = bot.messages[i].text;
        Serial.println(text);

        String fromName = bot.messages[i].from_name;

        if (text == "/flash") {
            flashState = !flashState;
            digitalWrite(FLASH_LED_PIN, flashState);
        }
        if (text == "/photo") {
            sendPhoto = true;
            Serial.println("New photo request");
        }
        if (text == "/motion_on") {
            motionEnabled = true;
            bot.sendMessage(chatId, "Motion notifications ON.", "");
        }
        if (text == "/motion_off") {
            motionEnabled = false;
            bot.sendMessage(chatId, "Motion notifications OFF.", "");
        }
        if (text == "/motion_state") {
            if (motionEnabled == false) {
                bot.sendMessage(chatId, "Motion notifications are OFF.", "");
            }
            else{
                bot.sendMessage(chatId, "Motion notifications are ON.", "");
            }
        }
        if (text == "/start"){
            String welcome = "Welcome " + fromName + " to the ESP32-CAM
Telegram bot.\n";
            welcome += "/photo : takes a new photo\n";
            welcome += "/flash : toggle flash LED\n";
            welcome += "/motion_on : enables motion sensor\n";
            welcome += "/motion_off: disables motion sensor \n";
        }
    }
}

```

```
        welcome += "/motion_state: get motion sensor state (ON or OFF)";
        bot.sendMessage(chatId, welcome, "");
    }
}
```

How the Code Works

This code is very similar with the one from the previous unit, so we'll just take a look at the relevant parts for this project.

The `motionEnabled` variable indicates whether motion notifications are enabled.

The `motionDetected` variable indicates whether motion was detected or not.

```
bool motionEnabled = false;
bool motionDetected = false;
```

The `detectsMovement()` function will be called when motion is detected. It will change the `motionDetected` variable to `true`.

```
static void IRAM_ATTR detectsMovement(void * arg) {
    //Serial.println("MOTION DETECTED!!!");
    motionDetected = true;
}
```

In the `setup()`, the following lines set GPIO 13 as an interrupt with the `detectsMovement()` callback function.

```
err = gpio_isr_handler_add(GPIO_NUM_13, &detectsMovement, (void *)13);
if (err != ESP_OK) {
    Serial.printf("handler add failed with error 0x%x \r\n", err);
}
err = gpio_set_intr_type(GPIO_NUM_13, GPIO_INTR_POSEDGE);
if (err != ESP_OK) {
    Serial.printf("set intr type failed with error 0x%x \r\n", err);
}
```

In the `loop()`, when motion is detected and motion notifications are enabled, send a notification and a photo to your Telegram account. Finally, set the `motionDetected` variable to `false`.

```
if(motionDetected && motionEnabled) {
```

```

bot.sendMessage(chatId, "Motion detected!!", "");
Serial.println("Motion Detected");
sendPhotoTelegram();
motionDetected = false;
}

```

Demonstration

Upload the code to your ESP32-CAM board. After uploading the code, press the ESP32-CAM on-board RST button so that it starts running the code. Then, you can open the Serial Monitor to check what's happening in the background.

When the ESP32-CAM first starts, you should receive a message in your Telegram account indicating the bot has started: "Bot started up". You'll only receive this message, if you've started a conversation with the bot previously.

```

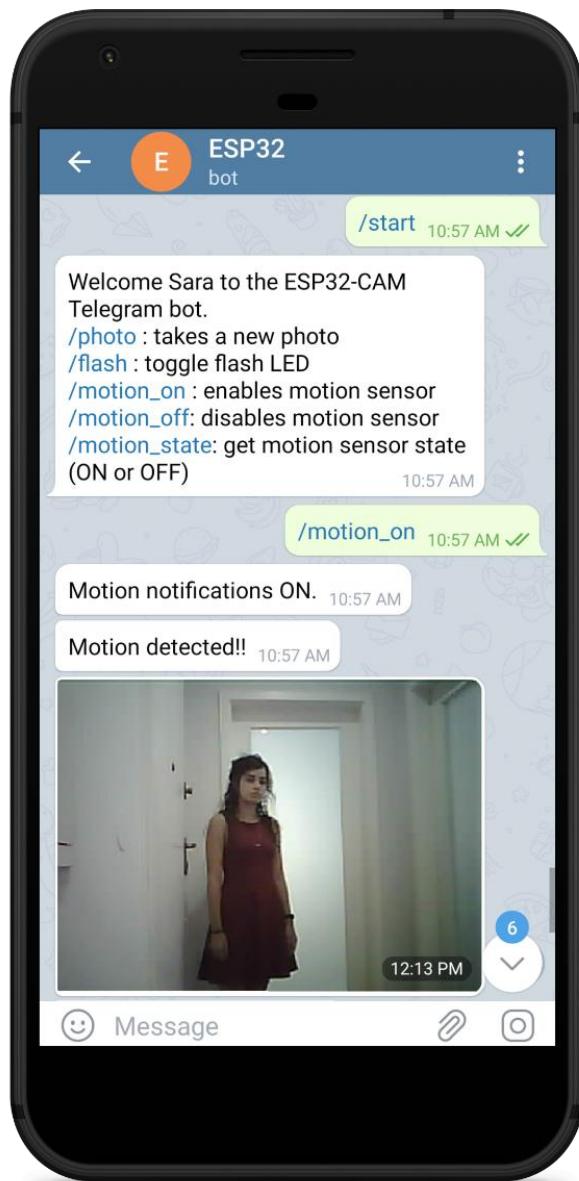
COM4
| Send
ESP32-CAM IP Address: 192.168.1.117
got response
Handle New Messages: 1
/start
got response
Handle New Messages: 1
/motion_on
Motion Detected
Connect to api.telegram.org
Connection successful
...
{"ok":true,"result":{"message_id":1387,"from":{"id":1041356751,"i
Motion Detected
Connect to api.telegram.org
Connection successful
...
{"ok":true,"result":{"message_id":1389,"from":{"id":1041356751,"i
<          >
 Autoscroll  Show timestamp  Newline  115200 baud  Clear output

```

Go to your Telegram account and open a conversation with your bot. Send the following commands and see the bot responding:

- **/start** shows the welcome message with the valid commands;

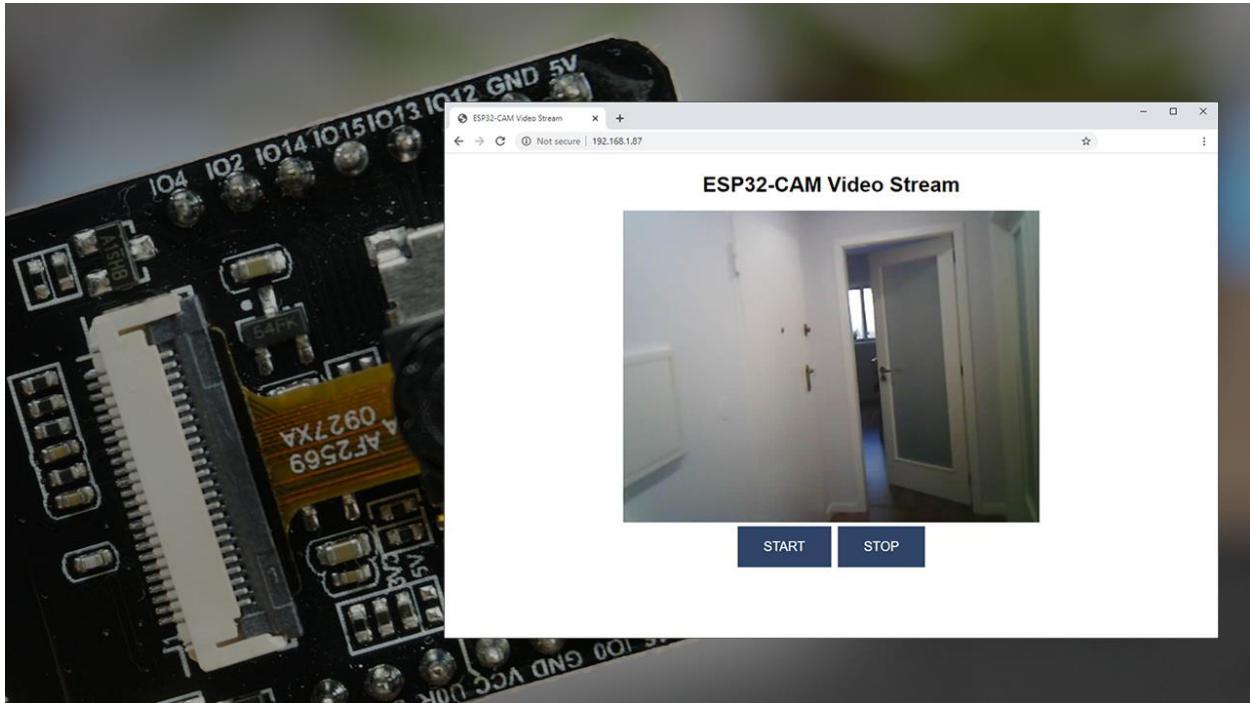
- **/motion_on** enables motion notifications. When motion is detected, you'll receive a photo;
- **/motion_off** disables motion notifications. You'll not longer receive motion notifications until you enabled them again;
- **/motion_state** shows if motion notifications are enabled or disabled;
- **/flash** inverts the state of the LED flash;
- **/photo** takes a new photo and sends it to your Telegram account no matter the notification's state.



MODULE 4

**Video Streaming, Pan and
Tilt, Car Robot**

Unit 1: Video Streaming Web Server (Start and Stop Buttons)



In this project you'll build a video streaming web server with START and STOP buttons that you can access on your local network.

Compatibility: this project is compatible with any ESP32 camera board with the OV2640 camera. You just need to make sure you use the right pinout for the board you're using. To get good results with video streaming projects, we recommend using an external antenna or placing your ESP32 camera very close to your router.

Code

Copy the following code to your Arduino IDE.

CODE

https://github.com/RuiSantosdotme/ESP32-CAM-eBook/blob/master/Code/Module%204/Video%20Streaming%20Web%20Server/Video_Streaming_Web_Server.ino

```
#include "esp_camera.h"
#include <WiFi.h>
#include "esp_timer.h"
#include "img_converters.h"
#include "Arduino.h"
#include "fb_gfx.h"
#include "soc/soc.h"                  // disable brownout problems
#include "soc/rtc_ctrl_reg.h"        // disable brownout problems
#include "esp_http_server.h"

// Replace with your network credentials
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

#define PART_BOUNDARY "12345678900000000000987654321"

#define CAMERA_MODEL_AI_THINKER

#if defined(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

#else
#error "Camera model not selected"
#endif
```

```

static const char* _STREAM_CONTENT_TYPE = "multipart/x-mixed-
replace;boundary=" PART_BOUNDARY;
static const char* _STREAM_BOUNDARY = "\r\n--" PART_BOUNDARY "\r\n";
static const char* _STREAM_PART = "Content-Type: image/jpeg\r\nContent-
Length: %u\r\n\r\n";

```

```

httpd_handle_t camera_httpd = NULL;
httpd_handle_t stream_httpd = NULL;

static const char PROGMEM INDEX_HTML[] = R"rawliteral(
<html>
  <head>
    <title>ESP32-CAM Video Stream</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <style>
      body {
        font-family: Arial;
        text-align: center;
        margin: 0px auto;
        padding-top: 30px;
      }
      table { margin-left: auto; margin-right: auto; }
      td { padding: 8px; }
      .button {
        background-color: #2f4468;
        border: none;
        color: white;
        padding: 20px 40px;
        text-align: center;
        text-decoration: none;
        display: inline-block;
        font-size: 20px;
        margin: 6px 3px;
        cursor: pointer;
        -webkit-touch-callout: none;
        -webkit-user-select: none;
        -khtml-user-select: none;
        -moz-user-select: none;
        -ms-user-select: none;
        user-select: none;
        -webkit-tap-highlight-color: rgba(0,0,0,0);
      }
      img { width: auto ;
        max-width: 100% ;
        height: auto ;
      }
    </style>
  </head>
  <body>
    <h1>ESP32-CAM Video Stream</h1>
    <img src="" id="photo" >
    <div align="center">
      <button class="button" onclick="startStream('start');">START</button>
      <button class="button" onclick="startStream('stop');">STOP</button>

```

```

</div>
<script>
    function startStream(x) {
        if(x=='start'){
            document.getElementById("photo").src =
window.location.href.slice(0,-1) + ":81/stream";
        }
        else if(x=='stop'){
            document.getElementById("photo").src = "";
        }
    }
    window.onload = document.getElementById("photo").src =
window.location.href.slice(0, -1) + ":81/stream";
</script>
</body>
</html>
)rawliteral";

static esp_err_t index_handler(httpd_req_t *req) {
    httpd_resp_set_type(req, "text/html");
    return httpd_resp_send(req, (const char *)INDEX_HTML, strlen(INDEX_HTML));
}

static esp_err_t stream_handler(httpd_req_t *req) {
    camera_fb_t * fb = NULL;
    esp_err_t res = ESP_OK;
    size_t _jpg_buf_len = 0;
    uint8_t * _jpg_buf = NULL;
    char * part_buf[64];

    res = httpd_resp_set_type(req, _STREAM_CONTENT_TYPE);
    if(res != ESP_OK){
        return res;
    }
    while(true){
        fb = esp_camera_fb_get();
        if (!fb) {
            Serial.println("Camera capture failed");
            res = ESP_FAIL;
        } else {
            if(fb->width > 400){
                if(fb->format != PIXFORMAT_JPEG){
                    bool jpeg_converted = frame2jpg(fb, 80, &_jpg_buf,&_jpg_buf_len);
                    esp_camera_fb_return(fb);
                    fb = NULL;
                    if(!jpeg_converted){
                        Serial.println("JPEG compression failed");
                        res = ESP_FAIL;
                    }
                } else {
                    _jpg_buf_len = fb->len;
                    _jpg_buf = fb->buf;
                }
            }
        }
    }
}

```

```

    }
    if(res == ESP_OK) {
        size_t hlen=snprintf((char *)part_buf,64,_STREAM_PART, _jpg_buf_len);
        res = httpd_resp_send_chunk(req, (const char *)part_buf, hlen);
    }
    if(res == ESP_OK) {
        res = httpd_resp_send_chunk(req, (const char *)_jpg_buf, _jpg_buf_len);
    }
    if(res == ESP_OK) {
        res = httpd_resp_send_chunk(req, _STREAM_BOUNDARY,strlen(_STREAM_BOUNDARY));
    }
    if(fb) {
        esp_camera_fb_return(fb);
        fb = NULL;
        _jpg_buf = NULL;
    } else if(_jpg_buf) {
        free(_jpg_buf);
        _jpg_buf = NULL;
    }
    if(res != ESP_OK) {
        break;
    }
    //Serial.printf("MJPG: %uB\n", (uint32_t) (_jpg_buf_len));
}
return res;
}

void startCameraServer(){
    httpd_config_t config = HTTPD_DEFAULT_CONFIG();
    config.server_port = 80;
    httpd_uri_t index_uri = {
        .uri      = "/",
        .method   = HTTP_GET,
        .handler  = index_handler,
        .user_ctx = NULL
    };
    httpd_uri_t stream_uri = {
        .uri      = "/stream",
        .method   = HTTP_GET,
        .handler  = stream_handler,
        .user_ctx = NULL
    };
    if (httpd_start(&camera_httpd, &config) == ESP_OK) {
        httpd_register_uri_handler(camera_httpd, &index_uri);
    }
    config.server_port += 1;
    config.ctrl_port += 1;
    if (httpd_start(&stream_httpd, &config) == ESP_OK) {
        httpd_register_uri_handler(stream_httpd, &stream_uri);
    }
}

void setup() {
    WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0); //disable brownout detector
}

```

```

Serial.begin(115200);
Serial.setDebugOutput(false);

camera_config_t config;
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;

if(psramFound()) {
    config.frame_size = FRAMESIZE_VGA;
    config.jpeg_quality = 10;
    config.fb_count = 2;
} else {
    config.frame_size = FRAMESIZE_SVGA;
    config.jpeg_quality = 12;
    config.fb_count = 1;
}

// Camera init
esp_err_t err = esp_camera_init(&config);
if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
}
// Wi-Fi connection
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}
Serial.println("");
Serial.println("WiFi connected");

Serial.print("Camera Stream Ready! Go to: http://");
Serial.println(WiFi.localIP());

```

```
// Start streaming web server
startCameraServer();
}

void loop() {
```

How the Code Works

The functions for video streaming are a bit complex to explain. We won't go into detail about how they work. We'll take a quick look at the code so that you understand how it works.

First, make sure you insert your network credentials in the following variables:

```
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
```

Then, check that you have the right pinout for the board you're using. For the ESP32-CAM AI-Thinker, this is the pinout.

```
#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
```

Web Page

The INDEX_HTML variable contains the HTML text to build the web page.

```

static const char PROGMEM INDEX_HTML[] = R"rawliteral(
<html>
  <head>
    <title>ESP32-CAM Video Stream</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <style>
      body {
        font-family: Arial;
        text-align: center;
        margin: 0px auto;
        padding-top: 30px;
      }
      table { margin-left: auto; margin-right: auto; }
      td { padding: 8px; }
      .button {
        background-color: #2f4468;
        border: none;
        color: white;
        padding: 20px 40px;
        text-align: center;
        text-decoration: none;
        display: inline-block;
        font-size: 20px;
        margin: 6px 3px;
        cursor: pointer;
        -webkit-touch-callout: none;
        -webkit-user-select: none;
        -khtml-user-select: none;
        -moz-user-select: none;
        -ms-user-select: none;
        user-select: none;
        -webkit-tap-highlight-color: rgba(0,0,0,0);
      }
      img { width: auto ;
        max-width: 100% ;
        height: auto ;
      }
    </style>
  </head>
  <body>
    <h1>ESP32-CAM Video Stream</h1>
    <img src="" id="photo" >
    <div align="center">
      <button class="button" onclick="startStream('start');">START</button>
      <button class="button" onclick="startStream('stop');">STOP</button>
    </div>
    <script>
      function startStream(x) {
        if(x=='start'){
          document.getElementById("photo").src =
window.location.href.slice(0,-1) + ":81/stream";
        }
        else if(x=='stop'){
          document.getElementById("photo").src = "";
        }
      }
    </script>
  </body>
</html>
)rawliteral";

```

```

        }
    }
    window.onload = document.getElementById("photo").src =
window.location.href.slice(0, -1) + ":81/stream";
</script>
</body>
</html>
) rawliteral";

```

About the Video Streaming Web Server: it serves a web page that contains:

- One **heading**: “ESP32-CAM Video Stream”. This goes between the `<h1>` and `</h1>` tags. You can change the text if you want.

```
<h1>ESP32-CAM Video Stream</h1>
```

- The **video streaming**: it is embedded on the web page as an image. When you first access to the web server, the source of the image is not defined (`src=""`).

```
<img src="" id="photo" >
```

The camera web server functions in our code create a sequence of jpeg images (video) on the `:81/stream` URL. So, to display the video streaming, the image source should be `<ESP_IP_ADDRESS>:81/stream`. This is done when you click the START button.

- Two buttons: **START** and **STOP**.

```
<button class="button" onclick="startStream('start');">START</button>
<button class="button" onclick="startStream('stop');">STOP</button>
```

When you click the START or STOP buttons, you’re calling the `startStream()` JavaScript function.

In case of the START button, we define the source of the image to `:81/stream` so that it embeds the video.

```
<script>
  function startStream(x) {
    if(x=='start'){
      document.getElementById("photo").src = window.location.href.slice(0,-1)
        + ":81/stream";
    }
  }

```

In case you've pressed the STOP button, the source is empty, so it won't show the video stream.

```
else if(x=='stop') {
    document.getElementById("photo").src = "";
}
```

Every time you refresh the web page, it automatically displays the stream.

```
window.onload = document.getElementById("photo").src =
window.location.href.slice(0, -1) + ":81/stream";
```

That's pretty much how the START and STOP buttons work.

setup()

You should already be familiar with what happens in the `setup()`: initialize the camera, connect to Wi-Fi and display the ESP32-CAM IP address.

Finally, call the `startCameraWebServer()` function to start the web server.

```
// Start streaming web server
startCameraServer();
```

Demonstration

After uploading the code, open the Serial Monitor at a baud rate of 115200 to get the ESP32 IP address.



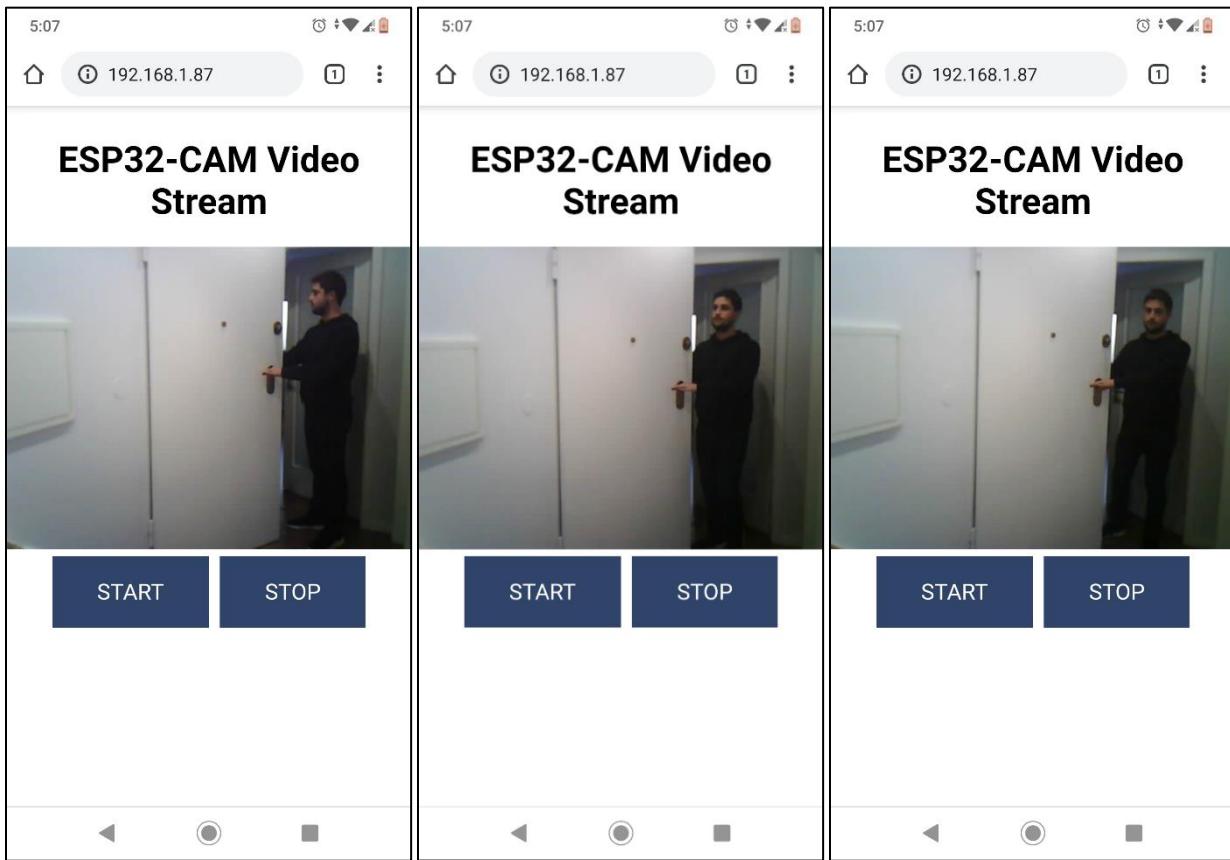
The screenshot shows the Arduino Serial Monitor window. The text output is as follows:

```
load:0x40080400, len:6352
entry 0x400806b8
.
WiFi connected
Camera Stream Ready! Go to: http://192.168.1.87
```

At the bottom of the window, there are several status indicators and settings:

- Autoscroll
- Show timestamp
- Both NL & CR
- 115200 baud
- Clear output

Open a browser on your computer or smartphone and type the ESP32-CAM IP address to see the video streaming web server.



You can click the **STOP** and **START** buttons to stop or start video streaming again.

You can also access the following URL to see the video stream source.

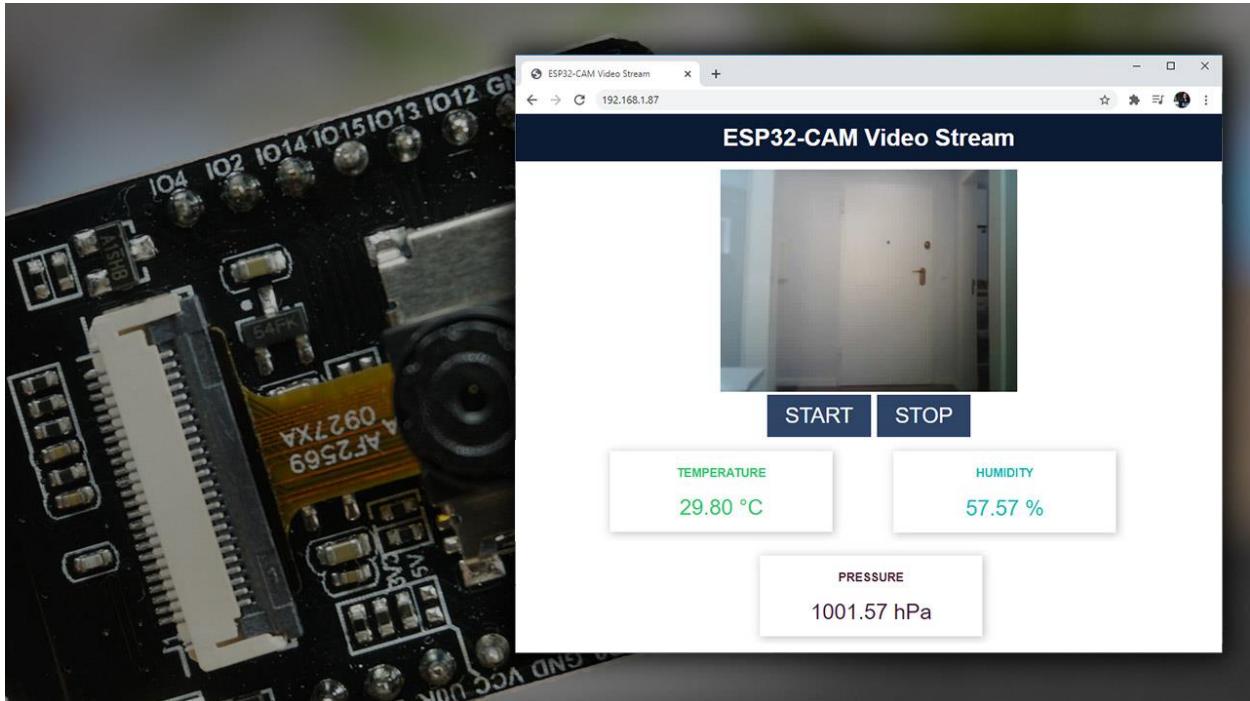
<ESP_IP_ADDRESS>:81/stream

You can only access one URL at a time, so close the previous tab to get access to this one.

Wrapping Up

In this project, you've learned how to embed video stream from the ESP32-CAM on a web page. If you're getting issues with video streaming (for example: very slow video streaming and lag) you may consider getting an [external antenna](#) for your board to increase the Wi-Fi range. We've found that in case of the ESP32-CAM AI-Thinker, if you're not very close to the router, the video stream will crash constantly. With an external antenna this doesn't happen and the project works flawlessly.

Unit 2: Video Streaming Web Server with Sensor Readings



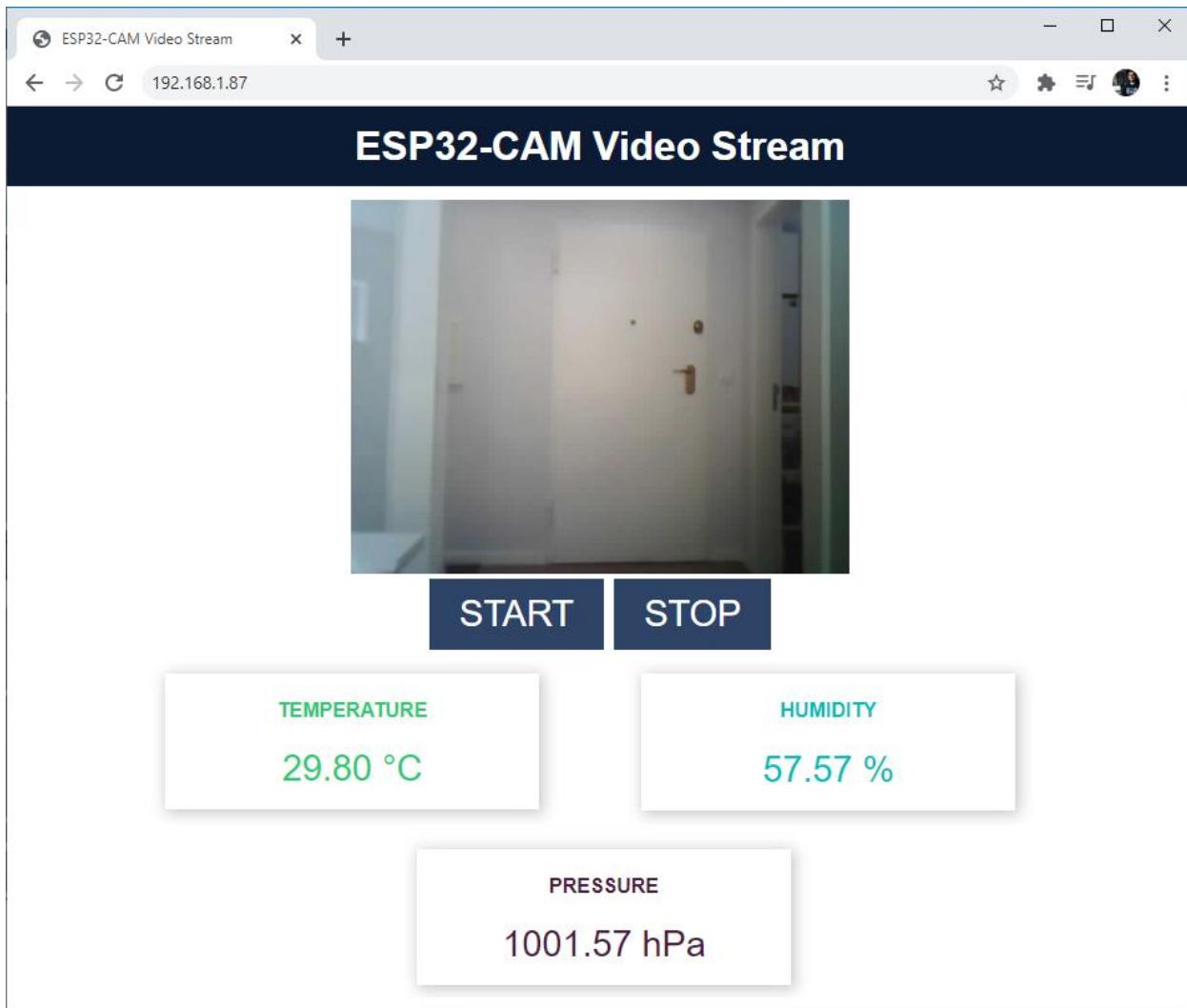
In this Unit, we'll take the previous project further and add sensor readings from a BME280 sensor (temperature, humidity and pressure) to the video streaming web server page. The readings are updated automatically on the web page using server-sent events (SSE).

The code will be modified to use the AsyncWebServer library to build the web server.

Compatibility: this project is compatible with the ESP32 Ai-Thinker board or any other board with two accessible GPIOs to connect the BME280 sensor using I2C communication.

Project Overview

The following print screen shows the web server you'll build in this Unit.



- The START and STOP buttons start and stop the video streaming. When you press the STOP button, the video disappears from the web page.
- There are three cards to display temperature, humidity and pressure from the BME280 sensor.
- The sensor readings are updated automatically every 30 seconds using server-sent events (SSE). You can change that period of time on the code. SSE allow a web page (client) to get updates from a server (ESP32-CAM).

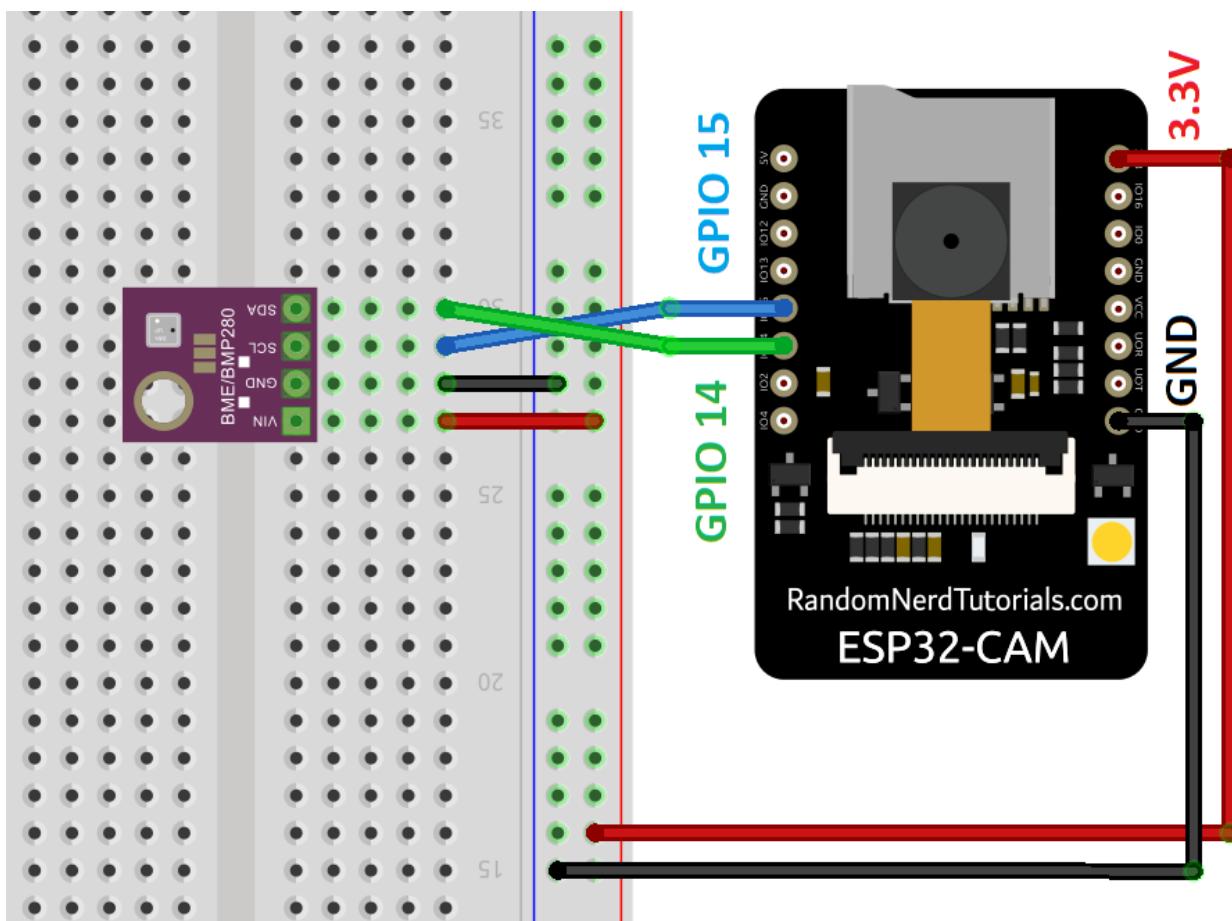
Parts Required

For this project, you need the following parts:

- [ESP32-CAM](#)
 - [BME280 sensor](#)
 - [Jumper wires](#)
 - [Breadboard \(optional\)](#)

Schematic Diagram

Wire the BME280 to the ESP32-CAM by following the next schematic diagram. The sensor SDA pin connects to GPIO 14 and the SCL pin to GPIO 15.



Installing Libraries

To build the web server, we'll use the [ESPAsyncWebServer library](#). This library also requires the [AsyncTCP library](#) to work properly. Follow the next steps to install those libraries if you haven't already.

Installing the ESPAsyncWebServer library

Follow the next steps to install the ESPAsyncWebServer library:

- 1) [Click here to download the ESPAsyncWebServer library](#). You should have a .zip folder in your *Downloads* folder.
- 2) Unzip the .zip folder and you should get *ESPAsyncWebServer-master* folder.
- 3) Rename your folder from *ESPAsyncWebServer-master* to *ESPAsyncWebServer*.
- 4) Move the *ESPAsyncWebServer* folder to your Arduino IDE installation libraries folder.
- 5) Finally, re-open your Arduino IDE.

Alternatively, after downloading the library, you can go to **Sketch** ▶ **Include Library** ▶ **Add .ZIP library...** and select the library you've just downloaded.

Installing the AsyncTCP library

The ESPAsyncWebServer library requires the AsyncTCP library to work. Follow the next steps to install the AsyncTCP library:

- 1) [Click here to download the AsyncTCP library](#). You should have a .zip folder in your Downloads folder.
- 2) Unzip the .zip folder and you should get *AsyncTCP-master* folder.
- 3) Rename your folder from *AsyncTCP-master* to *AsyncTCP*.
- 4) Move the *AsyncTCP* folder to your Arduino IDE installation libraries folder.

- 5) Finally, re-open your Arduino IDE.

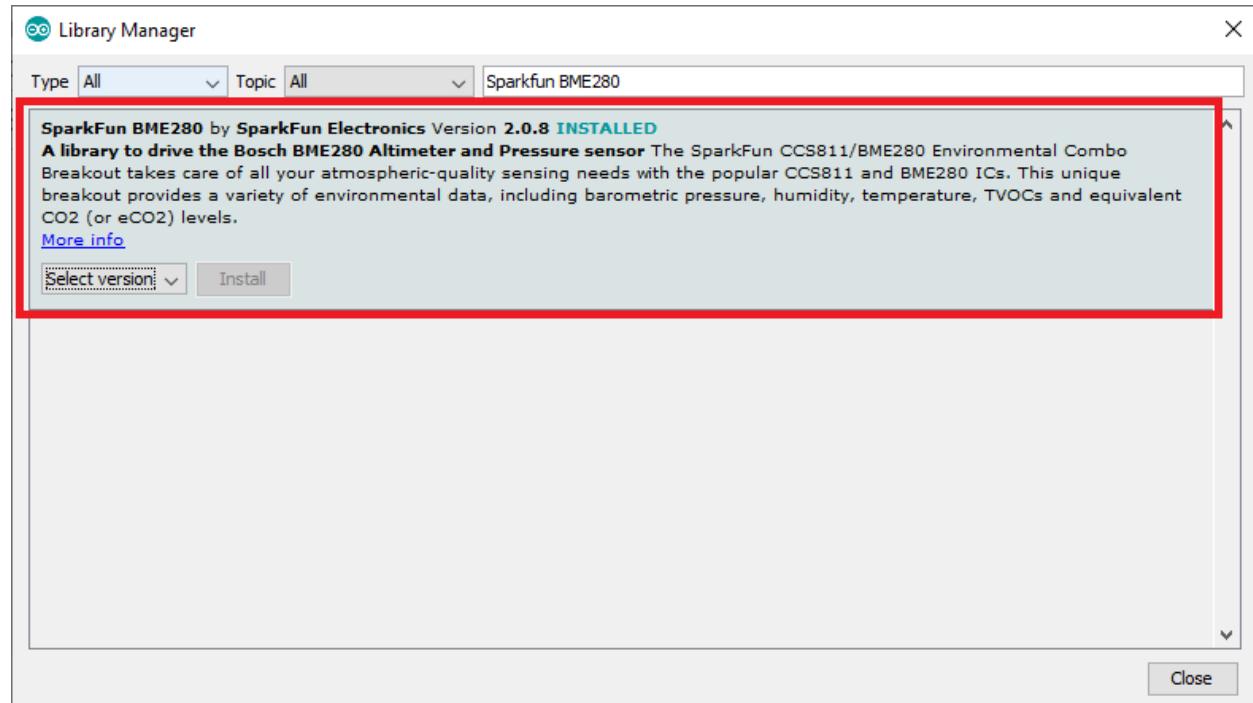
Alternatively, after downloading the library, you can go to **Sketch > Include Library > Add .ZIP library...** and select the library you've just downloaded.

BME280 SparkFun Library

To read from the BME280 sensor, we'll use the BME280 Sparkfun library. Follow the next steps to install the library:

Note: the Adafruit_BME280 sensor library conflicts with the camera library for the ESP32-CAM. Use the BME280 Sparkfun library instead. It is easy to use and works well.

- 1) Go to **Sketch > Include Library > Manage Libraries**.
- 2) Search for "**Sparkfun BME280**".
- 3) Install the library.



Code

Upload the following code to your ESP32-CAM. Insert your network credentials (SSID and password) and the code will work straight away.

CODE

https://github.com/RuiSantosdotme/ESP32-CAM-eBook/blob/master/Code/Module_4/Video_Streaming_Web_Server_Sensor_Readings/Video_Streaming_Web_Server_Sensor_Readings.ino

```
#include "Arduino.h"
#include "esp_camera.h"
#include "ESPAsyncWebServer.h"
#include <WiFi.h>
#include <Wire.h>
#include "SparkFunBME280.h"
// Replace with your network credentials
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = " REPLACE_WITH_YOUR_PASSWORD";
// Define I2C Pins for BME280
#define I2C_SDA 14
#define I2C_SCL 15
BME280 bme;
// ESP32 AI Thinker Module Pinout
#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
typedef struct {
    camera_fb_t * fb;
    size_t index;
} camera_frame_t;

#define PART_BOUNDARY "12345678900000000000987654321"
static const char* STREAM_CONTENT_TYPE = "multipart/x-mixed-replace;boundary=" PART_BOUNDARY;
```

```

static const char* STREAM_BOUNDARY = "\r\n--" PART_BOUNDARY "\r\n";
static const char* STREAM_PART = "Content-Type: %s\r\nContent-Length:
%u\r\n\r\n";
static const char * JPG_CONTENT_TYPE = "image/jpeg";
static const char * BMP_CONTENT_TYPE = "image/x-windows-bmp";
class AsyncJpegStreamResponse: public AsyncAbstractResponse {
private:
    camera_frame_t _frame;
    size_t _index;
    size_t _jpg_buf_len;
    uint8_t * _jpg_buf;
    uint64_t lastAsyncRequest;
public:
    AsyncJpegStreamResponse() {
        _callback = nullptr;
        _code = 200;
        _contentLength = 0;
        _contentType = STREAM_CONTENT_TYPE;
        _sendContentLength = false;
        _chunked = true;
        _index = 0;
        _jpg_buf_len = 0;
        _jpg_buf = NULL;
        lastAsyncRequest = 0;
        memset(&_frame, 0, sizeof(camera_frame_t));
    }
    ~AsyncJpegStreamResponse() {
        if(_frame.fb) {
            if(_frame.fb->format != PIXFORMAT_JPEG) {
                free(_jpg_buf);
            }
            esp_camera_fb_return(_frame.fb);
        }
    }
    bool _sourceValid() const {
        return true;
    }
    virtual size_t _fillBuffer(uint8_t *buf, size_t maxlen) override {
        size_t ret = _content(buf, maxlen, _index);
        if(ret != RESPONSE_TRY AGAIN) {
            _index += ret;
        }
        return ret;
    }
    size_t _content(uint8_t *buffer, size_t maxlen, size_t index) {
        if(!_frame.fb || _frame.index == _jpg_buf_len) {
            if(index && _frame.fb) {
                uint64_t end = (uint64_t)micros();
                int fp = (end - lastAsyncRequest) / 1000;
                log_printf("Size: %uKB, Time: %ums (%.1ffps)\n",
                _jpg_buf_len/1024, fp);
                lastAsyncRequest = end;
                if(_frame.fb->format != PIXFORMAT_JPEG) {
                    free(_jpg_buf);
                }
            }
        }
    }
}

```

