EE3080 DIP Guide

UAVIONICS

version 1.1
(12 January 2021)

Notes:

- This guide can get updated along the way. Please keep an eye on newer versions.
- Codes for this project can be found here:
  https://github.com/NelsenEW/EEE-DIP-UAVONICS

Table of Content

Overview

DIP Overview

UAVONICS DIP is administered in 2 phases.

The first phase emphasizes on cultivating direct individualized hands-on experience specific to electronics and coding. To achieve this, students will engage in building a flying camera, driven by ESP32 Microcontrollers. The camera will have a small form factor (smaller than 10cm x 10cm) maneuvered by user from a smart phone.

The second phase emphasizes on collaborative team work to achieve a common objective of crafting novel solution to a real-world problem scenario to be unveiled during the project. The whole project brings forth a holistic experience of hands-on practicality; harnessing creativity to innovate and improvise in problem-solving.

In short, the objective of the first phase is building a camera-equipped quadcopter, while the aim of the second phase is to utilize the quadcopter for real-world applications.

Overview on the Hardware

We will use various hardware for this project. The main ones are:

1. Quadcopter drone
2. ESP32 development module with camera
3. IMU sensor module
4. Other sensor modules (if needed)

i. Quadcopter

The quadcopter used for this project is small (also known as MAV; Micro Aerial Vehicle) for safety and legal reasons. Some specifications of the quadcopter:

- **Model:** Eachine E010 Nano Drone

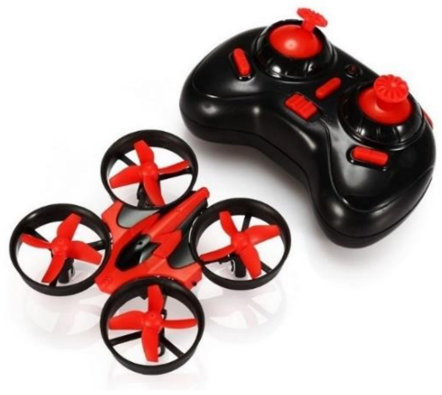


Figure 1. Eachine E010 Nano Drone.

- **Specifications:**
  - Motor: 4x brushed motors
  - Flight time: 5-7 minutes (onboard LED blinks if battery goes low)
  - Charge time: approx. 40 minutes
  - Remote control range: 30 m
  - Dimension/weight: 85 x 85 x 30 mm
  - Net weight: 25 gr

Due to the small size of the drone, it has **short flight time** and **small payload** (we cannot add much weight to the drone), which are the limiting factors for this project.

ii. ESP32 Development Board with Camera

The ESP32 is a widely popular family of development boards. It has a powerful *dual-core* microcontroller and Wi-Fi and Bluetooth capability built into the board. It is mostly used for IoT (Internet of Things) applications.

A microcontroller is a device with multiple input/output pins that can be programmed to perform lots of different tasks. You can think of it as a single-chip, tiny computer (which is obviously not as powerful as a "real" computer) that can interface with sensors and actuators (motors, etc.). Arduino boards, if you are familiar with them, are also development boards featuring microcontrollers on them.

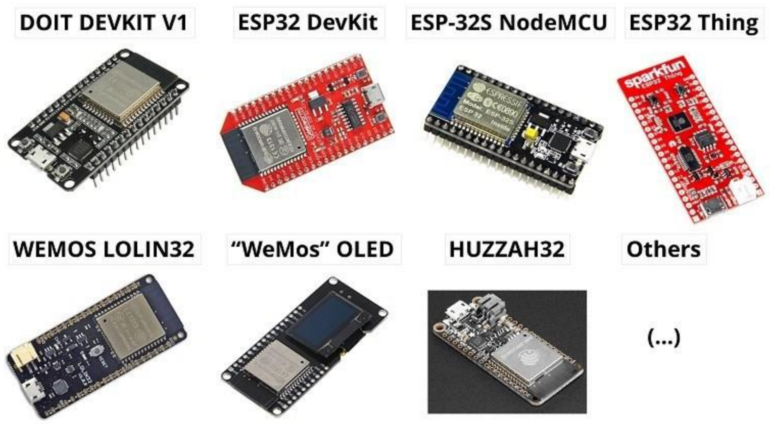


Figure 2. Various members of the ESP32 family.
From https://randomnerdtutorials.com/getting-started-with-esp32/.

