EE3080 UAVIONICS DIP Guide

Micro-drone and Its Applications

Version 1.0

(12 August 2021)

School of EEE

Nanyang Technological University

Table of Content

Changelog

This section lists the changes to this document as it is updated.

Version 1.0

Initial version.

Overview

DIP Overview

UAVONICS DIP is administered in 2 phases.

The first phase involves building and constructing a drone with a form factor of less than 10x10 cm. To achieve this, students will engage in building a flying drone equipped with a camera, powered by the ESP32 Microcontroller.

The second phase emphasizes on collaborative work to create a novel solution to a real-world problem scenario to be unveiled during the project. The application solution should demonstrate aspects of embedded AI, leveraging on the hardware built in phase one. 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 and getting familiar with the camera-equipped micro drone, while the aim of the second phase is to utilize the quadcopter for real-world applications.

Overview of the Micro Drone Hardware

The quadcopter we will be using for this project is small and lightweight (also known as MAV; Micro Aerial Vehicle) for safety and legal reasons. The micro drone is based on **ESP-Drone**, an open-source project for a micro drone powered by the ESP32 microcontroller.

For this DIP, we designed our own circuit board (PCB) for the drone and will use a modified version of the firmware of the ESP-drone project. We will refer to the micro drone as "ESP-drone" henceforth.

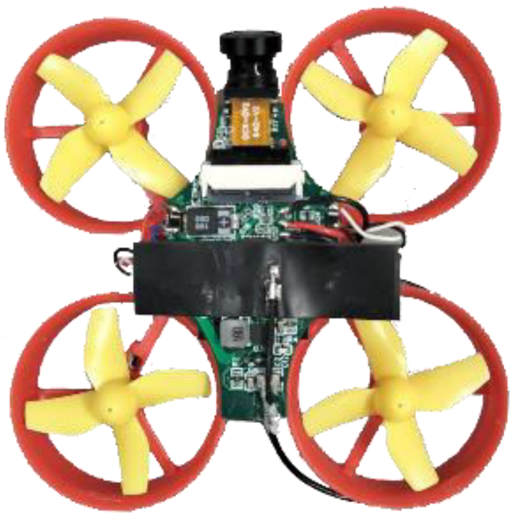


Figure 1. The ESP-drone specifically designed for this DIP project.

General specifications:

- Motor: 4x brushed motors

- Flight time: around 5 minutes
- Charge time: approx. 30 minutes (battery is removable)
- Dimension: approx. 85 x 85 x 30 mm
- Net weight: 26 gr
- Main processor: ESP32-S microcontroller
- Features: onboard Wi-Fi and Bluetooth, onboard camera, connector for external antenna

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 one of the limiting factors for this project.

The main features of this drone are Wi-Fi connectivity and onboard camera. With the Wi-Fi capability, it can communicate with a computer (or **even other drones**) and exchange data. We discuss the main components of the ESP-drone circuit board below.

i. ESP32 Microcontroller

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 that can interface with sensors and actuators (motors, etc.). Arduino boards are also development boards featuring microcontrollers on them.

The ESP32 is a widely popular family of microcontroller. It has a quite powerful CPU as well as Wi-Fi and Bluetooth capability built-in, which makes it highly suitable for IoT (Internet of Things) or applications requiring internet connectivity.

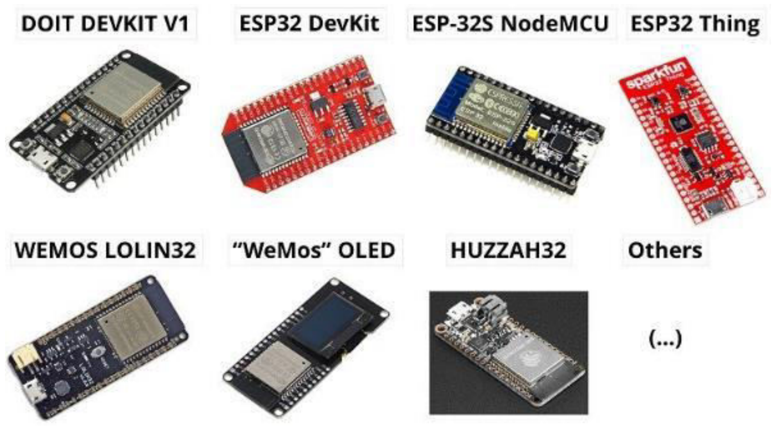


Figure 2. Various development boards featuring ESP32 microcontroller.
From https://randomnerdtutorials.com/getting-started-with-esp32/.