```

    }
    esp_camera_fb_return(_frame.fb);
    _frame.fb = NULL;
    _jpg_buf_len = 0;
    _jpg_buf = NULL;
}
if(maxLen < (strlen(STREAM_BOUNDARY) + strlen(STREAM_PART) +
strlen(JPG_CONTENT_TYPE) + 8)) {
    //log_w("Not enough space for headers");
    return RESPONSE_TRY AGAIN;
}
//get frame
_frame.index = 0;

_frame.fb = esp_camera_fb_get();
if (_frame.fb == NULL) {
    log_e("Camera frame failed");
    return 0;
}

if(_frame.fb->format != PIXFORMAT_JPEG) {
    unsigned long st = millis();
    bool jpeg_converted = frame2jpg(_frame.fb, 80, &_jpg_buf,
&_jpg_buf_len);
    if(!jpeg_converted){
        log_e("JPEG compression failed");
        esp_camera_fb_return(_frame.fb);
        _frame.fb = NULL;
        _jpg_buf_len = 0;
        _jpg_buf = NULL;
        return 0;
    }
    log_i("JPEG: %lums, %uB", millis() - st, _jpg_buf_len);
} else {
    _jpg_buf_len = _frame.fb->len;
    _jpg_buf = _frame.fb->buf;
}

//send boundary
size_t blen = 0;
if(index){
    blen = strlen(STREAM_BOUNDARY);
    memcpy(buffer, STREAM_BOUNDARY, blen);
    buffer += blen;
}
//send header
size_t hlen = sprintf((char *)buffer, STREAM_PART,
JPG_CONTENT_TYPE, _jpg_buf_len);
buffer += hlen;
//send frame
hlen = maxLen - hlen - blen;
if(hlen > _jpg_buf_len){
    maxLen -= hlen - _jpg_buf_len;
    hlen = _jpg_buf_len;
}

```

```

    }
    memcpy(buffer, _jpg_buf, hlen);
    _frame.index += hlen;
    return maxLen;
}
size_t available = _jpg_buf_len - _frame.index;
if(maxLen > available){
    maxLen = available;
}
memcpy(buffer, _jpg_buf+_frame.index, maxLen);
_frame.index += maxLen;

return maxLen;
}
};

void streamJpg(AsyncWebServerRequest *request) {
    AsyncJpegStreamResponse *response = new AsyncJpegStreamResponse();
    if(!response){
        request->send(501);
        return;
    }
    response->addHeader("Access-Control-Allow-Origin", "*");
    request->send(response);
}

float temperature;
float humidity;
float pressure;
// Get BME280 sensor readings and return them as a String variable
String getReadings(){
    temperature = bme.readTempC();
    //temperature = bme.readTempF();
    humidity = bme.readFloatHumidity();
    pressure = bme.readFloatPressure() / 100.0F;
    String message = "Temperature: " + String(temperature) + " °C \n";
    message += "Humidity: " + String(humidity) + " % \n";
    message += "Pressure: " + String(pressure) + " hPa \n";
    return message;
}
const char index_html[] PROGMEM = R"rawliteral(
<html>
    <head>
        <title>ESP32-CAM Video Stream</title>
        <meta name="viewport" content="width=device-width, initial-scale=1">
        <style>
            .button {background-color: #2f4468; border: none; color: white; padding: 12px 24px; text-align: center; text-decoration: none;
                display: inline-block; font-size: 1.8rem; margin: 4px 2px; cursor: pointer; -webkit-touch-callout: none; -webkit-user-select: none;
                -khtml-user-select: none; -moz-user-select: none; -ms-user-select: none; user-select: none; -webkit-tap-highlight-color: rgba(0,0,0,0);}
            img { width: auto ; max-width: 400px ; height: auto ; padding-top: 20px;}
            html {font-family: Arial; display: inline-block; text-align: center;}
            body { margin: 0; }
        </style>
    </head>
    <body>
        <h1>ESP32-CAM Video Stream</h1>
        
    </body>
</html>
)rawliteral";

```

```

.topnav { overflow: hidden; background-color: #0c1b35; color: white;
font-size: 1.7rem; }
    .content { padding: 12px; }
        .card { background-color: white; box-shadow: 2px 2px 12px 1px
rgba(140,140,140,.5); }
            .cards { max-width: 300px; margin: 0 auto; display: grid; grid-gap: 2rem;
grid-template-columns: repeat(auto-fit, minmax(300px, 1fr)); }
                .reading { font-size: 1.8rem; }
                .card.temperature { color: #3fca6b; }
                .card.humidity { color: #17bebb; }
                .card.pressure { color: #4b1d3f; }
            </style>
</head>
<body>
<div class="topnav">
    <h3>ESP32-CAM Video Stream</h3>
</div>
<img src="" id="photo" >
<div align="center">
    <button class="button" onclick="startStream('start');">START</button>
    <button class="button" onclick="startStream('stop');">STOP</button>
</div>
<div class="content">
    <div class="cards">
        <div class="card temperature">
            <h4><i class="fas fa-thermometer-half"></i> TEMPERATURE</h4><p><span
class="reading"><span id="temp">%TEMPERATURE%</span> &deg;C</span></p>
            </div>
            <div class="card humidity">
                <h4><i class="fas fa-tint"></i> HUMIDITY</h4><p><span class="reading"><span
id="hum">%HUMIDITY%</span> &percnt;</span></p>
            </div>
            <div class="card pressure">
                <h4><i class="fas fa-angle-double-down"></i> PRESSURE</h4><p><span
class="reading"><span id="pres">%PRESSURE%</span> hPa</span></p>
            </div>
        </div>
    </script>
    function startStream(x) {
        if(x=='start'){
            document.getElementById("photo").src = window.location.href +
"stream";
        }
        else if(x=='stop'){
            document.getElementById("photo").src = "";
        }
    }
    window.onload = document.getElementById("photo").src =
window.location.href + "stream";
    if (!!window.EventSource) {
        var source = new EventSource('/events');
        source.addEventListener('open', function(e) {
            console.log("Events Connected");
        }, false);
    }
</body>

```

```

        source.addEventListener('error', function(e) {
            if (e.target.readyState != EventSource.OPEN) {
                console.log("Events Disconnected");
            }
        }, false);
        source.addEventListener('message', function(e) {
            console.log("message", e.data);
        }, false);
        source.addEventListener('temperature', function(e) {
            console.log("temperature", e.data);
            document.getElementById("temp").innerHTML = e.data;
        }, false);
        source.addEventListener('humidity', function(e) {
            console.log("humidity", e.data);
            document.getElementById("hum").innerHTML = e.data;
        }, false);
        source.addEventListener('pressure', function(e) {
            console.log("pressure", e.data);
            document.getElementById("pres").innerHTML = e.data;
        }, false);
    }
</script>
</body>
</html>
)rawliteral";
String processor(const String& var) {
    getReadings();
    //Serial.println(var);
    if(var == "TEMPERATURE") {
        return String(temperature);
    }
    else if(var == "HUMIDITY") {
        return String(humidity);
    }
    else if(var == "PRESSURE") {
        return String(pressure);
    }
}
unsigned long lastTime = 0;
unsigned long timerDelay = 30000;
AsyncWebServer server(80);
AsyncEventSource events("/events");
void setup(){
    Serial.begin(115200);
    Serial.setDebugOutput(false);
    // Init BME280 sensor
    Wire.begin(I2C_SDA, I2C_SCL);
    bme.settings.commInterface = I2C_MODE;
    bme.settings.I2CAddress = 0x76;
    bme.settings.runMode = 3;
    bme.settings.tStandby = 0;
    bme.settings.filter = 0;
    bme.settings.tempOverSample = 1;
    bme.settings.pressOverSample = 1;
}

```

```

bme.settings.humidOverSample = 1;
bme.begin();
camera_config_t config;
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;

if(psramFound()) {
    config.frame_size = FRAMESIZE_VGA;
    config.jpeg_quality = 15;
    config.fb_count = 2;
} else {
    config.frame_size = FRAMESIZE_SVGA;
    config.jpeg_quality = 12;
    config.fb_count = 1;
}
// Camera init
esp_err_t err = esp_camera_init(&config);
if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
}
// Wi-Fi connection
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}
Serial.println("");
Serial.println("WiFi connected");
Serial.print("Camera Stream Ready! Go to: http://");
Serial.println(WiFi.localIP());
// Handle Web Server
server.on("/", HTTP_GET, [] (AsyncWebServerRequest *request) {
    request->send_P(200, "text/html", index_html, processor);
});
server.on("/stream", HTTP_GET, streamJpg);

```

```

// Handle Web Server Events
events.onConnect([](AsyncEventSourceClient *client) {
    if(client->lastId()) {
        Serial.printf("Client reconnected! Last message ID that it got is:
%u\n", client->lastId());
    }
    // send event with message "hello!", id current millis
    // and set reconnect delay to 1 second
    client->send("hello!", NULL, millis(), 10000);
});
server.addHandler(&events);
server.begin();
}

void loop(){
    if ((millis() - lastTime) > timerDelay) {
        String readings = getReadings();
        Serial.println(readings);
        lastTime = millis();
        // Send Events to the Web Server with the Sensor Readings
        events.send("ping", NULL, millis());
        events.send(String(temperature).c_str(), "temperature", millis());
        events.send(String(humidity).c_str(), "humidity", millis());
        events.send(String(pressure).c_str(), "pressure", millis());
    }
}

```

How the Code Works

Let's take a quick look at the relevant parts of this code.

Insert your network credentials on the following lines, so that the ESP32-CAM connects to your local network.

```
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_NETWORK";
```

Define the I2C pins that the BME280 sensor is connected to.

```
#define I2C_SDA 14
#define I2C_SCL 15
```

The following lines define the pins of the ESP32-CAM AI-Thinker module. If you're using another module, don't forget to modify the pin assignment.

```
// ESP32 AI Thinker Module Pinout
#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
```

The `AsyncJpegStreamResponse` class and `streamJpg()` function are responsible for the video streaming handling.

The `getReadings()` function gets temperature, humidity and pressure readings from the BME280 sensor.

```
String getReadings() {
    temperature = bme.readTempC();
    //temperature = bme.readTempF();
    humidity = bme.readFloatHumidity();
    pressure = bme.readFloatPressure() / 100.0F;
    String message = "Temperature: " + String(temperature) + " °C \n";
    message += "Humidity: " + String(humidity) + " % \n";
    message += "Pressure: " + String(pressure) + " hPa \n";
    return message;
}
```

The `index_html` variable contains the HTML, CSS and JavaScript to build the web server. The following tag defines a place for the video streaming on the web page.

```
<img src="" id="photo" >
```

The next lines create the ON and OFF buttons. When you click on the buttons, the `startStream()` JavaScript function is called with the “start” or “stop” argument.

```
<button class="button" onclick="startStream('start');">START</button>
<button class="button" onclick="startStream('stop');">STOP</button>
```

Then, define the cards to place the sensor readings. Notice that the place for each reading has a different id (“temp”, “hum” and “pres”).

```
<div class="card temperature">
```

```
<h4><i class="fas fa-thermometer-half"></i> TEMPERATURE</h4><p><span class="reading"><span id="temp">%TEMPERATURE%</span> &deg;C</span></p></div>
<div class="card humidity">
  <h4><i class="fas fa-tint"></i> HUMIDITY</h4><p><span class="reading"><span id="hum">%HUMIDITY%</span> &percnt;</span></p></div>
<div class="card pressure">
  <h4><i class="fas fa-angle-double-down"></i> PRESSURE</h4><p><span class="reading"><span id="pres">%PRESSURE%</span> hPa</span></p></div>
```

The `startStream()` JavaScript function starts and stops the streaming:

```
function startStream(x) {
  if(x=='start'){
    document.getElementById("photo").src = window.location.href +
"stream";
  }
  else if(x=='stop'){
    document.getElementById("photo").src = "";
  }
}
```

Handle Events

You need some JavaScript code to handle the events sent by the server (events with the sensor readings).

Create a new `EventSource` object and specify the URL of the page sending the updates. In our case, it's `/events`.

```
if (!!window.EventSource) {
  var source = new EventSource('/events');
```

Once you've instantiated an event source, you can start listening for messages from the server with `addEventListener()`.

These are the default event listeners, as shown here in the [AsyncWebServer documentation](#).

```
source.addEventListener('error', function(e) {
  if (e.target.readyState != EventSource.OPEN) {
    console.log("Events Disconnected");
  }
}, false);
```

```
source.addEventListener('message', function(e) {
  console.log("message", e.data);
}, false);
```

Then, add the event listener for "temperature", "humidity", and "pressure".

```
source.addEventListener('temperature', function(e) {
  console.log("temperature", e.data);
  document.getElementById("temp").innerHTML = e.data;
}, false);
source.addEventListener('humidity', function(e) {
  console.log("humidity", e.data);
  document.getElementById("hum").innerHTML = e.data;
}, false);
source.addEventListener('pressure', function(e) {
  console.log("pressure", e.data);
  document.getElementById("pres").innerHTML = e.data;
}, false);
```

Every 30 seconds, the ESP32 sends new readings as events to the client. When the client receives those events, it places the received data into the elements with the corresponding id on the web page.

For example, the following line searches the HTML element with the "pres" id and places the data received (e.data) into that place.

```
document.getElementById("pres").innerHTML = e.data;
```

The `processor()` function replaces the placeholders on the HTML text (%TEMPERATURE%, %HUMIDITY%, and %PRESSURE%) with the values currently saved on the `temperature`, `humidity` and `pressure` variables when you open the web browser for the first time.

```
String processor(const String& var) {
getReadings();
//Serial.println(var);
if(var == "TEMPERATURE") {
  return String(temperature);
}
else if(var == "HUMIDITY") {
  return String(humidity);
}
else if(var == "PRESSURE") {
  return String(pressure);
}
```

New readings are sent to the browser every 30 seconds. You can change that delay time on the `timerDelay` variable.

```
unsigned long timerDelay = 30000;
```

In the `setup()`, the following lines initialize the BME280 sensor.

```
Wire.begin(I2C_SDA, I2C_SCL);
bme.settings.commInterface = I2C_MODE;
bme.settings.I2CAddress = 0x76;
bme.settings.runMode = 3;
bme.settings.tStandby = 0;
bme.settings.filter = 0;
bme.settings.tempOverSample = 1;
bme.settings.pressOverSample = 1;
bme.settings.humidOverSample = 1;
bme.begin();
```

Then, handle the web server requests:

```
// Handle Web Server
server.on("/", HTTP_GET, [] (AsyncWebServerRequest *request) {
    request->send_P(200, "text/html", index_html, processor);
});
server.on("/stream", HTTP_GET, streamJpg);
```

And handle the web server events:

```
// Handle Web Server Events
events.onConnect([] (AsyncEventSourceClient *client) {
    if(client->lastId()) {
        Serial.printf("Client reconnected! Last message ID that it got is: %u\n", client->lastId());
    }
    // send event with message "hello!", id current millis
    // and set reconnect delay to 1 second
    client->send("hello!", NULL, millis(), 10000);
});

server.addHandler(&events);
```

In the `loop()`, send new events with the current sensor readings every 30 seconds.

```
void loop() {
    if ((millis() - lastTime) > timerDelay) {
        String readings = getReadings();
        Serial.println(readings);
        lastTime = millis();
        // Send Events to the Web Server with the Sensor Readings
        events.send("ping", NULL, millis());
```

```
    events.send(String(temperature).c_str(), "temperature", millis());
    events.send(String(humidity).c_str(), "humidity", millis());
    events.send(String(pressure).c_str(), "pressure", millis());
}
}
```

Demonstration

After uploading the code, open the Serial Monitor at a baud rate of 115200. Press the ESP32-CAM on-board RST button and its IP address should be printed as shown below.



The screenshot shows the Arduino Serial Monitor window titled 'COM5'. It displays the following text output:

```
.
```

WiFi connected

Camera Stream Ready! Go to: http://192.168.1.87

Temperature: 28.08 °C

Humidity: 57.13 %

Pressure: 1002.28 hPa

Temperature: 27.88 °C

Humidity: 55.80 %

Pressure: 1002.27 hPa

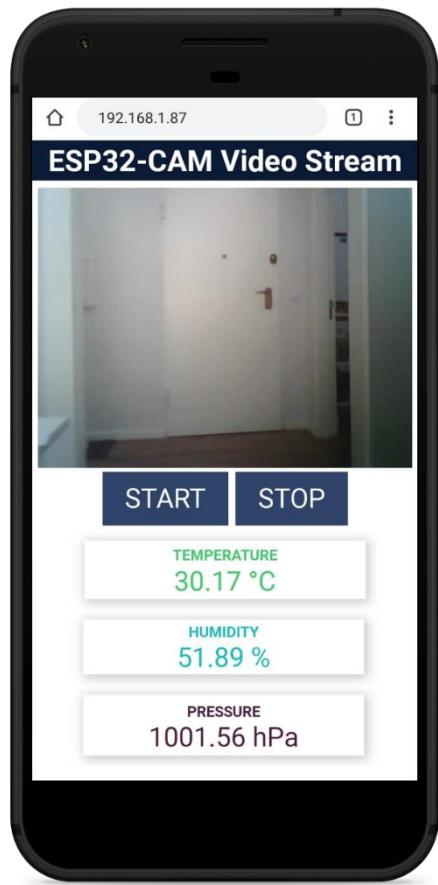
Temperature: 27.83 °C

Humidity: 55.85 %

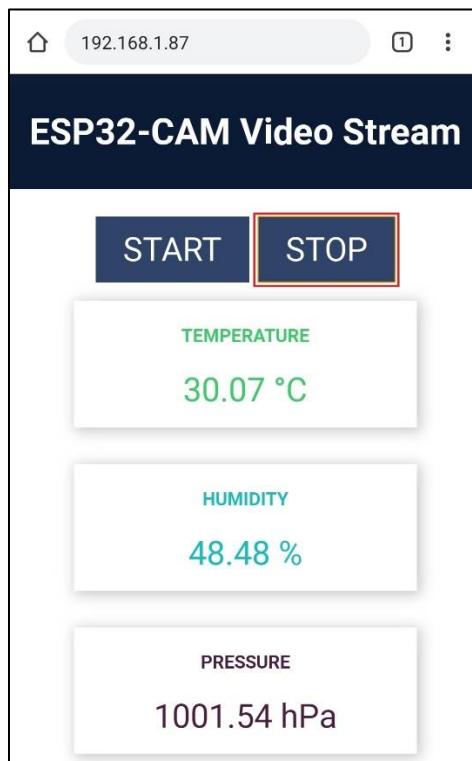
Pressure: 1002.27 hPa

At the bottom, there are three checkboxes: 'Autoscroll' (unchecked), 'Show timestamp' (unchecked), and 'Clear output' (unchecked). To the right of these are dropdown menus for 'Newline' (set to 'Newline') and '115200 baud' (set to '115200 baud').

Open a browser on your computer or smartphone and type the ESP32-CAM IP address to see the video streaming web server and the BME280 sensor readings.



You can click the **STOP** and **START** buttons to stop or start video streaming again.



Unit 3: Video Streaming IP Camera



In this project you're going to build an IP surveillance camera. The ESP32 camera hosts a video streaming web server that you can access with any device in your local network.

You can integrate this video streaming web server with popular home automation platforms like Home Assistant or Node-RED. In this tutorial, we'll show you how to integrate it with Home Assistant and Node-RED.

Compatibility: this project is compatible with any ESP32 camera board with the OV2640 camera. You just need to make sure you use the right pinout for the board you're using. To get good results with video streaming projects, we recommend using an external antenna or placing your ESP32 camera very close to your router.

Code

Copy the following code to your Arduino IDE.

CODE

https://github.com/RuiSantosdotme/ESP32-CAM-eBook/blob/master/Code/Module_4/IP_Camera/IP_Camera.ino

```
#include "esp_camera.h"
#include <WiFi.h>
#include "esp_timer.h"
#include "img_converters.h"
#include "Arduino.h"
#include "fb_gfx.h"
#include "soc/soc.h" //disable brownout problems
#include "soc/rtc_ctrl_reg.h" //disable brownout problems
#include "esp_http_server.h"

//Replace with your network credentials
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

#define PART_BOUNDARY "12345678900000000000987654321"

//AI Thinker Model
#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

static const char* _STREAM_CONTENT_TYPE = "multipart/x-mixed-replace;boundary=" PART_BOUNDARY;
static const char* _STREAM_BOUNDARY = "\r\n--" PART_BOUNDARY "\r\n";
static const char* _STREAM_PART = "Content-Type: image/jpeg\r\nContent-Length: %u\r\n\r\n";
```

```

httpd_handle_t stream_httpd = NULL;

static esp_err_t stream_handler(httpd_req_t *req) {
    camera_fb_t * fb = NULL;
    esp_err_t res = ESP_OK;
    size_t _jpg_buf_len = 0;
    uint8_t * _jpg_buf = NULL;
    char * part_buf[64];

    res = httpd_resp_set_type(req, _STREAM_CONTENT_TYPE);
    if(res != ESP_OK) {
        return res;
    }

    while(true) {
        fb = esp_camera_fb_get();
        if (!fb) {
            Serial.println("Camera capture failed");
            res = ESP_FAIL;
        } else {
            if(fb->width > 400){
                if(fb->format != PIXFORMAT_JPEG) {
                    bool jpeg_converted = frame2jpg(fb, 80, &_jpg_buf, &_jpg_buf_len);
                    esp_camera_fb_return(fb);
                    fb = NULL;
                    if(!jpeg_converted) {
                        Serial.println("JPEG compression failed");
                        res = ESP_FAIL;
                    }
                } else {
                    _jpg_buf_len = fb->len;
                    _jpg_buf = fb->buf;
                }
            }
        }
        if(res == ESP_OK){
            size_t hlen = snprintf((char *)part_buf, 64, _STREAM_PART, _jpg_buf_len);
            res = httpd_resp_send_chunk(req, (const char *)part_buf, hlen);
        }
        if(res == ESP_OK){
            res = httpd_resp_send_chunk(req, (const char *)_jpg_buf, _jpg_buf_len);
        }
        if(res == ESP_OK){
            res=httpd_resp_send_chunk(req, _STREAM_BOUNDARY, strlen(_STREAM_BOUNDARY));
        }
        if(fb) {
            esp_camera_fb_return(fb);
            fb = NULL;
            _jpg_buf = NULL;
        } else if(_jpg_buf){
            free(_jpg_buf);
            _jpg_buf = NULL;
        }
        if(res != ESP_OK){
    }
}

```

```

        break;
    }
    //Serial.printf("MJPG: %uB\n", (uint32_t) (_jpg_buf_len));
}
return res;
}

void startCameraServer(){
    httpd_config_t config = HTTPD_DEFAULT_CONFIG();
    config.server_port = 80;

    httpd_uri_t index_uri = {
        .uri      = "/",
        .method   = HTTP_GET,
        .handler  = stream_handler,
        .user_ctx = NULL
    };

    //Serial.printf("Starting web server on port: '%d'\n",
    config.server_port);
    if (httpd_start(&stream_httpd, &config) == ESP_OK) {
        httpd_register_uri_handler(stream_httpd, &index_uri);
    }
}

void setup() {
    WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0); //disable brownout detector

    Serial.begin(115200);
    Serial.setDebugOutput(false);

    camera_config_t config;
    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;

    if(psramFound()) {

```

```

    config.frame_size = FRAMESIZE_UXGA;
    config.jpeg_quality = 10;
    config.fb_count = 2;
} else {
    config.frame_size = FRAMESIZE_SVGA;
    config.jpeg_quality = 12;
    config.fb_count = 1;
}

// Camera init
esp_err_t err = esp_camera_init(&config);
if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
}
// Wi-Fi connection
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}
Serial.println("");
Serial.println("WiFi connected");

Serial.print("Camera Stream Ready! Go to: http://");
Serial.print(WiFi.localIP());

// Start streaming web server
startCameraServer();
}

void loop() {
    delay(1);
}

```

Network Credentials

Before uploading the code, search for the following lines and add your network credentials.

```

const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

```

Also, make sure you have the right pinout for the camera board you're using.

Getting the IP Address

After uploading the code, disconnect GPIO 0 from GND. Open the Serial Monitor at a baud rate of 115200. Press the ESP32-CAM on-board Reset button.

The ESP32 IP address should be printed on the Serial Monitor.

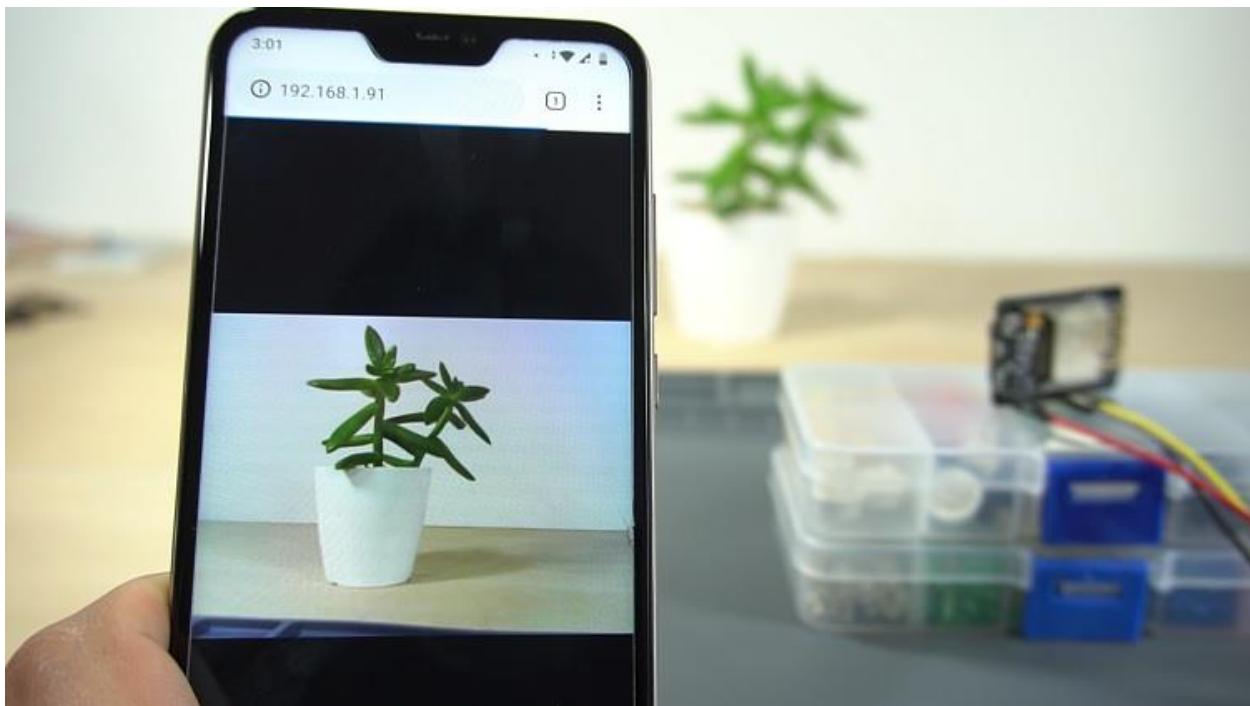


```
...
WiFi connected
Camera Stream Ready! Go to: http://192.168.1.91
```

Autoscroll Show timestamp Newline 115200 baud Clear output

Accessing the Video Streaming Server

Now, you can access your camera streaming server on your local network. Open a browser and type the ESP32-CAM IP address. A page with the current video streaming should load.



Home Assistant Integration



Having just the ESP32-CAM working via IP might be useful for most people, but you can integrate this project with Home Assistant (or any other home automation platforms). Continue reading to learn how to integrate with Home Assistant.

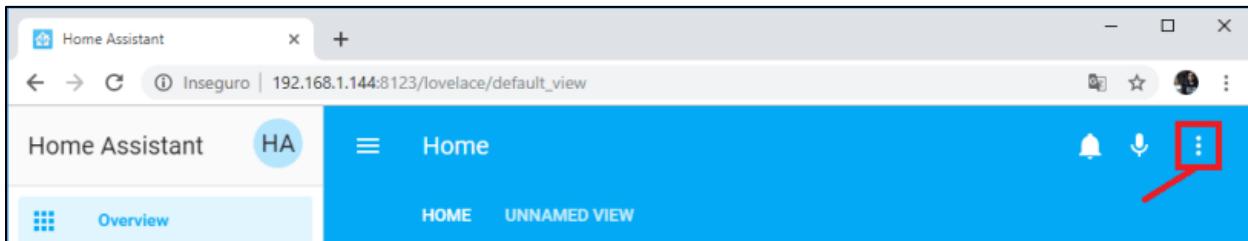
Prerequisites

You should be familiar with the Raspberry Pi. You can read the following guides:

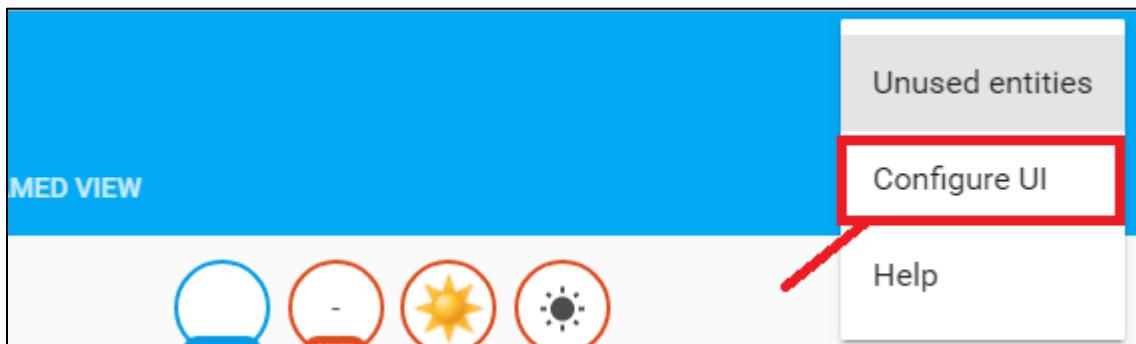
- [Getting Started with Raspberry Pi](#)
- [Getting Started with Home Assistant on Raspberry Pi](#)

Adding ESP32-CAM to Home Assistant

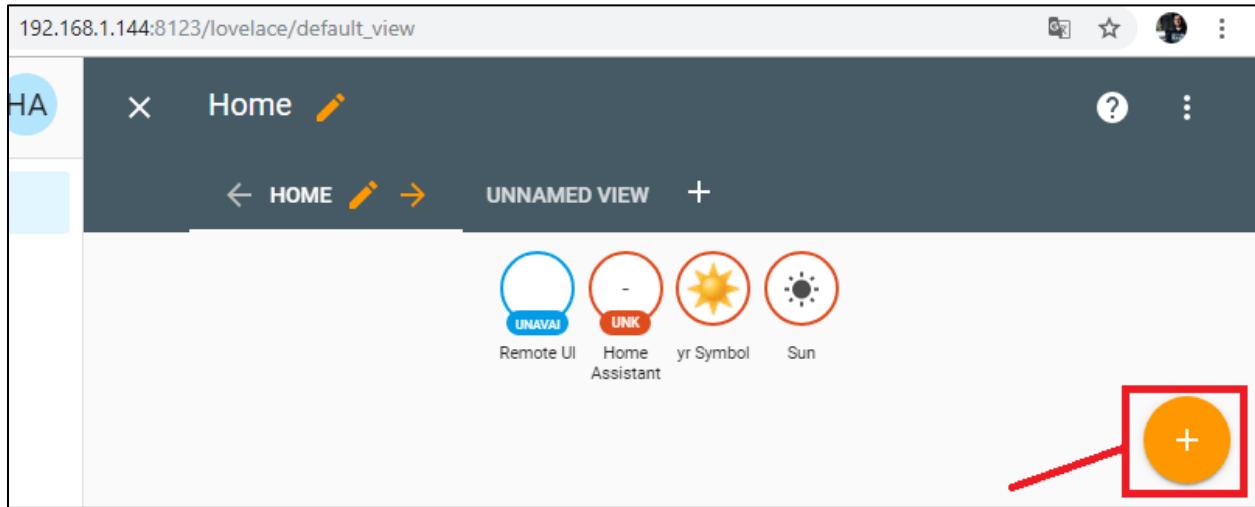
Open your Home Assistant dashboard and go to the more Settings menu.



Click on **Configure UI**:



Add a new card to your Dashboard (click the + sign):



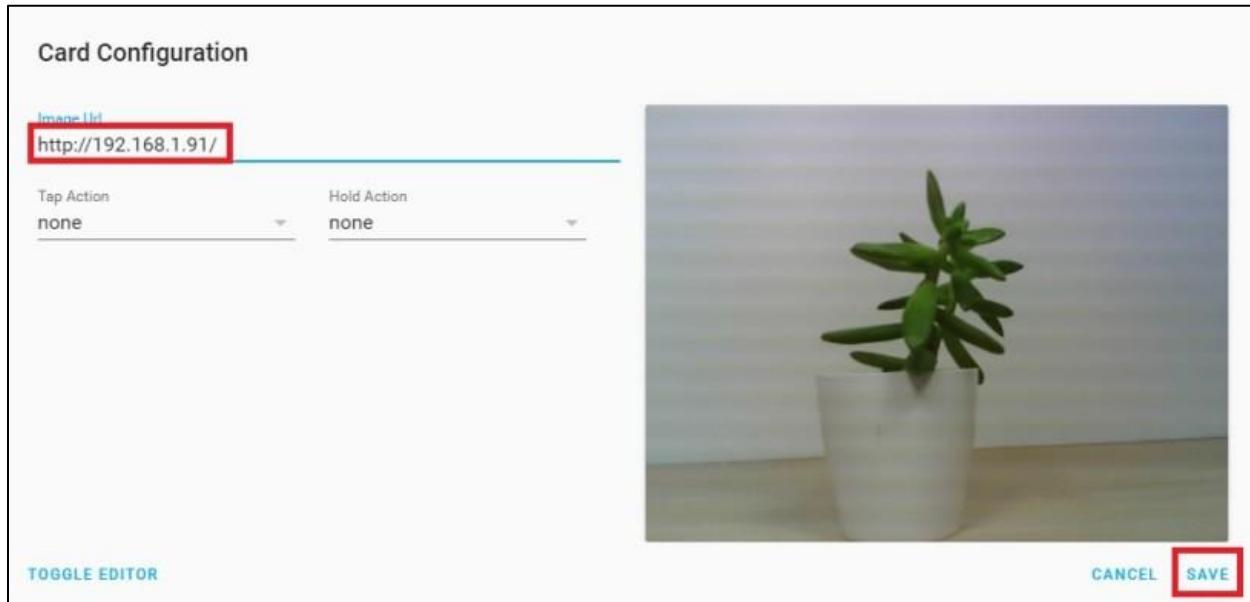
Pick a card of type **PICTURE**.

Card Configuration

Pick the card you want to add.

| | | |
|------------------|----------------|------------------|
| ALARM PANEL | CONDITIONAL | ENTITIES |
| ENTITY BUTTON | ENTITY FILTER | GAUGE |
| GLANCE | HISTORY GRAPH | HORIZONTAL STACK |
| IFRAME | LIGHT | MAP |
| MARKDOWN | MEDIA CONTROL | PICTURE |
| PICTURE ELEMENTS | PICTURE ENTITY | PICTURE GLANCE |
| PLANT STATUS | SENSOR | SHOPPING LIST |
| THERMOSTAT | VERTICAL STACK | WEATHER FORECAST |

In the Image URL field, enter your ESP32-CAM IP address. Then, click the “**SAVE**” button and return to the main dashboard.

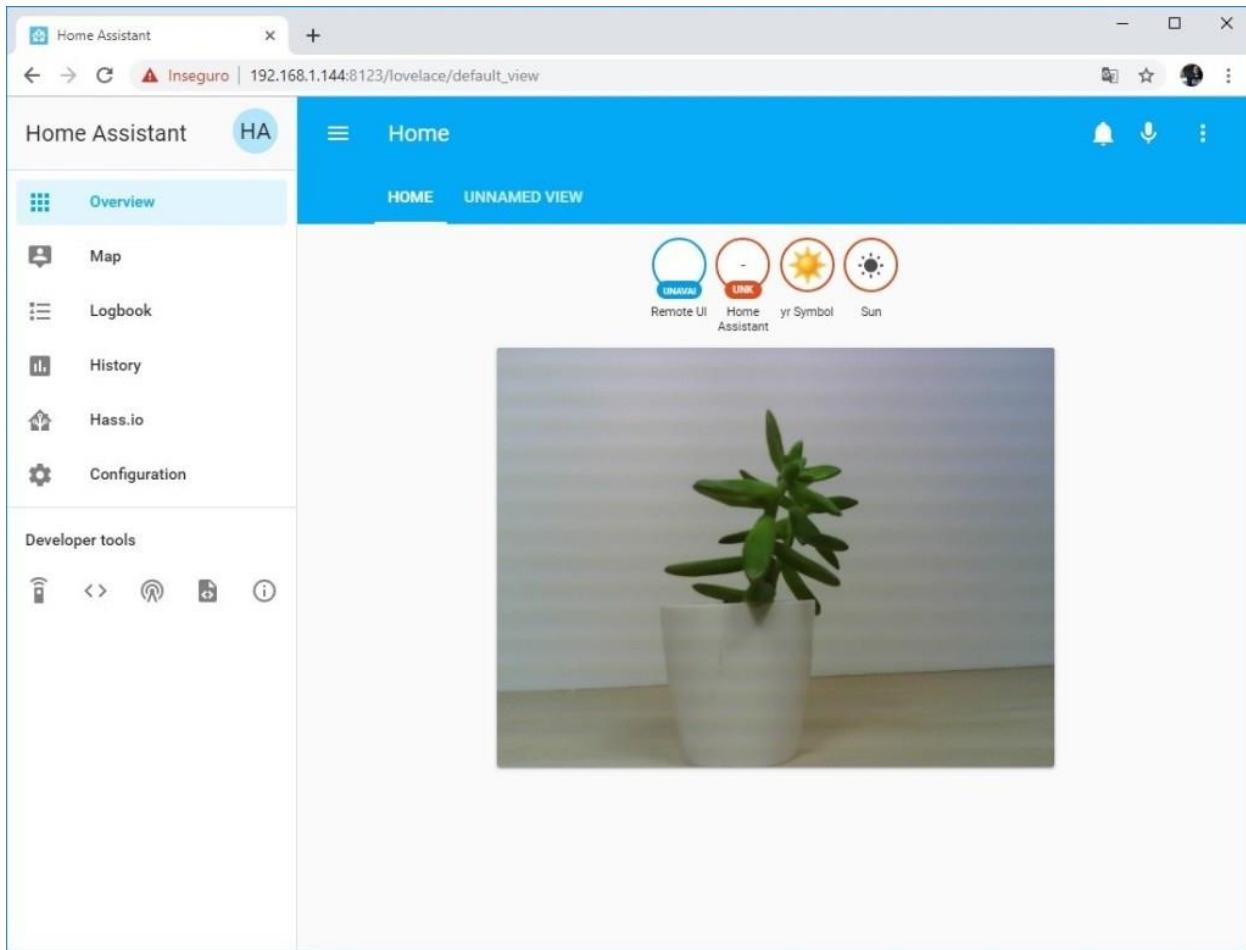


If you're using the configuration file, this is what you need to add.

Card Configuration

```
1 type: picture
2 tap_action:
3   action: none
4 hold_action:
5   action: none
6 image: 'http://192.168.1.91/'
7
```

After that, Home Assistant can display the ESP32-CAM video streaming.



Node-RED Integration

The video streaming web server also integrates with [Node-RED](#) and [Node-RED Dashboard](#). You just need to create a **Template** node and add the following:

```
<div style="margin-bottom: 10px;">

</div>
```

In the `src` attribute, you need to type your ESP32-CAM IP address, for example:

```
<div style="margin-bottom: 10px;">

</div>
```

Taking It Further

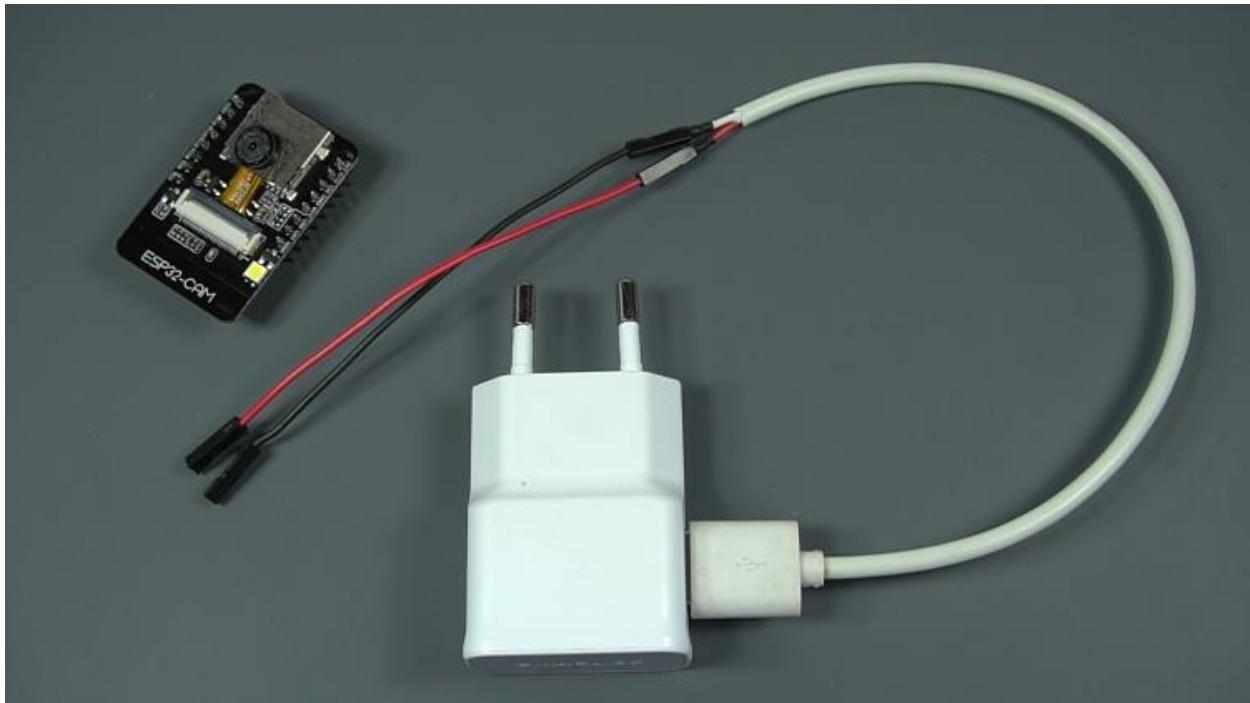
To take this project further, you can use one [fake dummy camera](#) and place the ESP32-CAM inside.



The ESP32-CAM board fits perfectly into the dummy camera enclosure.



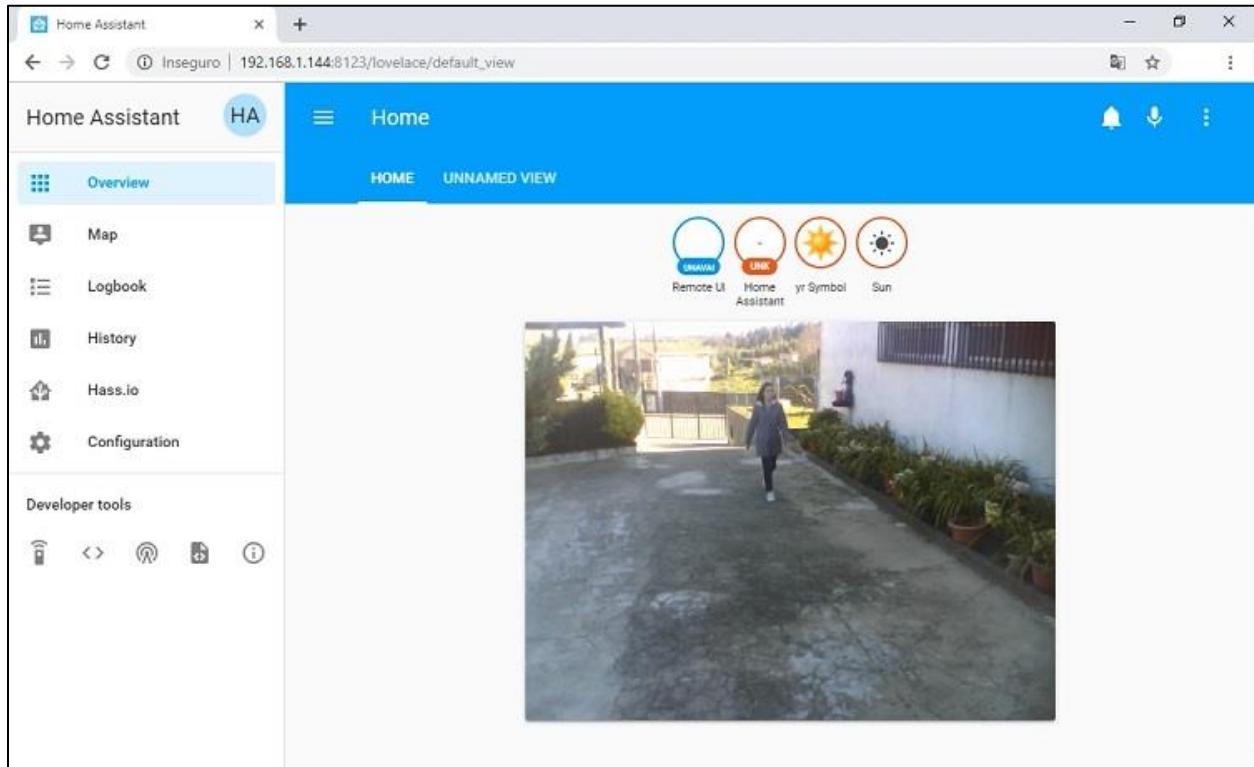
You can power the ESP32-CAM with a 5V power adapter through the GND and 5V pins. You can strip a USB cable as shown in the following image.



Place the surveillance camera in a suitable place.



After that, go to the camera IP address or to your Home Assistant dashboard and see in real time what's happening.

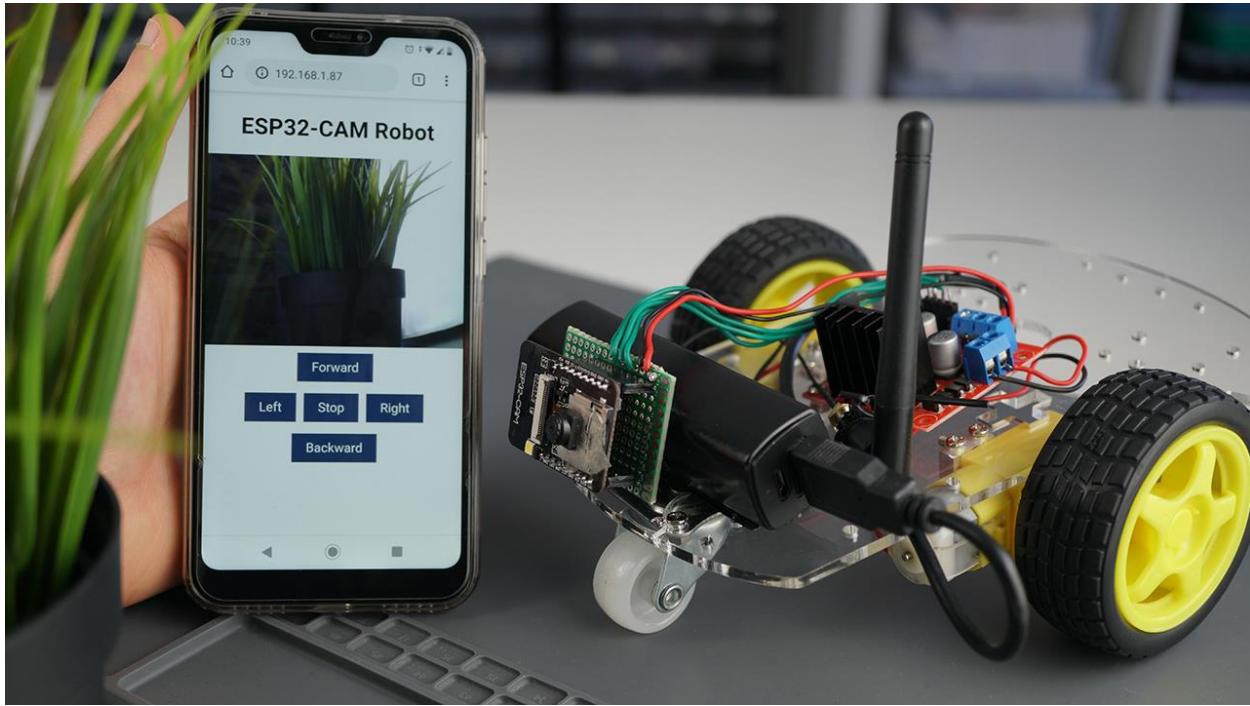


It's impressive what this tiny ESP32 camera module can. Now, we can use the surveillance camera to see in real time what's happening in the front entrance.

Wrapping Up

In this tutorial we've shown you how to build a simple video streaming web server with the ESP32-CAM board to build an IP camera. The web server we've built can be easily integrated with home automation platforms like Node-RED, Home Assistant and others.

Unit 4: Remote Controlled Car Robot with Camera (Web Server)



In this project you'll build a Wi-Fi remote controlled car robot with camera. You'll be able to control the robot using a web server that displays a video streaming of what the robot "sees". You control your robot remotely even if it's out of your sight.

Compatibility: this project requires 4 GPIOs to control the DC motors. So, it is compatible with the ESP32-CAM AI-Thinker and the TTGO T-Journal. We'll use the **ESP32-CAM AI-Thinker** board.

Project Overview

Before starting the project, we'll highlight the most important features and components used to build the robot.

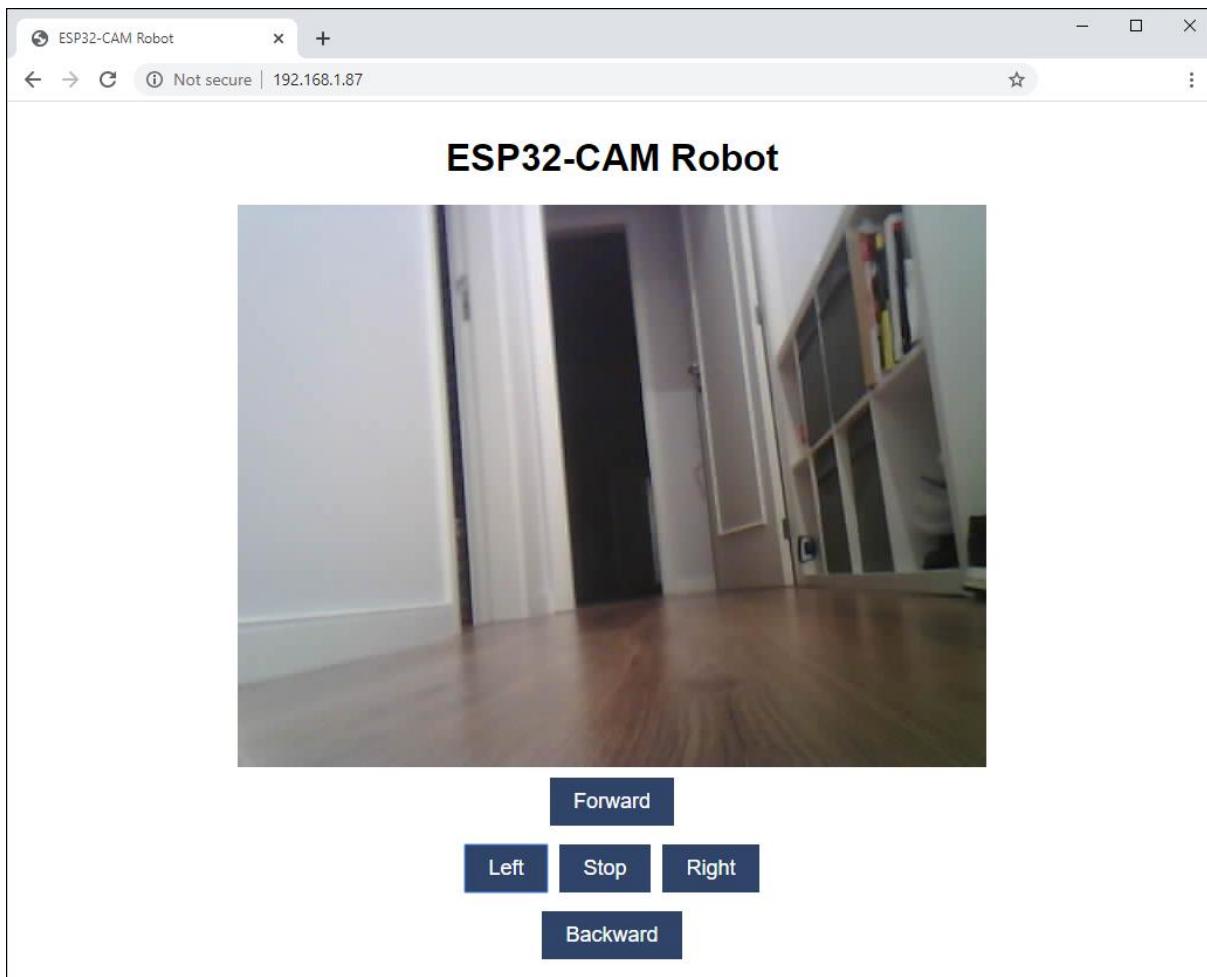
Wi-Fi

The robot will be controlled via Wi-Fi using your ESP32-CAM. We'll create a web-based interface to control the robot, that can be accessed in any device inside your local network. The web page also shows a video streaming of what the robot "sees". For good results with video streaming, we recommend using an ESP32-CAM with external antenna.

Important: without an external antenna the video stream lags and the web server is extremely slow to control the robot.

Robot Controls

The web server has 5 controls: **Forward**, **Backward**, **Left**, **Right**, and **Stop**.



The robot moves as long as you're pressing the buttons. When you release any button, the robot stops. However, we've included the Stop button that can be useful in case the ESP32 doesn't receive the stop command when you release a button.

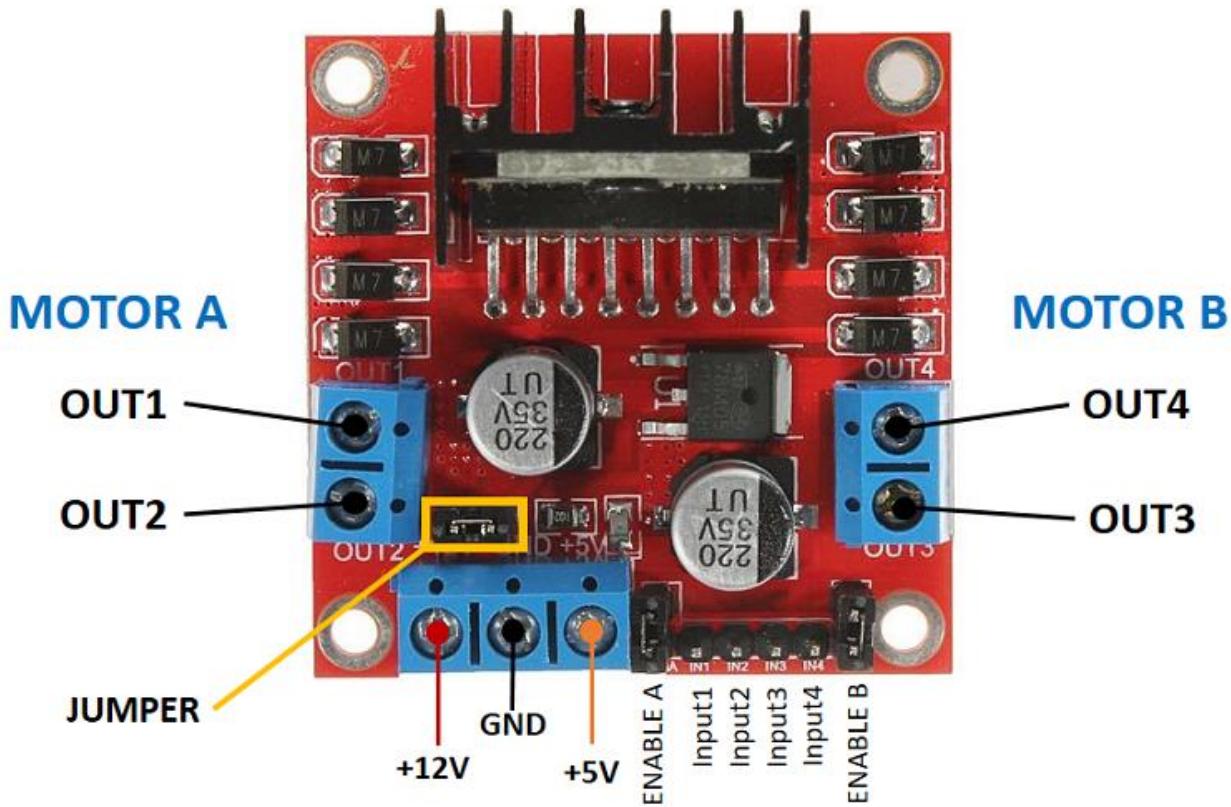
Smart Robot Chassis Kit

We're going to use the [Smart Robot Chassis Kit](#). You can find it in most online stores. The kit costs around \$10 and it's easy to assemble - [watch this video to see how to assemble the robot chassis kit](#).

You can use any other chassis kit as long as it comes with two DC motors.

L298N Motor Driver

There are many ways to control DC motors. We'll use the L298N motor driver that provides an easy way to control the speed and direction of 2 DC motors.



We won't explain how the L298N motor driver works. You can read the following article for an in-depth tutorial about the L298N motor driver:

- [ESP32 with DC Motor and L298N Motor Driver - Control Speed and Direction](#)

Power

To keep the circuitry simpler, we've powered the robot (motors) and the ESP32 using the same power source. We used a power bank/portable charger (like the ones used to charge your smartphone) and it worked well.



Note: the motors draw a lot of current, so if you feel your robot is not moving properly, you may need to use an external power supply for the motors. This means you need two different power sources. One to power the DC motors, and the other to power the ESP32.

Code

Copy the following code to your Arduino IDE.

CODE

[https://github.com/RuiSantosdotme/ESP32-CAM-eBook/blob/master/Code/Module 4/Car Robot Camera Web Server/Car Robot Camera Web Server.ino](https://github.com/RuiSantosdotme/ESP32-CAM-eBook/blob/master/Code/Module%204/Car%20Robot%20Camera%20Web%20Server/Car%20Robot%20Camera%20Web%20Server.ino)

```
#include "esp_camera.h"
#include <WiFi.h>
#include "esp_timer.h"
#include "img_converters.h"
#include "Arduino.h"
#include "fb_gfx.h"
#include "soc/soc.h"          // disable brownout problems
#include "soc/rtc_CNTL_REG.h" // disable brownout problems
#include "esp_http_server.h"

// Replace with your network credentials
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

#define PART_BOUNDARY "1234567890000000000000987654321"

#define CAMERA_MODEL_AI_THINKER

#if defined(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

#else
#error "Camera model not selected"
#endif

#define MOTOR_1_PIN_1      14
#define MOTOR_1_PIN_2      15
#define MOTOR_2_PIN_1      13
#define MOTOR_2_PIN_2      12
```

```

static const char* _STREAM_CONTENT_TYPE = "multipart/x-mixed-replace;boundary="
PART_BOUNDARY;
static const char* _STREAM_BOUNDARY = "\r\n--" PART_BOUNDARY "\r\n";
static const char* _STREAM_PART = "Content-Type: image/jpeg\r\nContent-Length:
%u\r\n\r\n";

```

```

httpd_handle_t camera_httpd = NULL;
httpd_handle_t stream_httpd = NULL;

```

```

static const char PROGMEM INDEX_HTML[] = R"rawliteral(
<html>
  <head>
    <title>ESP32-CAM Robot</title>
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <style>
      body {
        font-family: Arial;
        text-align: center;
        margin: 0px auto;
        padding-top: 30px;
      }
      table { margin-left: auto; margin-right: auto; }
      td { padding: 8px; }
      .button {
        background-color: #2f4468;
        border: none;
        color: white;
        padding: 10px 20px;
        text-align: center;
        text-decoration: none;
        display: inline-block;
        font-size: 18px;
        margin: 6px 3px;
        cursor: pointer;
        -webkit-touch-callout: none;
        -webkit-user-select: none;
        -khtml-user-select: none;
        -moz-user-select: none;
        -ms-user-select: none;
        user-select: none;
        -webkit-tap-highlight-color: rgba(0, 0, 0, 0);
      }
      img { width: auto ;
        max-width: 100% ;
        height: auto ;
      }
    </style>
  </head>
  <body>
    <h1>ESP32-CAM Robot</h1>
    <img src="" id="photo" >
    <table>
      <tr><td colspan="3" align="center"><button class="button"
onmousedown="toggleCheckbox('forward');"
ontouchstart="toggleCheckbox('forward');"
onmouseup="toggleCheckbox('stop');"
ontouchend="toggleCheckbox('stop');">Forward</button></td></tr>
      <tr><td align="center"><button class="button"
onmousedown="toggleCheckbox('left');"
ontouchstart="toggleCheckbox('left');"

```

```

onmouseup="toggleCheckbox('stop');"
ontouchend="toggleCheckbox('stop');">Left</button></td><td
align="center"><button class="button" onmousedown="toggleCheckbox('stop');"
ontouchstart="toggleCheckbox('stop');">Stop</button></td><td
align="center"><button class="button" onmousedown="toggleCheckbox('right');"
ontouchstart="toggleCheckbox('right');" onmouseup="toggleCheckbox('stop');"
ontouchend="toggleCheckbox('stop');">Right</button></td></tr>
<tr><td colspan="3" align="center"><button class="button"
onmousedown="toggleCheckbox('backward');"
ontouchstart="toggleCheckbox('backward');"
onmouseup="toggleCheckbox('stop');"
ontouchend="toggleCheckbox('stop');">Backward</button></td></tr>
</table>
<script>
function toggleCheckbox(x) {
    var xhr = new XMLHttpRequest();
    xhr.open("GET", "/action?go=" + x, true);
    xhr.send();
}
window.onload = document.getElementById("photo").src =
window.location.href.slice(0, -1) + ":81/stream";
</script>
</body>
</html>
)rawliteral";

static esp_err_t index_handler(httpd_req_t *req) {
    httpd_resp_set_type(req, "text/html");
    return httpd_resp_send(req, (const char *)INDEX_HTML, strlen(INDEX_HTML));
}

static esp_err_t stream_handler(httpd_req_t *req) {
    camera_fb_t * fb = NULL;
    esp_err_t res = ESP_OK;
    size_t _jpg_buf_len = 0;
    uint8_t * _jpg_buf = NULL;
    char * part_buf[64];

    res = httpd_resp_set_type(req, _STREAM_CONTENT_TYPE);
    if(res != ESP_OK){
        return res;
    }

    while(true) {
        fb = esp_camera_fb_get();
        if (!fb) {
            Serial.println("Camera capture failed");
            res = ESP_FAIL;
        } else {
            if(fb->width > 400){
                if(fb->format != PIXFORMAT_JPEG){
                    bool jpeg_converted = frame2jpg(fb, 80, &_jpg_buf, &_jpg_buf_len);
                    esp_camera_fb_return(fb);
                    fb = NULL;
                    if(!jpeg_converted){
                        Serial.println("JPEG compression failed");
                        res = ESP_FAIL;
                    }
                } else {

```

```

        _jpg_buf_len = fb->len;
        _jpg_buf = fb->buf;
    }
}
if(res == ESP_OK) {
    size_t hlen = snprintf((char *)part_buf, 64, _STREAM_PART, _jpg_buf_len);
    res = httpd_resp_send_chunk(req, (const char *)part_buf, hlen);
}
if(res == ESP_OK) {
    res = httpd_resp_send_chunk(req, (const char *)_jpg_buf, _jpg_buf_len);
}
if(res == ESP_OK) {
    res = httpd_resp_send_chunk(req, _STREAM_BOUNDARY, strlen(_STREAM_BOUNDARY));
}
if(fb) {
    esp_camera_fb_return(fb);
    fb = NULL;
    _jpg_buf = NULL;
} else if(_jpg_buf) {
    free(_jpg_buf);
    _jpg_buf = NULL;
}
if(res != ESP_OK) {
    break;
}
//Serial.printf("MJPG: %uB\n", (uint32_t) (_jpg_buf_len));
}
return res;
}

static esp_err_t cmd_handler(httpd_req_t *req) {
    char* buf;
    size_t buf_len;
    char variable[32] = {0,};

    buf_len = httpd_req_get_url_query_len(req) + 1;
    if (buf_len > 1) {
        buf = (char*)malloc(buf_len);
        if(!buf){
            httpd_resp_send_500(req);
            return ESP_FAIL;
        }
        if (httpd_req_get_url_query_str(req, buf, buf_len) == ESP_OK) {
            if (httpd_query_key_value(buf, "go", variable, sizeof(variable)) == ESP_OK) {
                free(buf);
                httpd_resp_send_404(req);
                return ESP_FAIL;
            }
        } else {
            free(buf);
            httpd_resp_send_404(req);
            return ESP_FAIL;
        }
        free(buf);
    } else {
        httpd_resp_send_404(req);
        return ESP_FAIL;
    }
}
```

```

}

sensor_t * s = esp_camera_sensor_get();
int res = 0;

if(!strcmp(variable, "forward")) {
    Serial.println("Forward");
    digitalWrite(MOTOR_1_PIN_1, 1);
    digitalWrite(MOTOR_1_PIN_2, 0);
    digitalWrite(MOTOR_2_PIN_1, 1);
    digitalWrite(MOTOR_2_PIN_2, 0);
}
else if(!strcmp(variable, "left")) {
    Serial.println("Left");
    digitalWrite(MOTOR_1_PIN_1, 0);
    digitalWrite(MOTOR_1_PIN_2, 1);
    digitalWrite(MOTOR_2_PIN_1, 1);
    digitalWrite(MOTOR_2_PIN_2, 0);
}
else if(!strcmp(variable, "right")) {
    Serial.println("Right");
    digitalWrite(MOTOR_1_PIN_1, 1);
    digitalWrite(MOTOR_1_PIN_2, 0);
    digitalWrite(MOTOR_2_PIN_1, 0);
    digitalWrite(MOTOR_2_PIN_2, 1);
}
else if(!strcmp(variable, "backward")) {
    Serial.println("Backward");
    digitalWrite(MOTOR_1_PIN_1, 0);
    digitalWrite(MOTOR_1_PIN_2, 1);
    digitalWrite(MOTOR_2_PIN_1, 0);
    digitalWrite(MOTOR_2_PIN_2, 1);
}
else if(!strcmp(variable, "stop")) {
    Serial.println("Stop");
    digitalWrite(MOTOR_1_PIN_1, 0);
    digitalWrite(MOTOR_1_PIN_2, 0);
    digitalWrite(MOTOR_2_PIN_1, 0);
    digitalWrite(MOTOR_2_PIN_2, 0);
}
else {
    res = -1;
}

if(res){
    return httpd_resp_send_500(req);
}

httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");
return httpd_resp_send(req, NULL, 0);
}

void startCameraServer(){
    httpd_config_t config = HTTPD_DEFAULT_CONFIG();
    config.server_port = 80;
    httpd_uri_t index_uri = {
        .uri      = "/",
        .method   = HTTP_GET,
        .handler  = index_handler,
}

```

```

    .user_ctx = NULL
};

httpd_uri_t cmd_uri = {
    .uri      = "/action",
    .method   = HTTP_GET,
    .handler  = cmd_handler,
    .user_ctx = NULL
};
httpd_uri_t stream_uri = {
    .uri      = "/stream",
    .method   = HTTP_GET,
    .handler  = stream_handler,
    .user_ctx = NULL
};
if (httpd_start(&camera_httpd, &config) == ESP_OK) {
    httpd_register_uri_handler(camera_httpd, &index_uri);
    httpd_register_uri_handler(camera_httpd, &cmd_uri);
}
config.server_port += 1;
config.ctrl_port += 1;
if (httpd_start(&stream_httpd, &config) == ESP_OK) {
    httpd_register_uri_handler(stream_httpd, &stream_uri);
}
}

void setup() {
    WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0); //disable brownout detector

    pinMode(MOTOR_1_PIN_1, OUTPUT);
    pinMode(MOTOR_1_PIN_2, OUTPUT);
    pinMode(MOTOR_2_PIN_1, OUTPUT);
    pinMode(MOTOR_2_PIN_2, OUTPUT);

    Serial.begin(115200);
    Serial.setDebugOutput(false);

    camera_config_t config;
    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;
}

```

```

if(psramFound()) {
    config.frame_size = FRAMESIZE_VGA;
    config.jpeg_quality = 10;
    config.fb_count = 2;
} else {
    config.frame_size = FRAMESIZE_SVGA;
    config.jpeg_quality = 12;
    config.fb_count = 1;
}

// Camera init
esp_err_t err = esp_camera_init(&config);
if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
}
// Wi-Fi connection
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}
Serial.println("");
Serial.println("WiFi connected");

Serial.print("Camera Stream Ready! Go to: http://");
Serial.println(WiFi.localIP());

// Start streaming web server
startCameraServer();
}

void loop() {
}

```

Insert your network credentials and the code should work straight away.

```

const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

```

How the Code Works

Let's take a look at the relevant parts to control the robot. Define the GPIOs that will control the motors. Each motor is controlled by two pins.

```

#define MOTOR_1_PIN_1      14
#define MOTOR_1_PIN_2      15
#define MOTOR_2_PIN_1      13
#define MOTOR_2_PIN_2      12

```

When you click the buttons, you make a request on a different URL.

```
<table>
  <tr><td colspan="3" align="center"><button class="button"
onmousedown="toggleCheckbox('forward');"
ontouchstart="toggleCheckbox('forward');" onmouseup="toggleCheckbox('stop');"
ontouchend="toggleCheckbox('stop');">Forward</button></td></tr>
  <tr><td align="center"><button class="button"
onmousedown="toggleCheckbox('left');" ontouchstart="toggleCheckbox('left');"
onmouseup="toggleCheckbox('stop');"
ontouchend="toggleCheckbox('stop');">Left</button></td><td
align="center"><button class="button" onmousedown="toggleCheckbox('stop');"
ontouchstart="toggleCheckbox('stop');">Stop</button></td><td
align="center"><button class="button" onmousedown="toggleCheckbox('right');"
ontouchstart="toggleCheckbox('right');" onmouseup="toggleCheckbox('stop');"
ontouchend="toggleCheckbox('stop');">Right</button></td></tr>
  <tr><td colspan="3" align="center"><button class="button"
onmousedown="toggleCheckbox('backward');"
ontouchstart="toggleCheckbox('backward');"
onmouseup="toggleCheckbox('stop');"
ontouchend="toggleCheckbox('stop');">Backward</button></td></tr></table>
<script>
  function toggleCheckbox(x) {
    var xhr = new XMLHttpRequest();
    xhr.open("GET", "/action?go=" + x, true);
    xhr.send();
  }
}
```

The previous snippet is not very readable, we recommend going to the [GitHub page](#) to see the code with better formatting.

Here's the requests made depending on the button that is being pressed:

- **Forward:**

```
<ESP_IP_ADDRESS>/action?go=forward
```

- **Backward:**

```
<ESP_IP_ADDRESS>/action?go=backward
```

- **Left:**

```
<ESP_IP_ADDRESS>/action?go=left
```

- **Right:**

```
<ESP_IP_ADDRESS>/action?go=right
```

- **Stop:**

<ESP_IP_ADDRESS>/action?go=stop

When you release the button, a request is made on the */action?go=stop* URL. The robot only moves as long as you're pressing the buttons.

Handle Requests

To handle what happens when we get requests on those URLs, we use these `if...`
`else` statements:

```
if(!strcmp(variable, "forward")) {  
    Serial.println("Forward");  
    digitalWrite(MOTOR_1_PIN_1, 1);  
    digitalWrite(MOTOR_1_PIN_2, 0);  
    digitalWrite(MOTOR_2_PIN_1, 1);  
    digitalWrite(MOTOR_2_PIN_2, 0);}  
else if(!strcmp(variable, "left")) {  
    Serial.println("Left");  
    digitalWrite(MOTOR_1_PIN_1, 0);  
    digitalWrite(MOTOR_1_PIN_2, 1);  
    digitalWrite(MOTOR_2_PIN_1, 1);  
    digitalWrite(MOTOR_2_PIN_2, 0);}  
else if(!strcmp(variable, "right")) {  
    Serial.println("Right");  
    digitalWrite(MOTOR_1_PIN_1, 1);  
    digitalWrite(MOTOR_1_PIN_2, 0);  
    digitalWrite(MOTOR_2_PIN_1, 0);  
    digitalWrite(MOTOR_2_PIN_2, 1);}  
else if(!strcmp(variable, "backward")) {  
    Serial.println("Backward");  
    digitalWrite(MOTOR_1_PIN_1, 0);  
    digitalWrite(MOTOR_1_PIN_2, 1);  
    digitalWrite(MOTOR_2_PIN_1, 0);  
    digitalWrite(MOTOR_2_PIN_2, 1);}  
else if(!strcmp(variable, "stop")) {  
    Serial.println("Stop");  
    digitalWrite(MOTOR_1_PIN_1, 0);  
    digitalWrite(MOTOR_1_PIN_2, 0);  
    digitalWrite(MOTOR_2_PIN_1, 0);  
    digitalWrite(MOTOR_2_PIN_2, 0);}
```

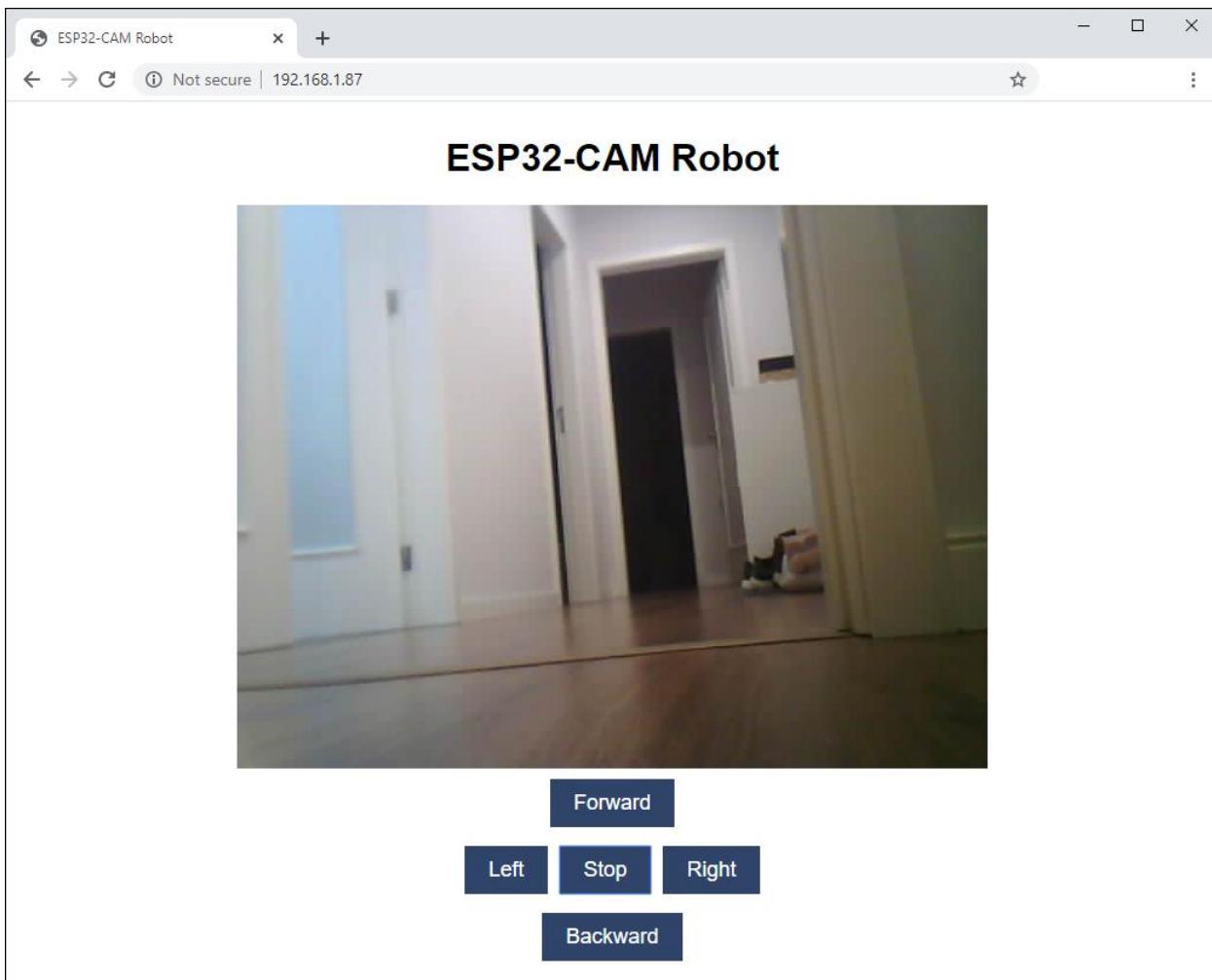
Testing the Code

After inserting your network credentials, you can upload the code to your ESP32-CAM board. After uploading, open the Serial Monitor to get its IP address.

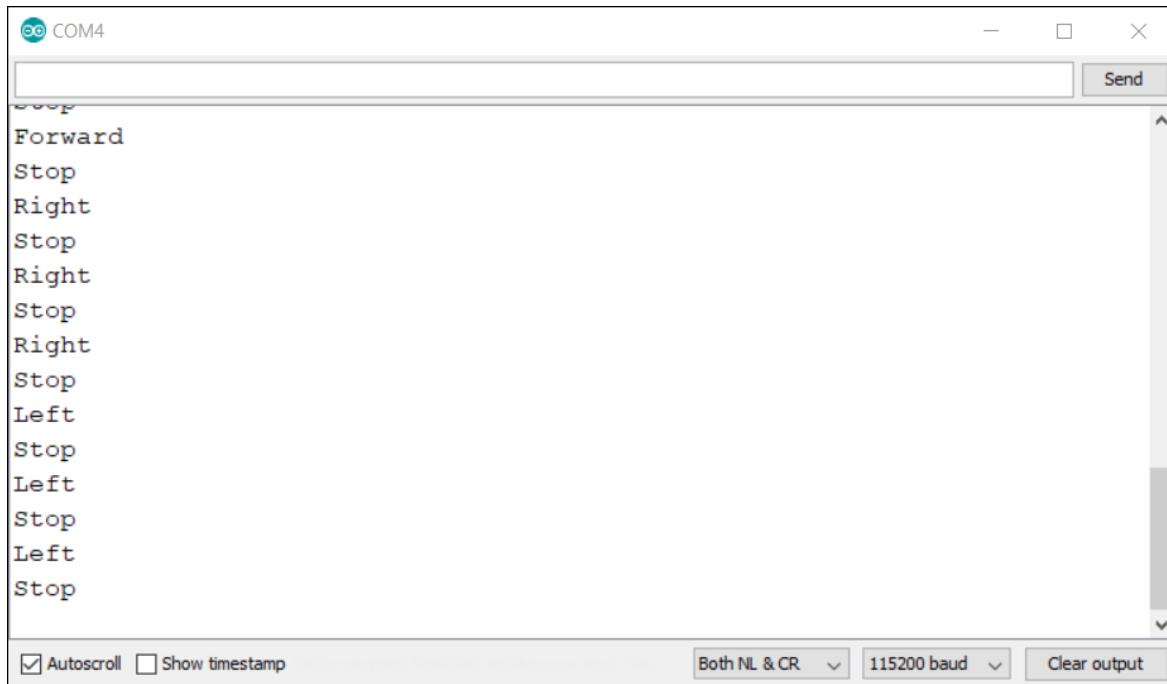
```
WiFi connected
Camera Stream Ready! Go to: http://192.168.1.87

 Autoscroll  Show timestamp      Both NL & CR      115200 baud      Clear output
```

Open a browser and type the ESP IP address. A similar web page should load:



Press the buttons and take a look at the Serial Monitor to see if it is streaming without lag and if it is receiving the commands without crashing.