For this project, we also need a camera. The ESP32-CAM is a member of the ESP32 development board family with built-in camera. We can either use this development board or create our own design based on this board's design.

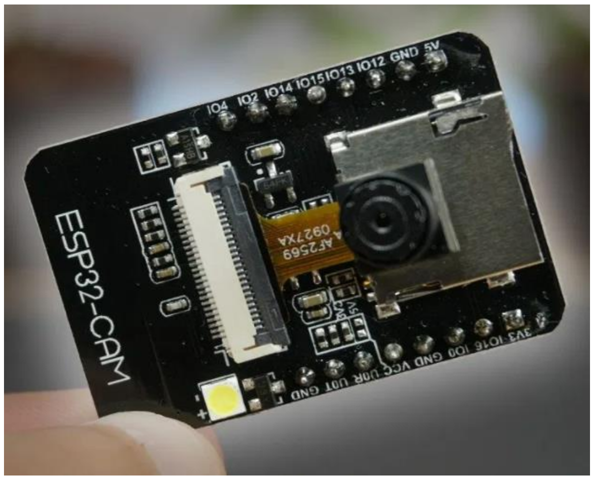

IO4 IO2 IO14 IO15 IO13 IO12 GND 5V
ESP32-CAM
3V3 IO16 IO0 GND VCC U0R U0T GND

Figure 3. ESP32-CAM development board.
From https://randomnerdtutorials.com/esp32-cam-video-streaming-face-recognition-arduino-ide/.

iii. IMU Sensor Module

To stabilize the drone, we need an Inertial Measurement Unit (IMU) sensor. This can be a 6-axis accelerometer & gyroscope[1] or a 9-axis accelerometer, gyroscope, and magnetometer. Although the additional magnetometer (which measures magnetic field strength) can be handy, 6-axis IMU sensors are usually enough.

The drone that we use already has built-in IMU sensor to determine how to control each of its 4 motors to achieve the desired maneuver. However, we cannot access the data from this sensor. Instead, we use our own IMU sensor module. An example of IMU sensor module is the GY-91 module based on the MPU9250 9-axis IMU chip. This module also has a built-in barometer (BMP280).

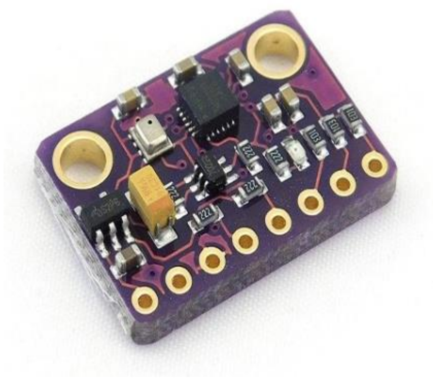


Figure 4. GY-91 9-axis IMU sensor module.

[1] Accelerometer measures linear acceleration, gyroscope measures angular velocity (rate of rotation). Both the accelerometer and gyroscope have 3 axis of measurement each (hence "6-axis accelerometer & gyroscope").

iv. Other Sensor Modules

We may also need additional sensor modules to augment the capability of the drone. For example, a ToF (time of flight) sensor module can be handy to measure the height of the drone above ground accurately, which is hard to get from IMU sensor alone. Keep in mind of the **payload limitation** of the drone, though.

End Goal and Examples of Application

The goal is to build a quadcopter with camera and sensors so that it is versatile enough to perform useful tasks. Some example of useful tasks:

- Flying camera
- Face tracking
- Waypoint following (flying in a pre-determined path and altitude)
- etc.

We have put several demo videos on the GitHub repository under the `demonstration` folder: https://github.com/NelsenEW/EEE-DIP-UAVONICS/tree/main/demonstartion. These are the example applications that you might look into developing.[2]

[2] These examples are done with different drone(s), so it is fine if your drone has different stability characteristics.