The ESP-drone is using the ESP32-S, a member of the ESP32 family featuring **dual-core** CPU, allowing us to perform multiple tasks concurrently. An example would be interfacing with a camera and streaming the image over Wi-Fi, maintaining communication with an external computer, and running control loops to stabilize the drone at the same time.

The ESP32-S board that we are using also features a connector for external antenna. We will use this to extend the range and signal quality of the drone's Wi-Fi.

ii. Inertial Measurement Unit

To stabilize the drone's flight, we need an Inertial Measurement Unit (IMU). The IMU we will be using is a 6-axis IMU[1]; 3-axis accelerometer and 3-axis gyroscope. The accelerometer measures the linear acceleration (including gravity) while the gyroscope measures angular velocity (rotational speed). The data from this sensor will be used as an input to the control loop of the drone.

iii. RGB Camera

Since the ESP32-S houses a dual-core CPU, we have enough processing power to integrate an onboard camera onto the drone. This is an RGB camera[2] which image can be streamed to an external computer via Wi-Fi. The computer can then perform image processing algorithms such as object detection or fiducial marker[3] detection. We will be using the camera for a vision-based positioning system for the ESP-drone.

iv. Other Sensor Modules

The circuit board of the ESP-drone has pads for adding an additional sensor to the drone[4]. 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. We need to keep in mind of the **payload limitation** of the drone, though.

The full hardware schematics of the ESP-drone's circuit board is available in **Appendix A** if you are interested to check it out.

Overview of The Software

Software is also a very important part of this project. The software part of this project can be divided into several categories based on the functionality:

i. `espdrone-nh` – The Drone's Firmware

This is the firmware running directly on the ESP32 microcontroller on the drone. It is responsible for all low-level controls of the drone (e.g. controlling all 4 motors, talking with the onboard camera, reading sensor data and filtering them, generating flight trajectory, etc.) as well as handling Wi-Fi communication to and from the drone. The firmware is written in **C** and makes use of ESP32's official development framework, *ESP-IDF* (we will see this in later parts).

ii. `espdrone-lib-python (edlib)` – The API

This is a library/API for communicating with the drone running on (external) computer. This API can be used to write scripts to fetch data and/or send instructions to the drone, or used by other software, as the backend engine, for similar purposes. It is written in **Python**.

[1] Specifically, we are using the popular MPU-6050 by Invensense.
[2] Specifically, the camera model is OV2640 by Omnivision.
[3] This is a marker which can be used as point of reference placed on the environment where the camera operates.
[4] The sensor added via this pad communicates with the ESP32 on the drone using the I^2C communication protocol.

iii. `espdrone-client` – GUI for Debugging and Testing

This is a graphical user interface written in Python with **PyQt5** for testing and debugging with the drone. With it we can view sensor and other readings, change the drone's parameters, and control the drone (e.g. using joysticks). This tool uses `espdrone-lib-python` as the backend.

iv. Others

Apart from the three mentioned above, we will also be using a simple Android app to fly the drone manually, as well as some other software to augment the capability of the drone (e.g. **ROS**). All these will be discussed in later parts.

Before we start, we will setup the toolchain and software needed to develop with the drone, discussed in the following part.

Setting Up the ESP32 Toolchain

Installing the Necessary Software

In order to develop and upload programs to the ESP32 on the drone, we need to set up the toolchain. We are going to use ***ESP-IDF*** (Espressif IoT Development Framework), which is provided directly by the company behind ESP32. We are not going to use Arduino IDE as it is not powerful enough for the task.

Before setting up the toolchain, install these on your computer if you have not already:

1. **Git** (not to be mistaken with **Github**), a version control software. You do *not* need to install Github on your desktop. Link: https://git-scm.com/download
2. **Microsoft Visual Studio Code (VS Code)**, not to be mistaken with **Visual Studio**. This is a powerful code editor that we will use. Link: https://code.visualstudio.com/download

Additional setup for Mac users using terminal (in order):