```
Forward
Stop
Right
Stop
Right
Stop
Right
Stop
Left
Stop
Left
Stop
Left
Stop
```

Autoscroll Show timestamp Both NL & CR 115200 baud Clear output

If everything is working properly, it's time to assemble the circuit.

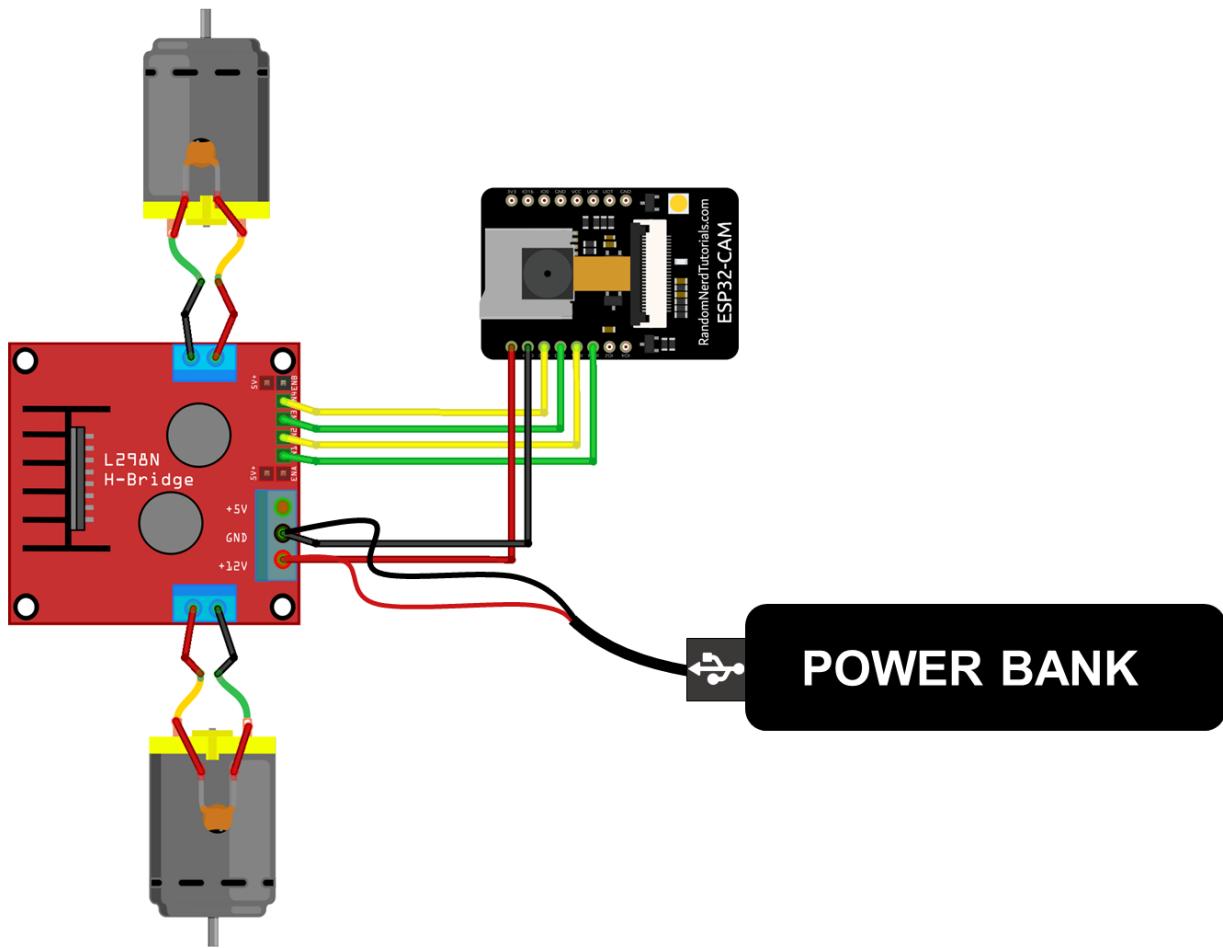
Parts Required

For this project, we'll use the following parts:

- [ESP32-CAM AI-Thinker with external antenna](#)
- [L298N Motor Driver](#)
- [Robot Car Chassis Kit](#)
- Power bank or other 5V power supply
- [Prototyping circuit board](#) (optional)

Circuit

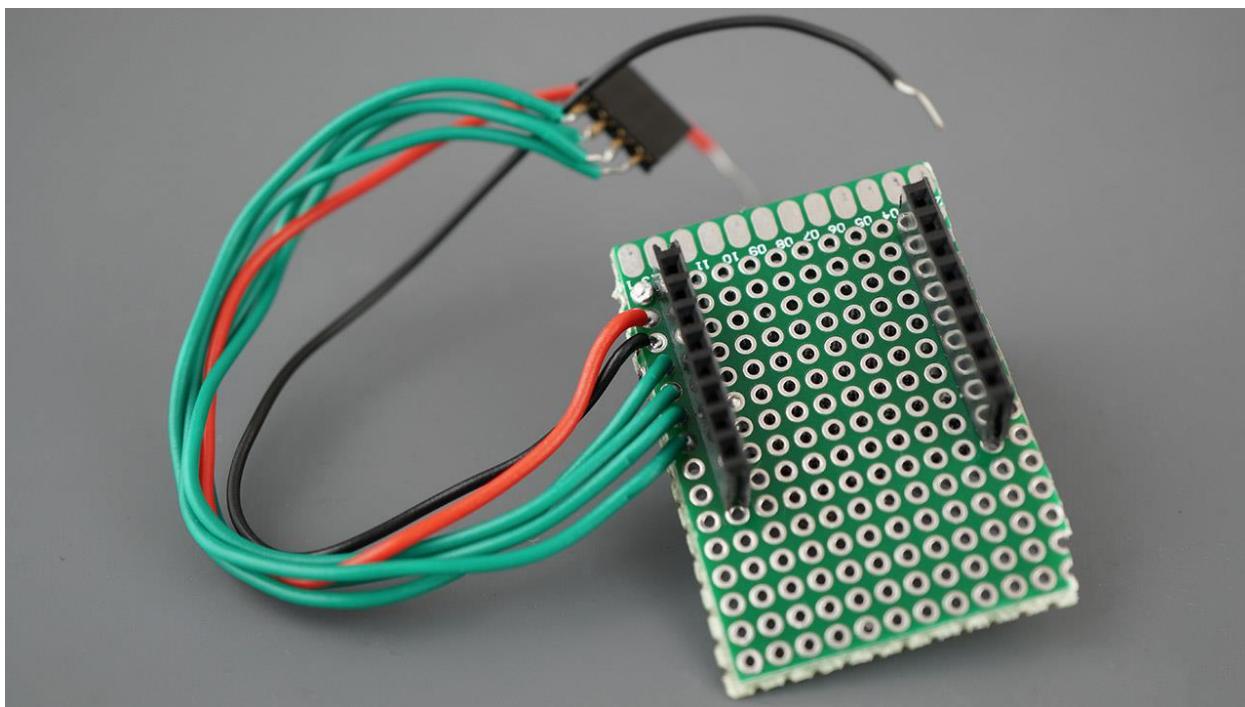
After assembling the robot chassis, you can wire the circuit by following the next schematic diagram.



Start by connecting the ESP32-CAM to the motor driver as shown in the schematic diagram. You can either use a mini breadboard or a stripboard to place your ESP32-CAM and build the circuit. The following table shows the connections between the ESP32-CAM and the L298N Motor Driver.

| L298N MOTOR DRIVER | ESP32-CAM |
|--------------------|-----------|
| IN1 | GPIO 14 |
| IN2 | GPIO 15 |
| IN3 | GPIO 13 |
| IN4 | GPIO 12 |

We assembled all the connections on a mini stripboard as shown below.



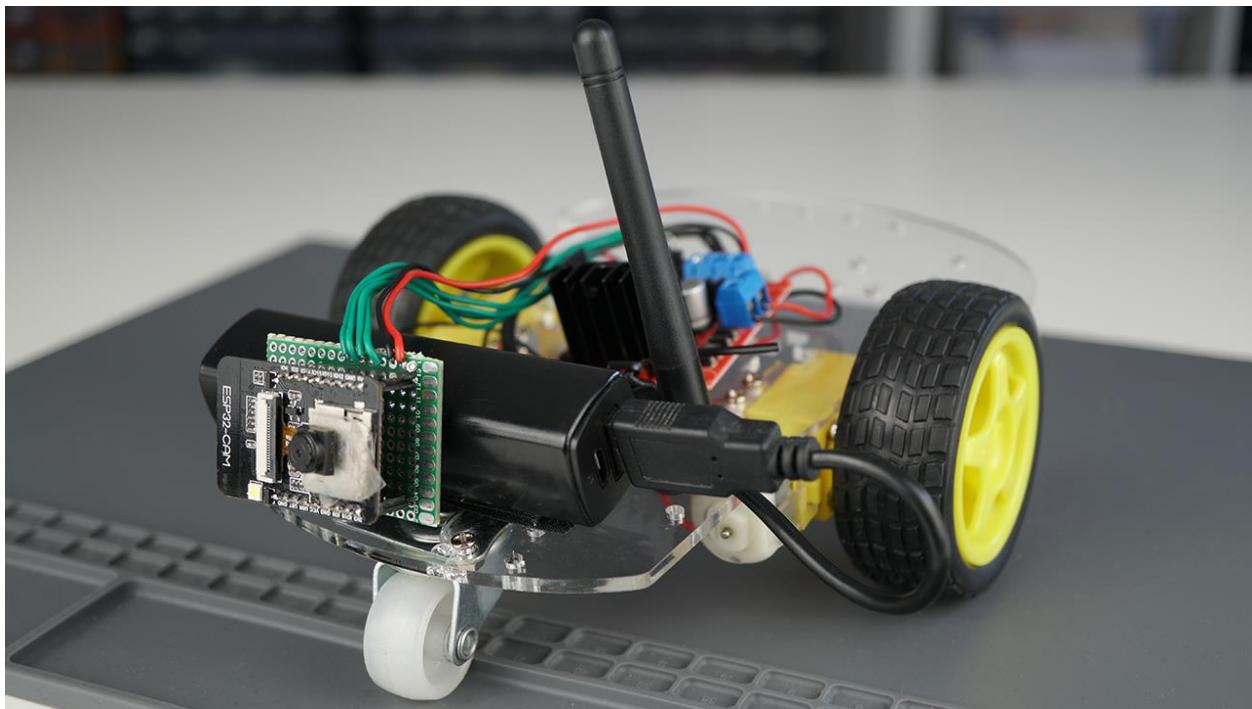
After that, wire each motor to its terminal block.

Note: we suggest soldering a 0.1 uF ceramic capacitor to the positive and negative terminals of each motor, as shown in the diagram to help smooth out any voltage spikes. Additionally, you can solder a slider switch to the red wire that comes from the power bank. This way, you can turn the power on and off.

Finally, apply power with a power bank as shown in the schematic diagram. You need to strip a USB cable. In this example, the ESP32-CAM and the motors are being powered using the same power source and it works well.

Note: the motors draw a lot of current, so if you feel your robot is not moving fast enough, you may need to use an external power supply for the motors. This means you need two different power sources. One to power the DC motors, and the other to power the ESP32. You can use a 4 AA battery pack to power the motors. When you get your robot chassis kit, you usually get a battery holder for 4 AA batteries.

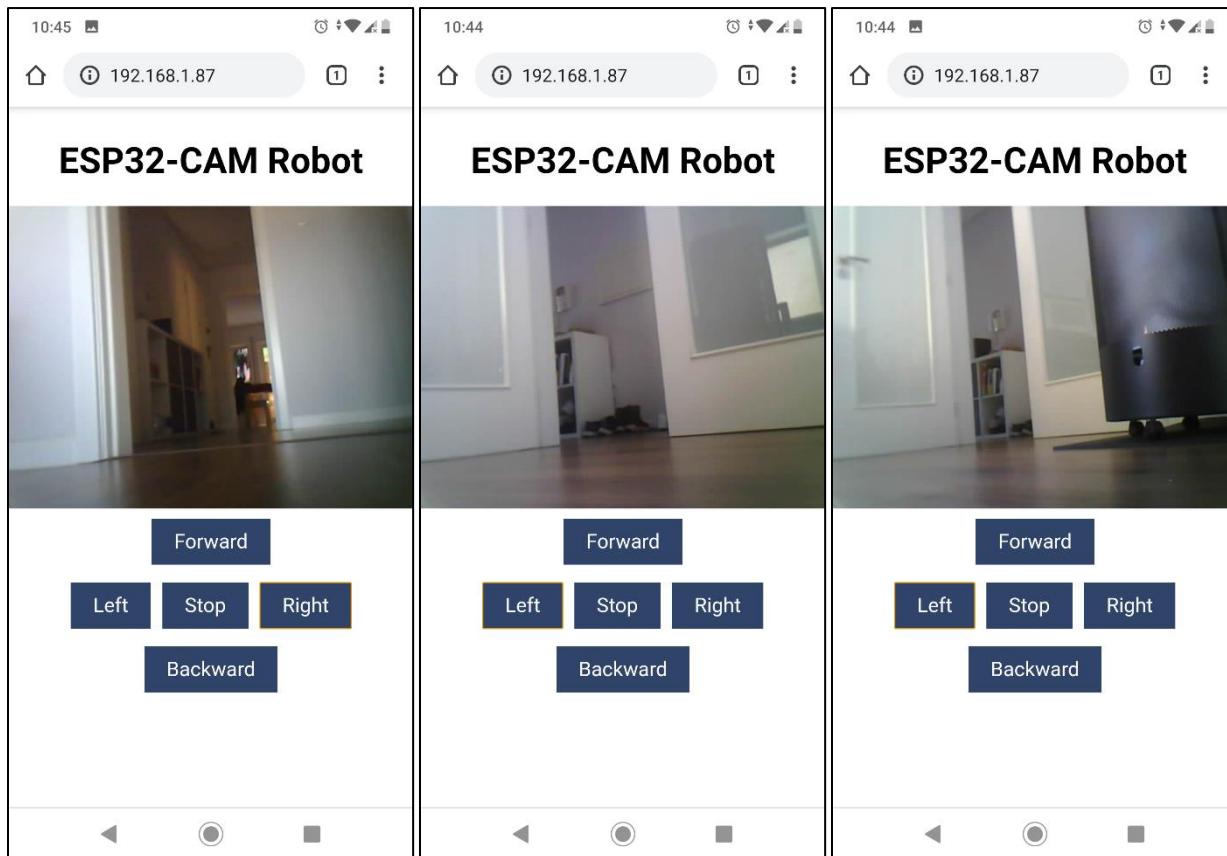
Your robot should look similar to the following figure:



Don't forget that you should use an external antenna with the ESP32-CAM, otherwise the web server might be extremely slow.

Demonstration

Open a browser on the ESP32-CAM IP address, and you should be able to control your robot. The web server works well on a laptop computer or smartphone. **You can only have the web server open in one device/tab at a time.**



Unit 5: Pan and Tilt Video Streaming (2 Axis)



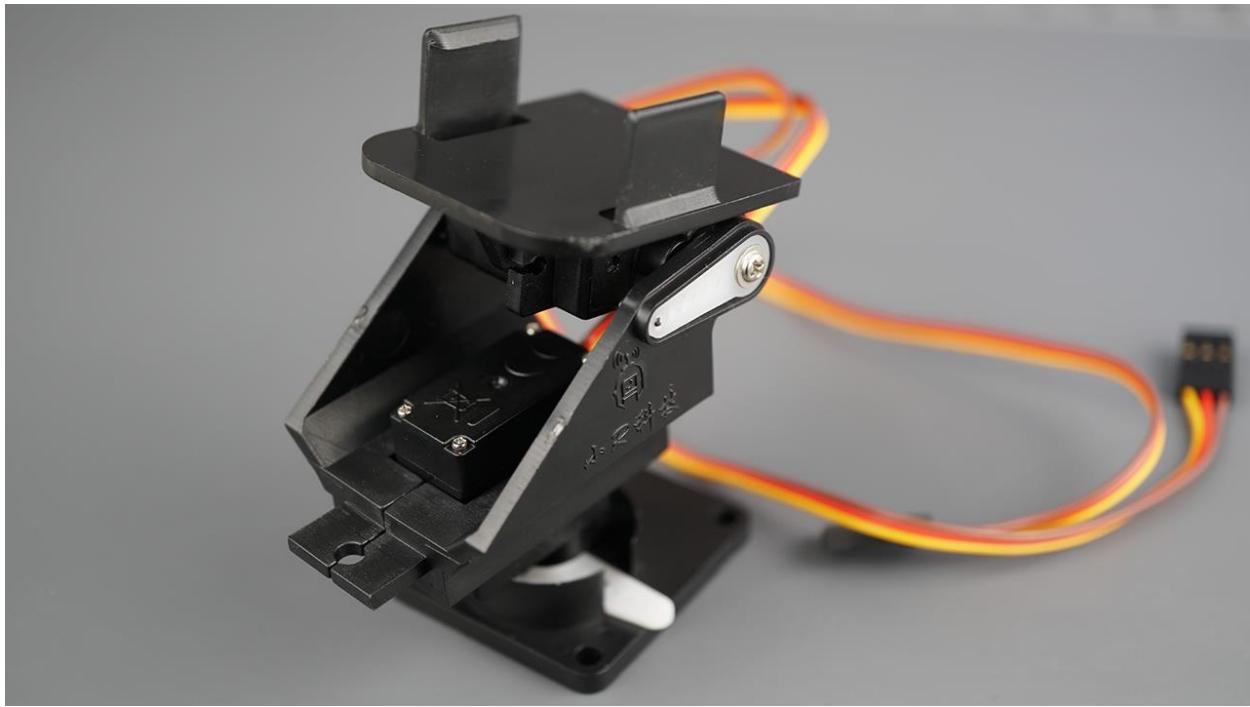
In this project we'll attach the ESP32-CAM to a pan and tilt stand with two SG90 servo motors. With a pan and tilt camera stand you can move the camera up, down, to the left and to the right – this is great for surveillance.

The ESP32-CAM hosts a web server that shows video streaming and buttons to control the servo motors to move the camera.

Compatibility: for this project you need an ESP32 camera development board with access to two GPIOs to control two servo motors. You can use: ESP32-CAM AI-Thinker, T-Journal or TTGO T-Camera Plus.

Pan and Tilt Stand and Motors

For this project we'll use a pan and tilt stand that already comes with two SG90 servo motors. The stand is shown in the following figure.



We got our stand from Banggood, but you can get yours from any other store.

- [Pan and Tilt stand with two servo motors](#)

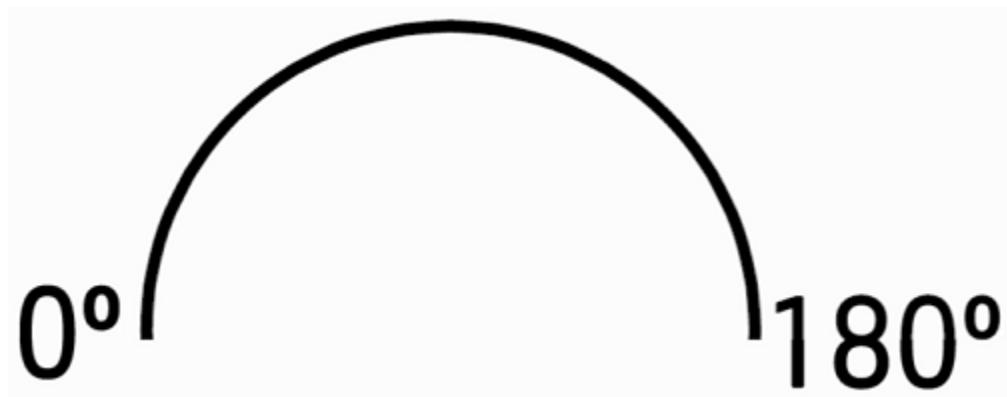
Alternatively, you can get two [SG90 servo motors](#) and 3D print your own stand.

Servo motors have three wires with different colors:

| Wire | Color |
|--------|-------------------------|
| Power | Red |
| GND | Black or brown |
| Signal | Yellow, orange or white |

How to Control a Servo?

You can position the servo's shaft in various angles from 0 to 180°. Servos are controlled using a pulse width modulation (PWM) signal. This means that the PWM signal sent to the motor will determine the shaft's position.



To control the servo motor, you can use the PWM capabilities of the ESP32 by sending a signal with the appropriate pulse width. Or you can use a library to simplify the code. We'll be using the **ESP32Servo** library.

Installing the **ESP32Servo** Library

To control servo motors, we'll use the **ESP32Servo** library. Make sure you install that library before proceeding. In your Arduino IDE, go to **Sketch > Include Library > Manage Libraries**. Search for **ESP32Servo** and install the library as shown below.



Code

The code for this project is very similar to the previous Unit, but instead of controlling two DC motors, we're controlling two servo motors.

Copy the following code to your Arduino IDE.

CODE

https://github.com/RuiSantosdotme/ESP32-CAM-eBook/blob/master/Code/Module%204/Pan%20and%20Tilt%20Video%20Streaming/Pan_and_Tilt_Video_Streaming.ino

```
#include "esp_camera.h"
#include <WiFi.h>
#include "esp_timer.h"
#include "img_converters.h"
#include "Arduino.h"
#include "fb_gfx.h"
#include "soc/soc.h"          // disable brownout problems
#include "soc/rtc_CNTL_REG.h" // disable brownout problems
#include "esp_http_server.h"
#include <ESP32Servo.h>

// Replace with your network credentials
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

#define PART_BOUNDARY "123456789000000000000987654321"

#define CAMERA_MODEL_AI_THINKER

#if defined(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

#else
#error "Camera model not selected"

```

```

#endif

#define SERVO_1      14
#define SERVO_2      15

#define SERVO_STEP   5

Servo servoN1;
Servo servoN2;
Servo servo1;
Servo servo2;

int servo1Pos = 0;
int servo2Pos = 0;

static const char* _STREAM_CONTENT_TYPE = "multipart/x-mixed-replace;boundary=" PART_BOUNDARY;
static const char* _STREAM_BOUNDARY = "\r\n--" PART_BOUNDARY "\r\n";
static const char* _STREAM_PART = "Content-Type: image/jpeg\r\nContent-Length: %u\r\n\r\n";
```

httpd_handle_t camera_httpd = NULL;
httpd_handle_t stream_httpd = NULL;

static const char PROGMEM INDEX_HTML[] = R"rawliteral(

<html>
 <head>
 <title>ESP32-CAM Robot</title>
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <style>
 body {
 font-family: Arial;
 text-align: center;
 margin: 0px auto;
 padding-top: 30px;
 }
 table { margin-left: auto; margin-right: auto; }
 td { padding: 8px; }
 .button {
 background-color: #2f4468;
 border: none;
 color: white;
 padding: 10px 20px;
 text-align: center;
 text-decoration: none;
 display: inline-block;
 font-size: 18px;
 margin: 6px 3px;
 cursor: pointer;
 -webkit-touch-callout: none;
 -webkit-user-select: none;
 -khtml-user-select: none;
 -moz-user-select: none;
 -ms-user-select: none;
 user-select: none;
 -webkit-tap-highlight-color: rgba(0,0,0,0);
 }
 img { width: auto ;
 max-width: 100% ;

```

        height: auto ;
    }
</style>
</head>
<body>
<h1>ESP32-CAM Pan and Tilt</h1>
<img src="" id="photo" >
<table>
<tr><td colspan="3" align="center"><button class="button" onmousedown="toggleCheckbox('up');"
ontouchstart="toggleCheckbox('up');" >Up</button></td></tr>
<tr><td align="center"><button class="button" onmousedown="toggleCheckbox('left');"
ontouchstart="toggleCheckbox('left');" >Left</button></td><td align="center"><button class="button" onmousedown="toggleCheckbox('right');"
ontouchstart="toggleCheckbox('right');" >Right</button></td></tr>
<tr><td colspan="3" align="center"><button class="button" onmousedown="toggleCheckbox('down');"
ontouchstart="toggleCheckbox('down');" >Down</button></td></tr>
</table>
<script>
function toggleCheckbox(x) {
    var xhr = new XMLHttpRequest();
    xhr.open("GET", "/action?go=" + x, true);
    xhr.send();
}
window.onload = document.getElementById("photo").src =
window.location.href.slice(0, -1) + ":81/stream";
</script>
</body>
</html>
) rawliteral";
}

static esp_err_t index_handler(httpd_req_t *req) {
    httpd_resp_set_type(req, "text/html");
    return httpd_resp_send(req, (const char *)INDEX_HTML, strlen(INDEX_HTML));
}

static esp_err_t stream_handler(httpd_req_t *req) {
    camera_fb_t * fb = NULL;
    esp_err_t res = ESP_OK;
    size_t _jpg_buf_len = 0;
    uint8_t * _jpg_buf = NULL;
    char part_buf[64];

    res = httpd_resp_set_type(req, _STREAM_CONTENT_TYPE);
    if(res != ESP_OK){
        return res;
    }

    while(true){
        fb = esp_camera_fb_get();
        if (!fb) {
            Serial.println("Camera capture failed");
            res = ESP_FAIL;
        } else {
            if(fb->width > 400){
                if(fb->format != PIXFORMAT_JPEG){

```

```

        bool jpeg_converted = frame2jpg(fb, 80, &_jpg_buf, &_jpg_buf_len);
        esp_camera_fb_return(fb);
        fb = NULL;
        if(!jpeg_converted){
            Serial.println("JPEG compression failed");
            res = ESP_FAIL;
        }
    } else {
        _jpg_buf_len = fb->len;
        _jpg_buf = fb->buf;
    }
}
}

if(res == ESP_OK){
    size_t hlen = snprintf((char *)part_buf, 64, _STREAM_PART, _jpg_buf_len);
    res = httpd_resp_send_chunk(req, (const char *)part_buf, hlen);
}
if(res == ESP_OK){
    res = httpd_resp_send_chunk(req, (const char *)_jpg_buf, _jpg_buf_len);
}
if(res == ESP_OK){
    res = httpd_resp_send_chunk(req, _STREAM_BOUNDARY, strlen(_STREAM_BOUNDARY));
}
if(fb){
    esp_camera_fb_return(fb);
    fb = NULL;
    _jpg_buf = NULL;
} else if(_jpg_buf){
    free(_jpg_buf);
    _jpg_buf = NULL;
}
if(res != ESP_OK){
    break;
}
//Serial.printf("MJPG: %uB\n", (uint32_t) (_jpg_buf_len));
}
return res;
}

static esp_err_t cmd_handler(httpd_req_t *req) {
    char* buf;
    size_t buf_len;
    char variable[32] = {0,};

    buf_len = httpd_req_get_url_query_len(req) + 1;
    if (buf_len > 1) {
        buf = (char*)malloc(buf_len);
        if(!buf){
            httpd_resp_send_500(req);
            return ESP_FAIL;
        }
        if (httpd_req_get_url_query_str(req, buf, buf_len) == ESP_OK) {
            if (httpd_query_key_value(buf, "go", variable, sizeof(variable)) == ESP_OK) {
                } else {
                    free(buf);
                    httpd_resp_send_404(req);
                    return ESP_FAIL;
                }
            }
        }
    }
}

```

```

    } else {
        free(buf);
        httpd_resp_send_404(req);
        return ESP_FAIL;
    }
    free(buf);
} else {
    httpd_resp_send_404(req);
    return ESP_FAIL;
}

sensor_t * s = esp_camera_sensor_get();
//flip the camera vertically
//s->set_vflip(s, 1);           // 0 = disable , 1 = enable
// mirror effect
//s->set_hmirror(s, 1);         // 0 = disable , 1 = enable

int res = 0;

if(!strcmp(variable, "up")) {
    if(servolPos <= 170) {
        servolPos += 10;
        servol.write(servolPos);
    }
    Serial.println(servolPos);
    Serial.println("Up");
}
else if(!strcmp(variable, "left")) {
    if(servo2Pos <= 170) {
        servo2Pos += 10;
        servo2.write(servo2Pos);
    }
    Serial.println(servo2Pos);
    Serial.println("Left");
}
else if(!strcmp(variable, "right")) {
    if(servo2Pos >= 10) {
        servo2Pos -= 10;
        servo2.write(servo2Pos);
    }
    Serial.println(servo2Pos);
    Serial.println("Right");
}
else if(!strcmp(variable, "down")) {
    if(servolPos >= 10) {
        servolPos -= 10;
        servol.write(servolPos);
    }
    Serial.println(servolPos);
    Serial.println("Down");
}
else {
    res = -1;
}

if(res){
    return httpd_resp_send_500(req);
}

```

```

    httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");
    return httpd_resp_send(req, NULL, 0);
}

void startCameraServer() {
    httpd_config_t config = HTTPD_DEFAULT_CONFIG();
    config.server_port = 80;
    httpd_uri_t index_uri = {
        .uri      = "/",
        .method   = HTTP_GET,
        .handler  = index_handler,
        .user_ctx = NULL
    };

    httpd_uri_t cmd_uri = {
        .uri      = "/action",
        .method   = HTTP_GET,
        .handler  = cmd_handler,
        .user_ctx = NULL
    };
    httpd_uri_t stream_uri = {
        .uri      = "/stream",
        .method   = HTTP_GET,
        .handler  = stream_handler,
        .user_ctx = NULL
    };
    if (httpd_start(&camera_httpd, &config) == ESP_OK) {
        httpd_register_uri_handler(camera_httpd, &index_uri);
        httpd_register_uri_handler(camera_httpd, &cmd_uri);
    }
    config.server_port += 1;
    config.ctrl_port += 1;
    if (httpd_start(&stream_httpd, &config) == ESP_OK) {
        httpd_register_uri_handler(stream_httpd, &stream_uri);
    }
}

void setup() {
    WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0); // disable brownout detector
    servo1.setPeriodHertz(50); // standard 50 hz servo
    servo2.setPeriodHertz(50); // standard 50 hz servo
    servoN1.attach(2, 1000, 2000);
    servoN2.attach(13, 1000, 2000);

    servo1.attach(SERVO_1, 1000, 2000);
    servo2.attach(SERVO_2, 1000, 2000);

    servo1.write(servo1Pos);
    servo2.write(servo2Pos);

    Serial.begin(115200);
    Serial.setDebugOutput(false);

    camera_config_t config;
    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;

if(psramFound()){
    config.frame_size = FRAMESIZE_VGA;
    config.jpeg_quality = 10;
    config.fb_count = 2;
} else {
    config.frame_size = FRAMESIZE_SVGA;
    config.jpeg_quality = 12;
    config.fb_count = 1;
}

// Camera init
esp_err_t err = esp_camera_init(&config);
if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
}
// Wi-Fi connection
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}
Serial.println("");
Serial.println("WiFi connected");

Serial.print("Camera Stream Ready! Go to: http://");
Serial.println(WiFi.localIP());

// Start streaming web server
startCameraServer();
}

void loop() {
}

```

Network Credentials

Insert your network credentials and the code should work straight away.

```
// Replace with your network credentials
```

```
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
```

How the Code Works

The code for this project is very similar with the code from the previous Unit, So, we'll take a look at the relevant parts to control the servo motors.

Define the pins the servo motors are connected to. In this case, they are connected to the ESP32-CAM GPIOs 14 and 15.

```
#define SERVO_1 14
#define SERVO_2 15
```

Create `Servo` objects to control each motor:

```
Servo servoN1;
Servo servoN2;
Servo servol;
Servo servo2;
```

You may be wondering why we are creating four `Servo` objects when we only have two servos. What happens is that the servo library we're using automatically assigns a PWM channel to each servo motor (`servoN1` → PWM channel 0; `servoN2` → PWM channel 1; `servol` → PWM channel 2; `servo2` → PWM channel 3).

The first channels are being used by the camera, so if we change the properties of those PWM channels, we'll get errors with the camera. So, we'll control `servol` and `servo2` that use PWM channels 2 and 3 that are not being used by the camera.

Define the servos initial position.

```
int servolPos = 0;
int servo2Pos = 0;
```

Web Page

The `INDEX_HTML` variable contains the HTML text to build the web page. The following lines display the buttons.

```


|                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                  |                                                                                                                             |  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                                                                                                                             |  |  |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------|--|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------|--|--|
| <button class="button" onmousedown="toggleCheckbox('up');" ontouchstart="toggleCheckbox('up');">Up</button></td></tr> <tr><td align="center">&lt;button class="button" onmousedown="toggleCheckbox('left');" ontouchstart="toggleCheckbox('left');"&gt;Left&lt;/button&gt;&lt;/td&gt;&lt;td align="center"&gt;&lt;button class="button" onmousedown="toggleCheckbox('right');" ontouchstart="toggleCheckbox('right');"&gt;Right&lt;/button&gt;&lt;/td&gt;&lt;/tr&gt; <tr><td align="center" colspan="3">&lt;button class="button" onmousedown="toggleCheckbox('down');" ontouchstart="toggleCheckbox('down');"&gt;Down&lt;/button&gt;&lt;/td&gt;&lt;/tr&gt; </td></tr></td></tr> |                                                                                                                             |  | <button class="button" onmousedown="toggleCheckbox('left');" ontouchstart="toggleCheckbox('left');">Left</button></td><td align="center"><button class="button" onmousedown="toggleCheckbox('right');" ontouchstart="toggleCheckbox('right');">Right</button></td></tr> <tr><td align="center" colspan="3">&lt;button class="button" onmousedown="toggleCheckbox('down');" ontouchstart="toggleCheckbox('down');"&gt;Down&lt;/button&gt;&lt;/td&gt;&lt;/tr&gt; </td></tr> | <button class="button" onmousedown="toggleCheckbox('down');" ontouchstart="toggleCheckbox('down');">Down</button></td></tr> |  |  |
| <button class="button" onmousedown="toggleCheckbox('left');" ontouchstart="toggleCheckbox('left');">Left</button></td><td align="center"><button class="button" onmousedown="toggleCheckbox('right');" ontouchstart="toggleCheckbox('right');">Right</button></td></tr> <tr><td align="center" colspan="3">&lt;button class="button" onmousedown="toggleCheckbox('down');" ontouchstart="toggleCheckbox('down');"&gt;Down&lt;/button&gt;&lt;/td&gt;&lt;/tr&gt; </td></tr>                                                                                                                                                                                                        | <button class="button" onmousedown="toggleCheckbox('down');" ontouchstart="toggleCheckbox('down');">Down</button></td></tr> |  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                                                                                                                             |  |  |
| <button class="button" onmousedown="toggleCheckbox('down');" ontouchstart="toggleCheckbox('down');">Down</button></td></tr>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |                                                                                                                             |  |                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                                                                                                                             |  |  |


```

The previous snippet is not very readable, we recommend going to the [GitHub page](#) to see the code with better formatting.

When you click the buttons, a JavaScript function is called. It makes a request on a different URL depending on the button clicked.

```

function toggleCheckbox(x) {
    var xhr = new XMLHttpRequest();
    xhr.open("GET", "/action?go=" + x, true);
    xhr.send();
}

```

Here's the requests made depending on the button that is being pressed:

- **Up:**

```
<ESP_IP_ADDRESS>/action?go=up
```

- **Down:**

```
<ESP_IP_ADDRESS>/action?go=down
```

- **Left:**

```
<ESP_IP_ADDRESS>/action?go=left
```

- **Right:**

```
<ESP_IP_ADDRESS>/action?go=right
```

Handle Requests

Then, we need to handle what happens when we get requests on those URLs. That's what's done in the following lines.

```
if(!strcmp(variable, "up")) {  
    if(servolPos <= 170) {  
        servolPos += 10;  
        servol.write(servolPos);  
    }  
    Serial.println(servolPos);  
    Serial.println("Up");  
}  
else if(!strcmp(variable, "left")) {  
    if(servo2Pos <= 170) {  
        servo2Pos += 10;  
        servo2.write(servo2Pos);  
    }  
    Serial.println(servo2Pos);  
    Serial.println("Left");  
}  
else if(!strcmp(variable, "right")) {  
    if(servo2Pos >= 10) {  
        servo2Pos -= 10;  
        servo2.write(servo2Pos);  
    }  
    Serial.println(servo2Pos);  
    Serial.println("Right");  
}  
else if(!strcmp(variable, "down")) {  
    if(servolPos >= 10) {  
        servolPos -= 10;  
        servol.write(servolPos);  
    }  
    Serial.println(servolPos);  
    Serial.println("Down");  
}
```

To move a motor, call the `write()` function on the `servol` or `servo2` objects and pass the angle (0 to 180) as an argument. For example:

```
servol.write(servolPos);
```

setup()

In the `setup()`, set the servo motor properties. Define the signal frequency.

```
servo1.setPeriodHertz(50); // standard 50 hz servo  
servo2.setPeriodHertz(50); // standard 50 hz servo
```

Use the `attach()` method to set the servo GPIO and minimum and maximum pulse width in microseconds.

```
servo1.attach(SERVO_1, 1000, 2000);  
servo2.attach(SERVO_2, 1000, 2000);
```

Set the motors to its initial position when the ESP32 first boots.

```
servo1.write(servo1Pos);  
servo2.write(servo2Pos);
```

That's pretty much how the code works when it comes to control the servo motors.

Testing the Code

After inserting your network credentials, you can upload the code to your board.

After uploading, open the Serial Monitor to get the board IP address.

```
load:0x40080400,len:6352  
entry 0x400806b8  
.br  
WiFi connected  
Camera Stream Ready! Go to: http://192.168.1.97
```

Autoscroll Show timestamp Both NL & CR 115200 baud Clear output

Open a browser and type the board IP address to get access to the web server.

Click on the buttons and check on the Serial Monitor if everything seems to be working as expected.

If everything is working as expected you can wire the servo motors to the ESP32-CAM and continue with the project.

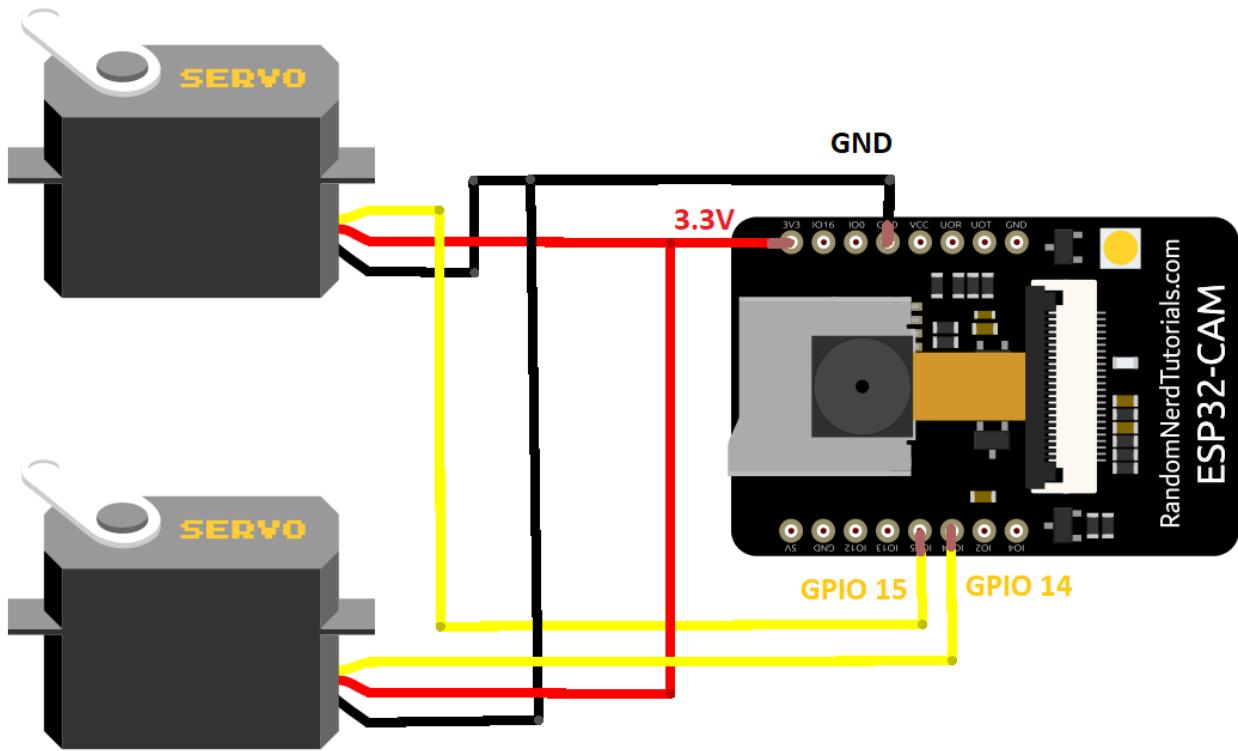
Parts Required

For this project, we'll use the following parts:

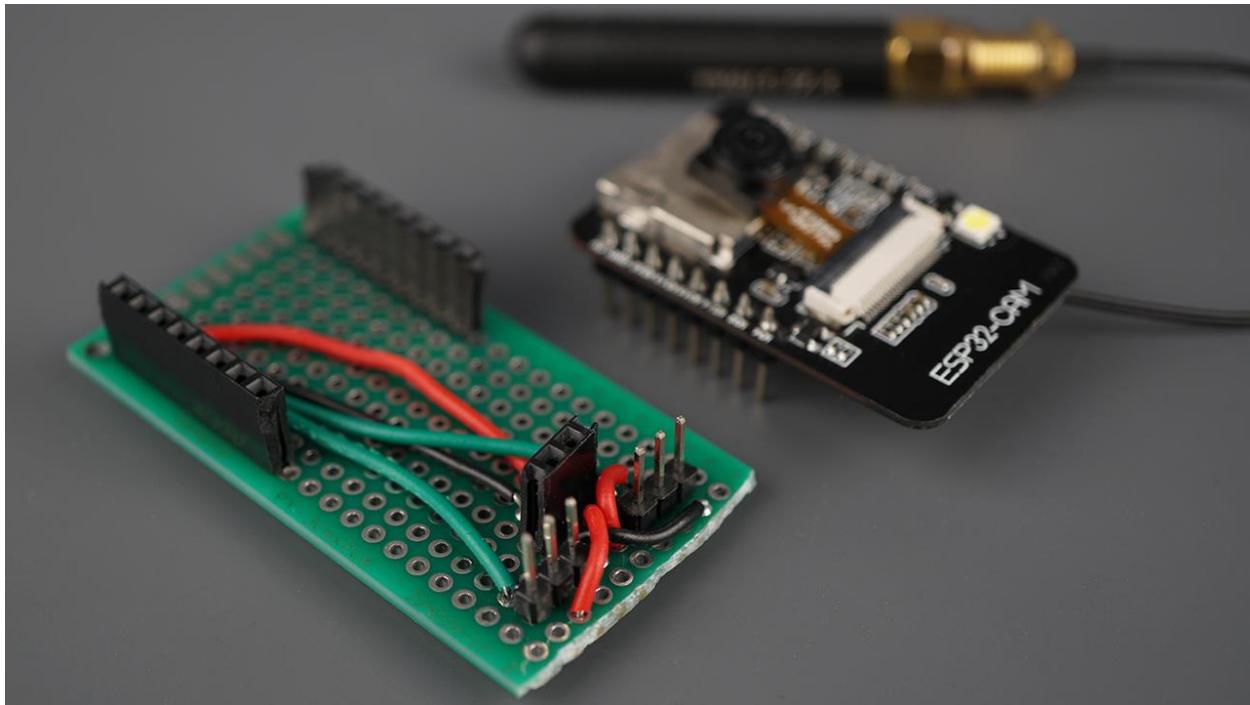
- [ESP32-CAM AI-Thinker with external antenna](#)
- [Pan and tilt stand with SG90 servo motors](#)
- [Prototyping circuit board](#) (optional)

Circuit

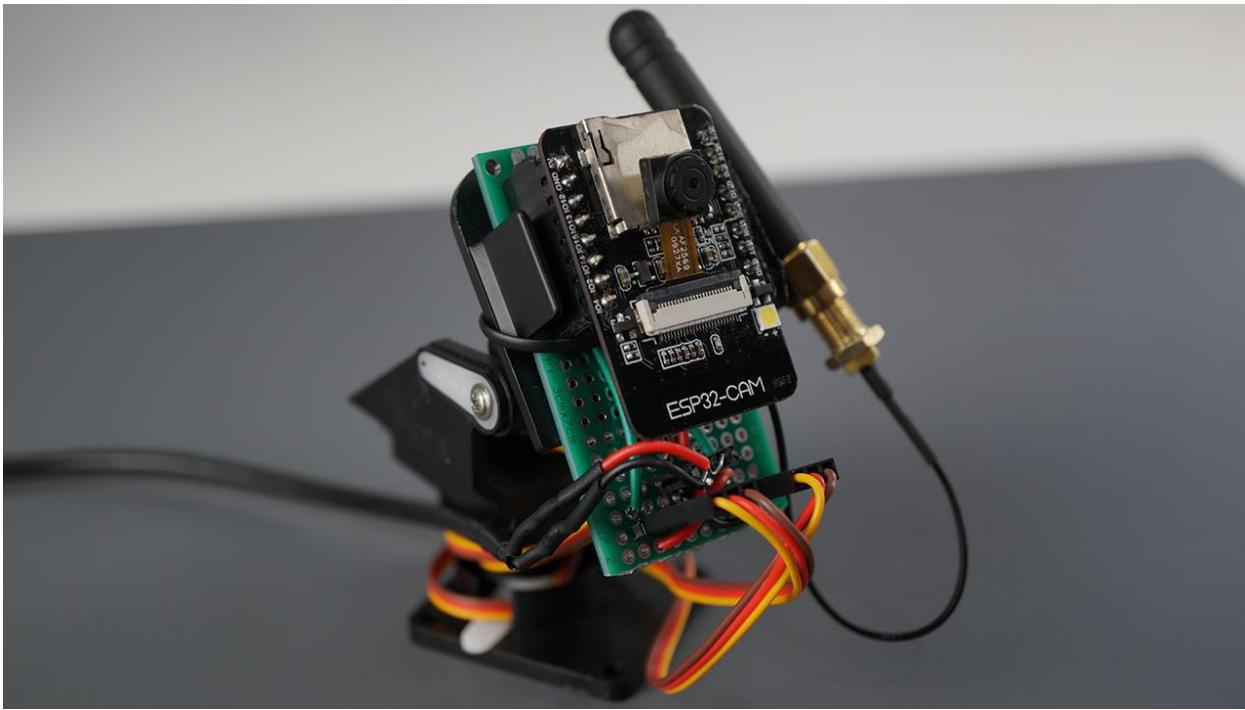
After assembling the pan and tilt stand, connect the servo motors to the ESP32-CAM as shown in the following schematic diagram. We're connecting the servo motor data pins to GPIO 15 and GPIO 14.



You can use a mini breadboard to assemble the circuit or build a mini stripboard with header pins to connect power, the ESP32-CAM and the motors as shown below.



The following figure shows how the pan and tilt stands looks like after assembling.

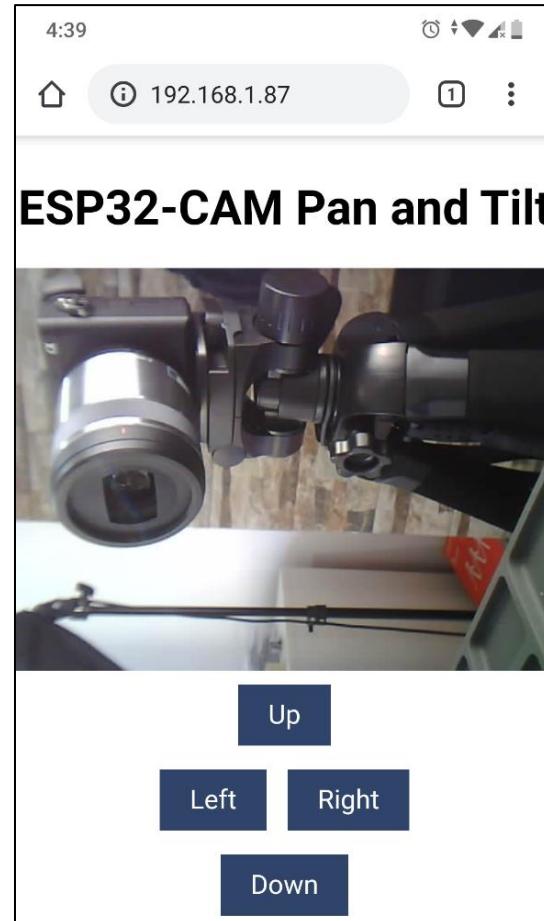


Demonstration

Apply power to your board. Open a browser and type the ESP32-CAM IP address.

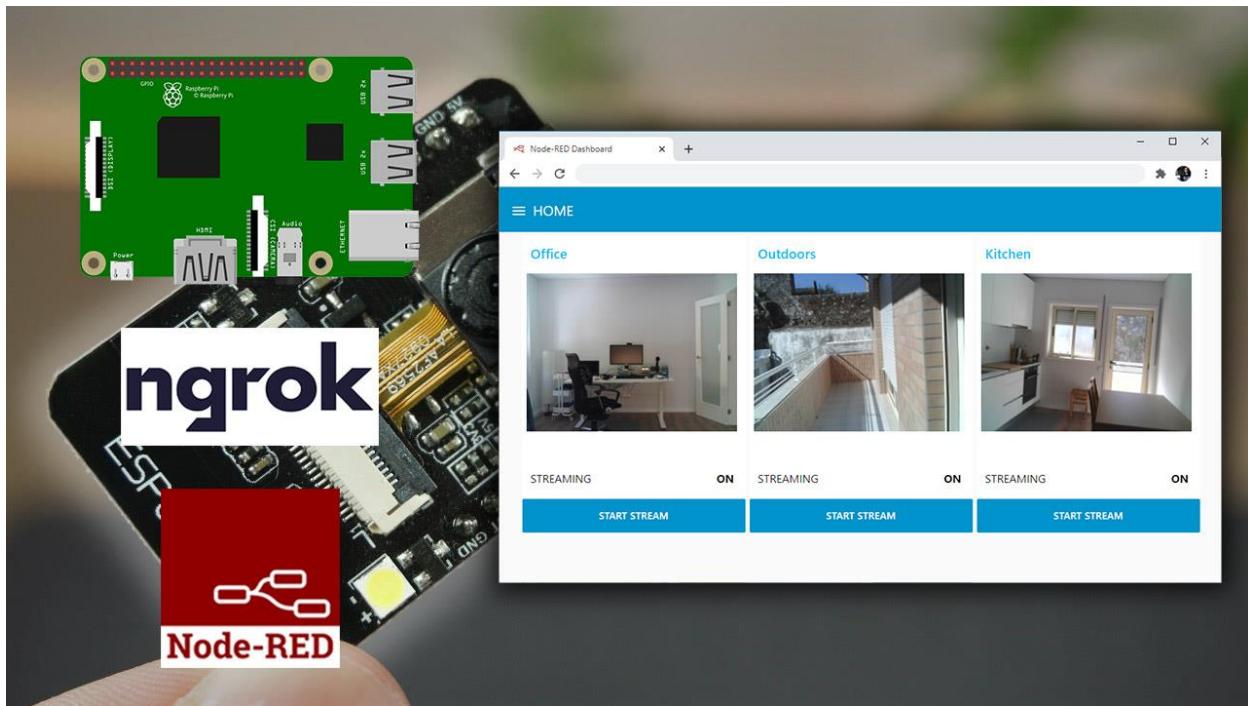
A web page with a real time video streaming should load. Click the buttons to move the camera up, down, left or right.

You can move the camera remotely using the buttons on the web page. This allows you to monitor a different area accordingly to the camera position. This is a great solution for surveillance applications.



Unit 6: ESP32-CAM IP Camera

Access from Anywhere in the World



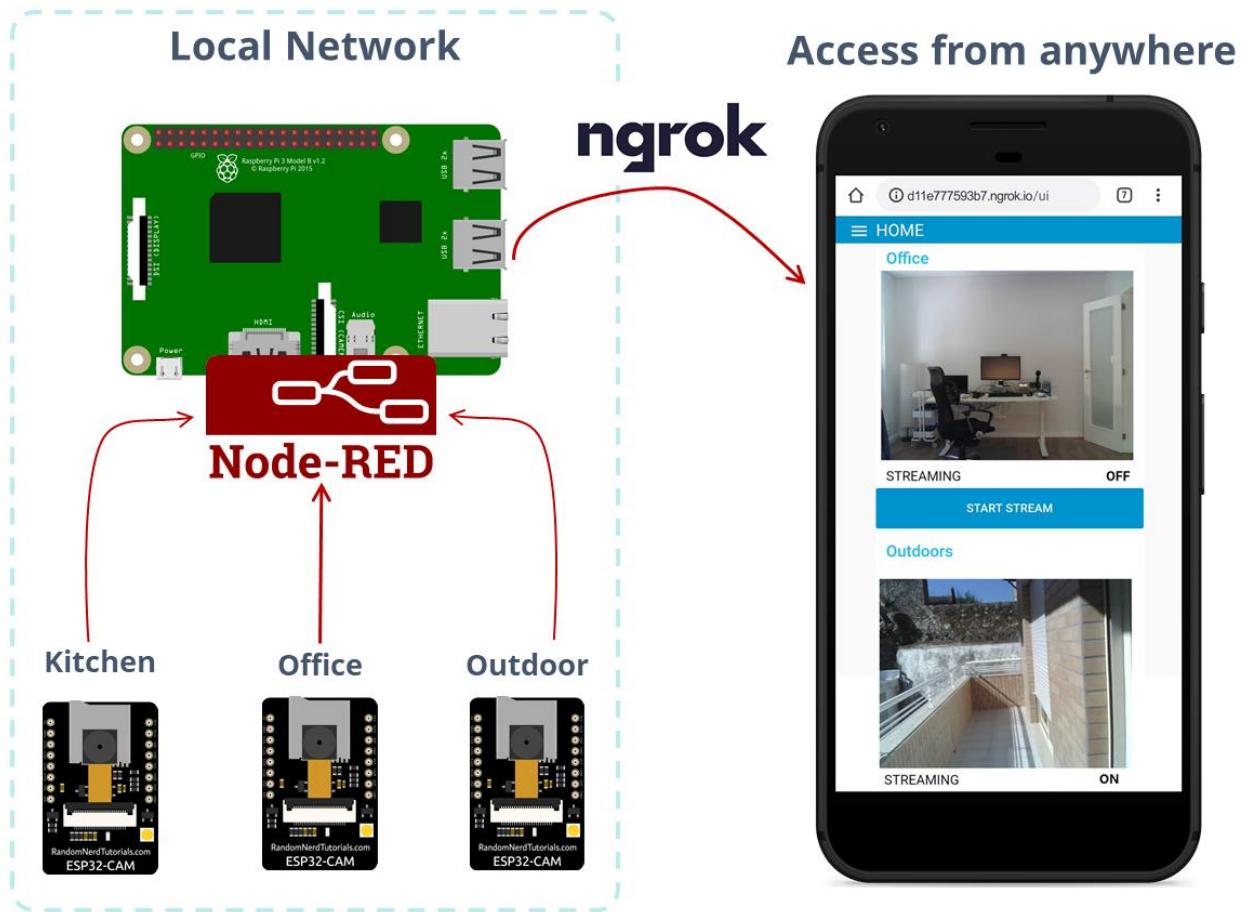
In this Unit you're going to make your ESP32-CAM IP Cameras accessible from anywhere in the world. In previous projects, the ESP32-CAM web servers were only accessible when you are connected to your local network. With this project, you'll be able to create a surveillance camera network to monitor your home from anywhere.

Note: this method requires having a computer running ngrok and Node-RED software. For a permanent solution, it is a good idea to run this software on the low-cost Raspberry Pi computer.

Compatibility: this project is compatible with any ESP32 camera board with the OV2640 camera. You just need to make sure you use the right pinout for the board you're using.

Project Overview

In this project you'll create a video streaming surveillance camera network with several ESP32-CAM boards that you can monitor from anywhere. As an example, we're using three different cameras. The following diagram shows a high-level overview on how everything works.

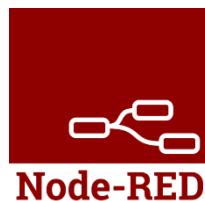


- There are three ESP32-CAM boards running a video streaming web server on your local network. You can easily delete or add more cameras;
- The Raspberry Pi with Node-RED software installed is running on your local network;
- The ESP32-CAM boards send the video streaming to Node-RED;

- Node-RED can display the video streaming from all cameras at the same time. On the interface, there's a button to start the streaming. You can start the streaming for one specific camera or you can start the streaming for all cameras. There's a label indicating whether the streaming is running or not.
- Once you click the “Start Stream” button, that particular ESP32-CAM starts streaming for a predefined number of seconds. This period of time is customizable. This approach prevents the ESP32-CAM from streaming without your permission. Otherwise, the ESP32 would be streaming even if you were not accessing Node-RED interface. This is a good idea to prevent overheating the ESP32-CAM;
- Raspberry Pi board is also running ngrok. This software allows you to create a safe tunnel to your local network. This way, you can monitor your cameras from anywhere in the world.

Installing Node-RED

We'll start by covering what's Node-RED and how to install it. We'll also install the Node-RED Dashboard nodes and the Multipart Decoder node.



Note: building complex flows with Node-RED or explore all its functionalities is not the purpose of this Unit. With this Unit we intend to provide the necessary information on how to connect the ESP32-CAM to Node-RED and access it from anywhere. Many of our readers use Node-RED in their home automation projects and are interested in interfacing the ESP32-CAM with Node-RED. This Unit is mainly addressed to them.

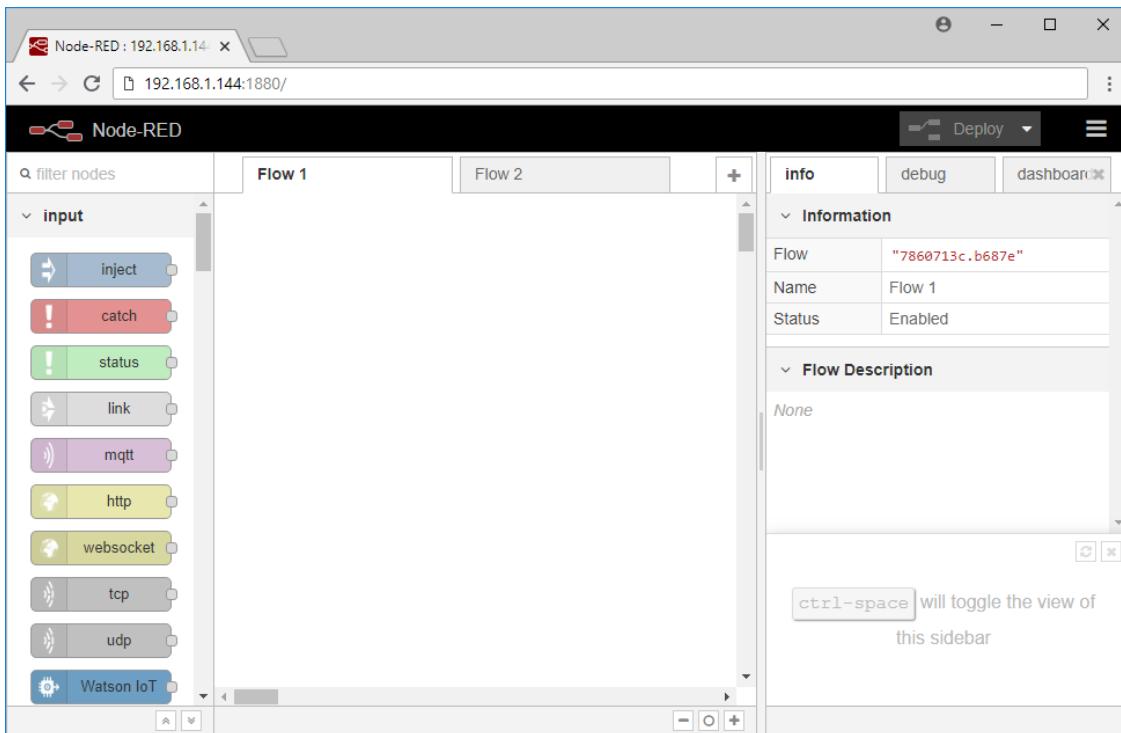
Prerequisites

Before continuing:

- You should be familiar with the Raspberry Pi board – [read Getting Started with Raspberry Pi](#);
- You should have the Raspberry Pi OS or Raspberry Pi OS (32-bit) Lite (previous called Raspbian) installed in your Raspberry Pi – [read Installing Raspbian Lite, Enabling and Connecting with SSH](#);
- You also need the following hardware: [Raspberry Pi board](#) – read [Best Raspberry Pi Starter Kits](#), [MicroSD Card – 16GB Class10](#), and [Power Supply \(5V 2.5A\)](#);

What's Node-RED?

[Node-RED](#) is a powerful open source visual wiring tool for building Internet of Things (IoT) applications. Node-RED uses visual programming that allows you to connect code blocks, known as nodes, together to perform a task. The nodes when wired together are called flows.



Installing Node-RED

Getting Node-RED installed in your Raspberry Pi is quick and easy. It just takes a few commands.

Having a Terminal window of your Raspberry Pi open, enter the next command to install Node-RED:

```
pi@raspberry:~ $ bash <(curl -sL https://raw.githubusercontent.com/node-red/raspbian-deb-package/master/resources/update-nodejs-and-nodered)
```

The installation should be completed after a couple of minutes.

Autostart Node-RED on boot

To automatically run Node-RED when the Pi boots up, you need to enter the following command:

```
pi@raspberry:~ $ sudo systemctl enable nodered.service
```

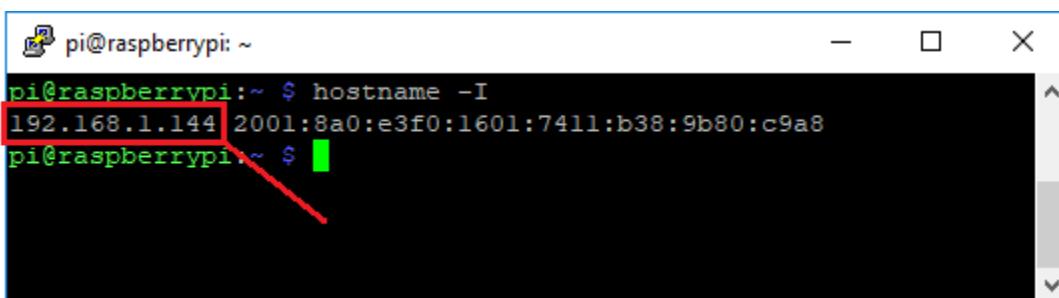
Now, restart your Raspberry Pi, so the auto-start takes effect:

```
pi@raspberry:~ $ sudo reboot
```

Raspberry Pi IP Address

To retrieve your Raspberry Pi IP address, type the next command in your Terminal window:

```
pi@raspberry:~ $ hostname -I
```



In our case, the Raspberry Pi IP address is **192.168.1.144** (save it, because you'll need it later).

Testing the Installation

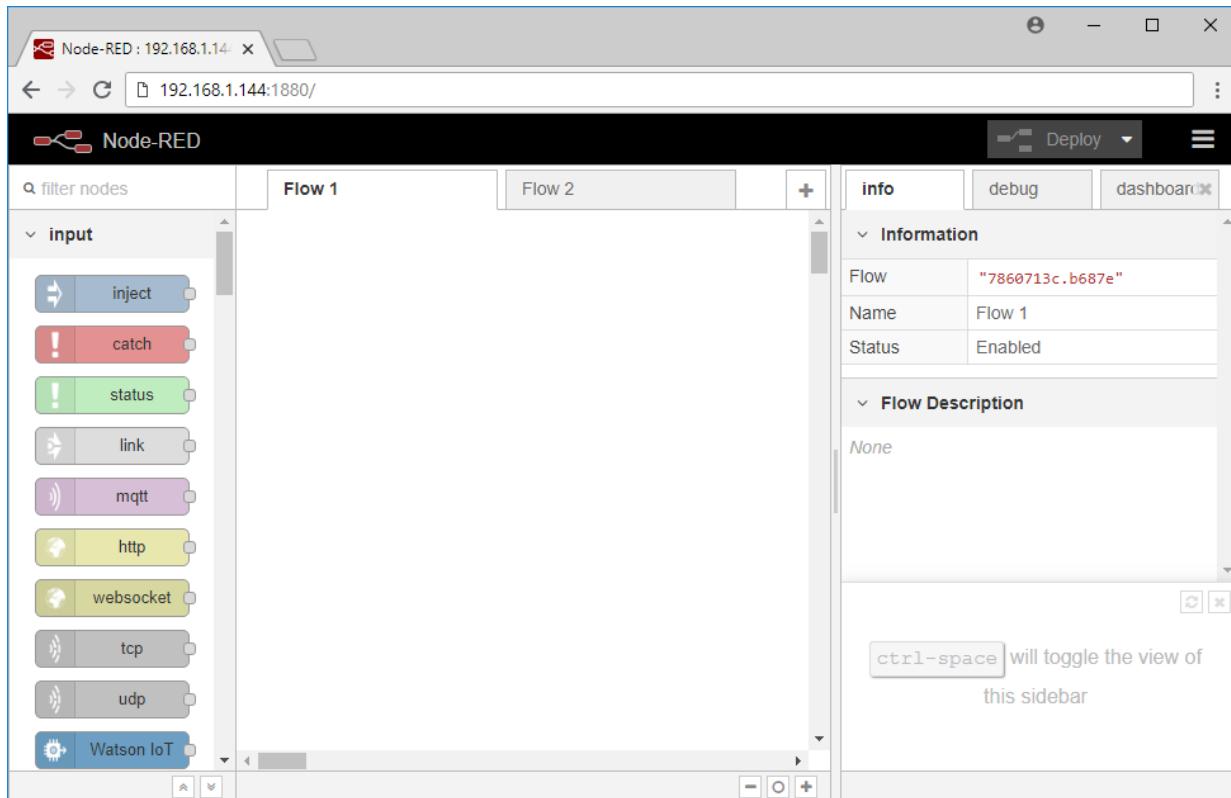
When your Pi is back on, you can test the installation by entering the IP address of your Pi in a web browser followed by the 1880 port number:

`http://YOUR_RPi_IP_ADDRESS:1880`

In my case is:

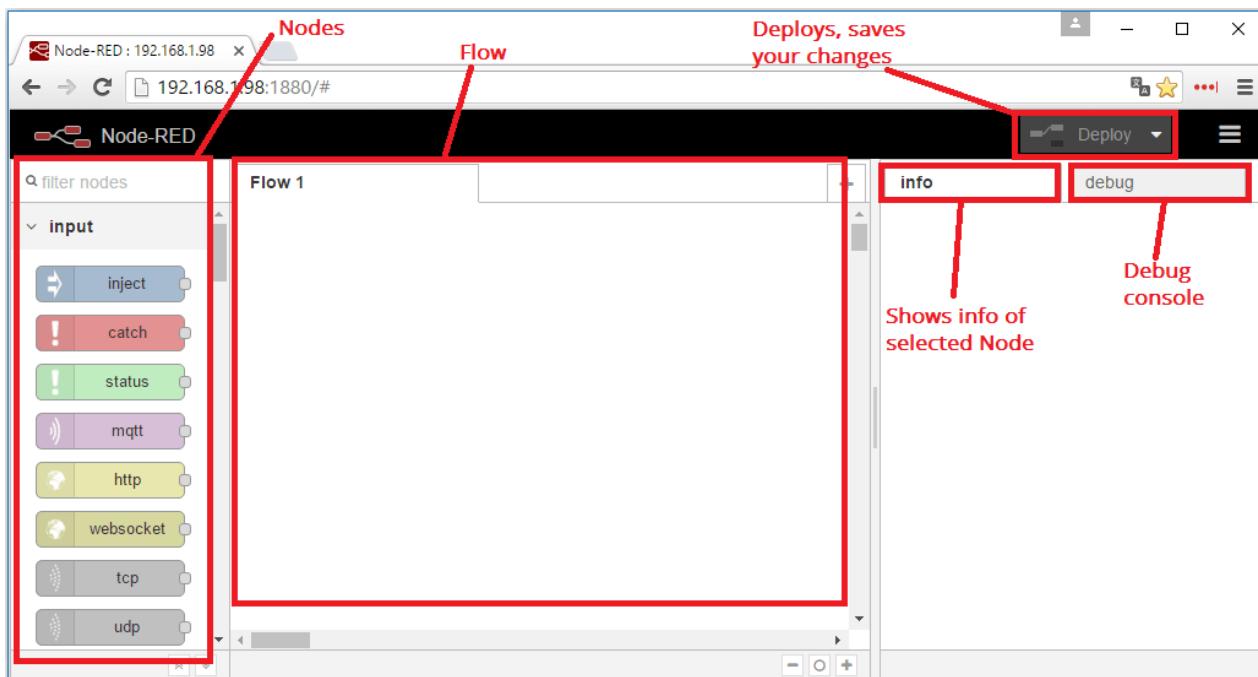
`http://192.168.1.144:1880`

A page as shown in the figure below should load:



Node-RED Overview

On the left-side, you can see a list with a bunch of blocks. Those blocks are called nodes and they are separated by their functionality. If you select a node, you can see how it works in the info tab. In the middle, you have the Flow and this is where you place the nodes.



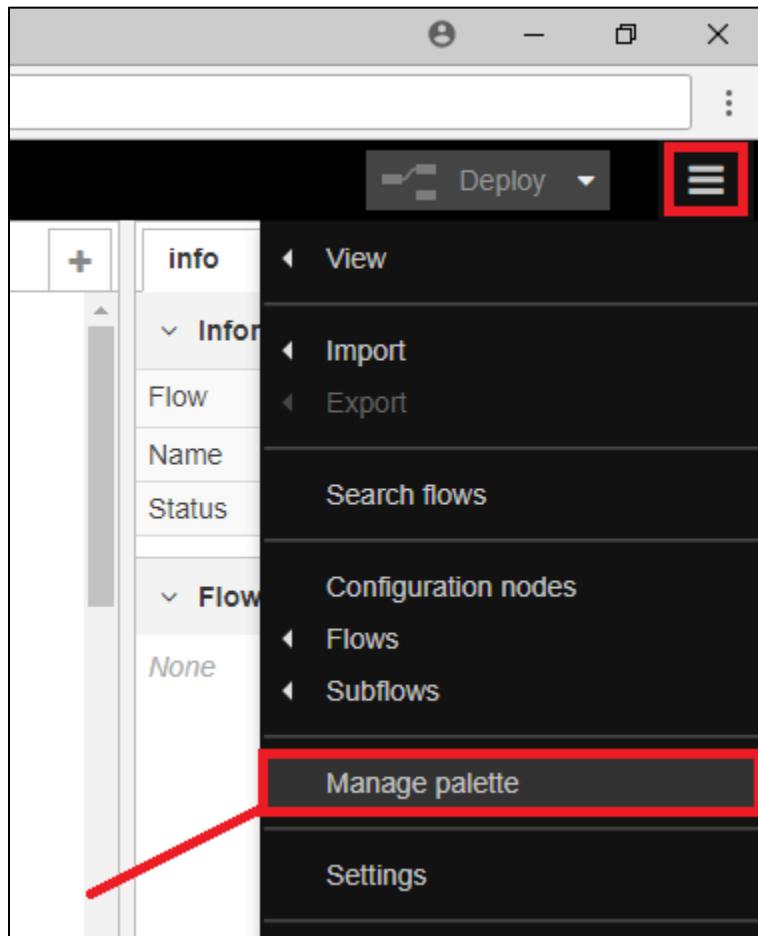
Installing Node-RED Dashboard

Node-RED Dashboard is a set of nodes that provides tools to create a user interface with buttons, charts, text, and much more.

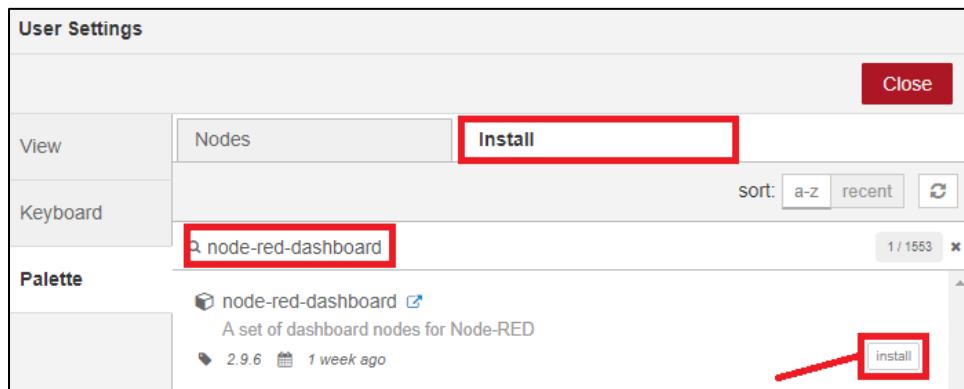
To learn more about Node-RED Dashboard you can check the following links:

- Node-RED site: <http://flows.nodered.org/node/node-red-dashboard>
- GitHub: <https://github.com/node-red/node-red-dashboard>

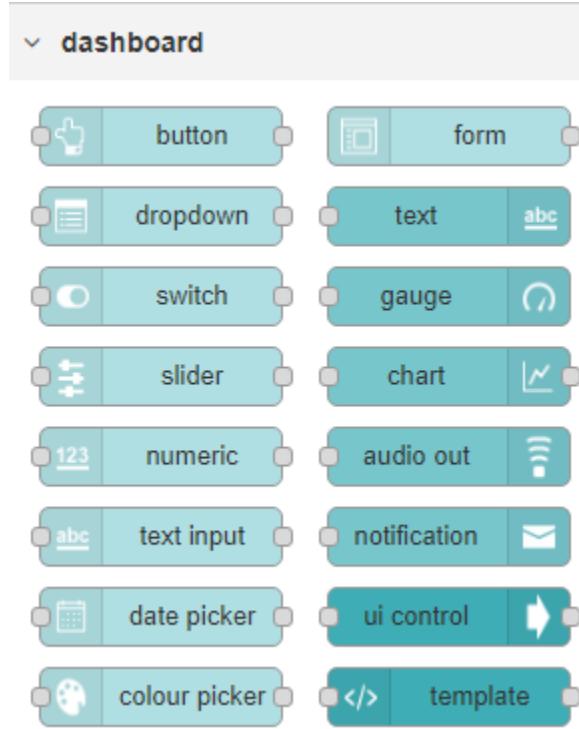
To install the Node-RED Dashboard, go to the “Settings” menu in the top-right corner and open the “Manage palette” menu:



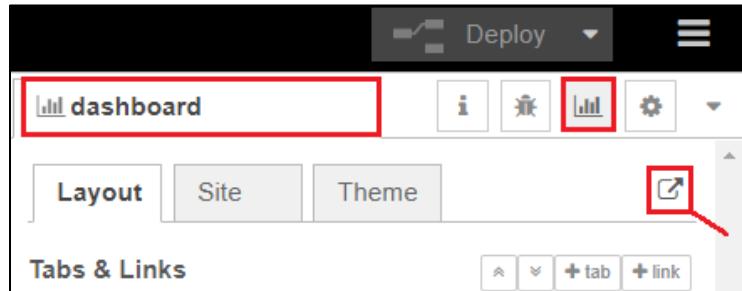
A new window opens with the User Settings. Open the “Install” tab, search for “node-red-dashboard”, and press the “Install” button:



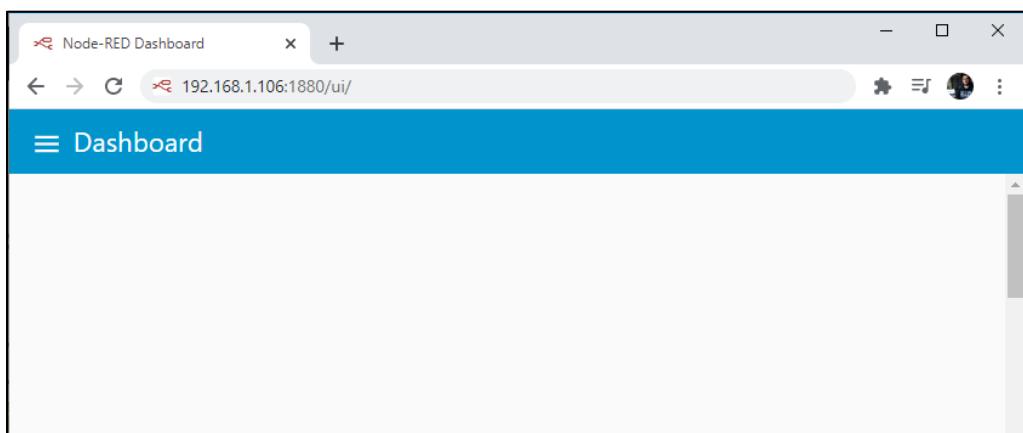
Close the menu and a new set of nodes should appear in your left window with the dashboard nodes, as shown in the following figure.



And in the menu, if you open the dashboard option:



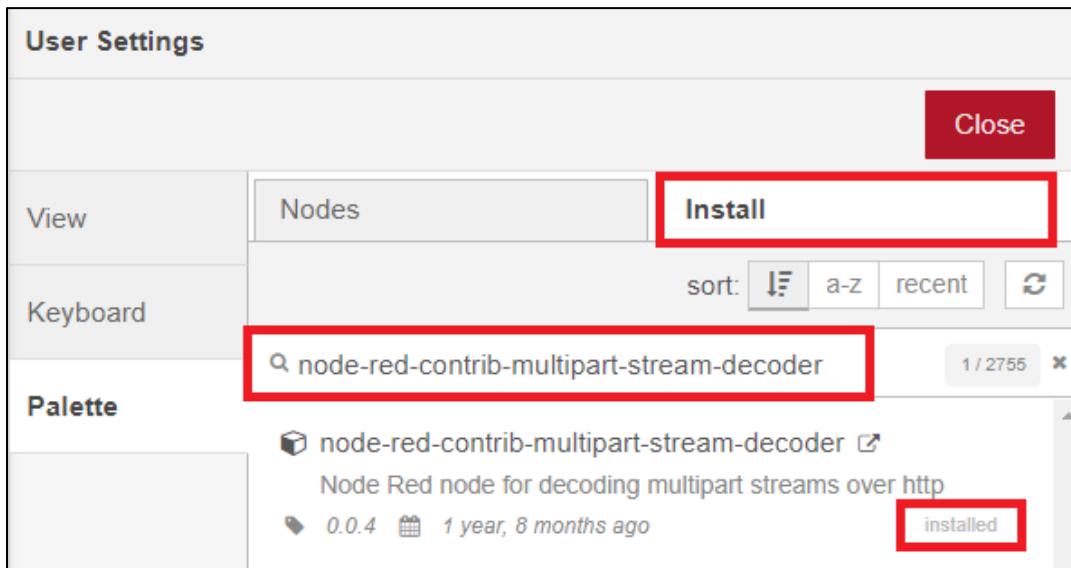
You'll be redirect to this page: <http://Your-RPi-IP-Address:1880/ui>. Your Dashboard should be empty as illustrated below.



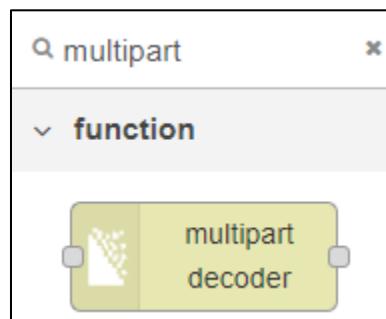
Installing Multipart Stream Decoder

To make this project work, you need to install the Multipart Stream Decoder node that allows you to decode http streams. Basically, this node will make an HTTP request to your ESP32-CAM and decode the image (video streaming).

With the “Install” tab opened, search for “node-red-contrib-multipart-stream-decoder”, and press the “Install” button:



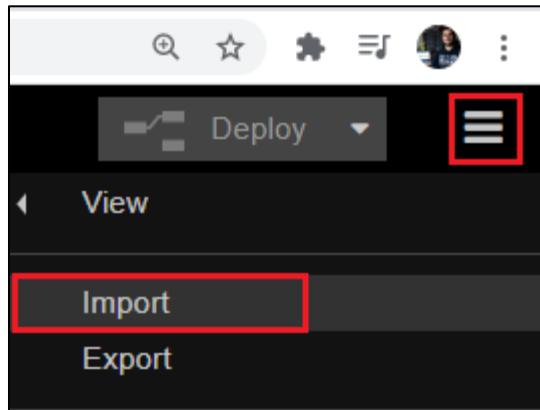
In your left menu, if you search for multipart, it should appear the “multipart decoder” node under the function section.



Now, you have everything ready to integrate Node-RED with your ESP32-CAM and create your home surveillance system.

Importing the Node-RED Flow

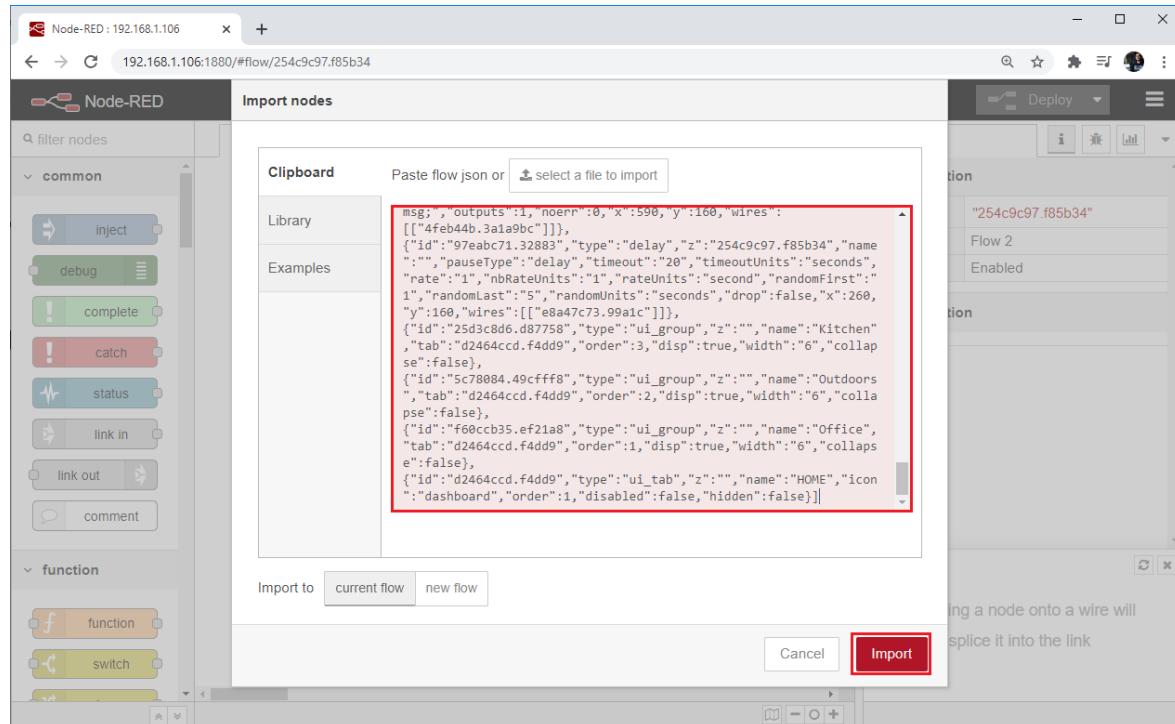
Open the top right menu and select the “Import” option:



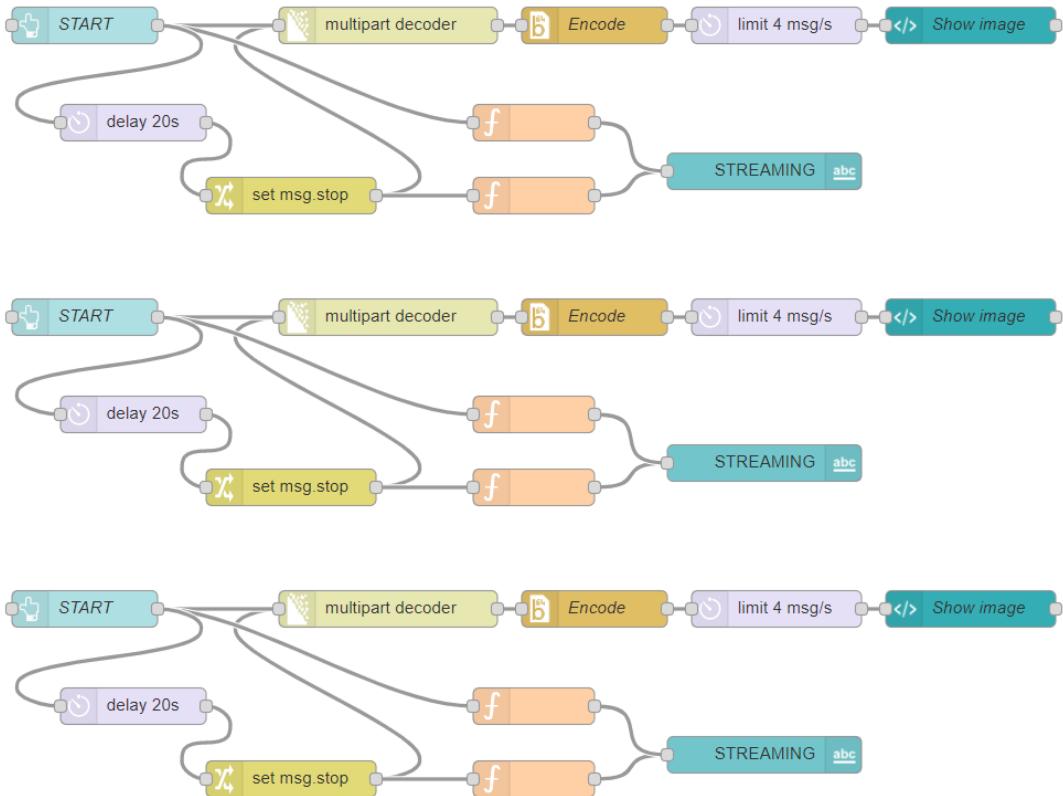
Copy the flow provided to the Clipboard and press the “Import” button. You can open the following link to get access to the Node-RED flow:

CODE

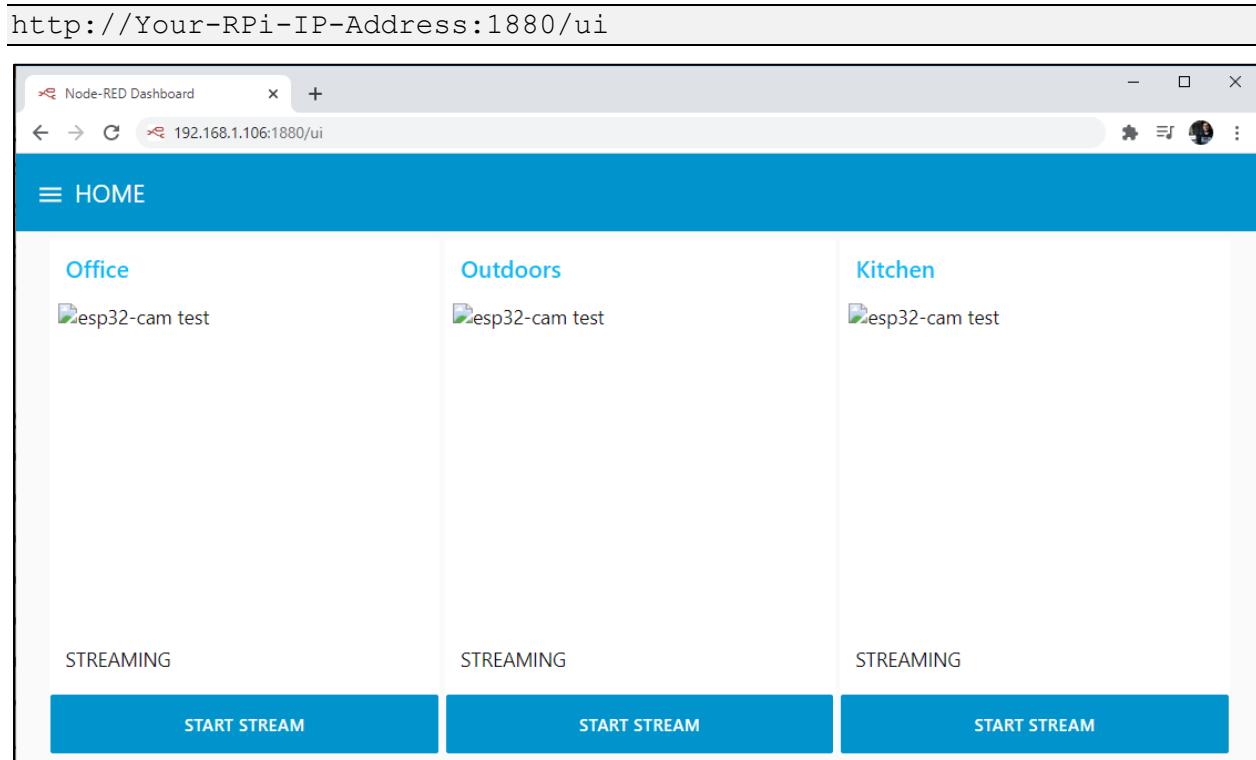
https://github.com/RuiSantosdotme/ESP32-CAM-eBook/blob/master/Code/Module_4/IP_Camera_Anywhere/node_red_flow.txt



The following image shows how your flow should look like:



Access your Raspberry Pi IP address followed by **:1880/ui** as shown below.



At the moment, you don't see any video streaming because you haven't prepared the ESP32-CAM boards yet. Additionally, it shows video streaming for three cameras. You can easily copy the provided nodes to add more cameras or delete some of them to delete a camera.

Preparing your ESP32-CAM

For this project, we'll program three ESP32-CAM boards with a sketch used in a previous Unit (Module 4, Unit 3). You can use one ESP32-CAM or as many boards as you want.

Copy the code provided and upload it to your board(s). This code turns your ESP32-CAM into an IP camera. We'll be using the ESP32-CAM AI Thinker Module, but you can use any other ESP32 camera, just check the pin assignment for the board you're using.

CODE

https://github.com/RuiSantosdotme/ESP32-CAM-eBook/blob/master/Code/Module_4/IP_Camera_Anywhere/IP_Camera_Anywhere.ino

```
#include "esp_camera.h"
#include <WiFi.h>
#include "esp_timer.h"
#include "img_converters.h"
#include "Arduino.h"
#include "fb_gfx.h"
#include "soc/soc.h" //disable brownout problems
#include "soc/rtc_CNTL_REG.h" //disable brownout problems
#include "esp_http_server.h"

//Replace with your network credentials
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

#define PART_BOUNDARY "123456789000000000000987654321"

// This project was tested with the AI Thinker Model, M5STACK PSRAM Model and
M5STACK WITHOUT PSRAM
#define CAMERA_MODEL_AI_THINKER
//##define CAMERA_MODEL_M5STACK_PSRAM
//##define CAMERA_MODEL_M5STACK_WITHOUT_PSRAM
//##define CAMERA_MODEL_M5STACK_WITHOUT_PSRAM_B
```

```

// Not tested with this model
//#define CAMERA_MODEL_WROVER_KIT

#if defined(CAMERA_MODEL_WROVER_KIT)
#define PWDN_GPIO_NUM      -1
#define RESET_GPIO_NUM     -1
#define XCLK_GPIO_NUM      21
#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         19
#define Y4_GPIO_NUM         18
#define Y3_GPIO_NUM         5
#define Y2_GPIO_NUM         4
#define VSYNC_GPIO_NUM      25
#define HREF_GPIO_NUM       23
#define PCLK_GPIO_NUM       22

#elif defined(CAMERA_MODEL_M5STACK_PSRAM)
#define PWDN_GPIO_NUM      -1
#define RESET_GPIO_NUM     15
#define XCLK_GPIO_NUM      27
#define SIOD_GPIO_NUM      25
#define SIOC_GPIO_NUM      23

#define Y9_GPIO_NUM         19
#define Y8_GPIO_NUM         36
#define Y7_GPIO_NUM         18
#define Y6_GPIO_NUM         39
#define Y5_GPIO_NUM         5
#define Y4_GPIO_NUM         34
#define Y3_GPIO_NUM         35
#define Y2_GPIO_NUM         32
#define VSYNC_GPIO_NUM      22
#define HREF_GPIO_NUM       26
#define PCLK_GPIO_NUM       21

#elif defined(CAMERA_MODEL_M5STACK_WITHOUT_PSRAM)
#define PWDN_GPIO_NUM      -1
#define RESET_GPIO_NUM     15
#define XCLK_GPIO_NUM      27
#define SIOD_GPIO_NUM      25
#define SIOC_GPIO_NUM      23

#define Y9_GPIO_NUM         19
#define Y8_GPIO_NUM         36
#define Y7_GPIO_NUM         18
#define Y6_GPIO_NUM         39
#define Y5_GPIO_NUM         5
#define Y4_GPIO_NUM         34
#define Y3_GPIO_NUM         35
#define Y2_GPIO_NUM         17
#define VSYNC_GPIO_NUM      22
#define HREF_GPIO_NUM       26
#define PCLK_GPIO_NUM       21

```

```

#define PWDN_GPIO_NUM      -1
#define RESET_GPIO_NUM     15
#define XCLK_GPIO_NUM      27
#define SIOD_GPIO_NUM      22
#define SIOC_GPIO_NUM      23

#define Y9_GPIO_NUM         19
#define Y8_GPIO_NUM         36
#define Y7_GPIO_NUM         18
#define Y6_GPIO_NUM         39
#define Y5_GPIO_NUM          5
#define Y4_GPIO_NUM         34
#define Y3_GPIO_NUM         35
#define Y2_GPIO_NUM         32
#define VSYNC_GPIO_NUM      25
#define HREF_GPIO_NUM       26
#define PCLK_GPIO_NUM       21

#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

#else
    #error "Camera model not selected"
#endif

static const char* _STREAM_CONTENT_TYPE = "multipart/x-mixed-replace;boundary="
PART_BOUNDARY;
static const char* _STREAM_BOUNDARY = "\r\n--" PART_BOUNDARY "\r\n";
static const char* _STREAM_PART = "Content-Type: image/jpeg\r\nContent-Length:
%u\r\n\r\n";

httpd_handle_t stream_httpd = NULL;

static esp_err_t stream_handler(httpd_req_t *req) {
    camera_fb_t * fb = NULL;
    esp_err_t res = ESP_OK;
    size_t jpg_buf_len = 0;
    uint8_t * jpg_buf = NULL;
    char * part_buf[64];

    res = httpd_resp_set_type(req, _STREAM_CONTENT_TYPE);
    if(res != ESP_OK) {

```

```

        return res;
    }

    while(true) {
        fb = esp_camera_fb_get();
        if (!fb) {
            Serial.println("Camera capture failed");
            res = ESP_FAIL;
        } else {
            if(fb->width > 400){
                if(fb->format != PIXFORMAT_JPEG){
                    bool jpeg_converted = frame2jpg(fb, 80, &_jpg_buf, &_jpg_buf_len);
                    esp_camera_fb_return(fb);
                    fb = NULL;
                    if(!jpeg_converted){
                        Serial.println("JPEG compression failed");
                        res = ESP_FAIL;
                    }
                } else {
                    _jpg_buf_len = fb->len;
                    _jpg_buf = fb->buf;
                }
            }
        }
        if(res == ESP_OK){
            size_t hlen = snprintf((char *)part_buf, 64, _STREAM_PART, _jpg_buf_len);
            res = httpd_resp_send_chunk(req, (const char *)part_buf, hlen);
        }
        if(res == ESP_OK){
            res = httpd_resp_send_chunk(req, (_jpg_buf, _jpg_buf_len));
        }
        if(res == ESP_OK){
            res = httpd_resp_send_chunk(req, _STREAM_BOUNDARY, strlen(_STREAM_BOUNDARY));
        }
        if(fb){
            esp_camera_fb_return(fb);
            fb = NULL;
            _jpg_buf = NULL;
        } else if(_jpg_buf){
            free(_jpg_buf);
            _jpg_buf = NULL;
        }
        if(res != ESP_OK){
            break;
        }
        //Serial.printf("MJPG: %uB\n", (uint32_t) (_jpg_buf_len));
    }
    return res;
}

void startCameraServer(){
    httpd_config_t config = HTTPD_DEFAULT_CONFIG();
    config.server_port = 80;

    httpd_uri_t index_uri = {
        .uri      = "/",
        .method   = HTTP_GET,
        .handler  = stream_handler,
        .user_ctx = NULL
}

```