Experiments with the Hardware

This section is for experimenting with each individual hardware used for this project and getting some sense of how they work. Try to do these experiments in Week 1 to allow enough time for further developments.

Playing with the Quadcopter

You are given an unmodified Eachine E010 drone. Fly and play around with the drone to get a sense of how it flies. Before taking off, **calibrate the drone** by connecting the remote control to the drone then holding both joysticks to the bottom left corner. Refer to the drone user manual for the controls.

A few things about the flight characteristic of the drone that you might notice upon flying it:

- The drone does not have advanced stabilization feature; it needs constant adjustment from the pilot to counter drifts (i.e. the drone does not hover in place and drifts if you let go of the remote controller).
- The drone quickly loses thrust as the battery discharges; overtime you need to apply more power to make it hover at the same height.

It might seem hard to fly the drone at first, but after some time you should be familiar with the handling and be able to fly it stably. The aim is to **make our camera-equipped drone as stable as this drone**.

Experiment with ESP32-CAM

This experiment is for you to get started with the ESP32-CAM. To program the ESP32-CAM board, we will use Arduino IDE. Install Arduino IDE on your computer if you have not done so: https://www.arduino.cc/en/software/.

After that, install the ESP32 add-on so you can program the ESP32-CAM board (which is not an Arduino board) using the Arduino IDE. Guide to do so:

- https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/ (for Windows)
- https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-mac-and-linux-instructions/ (for Mac OS and Linux)

Finally, there are some libraries that you will need to download on Arduino IDE for the example code to run. All the required libraries are listed in `packages.txt` file on the GitHub page (https://github.com/NelsenEW/EEE-DIP-UAVONICS/blob/main/packages.txt). As we progress, the list of libraries that you need to install might grow, so keep an eye on this list. Guide to install libraries on Arduino IDE: https://www.arduino.cc/en/guide/libraries.

We will now run example code available in the Arduino IDE to familiarize with the process of uploading program to the board and see the capability of the board. The example code (`CameraWebServer`) streams the video data taken by the onboard camera to a webpage.

Follow the guide here (serves as a nice getting-started guide): https://randomnerdtutorials.com/esp32-cam-video-streaming-face-recognition-arduino-ide/.

In the link above, an FTDI programmer is used to program the board. However, you will be using an Arduino board to replace the FTDI programmer. **See "*Uploading Codes to the ESP32-CAM Board*" below.**

Uploading Codes to the ESP32-CAM Board

The board does not have onboard USB connector, hence an external programmer is needed to upload programs to it. We will use an Arduino Nano as the programmer. Make the following connection:

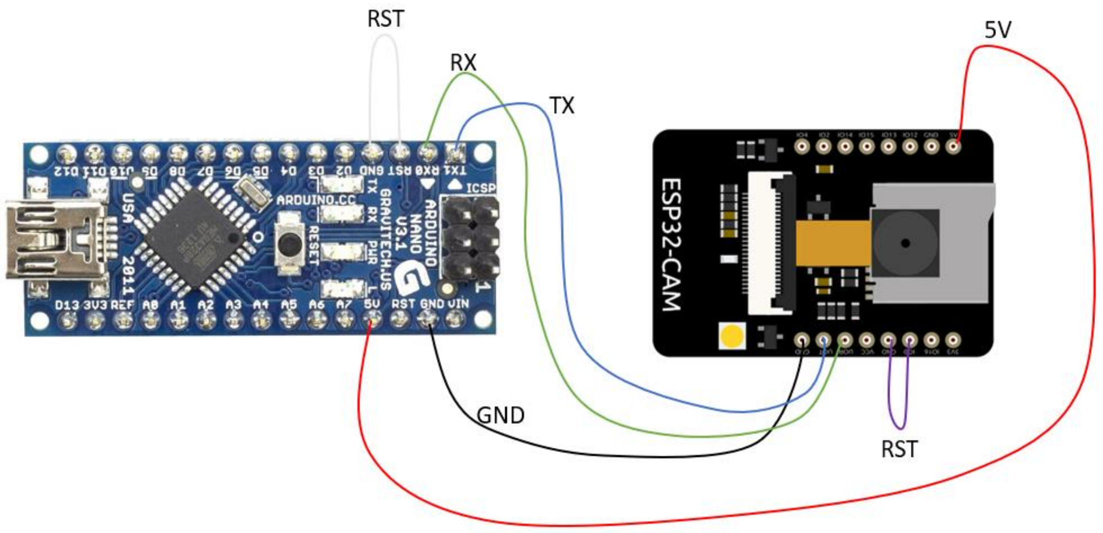


Figure 5. Configuration to upload program to the ESP32-CAM board.