3. **Homebrew**, MacOS package manager.

```
/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

4. **CMake**, build process integration for cross environment.

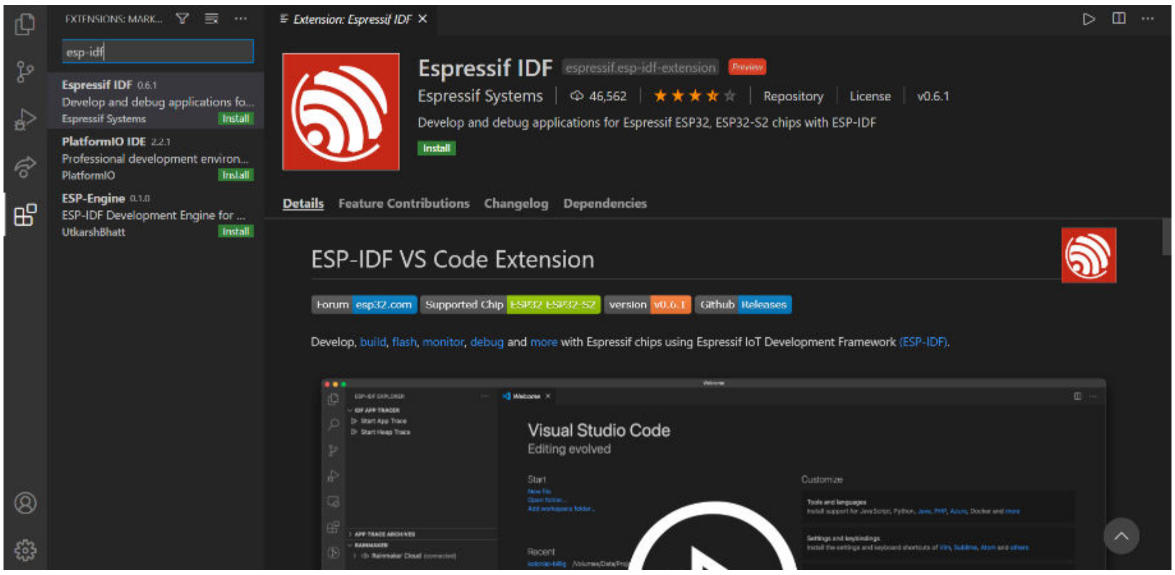
```
brew install cmake
```

5. **Ninja**, a small build system for speed.