```

};

//Serial.printf("Starting web server on port: '%d'\n", config.server_port);
if (httpd_start(&stream_httpd, &config) == ESP_OK) {
    httpd_register_uri_handler(stream_httpd, &index_uri);
}
}

void setup() {
    WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0); //disable brownout detector

    Serial.begin(115200);
    Serial.setDebugOutput(false);

    camera_config_t config;
    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;

    if(psramFound()) {
        config.frame_size = FRAMESIZE_UXGA;
        config.jpeg_quality = 10;
        config.fb_count = 2;
    } else {
        config.frame_size = FRAMESIZE_SVGA;
        config.jpeg_quality = 12;
        config.fb_count = 1;
    }

    // Camera init
    esp_err_t err = esp_camera_init(&config);
    if (err != ESP_OK) {
        Serial.printf("Camera init failed with error 0x%x", err);
        return;
    }
    // Wi-Fi connection
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.println("");
}

```

```

Serial.println("WiFi connected");

Serial.print("Camera Stream Ready! Go to: http://");
Serial.print(WiFi.localIP());

// Start streaming web server
startCameraServer();
}

void loop() {
  delay(1);
}

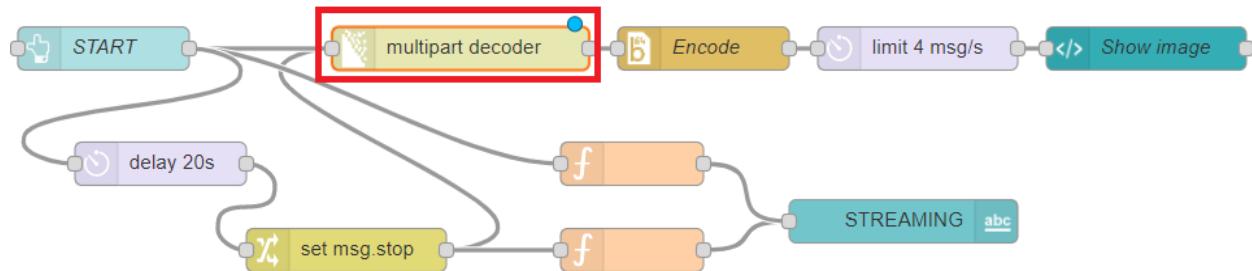
```

After uploading the code, open your Arduino IDE Serial Monitor and restart your ESP32-CAM to print its IP Address. Save the IP address. You'll need it to add the camera to Node-RED multipart node settings.

You can upload the same code to all your other ESP32-CAM boards. For demonstration purposes, we'll be using three boards.

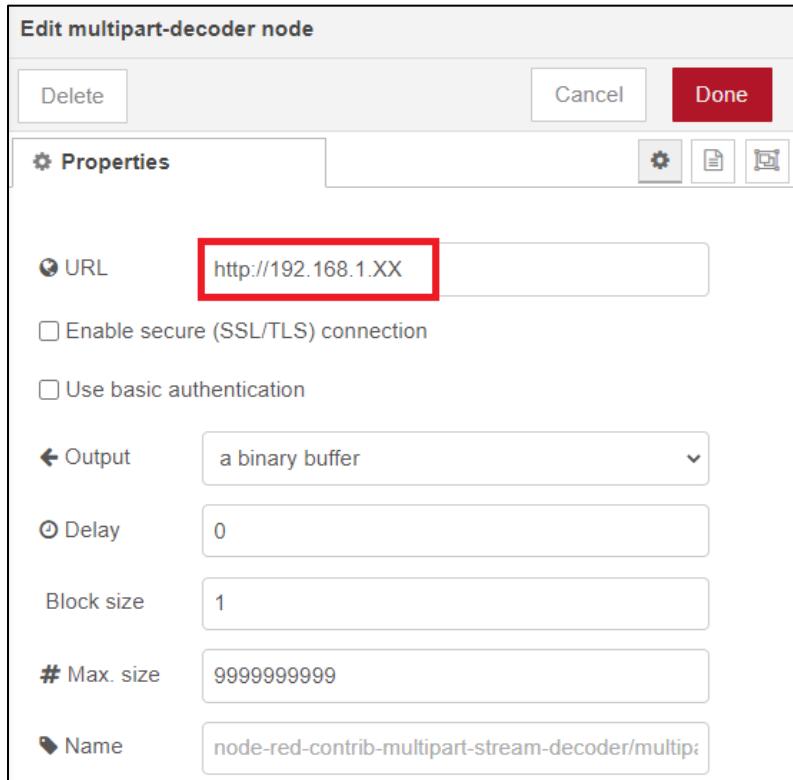
Adding the Camera to Node-RED

In your Node-RED dashboard, double-click the “multipart decoder” node and edit the URL field with your ESP32-CAM IP address.

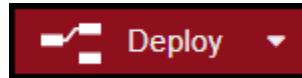


In my case, the URL will be replaced with <http://192.168.1.95>. Once, you make that change, press the “Done” button.

Edit all the other multipart decoder nodes with the IP address of the other cameras.



Finally, press the “Deploy” button for all changes to take effect.



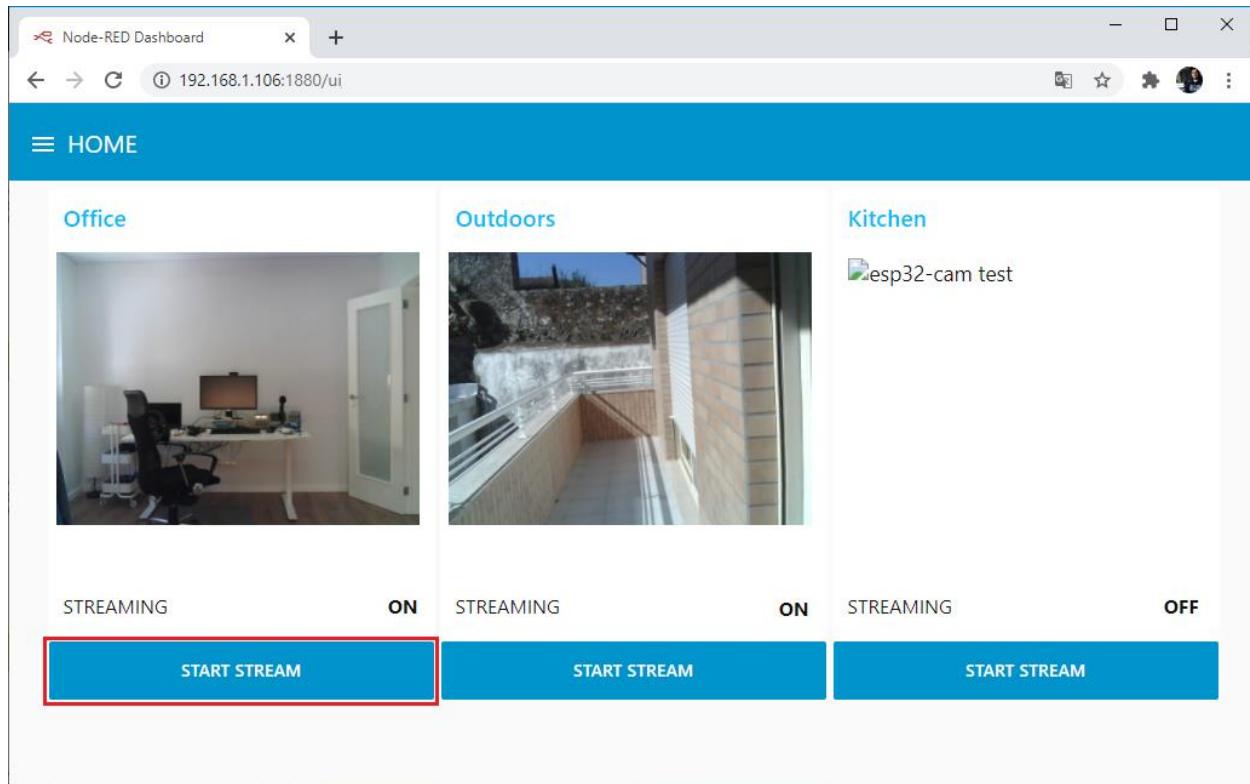
Demonstration

On your local network, access Node-RED UI on the following URL:

Your-RPi-IP-Address:1880/ui

Press the “Start Stream” button for each camera and you should get access to the video streaming.

Once you press the “Start Stream” button, it will stream video for 20 seconds (you can change this time on the Node-RED flow). Next to each video streaming, there’s a label indicating whether video streaming is on or off.

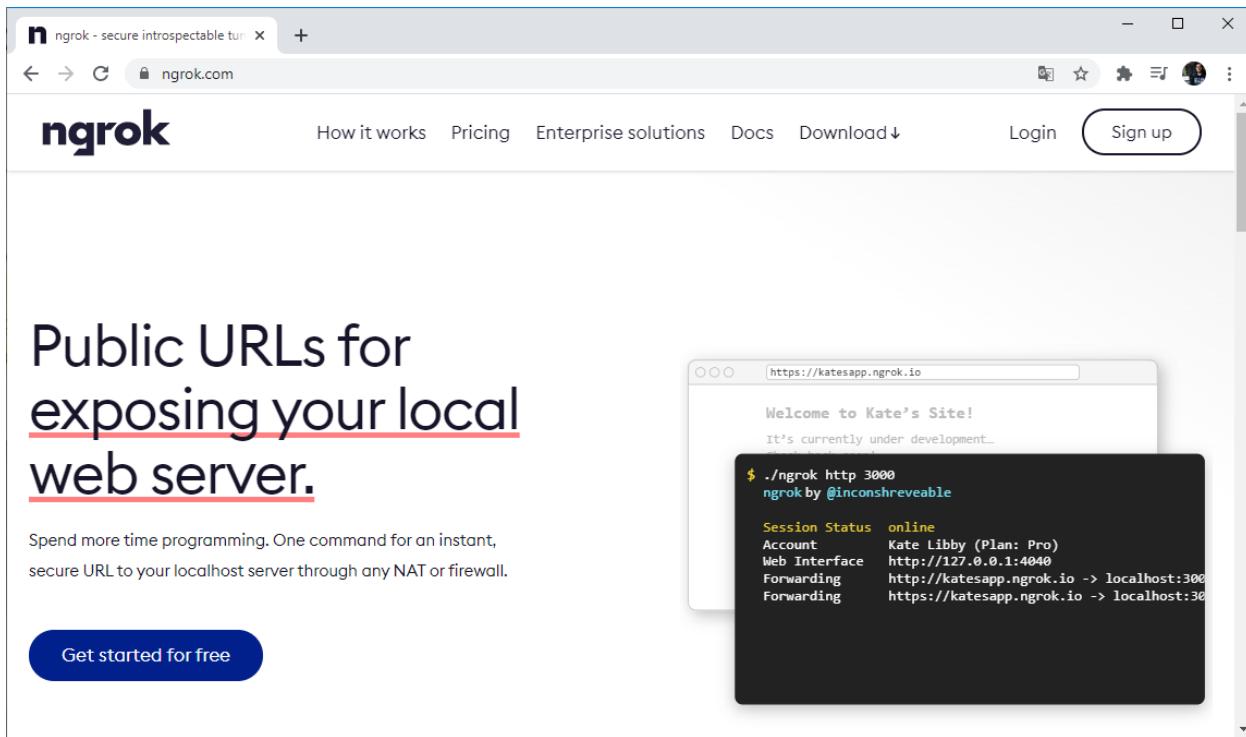


At the moment, you can only access this on your local network. Let's see how to make it accessible from anywhere in the next page.

Introducing ngrok

To make your camera streaming page accessible from anywhere, you are going to use a free service called **ngrok** to create a tunnel to your Node-RED dashboard.

Go to <https://ngrok.com> to create your account.



Click the blue “Sign up” button and enter your details in the fields.

After creating your account, login and go to the “**Authentication**” tab to find Your Tunnel Authtoken.

Copy your unique **Your Tunnel Authtoken** to a safe place (you'll need it later in this Unit).

The screenshot shows the ngrok dashboard interface. On the left, there's a sidebar with navigation links: Getting Started, Status, Endpoints, Authentication (which has a red box around it), Your AuthToken (which also has a red box around it), Tunnel Authtokens, API Keys, SSH Keys, and IP Whitelist. The main content area is titled 'Your AuthToken'. It contains a message: 'This is your personal Authtoken. Use this to authenticate the ngrok agent that you downloaded.' Below this is a text input field containing a long string of characters ('3V1MfHcNMD9rhBif8TRs_2whamY91tqX4kcKp5X---') which is also highlighted with a red box. To the right of the input field is a blue 'Copy' button.

My Tunnel Authtoken is:

```
3V1MfHcNMD9rhBif8TRs_2whamY91tqX4kcKp5X---
```

Installing ngrok in the RPi

Having an SSH communication established with your Raspberry Pi:



Send the following command to download the latest version of ngrok software:

```
pi@raspberry:~ $ wget http://randomnerdtutorials.com/ngrokARM
```

Unzip the ngrok software using:

```
pi@raspberry:~ $ unzip ngrokARM
```

Running ngrok

Enter the following command to authenticate your ngrok account (replace the red highlighted text with your own Tunnel Authtoken):

```
pi@raspberry:~ $ ./ngrok authtoken  
3V1MfHcNMD9rhBzg8TRs_2whamY91tqX4kcKp5X---
```

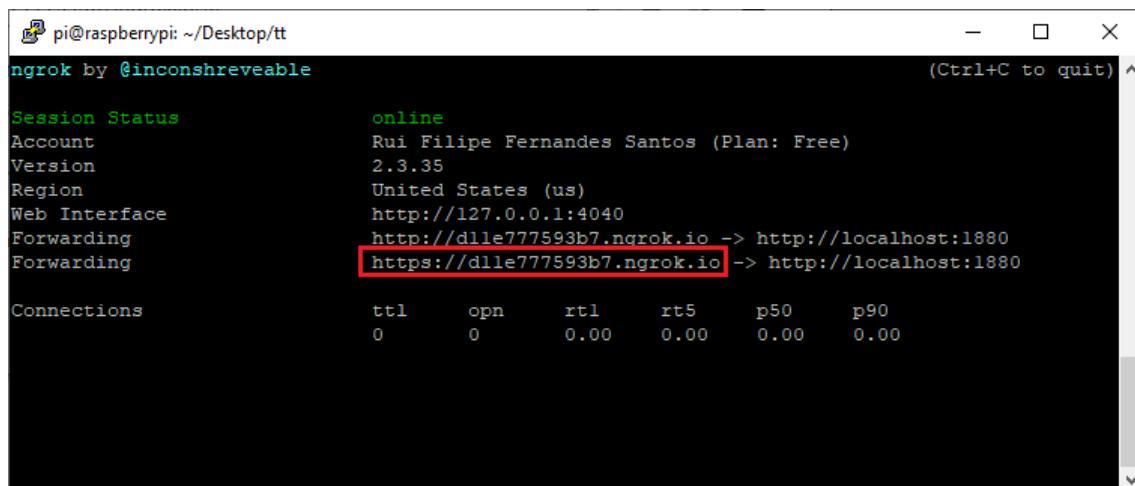
In your terminal window, enter the following command and replace the red text with your desired credentials:

```
pi@raspberry:~ $ ./ngrok http -auth="ruisan:pass" 1880
```

When you try to access the UI, you'll be asked to enter a username (**ruisan**) and a password (**pass**).



When you run the previous command, a new window shows up:



Copy your unique link (it is highlighted in the preceding figure):

```
https://d11e777593b7.ngrok.io
```

Important: do not use the **http** link, because that link is not encrypted:

http://d11e777593b7.ngrok.io

Important: you should always use the **https** link:

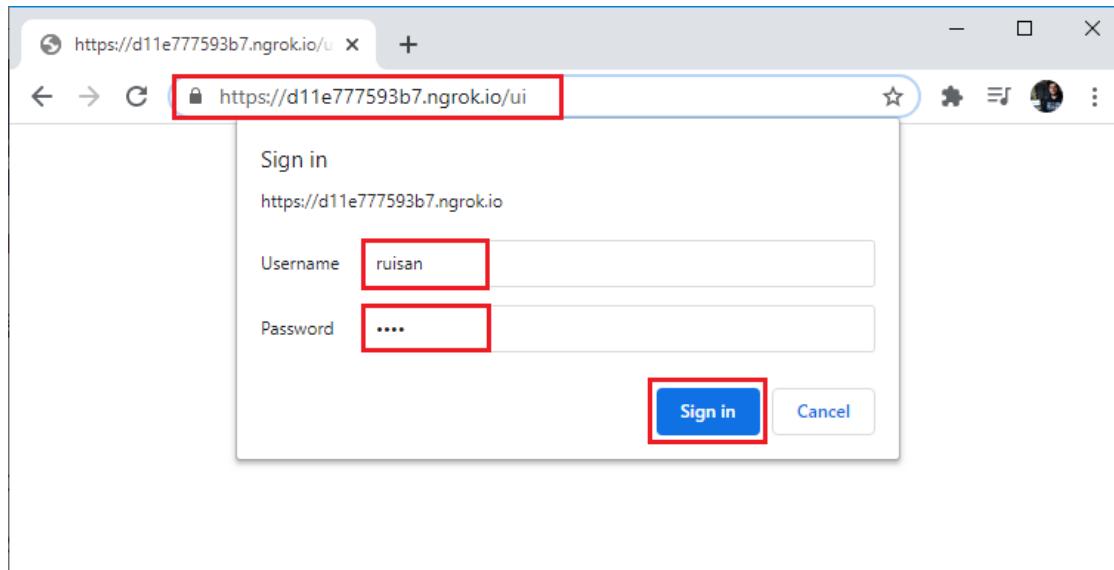
https://d11e777593b7.ngrok.io

Accessing Your Dashboard from Anywhere

If everything runs smoothly, you can open the Node-RED Dashboard from any web browser in the world. You just need to enter the URL you've got previously followed by **/ui**, as shown below:

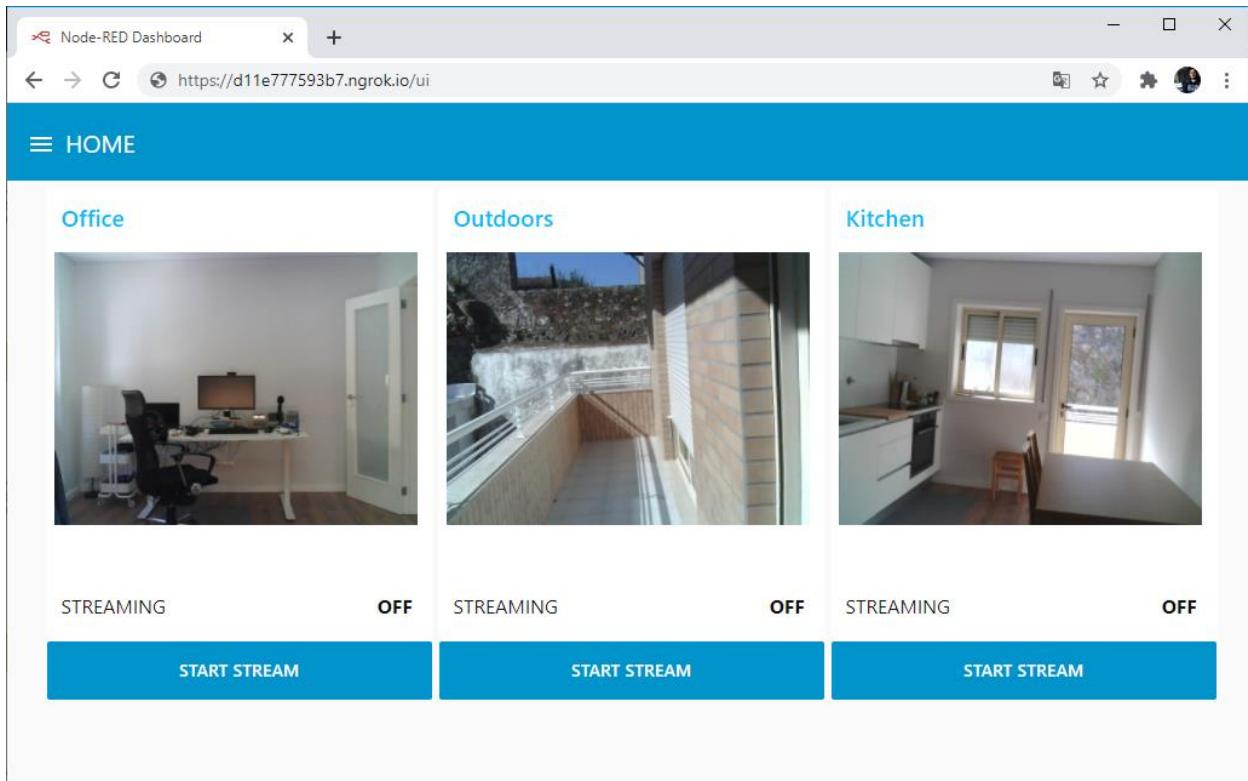
https://d11e777593b7.ngrok.io/ui

You'll be asked to enter a username (**ruisan**) and password (**pass**):



After entering the correct credentials for your tunnel, your Node-RED Dashboard loads.

Now, you have full control over your cameras from anywhere in the world.



Important: you can close the SSH window, but you can't quit the ngrok software, otherwise your tunnel stops. Additionally, you need to let your computer on and running ngrok in order to maintain the tunnel online.

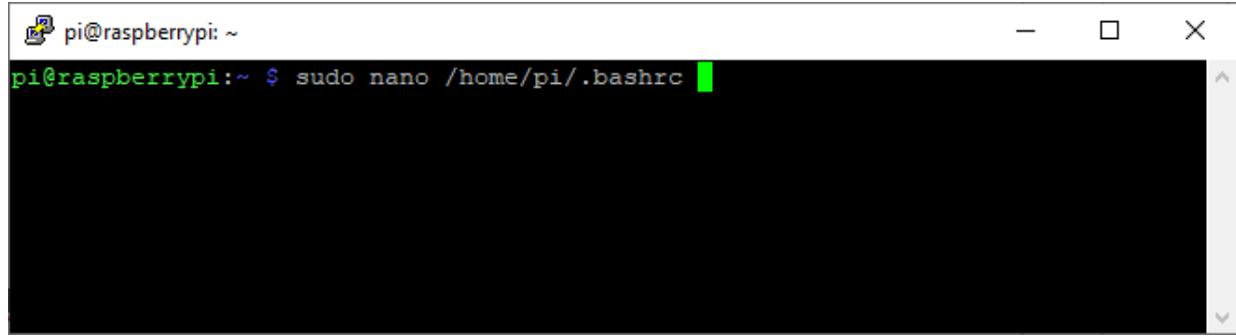
Autostart ngrok on boot

To make ngrok software autostart when the Raspberry Pi first boots, you need to install the screen software that allows you to run commands in the background:

```
pi@raspberry:~ $ sudo apt install screen -y
```

Finally, you need to edit the file /home/pi/.bashrc to tell your Raspberry Pi to run ngrok on start. Type the next command in your terminal window:

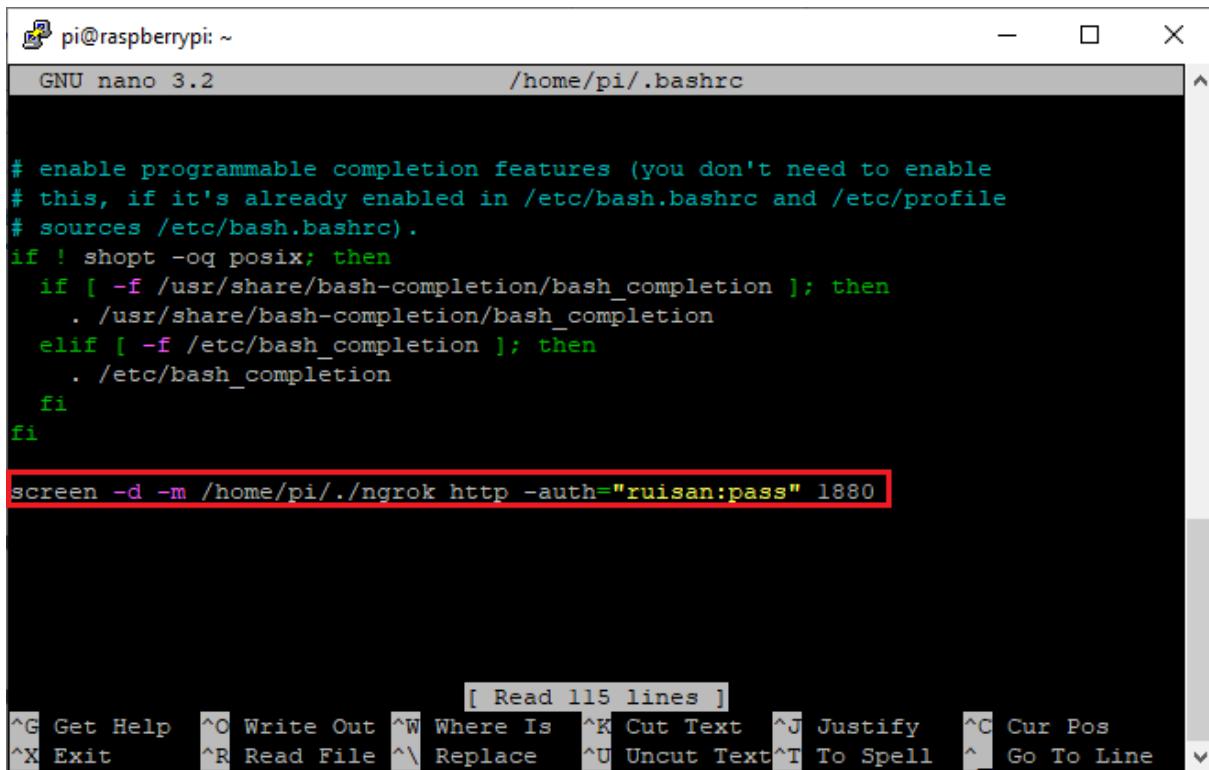
```
pi@raspberry:~ $ sudo nano /home/pi/.bashrc
```



```
pi@raspberrypi:~ $ sudo nano /home/pi/.bashrc
```

Scroll down to the bottom of the `.bashrc` file and add the following command:

```
screen -d -m /home/pi./ngrok http -auth="ruisan:pass" 1880
```



```
GNU nano 3.2          /home/pi/.bashrc

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
    if [ -f /usr/share/bash-completion/bash_completion ]; then
        . /usr/share/bash-completion/bash_completion
    elif [ -f /etc/bash_completion ]; then
        . /etc/bash_completion
    fi
fi

screen -d -m /home/pi./ngrok http -auth="ruisan:pass" 1880
```

[Read 115 lines]

^G Get Help **^O** Write Out **^W** Where Is **^K** Cut Text **^J** Justify **^C** Cur Pos
^X Exit **^R** Read File **^V** Replace **^U** Uncut Text **^T** To Spell **^** Go To Line

Press **Ctrl+X**, followed by **Y** and **Enter** key to save the file. Then, restart your Raspberry Pi to test if it's auto starting ngrok on boot:

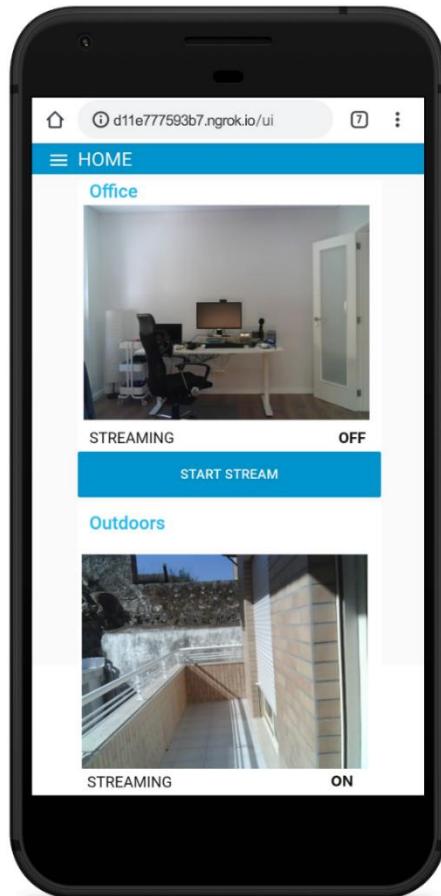
```
pi@raspberry:~ $ sudo reboot
```

Wait a few minutes for your Raspberry Pi to fully start all services. Then, if you go to ngrok Dashboard under the "Tunnels Online" menu, you should see the new URL that you must access to view your camera surveillance system from anywhere.

The screenshot shows the ngrok Tunnels Online interface. On the left, a sidebar menu includes 'Getting Started', 'Status' (with a red square highlight), 'Tunnels' (which is selected and highlighted in blue), 'Sessions', 'Endpoints', and 'Authentication'. The main area is titled 'Tunnels Online' and contains a table with two rows of tunnel information. The columns are 'Region', 'URL', and 'Client IP'. The first row shows 'US' as the region, 'https://f3efd5f5175d.ngrok.io' as the URL (also highlighted with a red box), and '2001:8a0:e3c1:a000:2600:8b00:3966:' as the Client IP. The second row shows 'US' as the region, 'http://f3efd5f5175d.ngrok.io' as the URL, and '2001:8a0:e3c1:a000:2600:8b00:3966:' as the Client IP.

| Region | URL | Client IP |
|--------|---|------------------------------------|
| US | https://f3efd5f5175d.ngrok.io | 2001:8a0:e3c1:a000:2600:8b00:3966: |
| US | http://f3efd5f5175d.ngrok.io | 2001:8a0:e3c1:a000:2600:8b00:3966: |

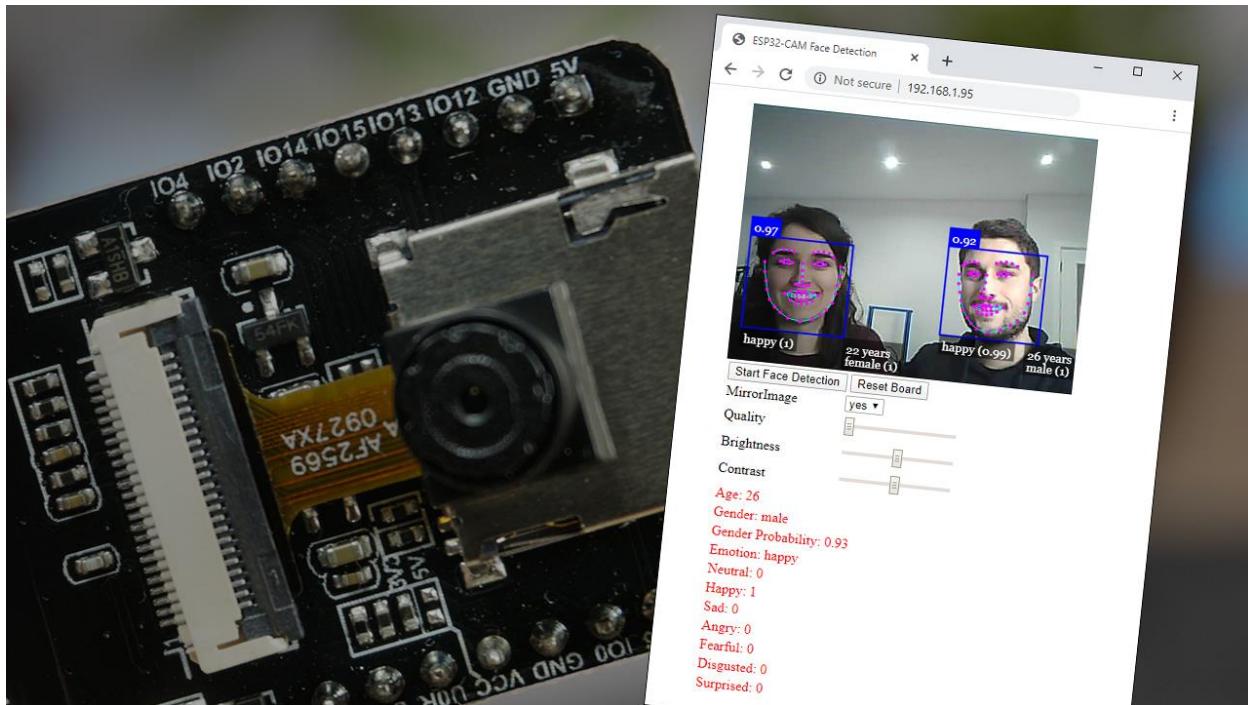
When you enter the preceding URL followed by `/ui` in your browser, you'll be asked to enter your username and password to open your Node-RED Dashboard with the cameras. Now, you have full control over your ESP32-CAM from anywhere!



MODULE 5

**Face Detection and
Face Recognition**

Unit 1: Face Detection Video Streaming (Age, Face Expression, etc.)



This Unit consists of a face detection video streaming project that estimates your age, gender and facial expression (happy, angry, sad, surprised, fearful and disgusted).

Compatibility: this project is compatible with any ESP32 camera board with the OV2640 camera. You just need to make sure you use the right pinout for the board you're using. To get good results with video streaming projects, we recommend using an external antenna or placing your ESP32 camera very close to your router.

Face recognition is done using a JavaScript library called [face-api.js](#). It runs on a web browser (we recommend using Google Chrome) – all the image processing is done using JavaScript on the browser. The ESP32-CAM hosts the video streaming web server. This example was inspired on a [project by fuchungyi](#).

Code

Copy the following code to your Arduino IDE.

CODE

https://github.com/RuiSantosdotme/ESP32-CAM-eBook/blob/master/Code/Module%205/Face%20Detection%20Video%20Streaming/Face_Detection_Video_Streaming.ino

```
#include <WiFi.h>
#include <WiFiClientSecure.h>
#include "esp_camera.h"
#include "soc/soc.h"
#include "soc/rtc_cntl_reg.h"
// Replace with your network credentials
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
String Feedback="";
String
Command="", cmd="", P1="", P2="", P3="", P4="", P5="", P6="", P7="", P8="", P9=""
";
byte
ReceiveState=0, cmdState=1, strState=1, questionstate=0, equalstate=0, semi
colonstate=0;
// 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

WiFiServer server(80);
void ExecuteCommand() {
    if (cmd!="getstill") {
        Serial.println("cmd= "+cmd+" ,P1= "+P1+" ,P2= "+P2+" ,P3= "+P3+
, P4= "+P4+" ,P5= "+P5+" ,P6= "+P6+" ,P7= "+P7+" ,P8= "+P8+" ,P9= "+P9);
        Serial.println("");
    }
}
```

```

if (cmd=="resetwifi") {
    WiFi.begin(P1.c_str(), P2.c_str());
    Serial.print("Connecting to ");
    Serial.println(P1);
    long int StartTime=millis();
    while (WiFi.status() != WL_CONNECTED)
    {
        delay(500);
        if ((StartTime+5000) < millis()) break;
    }
    Serial.println("");
    Serial.println("STAIP: "+WiFi.localIP().toString());
    Feedback="STAIP: "+WiFi.localIP().toString();
}
else if (cmd=="restart") {
    ESP.restart();
}
else if (cmd=="quality") {
    sensor_t * s = esp_camera_sensor_get();
    int val = P1.toInt();
    s->set_quality(s, val);
}
else if (cmd=="contrast") {
    sensor_t * s = esp_camera_sensor_get();
    int val = P1.toInt();
    s->set_contrast(s, val);
}
else if (cmd=="brightness") {
    sensor_t * s = esp_camera_sensor_get();
    int val = P1.toInt();
    s->set_brightness(s, val);
}
else {
    Feedback="Command is not defined.";
}
if (Feedback=="") {
    Feedback=Command;
}
}

void setup() {
    WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0);
    Serial.begin(115200);
    Serial.setDebugOutput(true);
    Serial.println();
    camera_config_t config;
    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;
//init with high specs to pre-allocate larger buffers
if(psramFound()){
    config.frame_size = FRAMESIZE_UXGA;
    config.jpeg_quality = 10; //0-63 lower number means higher quality
    config.fb_count = 2;
} else {
    config.frame_size = FRAMESIZE_SVGA;
    config.jpeg_quality = 12; //0-63 lower number means higher quality
    config.fb_count = 1;
}
// camera init
esp_err_t err = esp_camera_init(&config);
if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    delay(1000);
    ESP.restart();
}
//drop down frame size for higher initial frame rate
sensor_t * s = esp_camera_sensor_get();
s->set_framesize(s, FRAMESIZE_CIF); //UXGA|SXGA|XGA|SVGA|VGA|CIF|QVGA
//HQVGA|QQVGA
WiFi.mode(WIFI_AP_STA);
WiFi.begin(ssid, password);
delay(1000);
long int StartTime=millis();
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    if ((StartTime+10000) < millis())
        break;
}
if (WiFi.status() == WL_CONNECTED) {
    Serial.print("ESP IP Address: http://");
    Serial.println(WiFi.localIP());
}
server.begin();
}
static const char PROGMEM INDEX_HTML[] = R"rawliteral(
<!DOCTYPE html>
<head>
<title>ESP32-CAM Face Detection</title>
<meta charset="utf-8">
<meta name="viewport" content="width=device-width,initial-scale=1">
<script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>

```

```

<script src="https://cdn.jsdelivr.net/gh/justadudewhohacks/face-api.js@0.22.1/dist/face-api.min.js"></script>
</head><body>
<div id="container"></div>
<img id="ShowImage" src="" style="display:none">
<canvas id="canvas" style="display:none"></canvas>
<table>
<tr>
    <td colspan="2"><input type="button" id="getStill" value="Start Face Detection" style="display:none"></td>
    <td><input type="button" id="restart" value="Reset Board"></td>
</tr>
<tr>
    <td>MirrorImage</td>
    <td colspan="2">
        <select id="mirrorimage">
            <option value="1">yes</option>
            <option value="0">no</option>
        </select>
    </td>
</tr>
<tr>
    <td>Quality</td>
    <td colspan="2"><input type="range" id="quality" min="10" max="63" value="10"></td>
</tr>
<tr>
    <td>Brightness</td>
    <td colspan="2"><input type="range" id="brightness" min="-2" max="2" value="0"></td>
</tr>
<tr>
    <td>Contrast</td>
    <td colspan="2"><input type="range" id="contrast" min="-2" max="2" value="0"></td>
</tr>
</table>
<iframe id="ifr" style="display:none"></iframe>
<div id="message" style="color:red"><div>
</body>
</html>
<script>
    var getStill = document.getElementById('getStill');
    var ShowImage = document.getElementById('ShowImage');
    var canvas = document.getElementById("canvas");
    var context = canvas.getContext("2d");
    var mirrorimage = document.getElementById("mirrorimage");
    var message = document.getElementById('message');
    var ifr = document.getElementById('ifr');
    var myTimer;
    var restartCount=0;
    const modelPath = 'https://ruisantosdotme.github.io/face-api.js/weights/';
    let currentStream;
    let displaySize = { width:400, height: 296 }

```

```

let faceDetection;
Promise.all([
    faceapi.nets.tinyFaceDetector.load(modelPath),
    faceapi.nets.faceLandmark68TinyNet.load(modelPath),
    faceapi.nets.faceRecognitionNet.load(modelPath),
    faceapi.nets.faceExpressionNet.load(modelPath),
    faceapi.nets.ageGenderNet.load(modelPath)
])
getStill.onclick = function (event) {
    clearInterval(myTimer);
    myTimer = setInterval(function(){error_handle();},5000);
    ShowImage.src=location.origin+'/?getstill='+Math.random();
}
function error_handle() {
    restartCount++;
    clearInterval(myTimer);
    if (restartCount<=2) {
        message.innerHTML = "Get still error. <br>Restart ESP32-CAM"
            +restartCount+" times.";
        myTimer = setInterval(function(){getStill.click();},10000);
        ifr.src = document.location.origin+'?restart';
    }
    else
        message.innerHTML = "Get still error. <br>Please close the
page and check ESP32-CAM.";
}
getStill.style.display = "block";
ShowImage.onload = function (event) {
    clearInterval(myTimer);
    restartCount=0;
    canvas.setAttribute("width", ShowImage.width);
    canvas.setAttribute("height", ShowImage.height);
    canvas.style.display = "block";

    if (mirrorimage.value==1) {
        context.translate((canvas.width + ShowImage.width) / 2, 0);
        context.scale(-1, 1);
        context.drawImage(ShowImage, 0, 0, ShowImage.width, ShowImage.height);
        context.setTransform(1, 0, 0, 1, 0, 0);
    }
    else {
        context.drawImage(ShowImage,0,0,ShowImage.width,ShowImage.height);
    }
    DetectImage();
}
restart.onclick = function (event) {
    fetch(location.origin+'/?restart=stop');
}
quality.onclick = function (event) {
    fetch(document.location.origin+'/?quality='+this.value+';stop');
}
brightness.onclick = function (event) {
    fetch(document.location.origin+'/?brightness='+this.value+';stop
');
}

```

```

        }
        contrast.onclick = function (event) {
            fetch(document.location.origin+'/?contrast='+this.value+';stop')
        ;
    }
    async function DetectImage() {
        const detections = await faceapi.detectAllFaces(canvas, new
faceapi.TinyFaceDetectorOptions()).withFaceLandmarks(true).withFaceExp
ressions().withAgeAndGender()
        const resizedDetections = faceapi.resizeResults(detections, displaySize)
        faceapi.draw.drawDetections(canvas, resizedDetections)
        faceapi.draw.drawFaceLandmarks(canvas, resizedDetections)
        faceapi.draw.drawFaceExpressions(canvas, resizedDetections)
        resizedDetections.forEach(result => {
            const { detection, expressions, gender, genderProbability, age } =
result
            message.innerHTML = "";
            var maxEmotion="neutral";
            var maxProbability=expressions.neutral;
            if (expressions.happy>maxProbability) {
                maxProbability=expressions.happy;
                maxEmotion="happy";
            }
            if (expressions.sad>maxProbability) {
                maxProbability=expressions.sad;
                maxEmotion="sad";
            }
            if (expressions.angry>maxProbability) {
                maxProbability=expressions.angry;
                maxEmotion="angry";
            }
            if (expressions.fearful>maxProbability) {
                maxProbability=expressions.fearful;
                maxEmotion="fearful";
            }
            if (expressions.disgusted>maxProbability) {
                maxProbability=expressions.disgusted;
                maxEmotion="disgusted";
            }
            if (expressions.surprised>maxProbability) {
                maxProbability=expressions.surprised;
                maxEmotion="surprised";
            }
            document.getElementById("message").innerHTML =
"<table><tr><td>Age: " +Math.round(age)+"</td></tr><tr><td>Gender:
"+gender+"</td></tr><tr><td>Gender Probability:
"+Round(genderProbability)+"</td></tr><tr><td>Emotion:
"+maxEmotion+"</td></tr><tr><td>Neutral:
"+Round(expressions.neutral)+"</td></tr><tr><td>Happy:
"+Round(expressions.happy)+"</td></tr><tr><td>Sad:
"+Round(expressions.sad)+"</td></tr><tr><td>Angry:
"+Round(expressions.angry)+"</td></tr><tr><td>Fearful:
"+Round(expressions.fearful)+"</td></tr><tr><td>Disgusted:
"

```

```

"+Round(expressions.disgusted)+"</td></tr><tr><td>Surprised:<br>
"+Round(expressions.surprised)+"</td></tr></table>";
    var ageF = Math.round(age);
    var genderProbabilityF = Math.round(genderProbability);
    new faceapi.draw.DrawTextField(
        [
            `${ageF} years`,
            `${gender} (${genderProbabilityF})`
        ],
        result.detection.box.bottomRight
    ).draw(canvas)
})
try {
    document.createEvent("TouchEvent");
    setTimeout(function() {getStill.click();}, 500);
}
catch(e) {
    setTimeout(function() {getStill.click();}, 500);
}
}
function Round(n) { return Math.round(Number(n)*100)/100; }
</script>
)rawliteral";

void loop() {
    Feedback="";Command="";cmd="";P1="";P2="";P3="";P4="";P5="";P6="";P7
    ="";P8="";P9="";
    ReceiveState=0,cmdState=1,strState=1,questionstate=0,equalstate=0,se
    micolonstate=0;
    WiFiClient client = server.available();
    if (client) {
        String currentLine = "";
        while (client.connected()) {
            if (client.available()) {
                char c = client.read();
                getCommand(c);
                if (c == '\n') {
                    if (currentLine.length() == 0) {
                        if (cmd=="getstill") {
                            camera_fb_t * fb = NULL;
                            fb = esp_camera_fb_get();
                            if(!fb) {
                                Serial.println("Camera capture failed");
                                delay(1000);
                                ESP.restart();
                            }
                            client.println("HTTP/1.1 200 OK");
                            client.println("Access-Control-Allow-Origin:
                            *");
                            client.println("Access-Control-Allow-Headers: Origin, X-
                            Requested-With, Content-Type, Accept");
                        }
                    }
                }
            }
        }
    }
}

```

```

        client.println("Access-Control-Allow-Methods:
GET, POST, PUT, DELETE, OPTIONS");
        client.println("Content-Type: image/jpeg");
        client.println("Content-Disposition: form-data;
name=\"imageFile\"; filename=\"picture.jpg\"");
        client.println("Content-Length: " + String(fb->len));
        client.println("Connection: close");
        client.println();
        uint8_t *fbBuf = fb->buf;
        size_t fbLen = fb->len;
        for (size_t n=0;n<fbLen;n=n+1024) {
            if (n+1024<fbLen) {
                client.write(fbBuf, 1024);
                fbBuf += 1024;
            }
            else if (fbLen%1024>0) {
                size_t remainder = fbLen%1024;
                client.write(fbBuf, remainder);
            }
        }
        esp_camera_fb_return(fb);
    }
    else {
        client.println("HTTP/1.1 200 OK");
        client.println("Access-Control-Allow-Headers: Origin, X-
Requested-With, Content-Type, Accept");
        client.println("Access-Control-Allow-Methods: GET, POST, PUT,
DELETE, OPTIONS");
        client.println("Content-Type: text/html; charset=utf-8");
        client.println("Access-Control-Allow-Origin: *");
        client.println("Connection: close");
        client.println();
        String Data="";
        if (cmd!="")
            Data = Feedback;
        else {
            Data = String((const char *)INDEX_HTML);
        }
        int Index;
        for (Index = 0; Index < Data.length(); Index = Index+1000)
{
            client.print(Data.substring(Index, Index+1000));
}
        client.println();
    }

    Feedback="";
    break;
} else {
    currentLine = "";
}
}
else if (c != '\r') {
    currentLine += c;
}

```

```

    }
    if((currentLine.indexOf("/")!=-1)&&(currentLine.indexOf(" HTTP")!=-1)){
        if (Command.indexOf("stop")!=-1) {
            client.println();
            client.println();
            client.stop();
        }
        currentLine="";
        Feedback="";
        ExecuteCommand();
    }
}
delay(1);
client.stop();
}

void getCommand(char c){
    if (c=='?') ReceiveState=1;
    if ((c==' ') || (c=='\r') || (c=='\n')) ReceiveState=0;

    if (ReceiveState==1) {
        Command=Command+String(c);
        if (c=='=') cmdState=0;
        if (c==';') strState++;
        if((cmdState==1)&&((c!='?') || (questionstate==1)))
cmd=cmd+String(c);
        if((cmdState==0)&&(strState==1)&&((c!='=') || (equalstate==1)))
P1=P1+String(c);
        if ((cmdState==0)&&(strState==2)&&(c!=';')) P2=P2+String(c);
        if ((cmdState==0)&&(strState==3)&&(c!=';')) P3=P3+String(c);
        if ((cmdState==0)&&(strState==4)&&(c!=';')) P4=P4+String(c);
        if ((cmdState==0)&&(strState==5)&&(c!=';')) P5=P5+String(c);
        if ((cmdState==0)&&(strState==6)&&(c!=';')) P6=P6+String(c);
        if ((cmdState==0)&&(strState==7)&&(c!=';')) P7=P7+String(c);
        if ((cmdState==0)&&(strState==8)&&(c!=';')) P8=P8+String(c);
        if ((cmdState==0)&&(strState>=9)&&((c!=';') || (semicolonstate==1)))
P9=P9+String(c);
        if (c=='?') questionstate=1;
        if (c=='=') equalstate=1;
        if ((strState>=9)&&(c==';')) semicolonstate=1;
    }
}

```

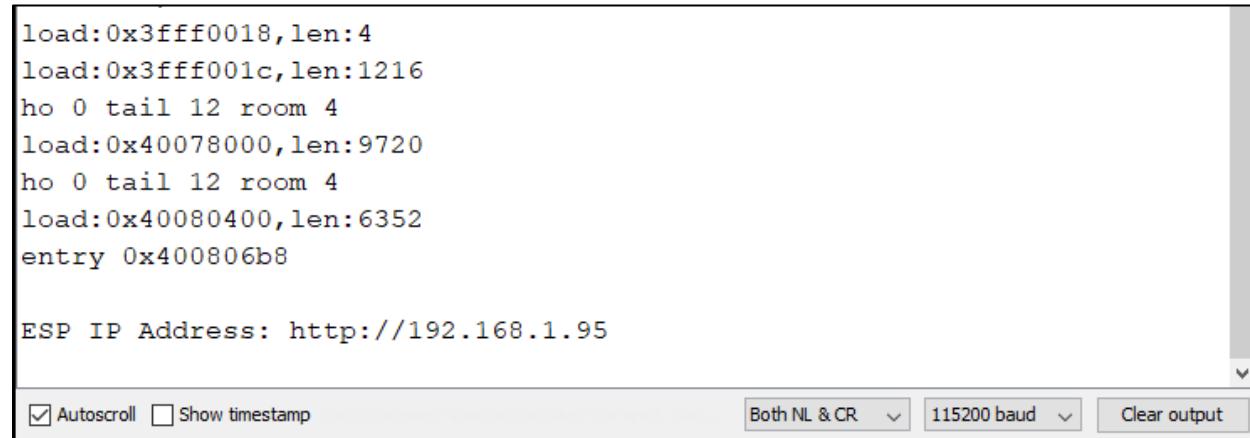
Network Credentials

Before uploading the code, make sure you insert your network credentials.