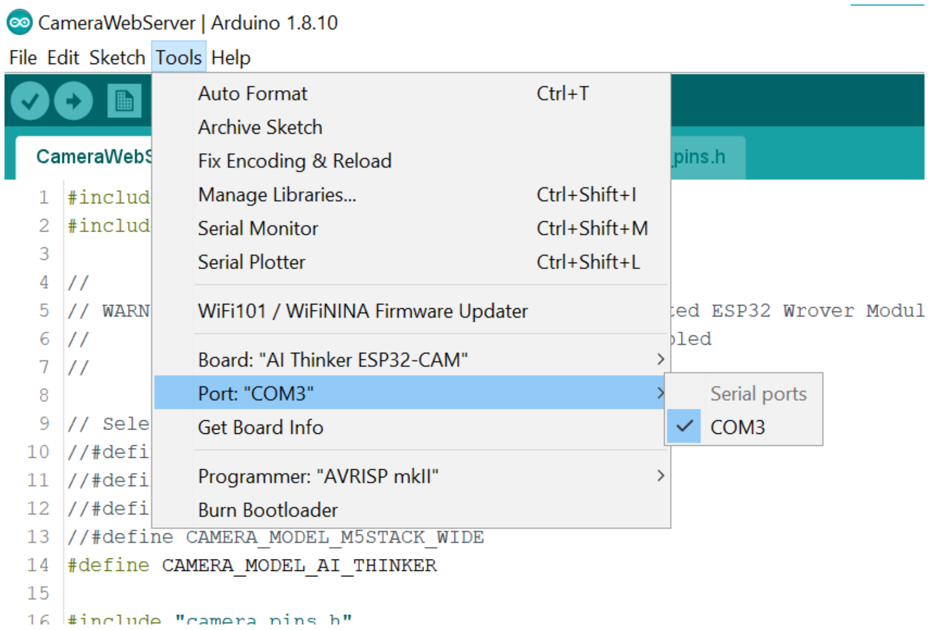
Note that the RX of Arduino Nano connects to RX of the ESP32-CAM[3], and TX to TX. After making the necessary connection, plug the Arduino Nano to your computer using the board's USB port. Reset the ESP32-CAM so it enters programming mode. Resetting the ESP32-CAM can be done in the following ways:

- Pressing the RESET button (normal way).

[3] If you are familiar with UART: you might think that this is a typo; it is not. Usually TX connects to RX and vice versa. For this case however, RX connects to RX and TX to TX because the ESP32-CAM "borrows" the Arduino Nano's onboard USB-TTL converter chip responsible for translating the USB protocol your computer uses into UART protocol that the ESP32-CAM understands. This is also why you need to connect the Arduino Nano's RST pin to GND; to keep the Arduino Nano's own microcontroller on reset so that it does not disturb the communication between your computer and the ESP32-CAM.

- If the RESET button is unreachable, the board can be power cycled: unplug and re-plug the GND cable going from the Arduino Nano to the ESP32 board (black cable), **not the 5V/red cable**[4].

On the Arduino IDE under the "Tools" menu, set the **board to "AI Thinker ESP32-CAM"** and the **port to the correct one** (one way to know is to unplug the Arduino Nano and see which one disappears from the list). See the following screenshot:


CameraWebServer | Arduino 1.8.10
File Edit Sketch Tools Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Board: "AI Thinker ESP32-CAM"
Port: "COM3"
Get Board Info
Programmer: "AVRISP mkII"
Burn Bootloader
Serial ports
COM3

```
//#define CAMERA_MODEL_M5STACK_WIDE
#define CAMERA_MODEL_AI_THINKER
```

Figure 6. Choose the correct Board and Port under the Tools menu.

Finally, press Upload (the right arrow logo below the "Edit" menu).

Remarks:

- **This will be the method that you use every time you upload codes to the ESP32-CAM board.**
- If the upload fails, try to reset the board again and reupload.
- After uploading the code, do not forget to **disconnect IO0 from GND (purple cable) and reset the board** to bring the ESP32-CAM out of programming mode. The connection from IO0 to GND is to bring the board to programming mode. Hence, forgetting to remove it will make the board stay in programming mode (does not run the uploaded code). In that case, this message will be printed on the Arduino IDE serial monitor:

[4] The purpose of "unplugging and re-plugging" the GND cable is to cut and restore power to the ESP32-CAM board, effectively resetting it. Normally, doing so on the 5V cable instead of the GND should also work. However, if you have other things connected to the ESP32-CAM board (which will be the case as we move further), unplugging the 5V cable does not always remove power from the ESP32-CAM. The board can still be "accidentally" powered via the GPIO pins due to the ESP32-CAM's internal circuitry.

TL;DR: unplugging the 5V cable does not always remove power from the board, unplugging GND does.

```
rst:0x1 (POWERON_RESET),boot:0x3 (DOWNLOAD_BOOT(UART0/UART1/SDIO_REI_REO_V2))
waiting for download
```

- When you need to upload another code to the board, reconnect IO0 to GND and reset the board to bring it back to programming mode.

Important Notes for Working with ESP32-CAM

- A website listing errors that might occur when using the ESP32-CAM board and how to fix them: https://randomnerdtutorials.com/esp32-cam-troubleshooting-guide/.
- Using Arduino IDE, the syntax and language used for programming the ESP32-CAM board are similar to that of Arduino boards (e.g. `digitalWrite()` also works for ESP32-CAM, as well as many other Arduino functions). This is helpful if you are familiar with Arduino boards.
- Like other microcontrollers, ESP32-CAM has multiple input/output pins (also referred to as GPIO) which can be programmed independently. Some GPIOs also have special functions, such as for communicating with other devices (including a computer with proper hardware).
  ESP32-CAM pinout:

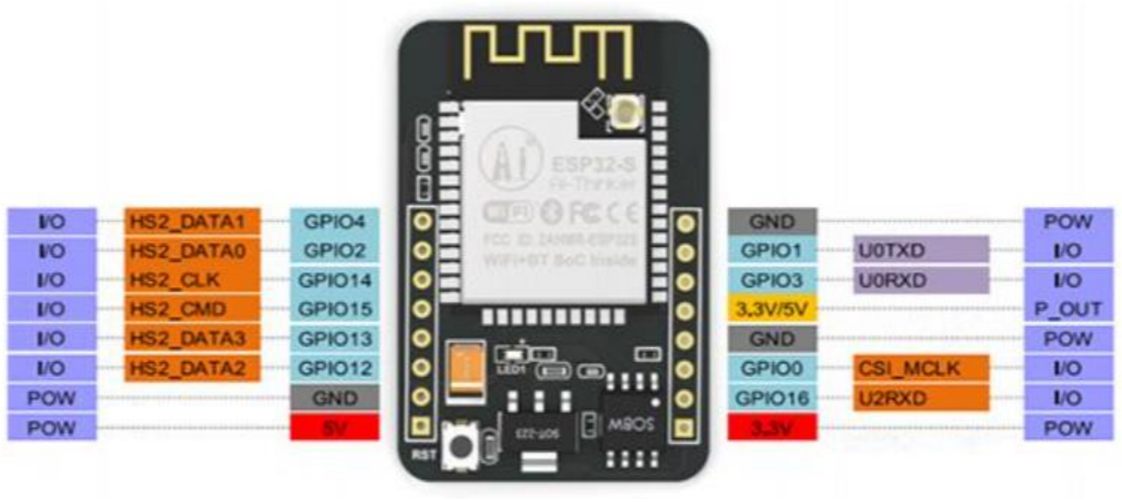


Figure 7. ESP32-CAM pinout.
From https://www.seeedstudio.com/ESP32-CAM-Development-Board-with-camer-p-3153.html.