```
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";
```

Demonstration

After uploading the code, open the Serial Monitor to get the ESP32-CAM IP address.



The screenshot shows the Serial Monitor window of a terminal application. The window title is "Serial Monitor". The main pane displays the following text:

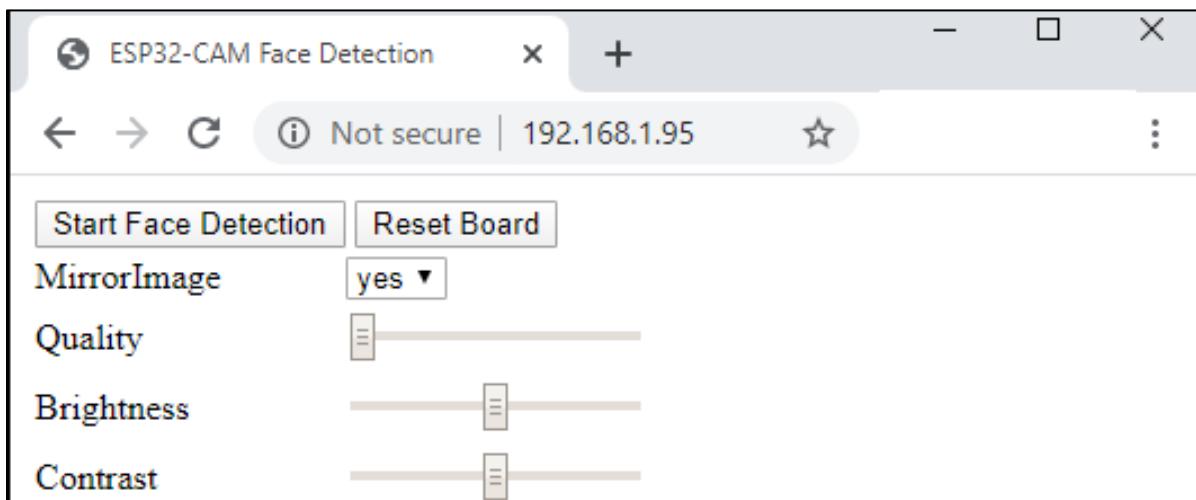
```
load:0x3fff0018,len:4
load:0x3fff001c,len:1216
ho 0 tail 12 room 4
load:0x40078000,len:9720
ho 0 tail 12 room 4
load:0x40080400,len:6352
entry 0x400806b8

ESP IP Address: http://192.168.1.95
```

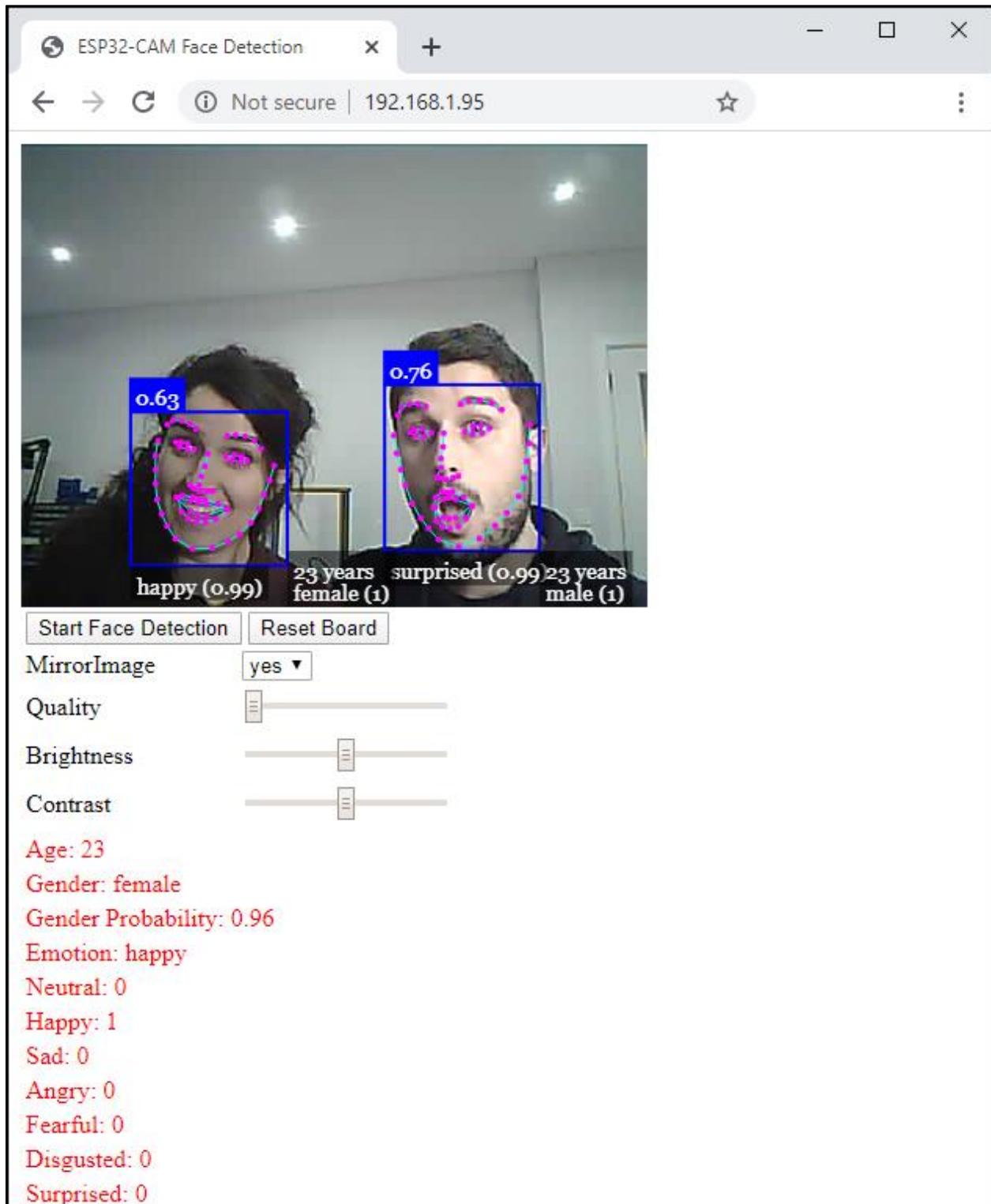
At the bottom of the window, there are several configuration options:

- Autoscroll
- Show timestamp
- Both NL & CR
- 115200 baud
- Clear output

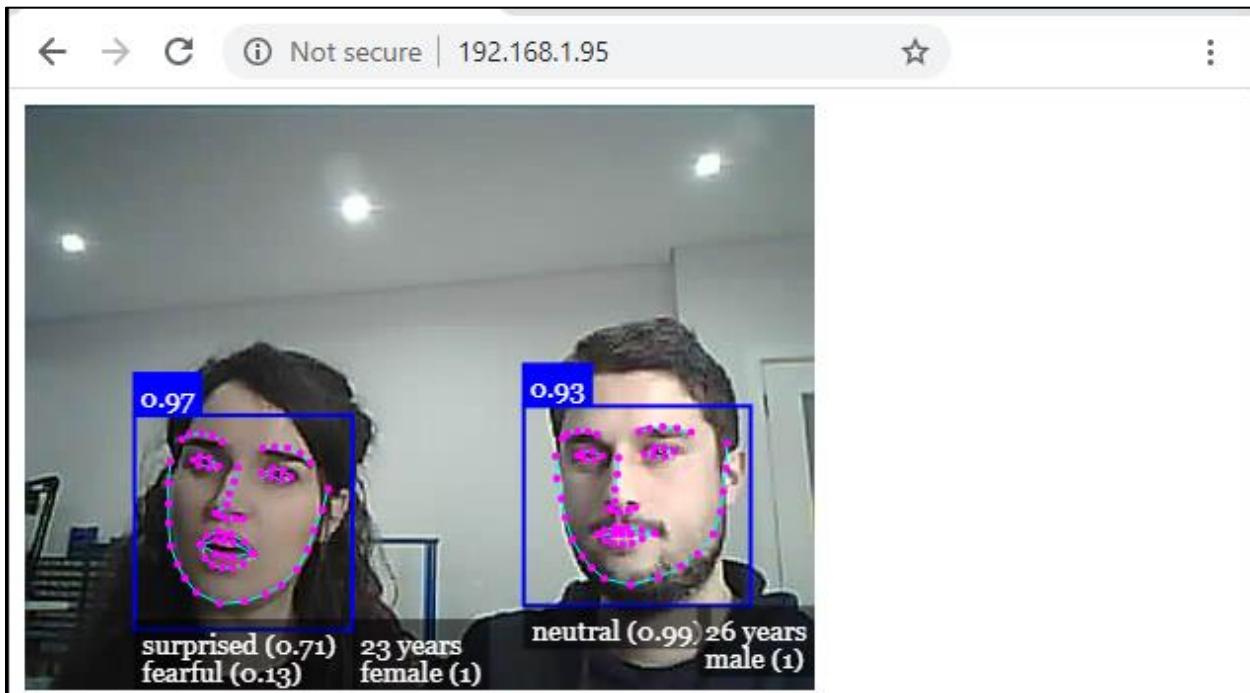
Open a browser and type the ESP32 IP address. The web server page should look as shown in the following screenshot. The web page shows a button to **Start Face Detection** and another button to reset the board in case it crashes. There's also the option to mirror the image or adjust quality, brightness and contrast.



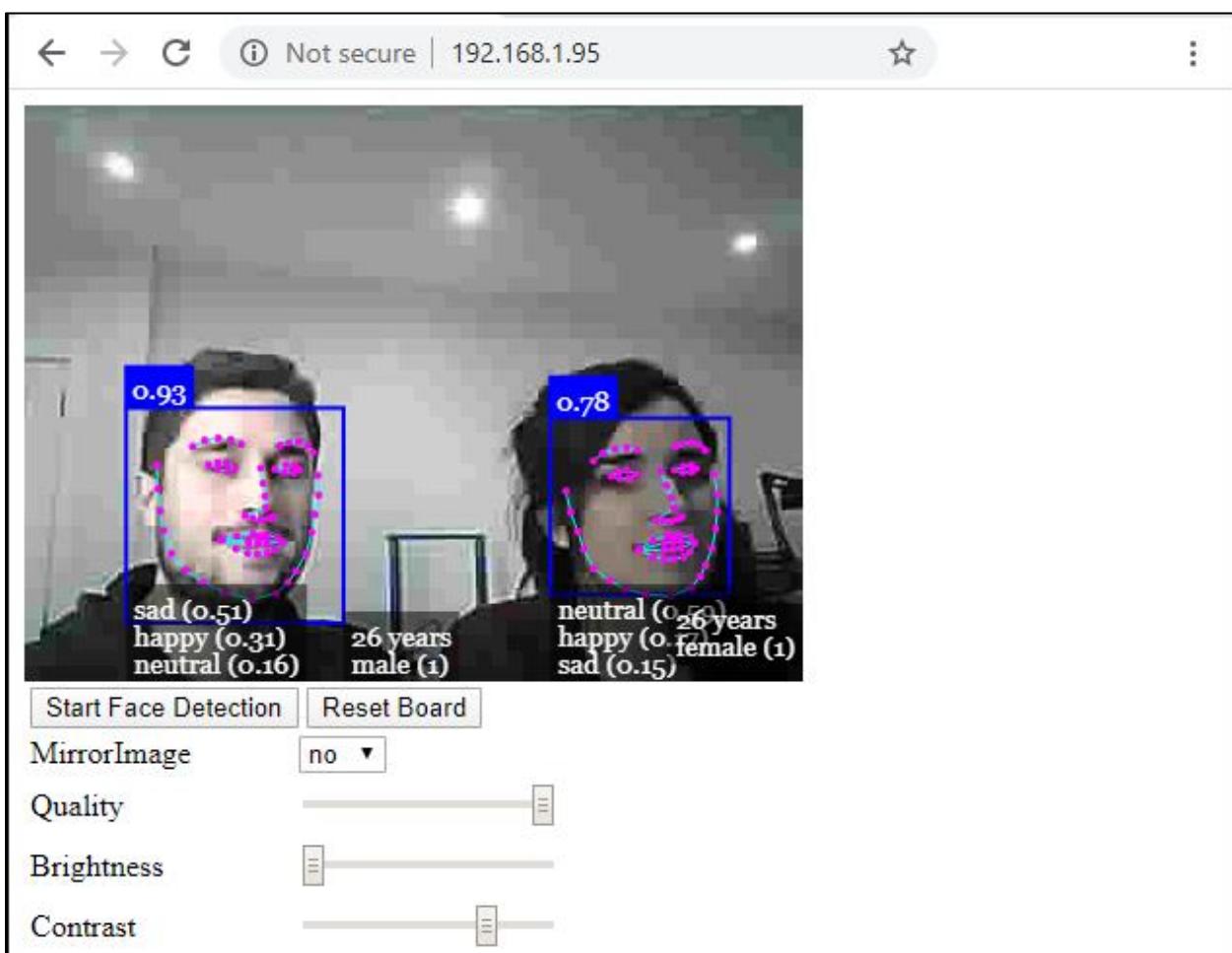
Click the **Start Face Detection** button to start streaming.



It can identify two faces at the same time. But only displays the information for one person at a time. Here's another example for a different face expression.



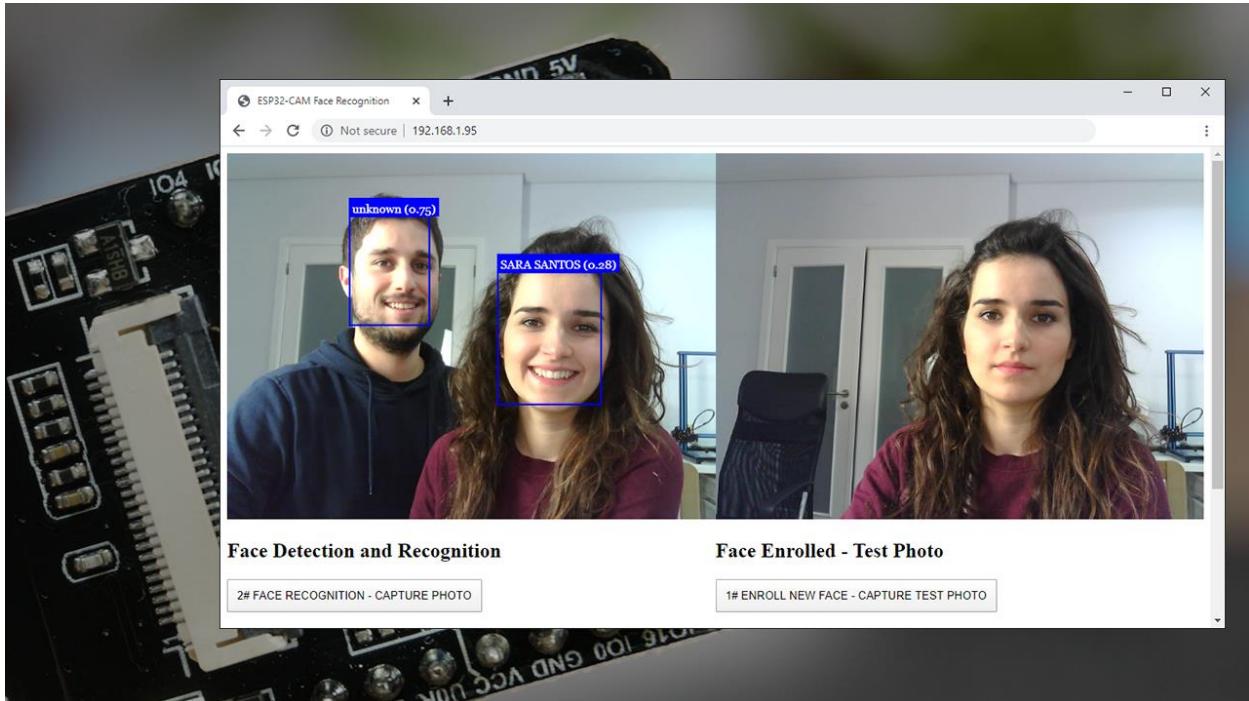
You can also change the image settings by moving the sliders.



Wrapping Up

We hope you had fun experimenting with this project. We were impressed how well the face detection algorithm works. It easily identifies your gender, facial expression, and the estimation for the age is pretty accurate too. We are both 26 years old at the time of testing this project, so we can say it performs quite well.

Unit 2: Face Recognition (Comparing Two Photos)



This is a face recognition project using photos. When the ESP32-CAM recognizes an enrolled face, it triggers an action – as an example, we print a message in the Serial Monitor, but you can control an output.

For face recognition, the ESP32-CAM hosts a web page that uses a JavaScript library called [face-api.js](#). This library makes it easy to compare two images for face recognition. It runs on a web browser (we recommend using Google Chrome) – all the image processing is done using JavaScript on the browser. The ESP32 is responsible for hosting the web page, taking photos and saving them on SPIFFS.

Compatibility: this project is compatible with any ESP32 camera board with the OV2640 camera. You just need to make sure you use the right pinout for the board you're using.

Limitations

- For a better experience with this project, we recommend using a laptop computer with Google Chrome. At the moment, this project is extremely slow on a smartphone (it works, but it is VERY slow).
- Sometimes your browser doesn't load the libraries properly. If you don't see a popup with your Face Recognition result after a few seconds, Refresh the web page again.
- The code for this project is able to identify just one single person. The name of that person can be changed on the code.
- Sometimes the algorithm is not able to identify your face if your head is tilted to the side.
- Regardless of these limitations, the project works well and it's quite fun to experiment with.

Code

Copy the following code to your Arduino IDE.

CODE

https://github.com/RuiSantosdotme/ESP32-CAM-eBook/blob/master/Code/Module_5/Face_Recognition_Comparing_two_Photos/Face_Recognition_Comparing_two_Photos.ino

```
#include "WiFi.h"
#include "esp_camera.h"
#include "esp_timer.h"
#include "img_converters.h"
#include "Arduino.h"
#include "soc/soc.h"          // Disable brownout problems
#include "soc/rtc_cntl_reg.h"  // Disable brownout problems
#include "driver/rtc_io.h"
#include <ESPAsyncWebServer.h>
#include <StringArray.h>
#include <SPIFFS.h>
#include <FS.h>
```

```

// Replace with your network credentials
const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

// Create AsyncWebServer object on port 80
AsyncWebServer server(80);

boolean takeNewPhoto = false;
boolean takeNewPhoto2 = false;

// Photo File Name to save in SPIFFS
String FILE_PHOTO = "/photo.jpg";
String FILE_PHOTO_2 = "/test-photo.jpg";

// OV2640 camera module pins (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

const char index_html[] PROGMEM = R"rawliteral(
<!DOCTYPE html>
<html>
  <head>
    <title>ESP32-CAM Face Recognition</title>
    <meta name="viewport" content="width=device-width,initial-scale=1">
    <meta http-equiv="Access-Control-Allow-Headers" content="Origin, X-Request-With, Content-Type, Accept">
    <meta http-equiv="Access-Control-Allow-Methods" content="GET,POST,PUT,DELETE,OPTIONS">
    <meta http-equiv="Access-Control-Allow-Origin" content="*">
    <title>ESP32-CAM Face Detection and Recognition</title>
    <script src="https://code.jquery.com/jquery-3.3.1.min.js"></script>
    <script src='https://cdn.jsdelivr.net/gh/justadudewhohacks/face-api.js@0.22.1/dist/face-api.min.js'></script>
    <style>
      #container { padding: 0; }
      canvas {
        position: absolute;
        top: 0;
        left: 0;

```

```

        z-index:999;
    }
    img {
        width: auto ;
        max-width: 100% ;
        height: auto ;
    }
    .flex-container { display: flex; }
    @media only screen and (max-width: 600px) {
        .flex-container {
            flex-direction: column;
        }
    }
    button {
        padding: 10px;
        margin-bottom: 30px;
    }
</style>
</head>
<body>
<div class="flex-container">
    <div>
        <div id="container"></div>
        </img>
        <h2>Face Detection and Recognition</h2>
        <button onclick="capturePhoto();">
            2# FACE RECOGNITION - CAPTURE PHOTO
        </button>
    </div>
    <div>
        </img>
        <h2>Face Enrolled - Test Photo</h2>
        <button onclick="captureTestPhoto()">
            1# ENROLL NEW FACE - CAPTURE TEST PHOTO
        </button>
    </div>
</div>
<div>
    <p><b>How to use this ESP32-CAM Face Recognition example:</b></p>
    <ol>
        <li>Press the "ENROLL NEW FACE - CAPTURE TEST PHOTO" button and refresh the page</li>
        <li>Press the "FACE RECOGNITION - CAPTURE PHOTO" button</li>
        <li>Refresh the page. The face recognition code runs automatically, so you have to wait 5 to 40 seconds for an alert with the face recognition result.</li>
    </ol>
    <p><b>IMPORTANT:</b> if you don't see a popup with your Face Recognition result, refresh the web page and wait a few seconds. I only recommend testing this web server on a laptop/desktop computer with Google Chrome web browser.</p>
    <p>You can only do face recognition for one subject - your TEST PHOTO should only have one face.</p>
</div>

```

```

</body>
<script>
    function capturePhoto() {
        var xhr = new XMLHttpRequest();
        xhr.open('GET', "/capture-photo", true);
        xhr.send();
    }
    function captureTestPhoto(){
        var xhr = new XMLHttpRequest();
        xhr.open('GET', "/capture-test-photo", true);
        xhr.send();
    }
    var ShowImage = document.getElementById('newImage');
    const modelPath      =      'https://ruisantosdotme.github.io/face-
api.js/weights/';
    let currentStream;
    let displaySize = { width:320, height: 400 }
    let canvas;
    let faceDetection;

    Promise.all([
        faceapi.nets.faceRecognitionNet.loadFromUri(modelPath),
        faceapi.nets.faceLandmark68Net.loadFromUri(modelPath),
        faceapi.nets.ssdMobilenetv1.loadFromUri(modelPath)
    ]).then(start());

    async function start() {
        const container = document.createElement('div')
        container.style.position = 'relative'
        document.body.append(container)
        const labeledFaceDescriptors = await loadLabeledImages()
        const faceMatcher = new faceapi.FaceMatcher(labeledFaceDescriptors,
0.6)
        if( document.getElementsByTagName("canvas").length == 0 ) {
            canvas = faceapi.createCanvasFromMedia(ShowImage)
            document.getElementById('container').append(canvas)
        }
        const displaySize = { width: ShowImage.width, height:
ShowImage.height }
        faceapi.matchDimensions(canvas, displaySize)
        const detections = await
faceapi.detectAllFaces(ShowImage).withFaceLandmarks().withFaceDescript
ors()
        const resizedDetections = faceapi.resizeResults(detections,
displaySize)
        const results = resizedDetections.map(d =>
faceMatcher.findBestMatch(d.descriptor))
        results.forEach((result, i) => {
            const box = resizedDetections[i].detection.box
            const drawBox = new faceapi.draw.DrawBox(box, { label:
result.toString() })
            drawBox.draw(canvas)
            if( result.toString().indexOf("unknown") == -1 ) {
                var xhr = new XMLHttpRequest();

```

```

        xhr.open('GET', "/trigger", true);
        xhr.send();
    }
}
alert("DONE!");
}

function loadLabeledImages() {
    const labels = ['SARA SANTOS']
    return Promise.all(
        labels.map(async label => {
            const descriptions = []
            const img = await
faceapi.fetchImage(window.location.href+"test-photo.jpg")
            const detections = await
faceapi.detectSingleFace(img).withFaceLandmarks().withFaceDescriptor()
            detections.push(detections.descriptor)
            return new faceapi.LabeledFaceDescriptors(label, descriptions)
        })
    )
}
</script>
</html>) rawliteral;

```



```

void setup() {
    // Serial port for debugging purposes
    Serial.begin(115200);

    // Connect to Wi-Fi
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.println("Connecting to WiFi...");
    }
    if (!SPIFFS.begin(true)) {
        Serial.println("An Error has occurred while mounting SPIFFS");
        ESP.restart();
    }
    else {
        delay(500);
        Serial.println("SPIFFS mounted successfully");
    }

    // Print ESP32 Local IP Address
    Serial.print("IP Address: http://");
    Serial.println(WiFi.localIP());

    // Turn-off the 'brownout detector'
    WRITE_PERI_REG RTC_CNTL_BROWN_OUT_REG, 0;

    // OV2640 camera module
    camera_config_t config;
    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;

if (psramFound()) {
    config.frame_size = FRAMESIZE_UXGA;
    config.jpeg_quality = 10;
    config.fb_count = 2;
} else {
    config.frame_size = FRAMESIZE_SVGA;
    config.jpeg_quality = 12;
    config.fb_count = 1;
}
// Camera init
esp_err_t err = esp_camera_init(&config);
if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    ESP.restart();
}

// Route for root / web page
server.on("/", HTTP_GET, [] (AsyncWebServerRequest * request) {
    request->send_P(200, "text/html", index_html);
});

server.on("/capture-photo", HTTP_GET, [] (AsyncWebServerRequest * request) {
    takeNewPhoto = true;
    request->send_P(200, "text/plain", "Taking Photo");
});

server.on("/trigger", HTTP_GET, [] (AsyncWebServerRequest * request) {
    Serial.println("Trigger action");
    request->send_P(200, "text/plain", "Trigger");
});

server.on("/capture-test-photo", HTTP_GET, [] (AsyncWebServerRequest * request) {
    takeNewPhoto2 = true;
    request->send_P(200, "text/plain", "Taking Photo");
});

```

```

    });

server.on("/photo.jpg", HTTP_GET, [] (AsyncWebServerRequest * request) {
    request->send(SPIFFS, FILE_PHOTO.c_str(), "image/jpg", false);
});

server.on("/test-photo.jpg", HTTP_GET, [] (AsyncWebServerRequest * request) {
    request->send(SPIFFS, FILE_PHOTO_2.c_str(), "image/jpg", false);
});

// Start server
server.begin();
}

void loop() {
    if (takeNewPhoto) {
        takeNewPhoto = capturePhotoSaveSpiffs(FILE_PHOTO.c_str());
    }
    if (takeNewPhoto2) {
        takeNewPhoto2 = capturePhotoSaveSpiffs(FILE_PHOTO_2.c_str());
    }
    delay(1);
}

// Check if photo capture was successful
bool checkPhoto( fs::FS &fs , const char* photoName) {
    File f_pic = fs.open( photoName );
    unsigned int pic_sz = f_pic.size();
    return ( pic_sz > 100 );
}

// Capture Photo and Save it to SPIFFS
bool capturePhotoSaveSpiffs( const char* photoName ) {
    camera_fb_t * fb = NULL; // pointer
    bool ok = 0;// Boolean indicating if picture was captured correctly

    do {
        // Take a photo with the camera
        Serial.println("Taking a photo...");

        fb = esp_camera_fb_get();
        if (!fb) {
            Serial.println("Camera capture failed");
            return true;
        }

        // Photo file name
        Serial.printf("Picture file name: %s\n", photoName);
        File file = SPIFFS.open(photoName, FILE_WRITE);

        // Insert the data in the photo file
        if (!file) {
            Serial.println("Failed to open file in writing mode");
        }
    }
}

```

```

    else {
        file.write(fb->buf, fb->len); // payload (image), payload length
        Serial.print("The picture has been saved in ");
        Serial.print(photoName);
        Serial.print(" - Size: ");
        Serial.print(file.size());
        Serial.println(" bytes");
    }
    // Close the file
    file.close();
    esp_camera_fb_return(fb);

    // check if file has been correctly saved in SPIFFS
    ok = checkPhoto(SPIFFS, photoName);
} while ( !ok );
return false;
}

```

Network Credentials

Before uploading the code, search for the following lines and add your network credentials.

```

const char* ssid = "REPLACE_WITH_YOUR_SSID";
const char* password = "REPLACE_WITH_YOUR_PASSWORD";

```

Add Your Name

Search for the following line and write your name on the `labels` variable.

```
const labels = [ 'SARA SANTOS' ]
```

Trigger an Action

When the browser identifies an enrolled face, it makes a request on the `/trigger` URL. When this happens, we print a message in the Serial Monitor. Instead of printing a message, you can control an output.

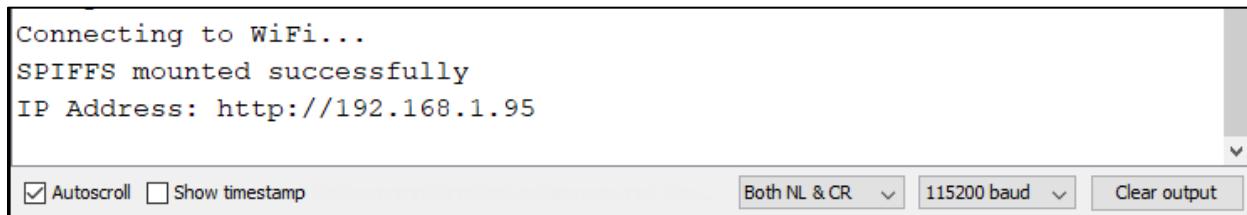
```

server.on("/trigger", HTTP_GET, [] (AsyncWebServerRequest * request) {
    Serial.println("Trigger action");
    request->send_P(200, "text/plain", "Trigger");
});

```

Demonstration

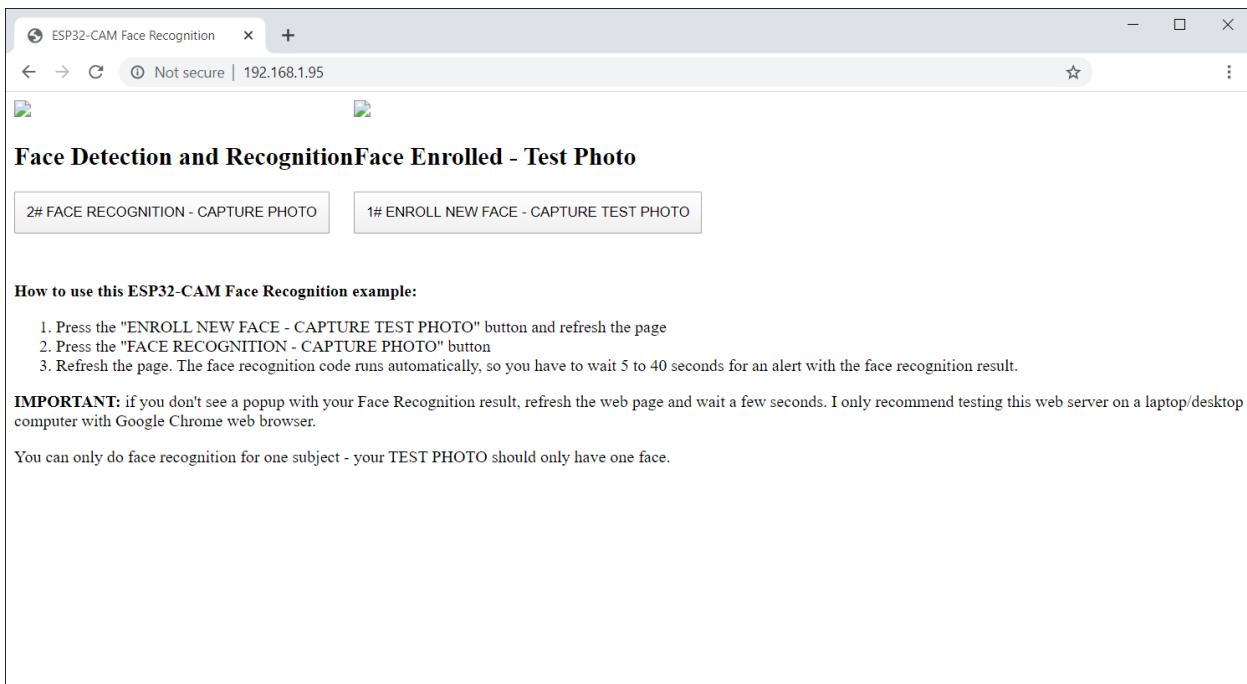
After uploading the code, open the Serial Monitor at a baud rate of 115200. Press the ESP32 on-board RST button to get its IP address.