- It is not easy to connect to enterprise network (NTUSECURE, NTUWL, etc.). Connect the ESP32-CAM board to a private router or to mobile hotspot (e.g. from your phone) for the `CameraWebServer` example and for further developments.
- The `CameraWebServer` example also has facial recognition feature (for demonstration purpose). The algorithm runs directly on the ESP32-CAM board and is very slow (approx. 2 FPS). Hence, if you want to run computationally heavy tasks (e.g. face recognition), **you have to send the image data from the ESP32-CAM to a computer** (e.g. your laptop), then the computer can perform the task and send the result back to the drone.
- The ESP32-CAM board has a dual-core microcontroller. You can make use of both cores to split the workload.

Experiment with GY-91 IMU Sensor

In this experiment, the ESP32-CAM reads data from the IMU sensor and prints the reading on Arduino IDE's serial monitor. You can download the code for this experiment from this project's GitHub repository under `examples` folder: `mpu9250_test.ino`.

Note: *some example codes on the GitHub page also include header files (file extension: .h) and/or other C++ codes to compile correctly (you can see what header files the code needs by looking at the* `#include<>` *directives inside the code). We have put these include files under the* `include` *folder on the GitHub repository. Move or copy these files to the same folder as the code before you compile it.*

The ESP32-CAM communicates with the GY-91 module via a communication protocol known as **Inter-Integrated Circuit (I^2C).** This protocol allows multiple devices (up to 112 for standard addressing scheme) to communicate with each other using only two wires. Arduino provides functions to perform communication using I^2C, but you can check this out if you want to know more about I^2C: https://www.robot-electronics.co.uk/i2c-tutorial.

For this experiment, make the following setup:

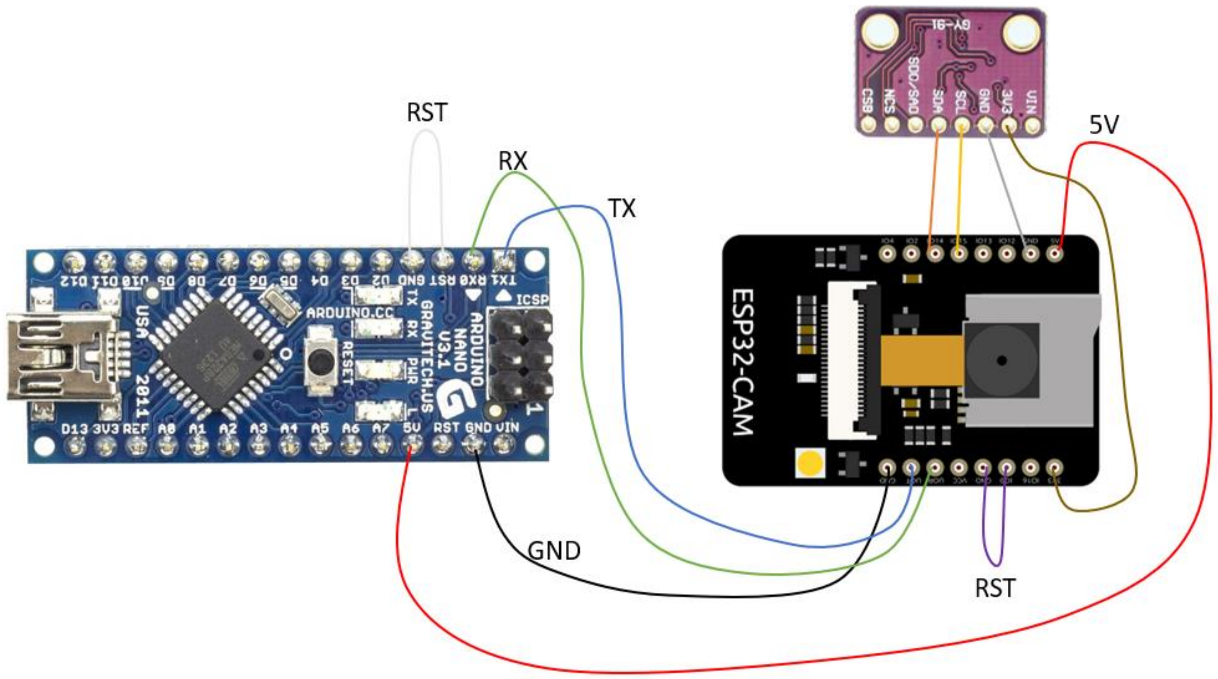


Figure 8. Hardware connection for experiment with GY-91 IMU sensor.