```
Connecting to WiFi...
SPIFFS mounted successfully
IP Address: http://192.168.1.95
```

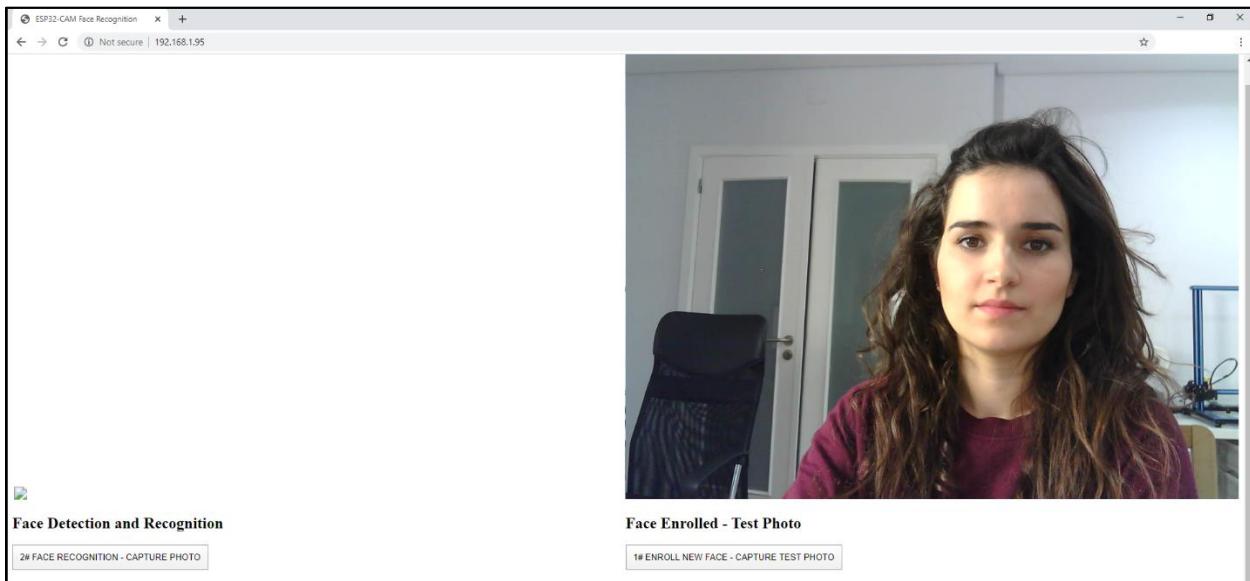
Autoscroll Show timestamp Both NL & CR 115200 baud Clear output

Open a browser and type the ESP32 IP address. The following web page should load.



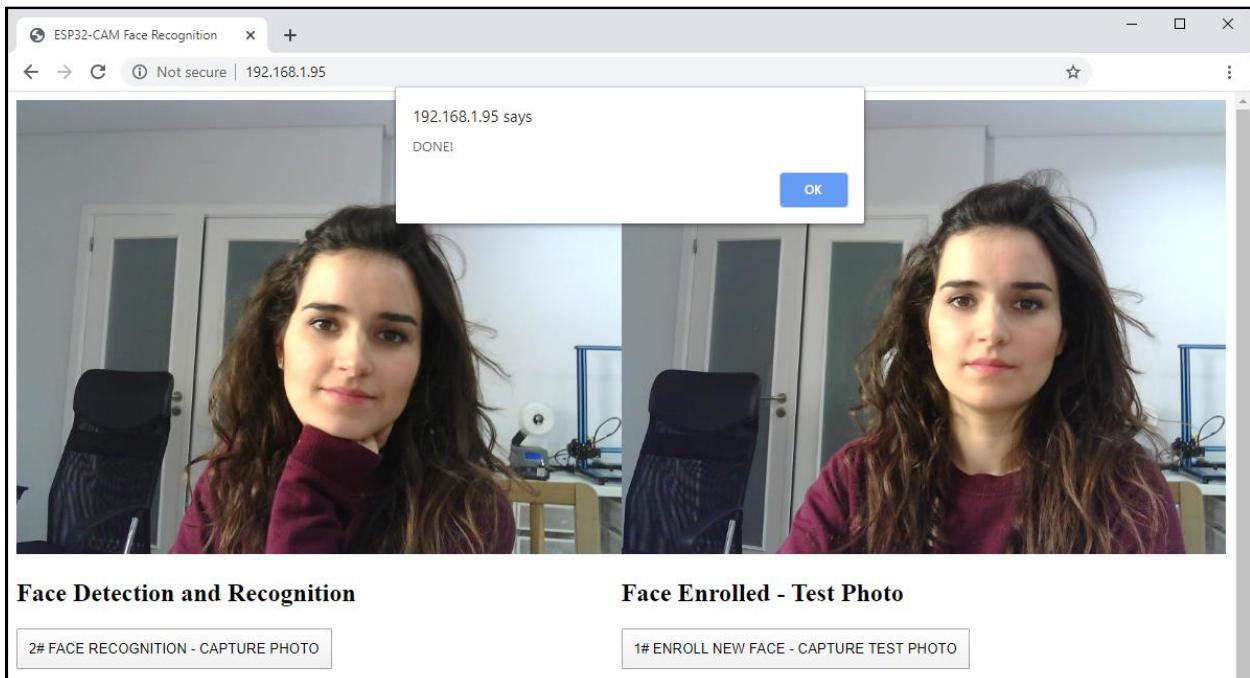
First, you need to enroll a new face. Click the button on the right "**ENROLL NEW FACE - CAPTURE TEST PHOTO**". This is the photo that will be saved on the ESP32 for your identification. The photos for face recognition will be compared with this one.

After clicking on that button, it takes a new photo (test photo). Refresh the web page, so that it displays the test photo.

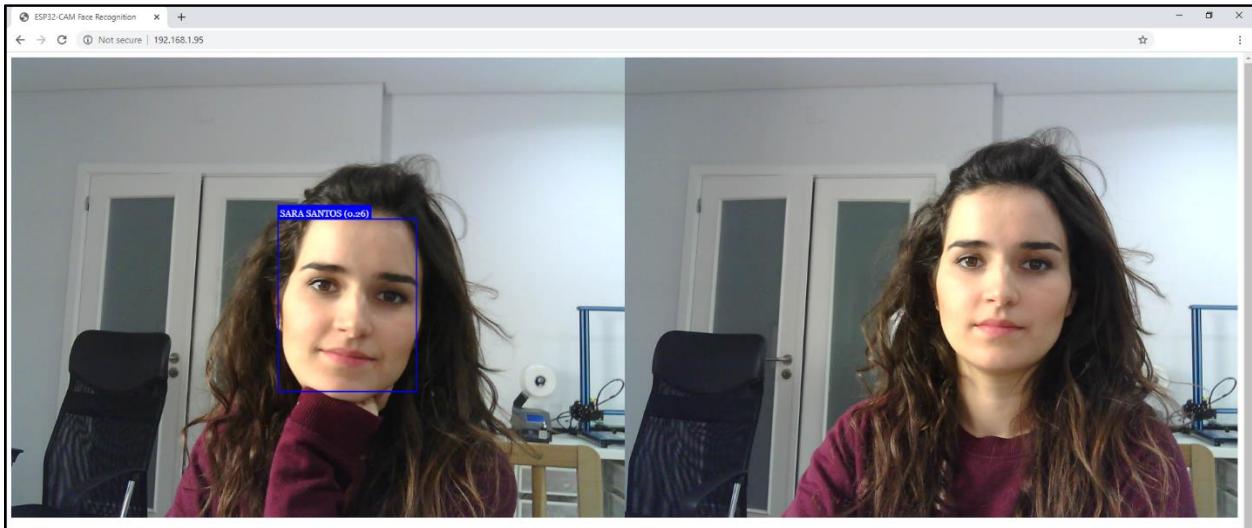


After that, you can take new photos to compare with that one. Click the "**FACE RECOGNITION – CAPTURE PHOTO**" button. This will take a new photo and compare it with the test photo.

Then, refresh the web page. The code for face recognition will run automatically. It may take several seconds to process the images. When it is ready, it will pop up a message saying "**DONE**" as shown below.



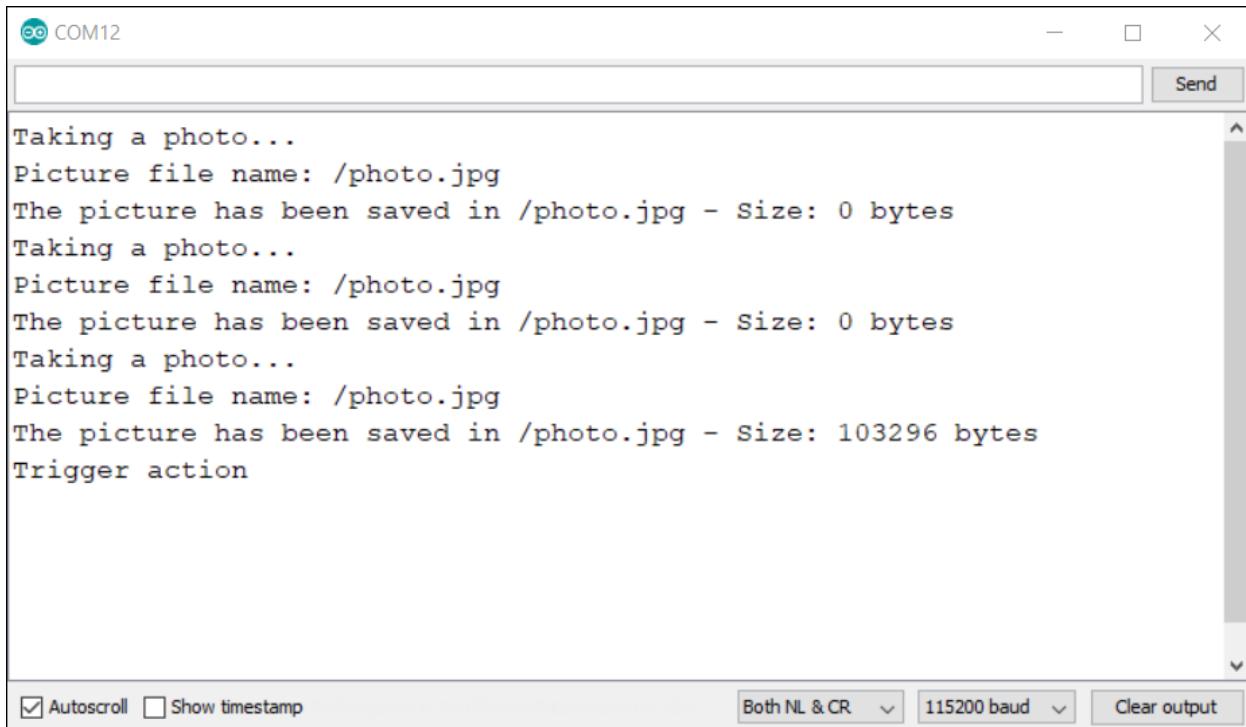
Click **OK** to see the result of the face recognition test.



It draws a blue rectangle around your face with a number between 0 and 1. A lower number means a higher probability that the person on both photos is the same. The result was 0.26, which means the algorithm is 74% sure it is Sara.



If you take a look at the Serial Monitor, it should display a message saying “Trigger action”. This happens when it identifies your face.

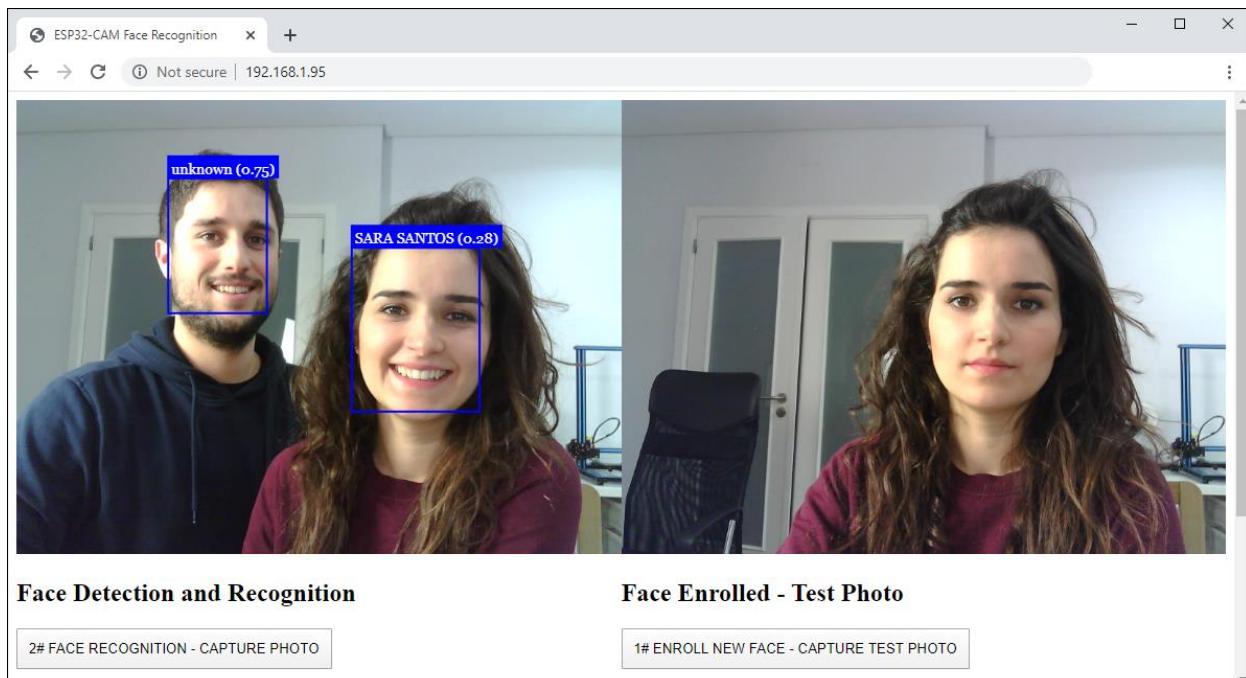


The screenshot shows a Windows-style serial monitor window titled "COM12". The text area contains the following log entries:

```
Taking a photo...
Picture file name: /photo.jpg
The picture has been saved in /photo.jpg - Size: 0 bytes
Taking a photo...
Picture file name: /photo.jpg
The picture has been saved in /photo.jpg - Size: 0 bytes
Taking a photo...
Picture file name: /photo.jpg
The picture has been saved in /photo.jpg - Size: 103296 bytes
Trigger action
```

At the bottom, there are checkboxes for "Autoscroll" and "Show timestamp", and dropdown menus for "Both NL & CR" (set to "Both NL & CR"), "115200 baud" (set to "115200 baud"), and "Clear output".

If it doesn't identify the person, it displays a blue rectangle with the “unknown” label and no “Trigger action” message on the Serial Monitor.



Wrapping Up

We hope you liked this project, despite its limitations. We had a lot of fun creating this face recognition example, we hope you had fun too.

The code for this project is quite complex and it is not easy to explain without having to explain a lot of other new concepts that go beyond the scope of this eBook.

However, modifying the code to trigger any other action instead of printing a message is quite easy. Instead of printing a message you can blink an LED, control a relay, etc. Also, don't forget to change the code to add your name on the blue label instead of saying "Sara Santos".

EXTRA UNITS

ESP32-CAM Access Point

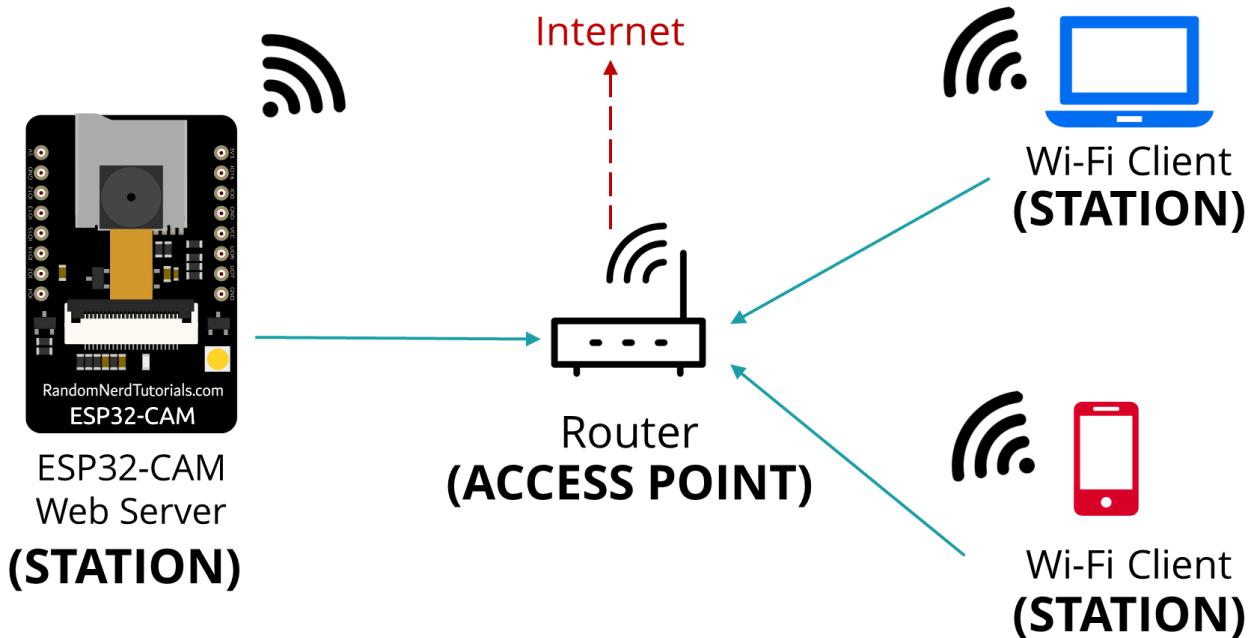


This extra Unit shows how to set your ESP32-CAM as an Access Point (AP) in your web server projects. This way, you don't need to be connected to a router to access the web server.

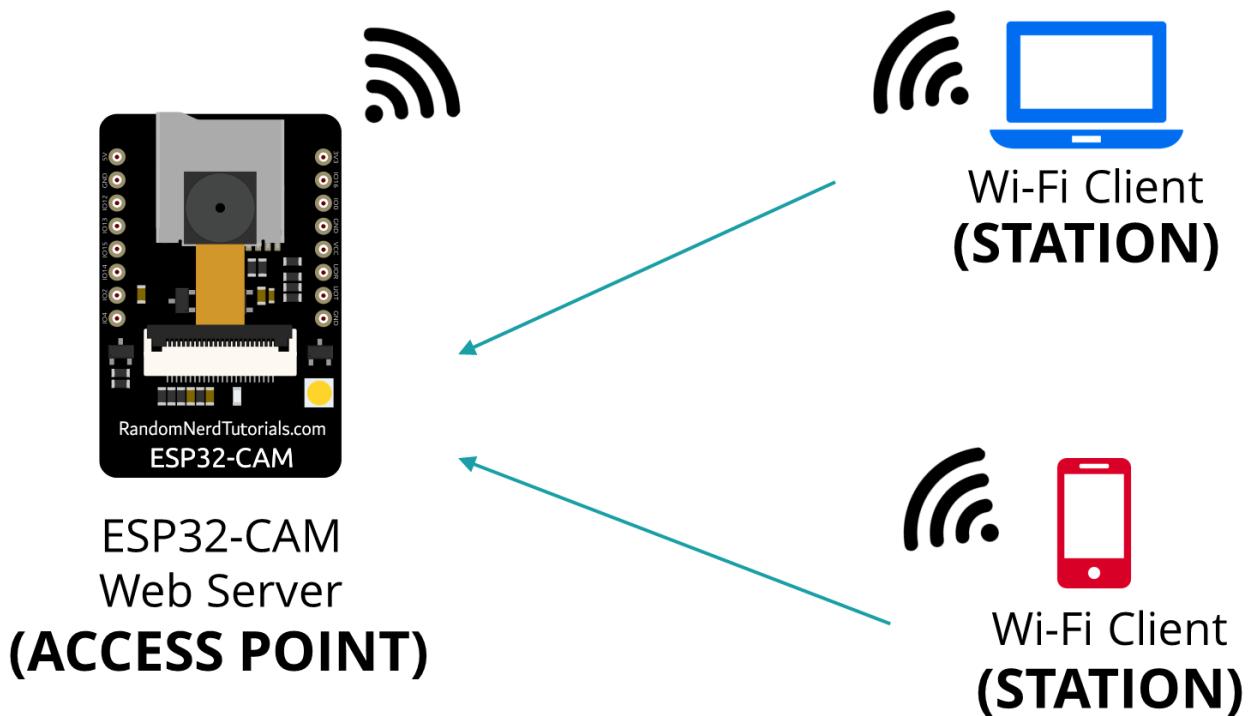
Access Point vs Station

In previous web server projects, we connect the ESP32-CAM to a wireless router. In this configuration, we can access the ESP32-CAM board through the local network.

In this scenario, the router acts as an access point and the ESP32-CAM board is set as a station. So, you need to be connected to your router (local network) to control and access the ESP32-CAM web server.



In some cases, this might not be the best configuration (when you don't have a router nearby). But if you set the ESP32-CAM boards as an access point (hotspot), you can be connected using any device with Wi-Fi capabilities without the need to connect to your router.



Basically, when you set the ESP32-CAM as an access point you create its own Wi-Fi network and nearby Wi-Fi devices (stations) can connect to it (like your smartphone or your computer).

Soft Access Point

Because the ESP32-CAM doesn't connect further to a wired network (like your router), it is called soft-AP (soft Access Point).

This means that if you try to load libraries or use firmware from the internet, it will not work (like including JavaScript libraries). It also doesn't work if you try to make HTTP requests to services on the internet (like sending an email with a photo, for example).

ESP32-CAM Video Web Server Access Point (AP)

In this tutorial, we'll show you how to set the ESP32 as an access point. As an example, we'll modify the CameraWebServer project that comes with the Arduino IDE. Then, you should be able to modify any of your projects to set the ESP32-CAM as an access point.

We'll use the code from Module 1, Unit 3. In your Arduino IDE, go to **File** ▶ **Examples** ▶ **ESP32** ▶ **Camera** ▶ **CameraWebServer**.

Then, modify the code to act as an access point as we'll explain.

Customize the SSID and Password

You need to define an SSID name and a password to access the ESP32-CAM access point. In this example we're setting the ESP32 SSID name to **ESP32-CAM Access**

Point. You can modify the name to whatever you want. The password is **123456789**, but you can and should also modify it.

```
const char* ssid = "ESP32-CAM Access Point";
const char* password = "123456789";
```

Setting the ESP32-CAM as an Access Point

In the `setup()`, **remove** the following lines (set the ESP32 as a station):

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

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

Serial.println("");
Serial.println("WiFi connected");
```

And **add** the following to set the ESP32 as an access point using the `softAP()` method:

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

There are also other optional parameters you can pass to the `softAP()` method. Here's all the parameters:

```
.softAP(const char* ssid, const char* password, int channel, int
ssid_hidden, int max_connection)
```

- **ssid** (defined earlier): maximum of 63 characters;
- **password** (defined earlier): minimum of 8 characters; set to NULL if you want the access point to be open
- **channel**: Wi-Fi channel number (1-13)
- **ssid_hidden**: (0 = broadcast SSID, 1 = hide SSID)
- **max_connection**: maximum simultaneous connected clients (1-4)

This is what you need to include in your web server sketches to set the ESP32-CAM as an access point.

You can download the complete code in the following link:

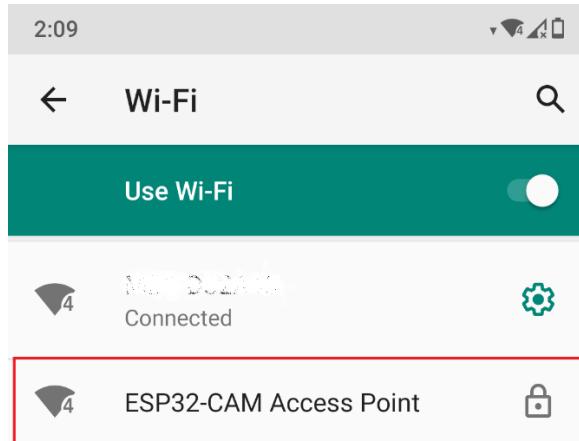
CODE

https://github.com/RuiSantosdotme/ESP32-CAM-eBook/blob/master/Code/Extra/CameraWebServer_Access_Point.zip?raw=true

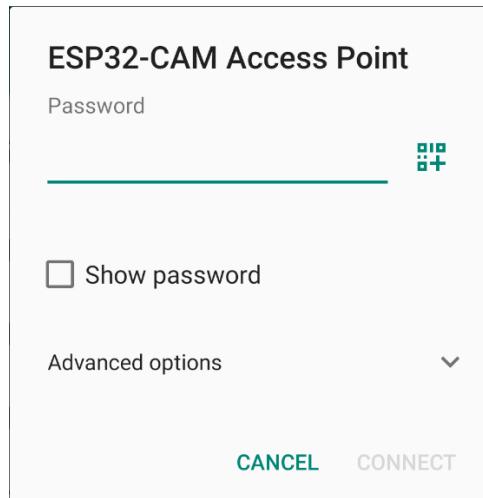
Connecting to the ESP32-CAM Access Point

After uploading the code, you can connect to the ESP32-CAM access point to access the web server. You don't need to connect to a router.

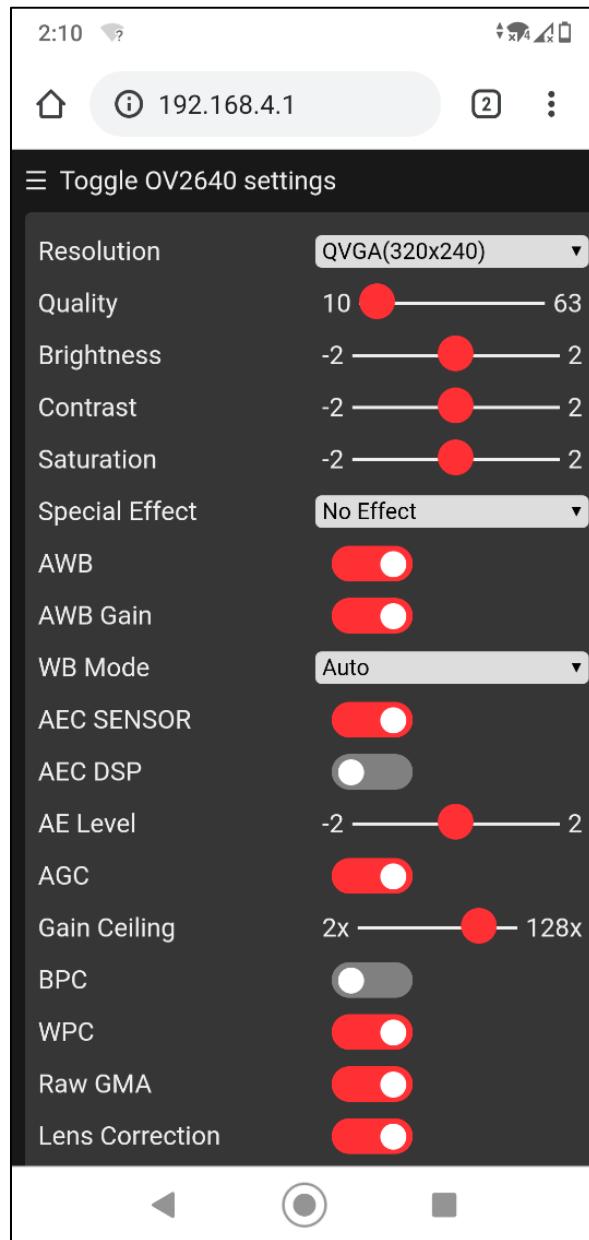
In your smartphone open your Wi-Fi settings and tap the **ESP32-CAM Access Point** network:



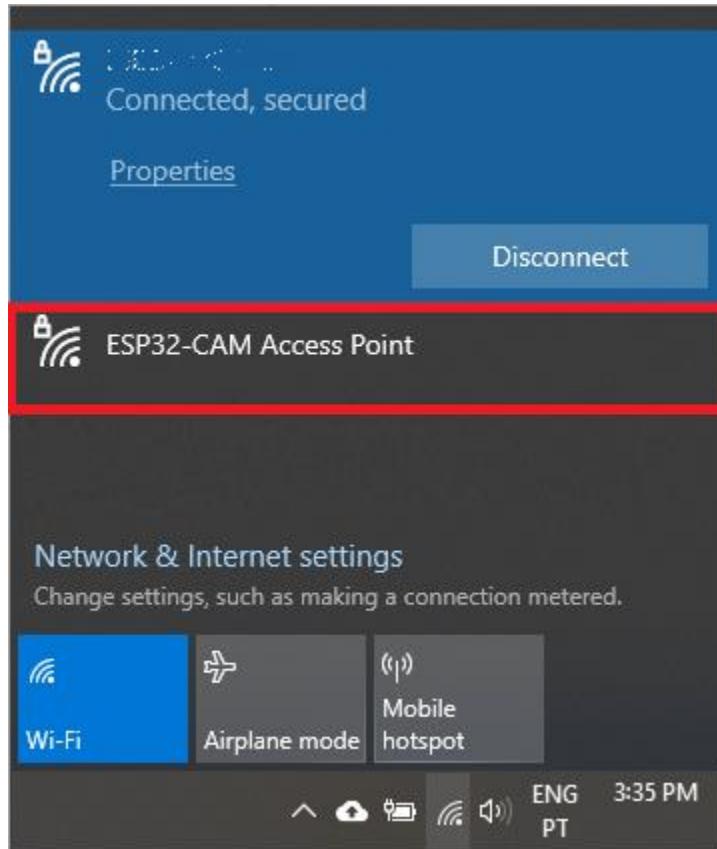
Type the password you've defined earlier in the code.



Open your web browser and type the IP address **192.168.4.1**. The video streaming web server page should load:



To connect to the access point on your computer, go to the Network and Internet Settings, select the “ESP32-Access-Point” and insert the password.



And it's done! Now, to access the ESP32-CAM web server page, you just need to type the IP address 192.168.4.1 in your browser.

Wrapping Up

With this extra Unit, you've learned how to set the ESP32-CAM as an access point on your web server sketches. When the ESP32 is set as an access point, devices with Wi-Fi capabilities like your smartphone can connect directly to the ESP without the need to connect to a router.

ESP32-CAM Static IP Address



This Unit shows how to set a static/fixed IP address for your ESP32-CAM board. If you're running a web server or Wi-Fi client with your ESP32-CAM and every time you restart your board, it has a new IP address, you can follow this unit to assign a static/fixed IP address.

Static/Fixed IP Address Sketch

To show you how to fix your ESP32-CAM IP address, we'll use the CameraWebServer Example from Module 1, Unit 3. By the end of our explanation you should be able to fix your IP address regardless of the web server or Wi-Fi project you're building.

In your Arduino IDE, go to **File** ▶ **Examples** ▶ **ESP32** ▶ **Camera** ▶ **CameraWebServer**.

Then, modify the code to fix the IP address as we'll explain.

Setting ESP8266 Static IP Address

Before the `setup()` and `loop()` functions, define the following variables with your own static IP address and corresponding gateway IP address.

By default, the next snippet assigns the IP address 192.168.1.184 that works in the gateway 192.168.1.1.

```
// Set your Static IP address
IPAddress local_IP(192, 168, 1, 184);
// Set your Gateway IP address
IPAddress gateway(192, 168, 1, 1);

IPAddress subnet(255, 255, 0, 0);
IPAddress primaryDNS(8, 8, 8, 8);      //optional
IPAddress secondaryDNS(8, 8, 4, 4); //optional
```

In the `setup()`, you need to call the `WiFi.config()` method to assign the configurations to your ESP32-CAM (before starting Wi-Fi).

```
if(!WiFi.config(local_IP, gateway, subnet, primaryDNS, secondaryDNS)) {
    Serial.println("STA Failed to configure");
}

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

The **primaryDNS** and **secondaryDNS** parameters are optional and you can remove them.

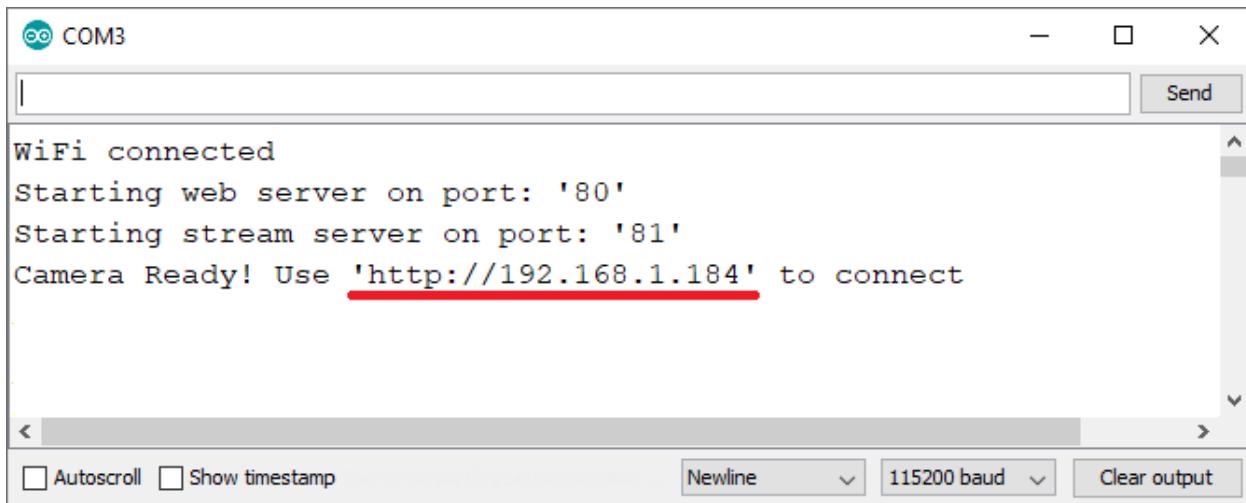
You can also download the code below that assigns the static IP address **192.168.1.184**. You just need to insert your network credentials and the code will work straight away.

CODE

```
https://github.com/RuiSantosdotme/ESP32-CAM-eBook/blob/master/Code/Extra/CameraWebServer\_Static\_IP.zip?raw=true
```

Testing

After uploading the code to your board, open the Arduino IDE Serial Monitor at the baud rate 115200. Restart your ESP32-CAM board and the IP address defined earlier should be assigned to your board.



```
WiFi connected
Starting web server on port: '80'
Starting stream server on port: '81'
Camera Ready! Use 'http://192.168.1.184' to connect
```

As you can see, it prints the IP address 192.168.1.184.

Now, you can access that IP address on your local network to see the Video Streaming Web Server.

Wrapping Up

In this unit, you've learned how to set up a static IP address for your ESP32-CAM. You can use what you've learned here in any of your ESP32-CAM sketches.

Congratulations for completing this course!

If you followed all the Modules presented in this course, now you should be able to build pretty cool projects with the ESP32-CAM.

You know how to:

- Take photos and save them on microSD card or SPIFFS;
- Build a web server to display and take photos;
- Send photos captured with the ESP32-CAM via email;
- Create a video streaming web server;
- Run an IP cam connected to Home Assistant or Node-RED;
- Build a remote controlled car robot with camera;
- Control a pan and tilt stand to move your camera remotely;
- Face detection with face-api.js;
- Face recognition comparing two photos with face-api.js;
- Build a surveillance system network that you can access from anywhere;
- And much more...

We hope you had fun following this course and you've learned something new! If you have something that you would like to share (your projects, suggestions, feedback) let us know in the [Private Forum](#) or [Facebook group](#).

Good luck with all your projects,

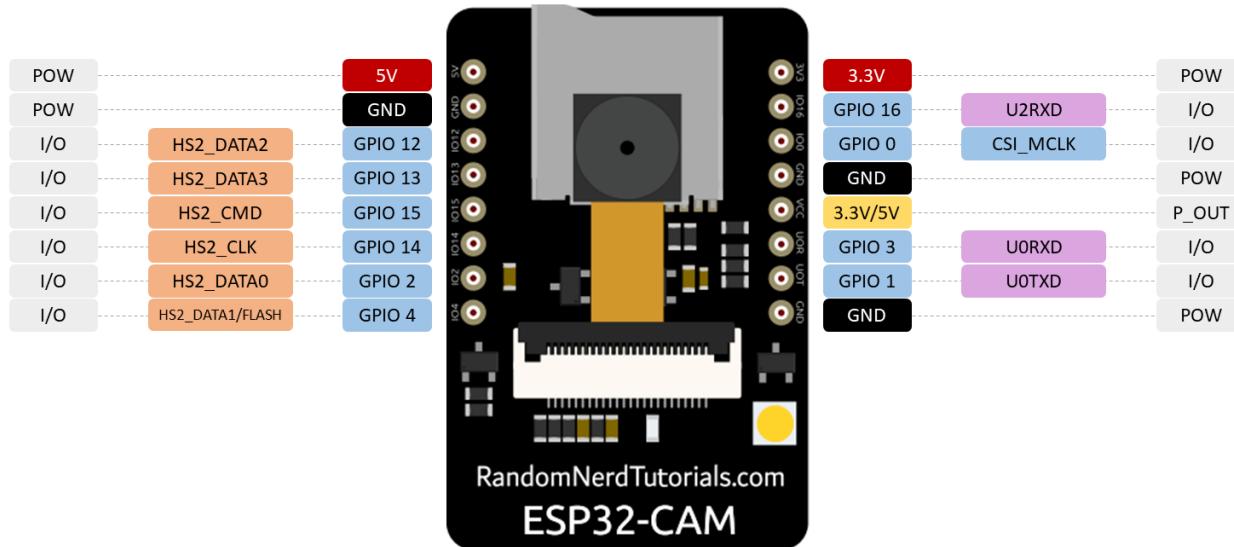
Rui Santos and Sara Santos

APPENDIX

Pinout for ESP32 Camera Boards

This appendix shows the pinout for the most common ESP32 camera development boards.

ESP32-CAM AI-Thinker

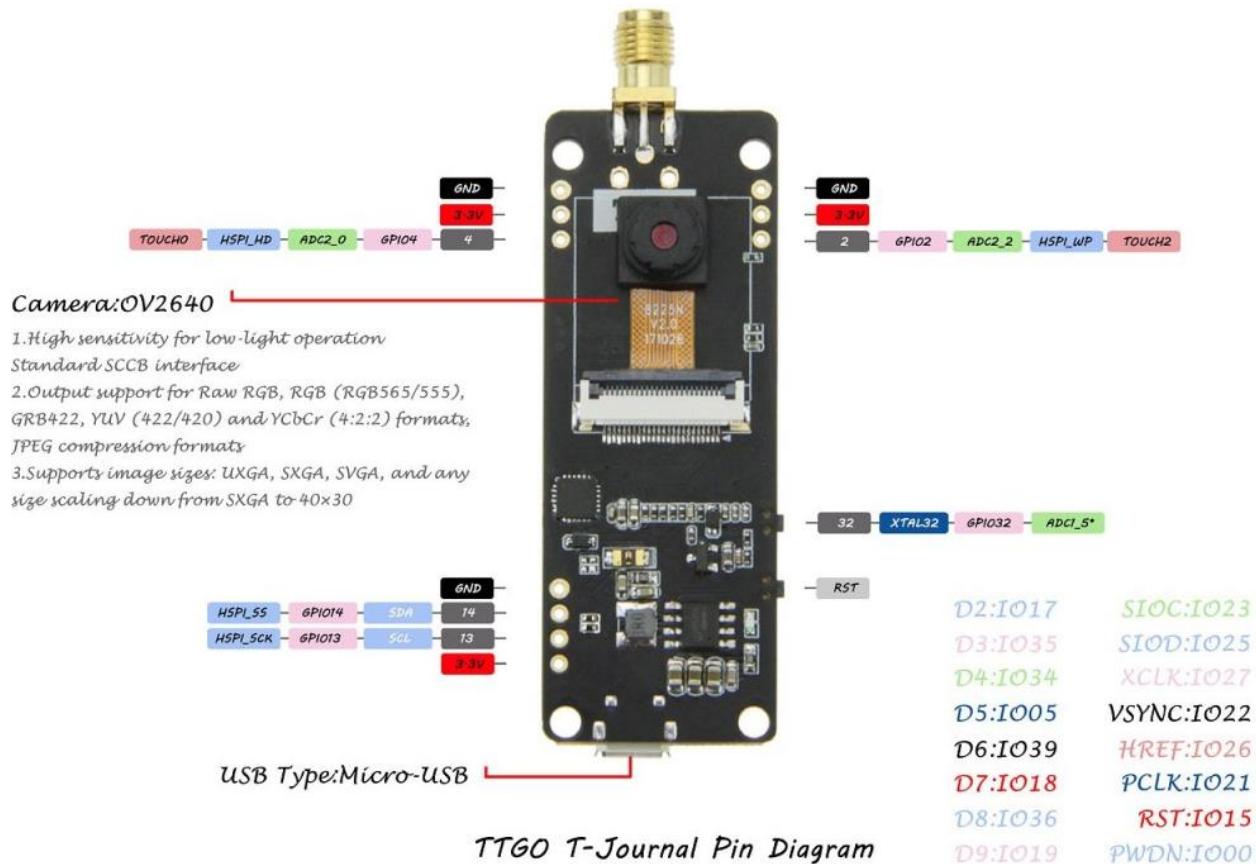


Pin assignment for the ESP32-CAM AI-Thinker.

```
#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
```

TTGO T-Journal



Pin assignment for the TTGO T-Journal.

```
#define PWDN_GPIO_NUM      32
#define RESET_GPIO_NUM      -1
#define XCLK_GPIO_NUM       27
#define SIOD_GPIO_NUM       25
#define SIOC_GPIO_NUM        23

#define Y9_GPIO_NUM          19
#define Y8_GPIO_NUM          36
#define Y7_GPIO_NUM          18
#define Y6_GPIO_NUM          39
#define Y5_GPIO_NUM          5
#define Y4_GPIO_NUM          34
#define Y3_GPIO_NUM          35
#define Y2_GPIO_NUM          17
#define VSYNC_GPIO_NUM       22
#define HREF_GPIO_NUM        26
#define PCLK_GPIO_NUM        21
```

M5-Camera Model A

There are two similar models: M5-Camera Model A and M5-Camera Model B. The model A looks as shown in the following figure.



Pin assignment for the M5-Camera Model A.

```
#define PWDN_GPIO_NUM      -1
#define RESET_GPIO_NUM       15
#define XCLK_GPIO_NUM        27
#define SIOD_GPIO_NUM        25
#define SIOC_GPIO_NUM        23

#define Y9_GPIO_NUM           19
#define Y8_GPIO_NUM           36
#define Y7_GPIO_NUM           18
#define Y6_GPIO_NUM           39
#define Y5_GPIO_NUM            5
#define Y4_GPIO_NUM           34
#define Y3_GPIO_NUM           35
#define Y2_GPIO_NUM           32
#define VSYNC_GPIO_NUM         22
#define HREF_GPIO_NUM          26
#define PCLK_GPIO_NUM          21
```

M5-Camera Model B

The M5-Camera Model B looks as follows:



Pin assignment for the M5-Camera Model B.

```
#define PWDN_GPIO_NUM      -1
#define RESET_GPIO_NUM     15
#define XCLK_GPIO_NUM      27
#define SIOD_GPIO_NUM      22
#define SIOC_GPIO_NUM      23

#define Y9_GPIO_NUM         19
#define Y8_GPIO_NUM         36
#define Y7_GPIO_NUM         18
#define Y6_GPIO_NUM         39
#define Y5_GPIO_NUM          5
#define Y4_GPIO_NUM         34
#define Y3_GPIO_NUM         35
#define Y2_GPIO_NUM         32
#define VSYNC_GPIO_NUM      25
#define HREF_GPIO_NUM       26
#define PCLK_GPIO_NUM       21
```

M5-Stack Without PSRAM



Pin assignment for the M5-stack camera without PSRAM.

```
#define PWDN_GPIO_NUM      -1
#define RESET_GPIO_NUM     15
#define XCLK_GPIO_NUM      27
#define SIOD_GPIO_NUM      25
#define SIOC_GPIO_NUM      23

#define Y9_GPIO_NUM         19
#define Y8_GPIO_NUM         36
#define Y7_GPIO_NUM         18
#define Y6_GPIO_NUM         39
#define Y5_GPIO_NUM          5
#define Y4_GPIO_NUM         34
#define Y3_GPIO_NUM         35
#define Y2_GPIO_NUM          7
#define VSYNC_GPIO_NUM       22
#define HREF_GPIO_NUM        26
#define PCLK_GPIO_NUM        21
```

ESP-EYE

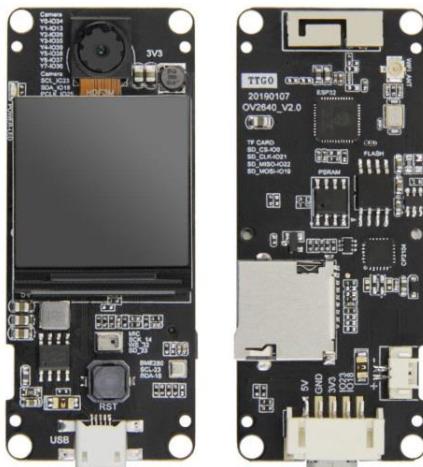


Pin assignment for the ESP-EYE camera.

```
#define PWDN_GPIO_NUM      -1
#define RESET_GPIO_NUM      -1
#define XCLK_GPIO_NUM        4
#define SIOD_GPIO_NUM        18
#define SIOC_GPIO_NUM        23

#define Y9_GPIO_NUM          36
#define Y8_GPIO_NUM          37
#define Y7_GPIO_NUM          38
#define Y6_GPIO_NUM          39
#define Y5_GPIO_NUM          35
#define Y4_GPIO_NUM          14
#define Y3_GPIO_NUM          13
#define Y2_GPIO_NUM          34
#define VSYNC_GPIO_NUM        5
#define HREF_GPIO_NUM         27
#define PCLK_GPIO_NUM         25
```

T-Camera Plus



Pin assignment for the TTGO T-Camera Plus.

```
#define PWDN_GPIO_NUM      -1
#define RESET_GPIO_NUM      -1
#define XCLK_GPIO_NUM        4
#define SIOD_GPIO_NUM        18
#define SIOC_GPIO_NUM        23

#define Y9_GPIO_NUM           36
#define Y8_GPIO_NUM           37
#define Y7_GPIO_NUM           38
#define Y6_GPIO_NUM           39
#define Y5_GPIO_NUM           35
#define Y4_GPIO_NUM           26
#define Y3_GPIO_NUM           13
#define Y2_GPIO_NUM           34
#define VSYNC_GPIO_NUM         5
#define HREF_GPIO_NUM          27
#define PCLK_GPIO_NUM          25
```

TTGO T-Camera with PIR Sensor



Pin definition for the T-Camera with PIR sensor (we haven't experimented with this board, so we're not sure if this is the right pinout).

```
#define PWDN_GPIO_NUM      -1
#define RESET_GPIO_NUM     -1
#define XCLK_GPIO_NUM       4
#define SIOD_GPIO_NUM      18
#define SIOC_GPIO_NUM      23

#define Y9_GPIO_NUM         36
#define Y8_GPIO_NUM         15
#define Y7_GPIO_NUM         12
#define Y6_GPIO_NUM         39
#define Y5_GPIO_NUM         35
#define Y4_GPIO_NUM         14
#define Y3_GPIO_NUM         13
#define Y2_GPIO_NUM         34
#define VSYNC_GPIO_NUM      5
#define HREF_GPIO_NUM       27
#define PCLK_GPIO_NUM       25
```

List of Parts Required

To build the projects in this eBook you need the following parts:

Parts Required

- [ESP32-CAM AI-Thinker with external antenna](#) (recommend) - Read [Best ESP32 Camera Development Boards](#)
- [FTDI programmer](#)
- [OV2640 camera probes](#) (optional)
- [Jumper wires](#)
- [Breadboard](#)
- [Prototyping circuit board](#) (optional)
- [Header socket connector](#) (optional)
- [Pushbutton](#)
- [10k Ohm resistor](#)
- [PIR motion sensor](#)
- [Fake dummy camera](#)
- [BME280 sensor](#)
- [2WD smart robot chassis kit](#)
- [L298N motor driver](#)
- [Pan and tilt camera stand](#)
- [Raspberry Pi Board](#) – read [Best Raspberry Pi Starter Kits](#)
- [MicroSD Card – 16GB Class10](#)
- [Raspberry Pi Power Supply \(5V 2.5A\)](#)

Useful Tools (optional)

- [Soldering Iron](#)
- [Multimeter](#)
- [Bench power supply](#)
- [Hot glue gun](#)
- [Wire stripper/cutter](#)

Other RNT Courses/eBooks

[Random Nerd Tutorials](#) is an online resource with electronics projects, tutorials and reviews. Currently, Random Nerd Tutorials has more than [200 free blog posts](#) with complete tutorials using open-source hardware that anyone can read, remix and apply to their own projects.

To keep free tutorials coming, there's also paid content or as we like to call "premium content". To support Random Nerd Tutorials, you can [download premium content here](#). If you enjoy this eBook, make sure you [check all our courses and resources](#).

Here's a list with all the courses/eBooks available to download at Random Nerd Tutorials:

- [Learn ESP32 with Arduino IDE](#) (Bestseller)
- [Home Automation Using ESP8266](#)
- [MicroPython Programming with ESP32/ESP8266](#)
- [20 Easy Raspberry Pi Projects](#) (available on Amazon)
- [Build a Home Automation System for \\$100](#)
- [Arduino Step-by-step Projects Course](#)
- [Android Apps for Arduino with MIT App Inventor 2](#)
- [Electronics For Beginners eBook](#)