Notice that the setup above is for both programming the ESP32-CAM and reading data from the GY-91 module. You can see that only 4 wires go to the GY-91 module; 2 for power, 2 for I^2C communication[5]. The GPIOs of the ESP32-CAM which we use for the I^2C communication can be specified from the code (and can be changed).

[5] The 2 wires are SDA for **s**erial **da**ta and SCL for **s**erial **cl**ock. The clock's purpose is to synchronize all devices on the bus; recall EE2004.

After uploading the code to the board, do not forget to remove the connection from GPIO0 to GND (purple cable) and reset the ESP32-CAM. **Do not** remove other wires or unplug the Arduino Nano from your computer. Then, open Serial Monitor on the Arduino IDE and set the baud rate of the monitor to 115200. You should see some output on the serial monitor.

If the code works successfully, try to move the sensor and observe the reading, and read through the code to get a glimpse of how it works. If it does not run, check that:

- You got all connections correct. The GY-91 module's red LED will light up if it receives power (does not necessarily mean that the I^{2}C connection is correct).
- You have disconnected GPIO0 from GND (purple cable) **and reset the ESP32-CAM** after uploading.
- You have selected the correct port on the Arduino IDE Tools menu.
- You have set the correct baud rate on the serial monitor.

Explanation for This Example

The ESP32-CAM queries the readings from the GY-91 module via the I^{2}C lines (as can be seen in the hardware connection). The readings from the sensor module are raw and need to be processed by the ESP32-CAM:

- The readings sent via I^{2}C are not in proper units of measurement. They need to be converted into proper units (e.g. m/s^2 for acceleration). The code already handles this.[6]
- Converting the readings alone might not be enough. Sometimes we would also like to apply some algorithm on the reading, for example to smooth out the reading using a filter algorithm, or to estimate the sensor's position and attitude (roll, pitch, and yaw).

After processing the readings, they are sent to the computer to be displayed on the Arduino IDE Serial Monitor. The computer understands USB protocol, while the ESP32-CAM does not. Thus, in order to send data to the computer, we have to convert the protocol that the ESP32-CAM supports into USB protocol that the computer knows.

The protocol on the ESP32-CAM that we use to send the data to the computer is called UART (Universal Asynchronous Receiver Transmitter). Similar to I^{2}C, it only uses two wires to send and receive data. However, unlike I^{2}C:

- UART only supports communication between 2 devices (one-to-one).
- The 2 wires used are RX for receiving and TX for transmitting (no clock signal). Since it has dedicated wires for receiving and transmitting, UART supports transmitting and receiving data at the same time (referred to as "full-duplex") provided that both the communicating devices support that capability. In contrast, I^{2}C is half-duplex (cannot send and receive at the same time).
- Due to the absence of clock signal, the two devices must therefore agree beforehand on how fast the communication should be performed. This is why you need to set the Serial Monitor's baud

[6] In case you want to know how to convert the reading into proper units, refer to the datasheet of the sensor used on the module (MPU9250).

rate to 115200; you are telling the computer to communicate at 115200 bits/sec. This same speed is also programmed to the ESP32-CAM in the code.

The ESP32-CAM does not have USB-UART converter onboard, neither does the computer. The Arduino Nano, however, has one. The connection we make from the ESP32-CAM to the Arduino Nano is to allow the ESP32-CAM to "borrow" the Arduino Nano's USB-UART converter (also referred to as USB-TTL) so that it can communicate with the computer. This is why we do not remove the Arduino Nano after programming the ESP32-CAM[7]; we still need it.

[7] By the way, when we program the ESP32-CAM, the board receives the new code via UART, too. This is why we need the Arduino Nano for programming the board.