```
brew install ninja
```

Installing ESP-IDF

After installing all the above software, launch VS Code and install the **ESP-IDF extension**.

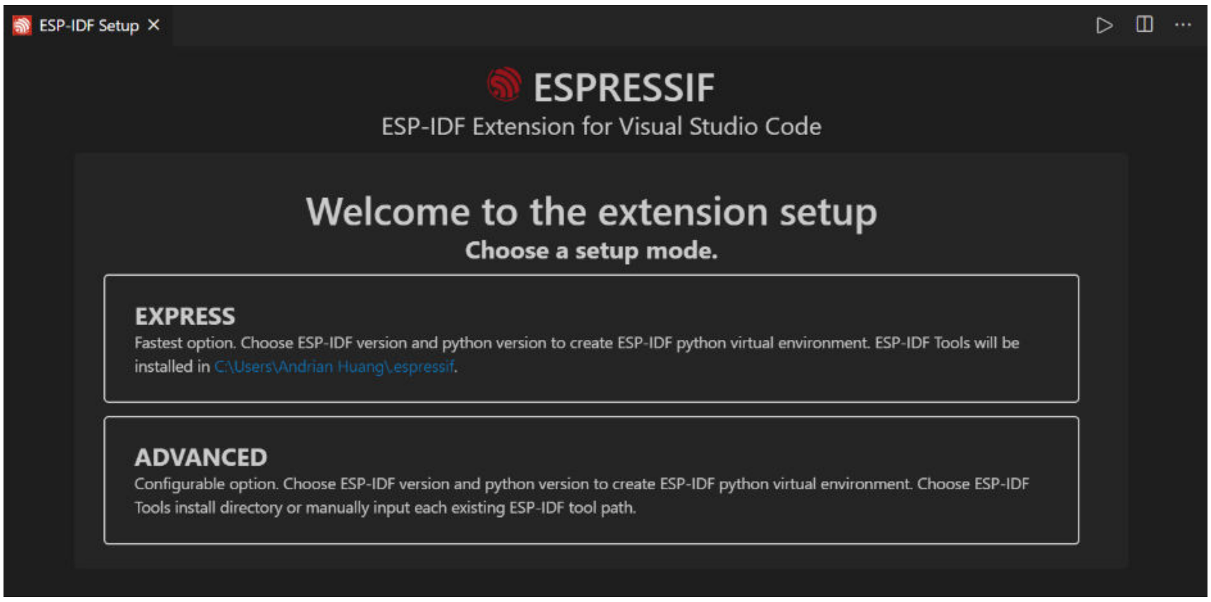


Develop and debug applications for Espressif ESP32, ESP32-S2 chips with ESP-IDF

Develop, build, flash, monitor, debug and more with Espressif chips using Espressif IoT Development Framework (ESP-IDF).

Figure 3. ESP-IDF extension on VS Code extension marketplace.

After you finish installing the extension, you should see this on VS code:


ESP-IDF Setup

ESPRESSIF

ESP-IDF Extension for Visual Studio Code

Welcome to the extension setup

Choose a setup mode.

EXPRESS

Fastest option. Choose ESP-IDF version and python version to create ESP-IDF python virtual environment. ESP-IDF Tools will be installed in C:\Users\Andrian Huang\.espressif.

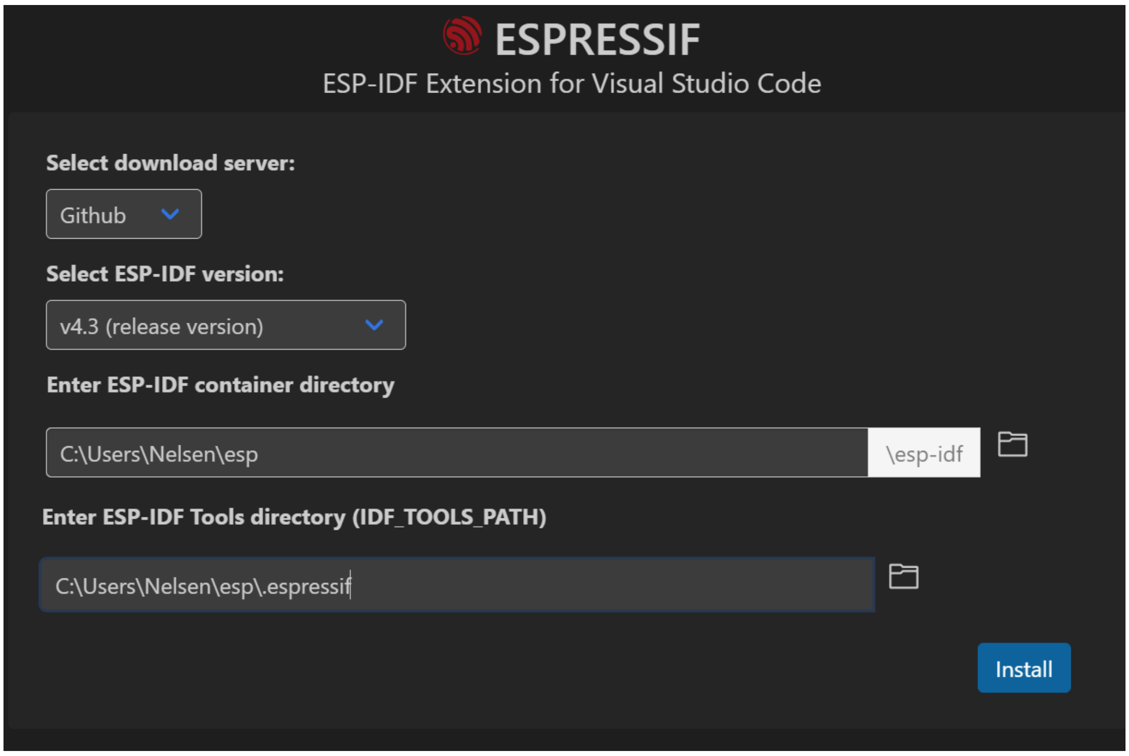
ADVANCED

Configurable option. Choose ESP-IDF version and python version to create ESP-IDF python virtual environment. Choose ESP-IDF Tools install directory or manually input each existing ESP-IDF tool path.

Figure 4. ESP-IDF setup screen after installing the extension.

Next, choose **"Express".**

The ESP-IDF container directory **does not support directory with whitespaces in its full path**. As such, choose a folder which full path does not contain spaces.


ESPRESSIF

ESP-IDF Extension for Visual Studio Code

Select download server: Github

Select ESP-IDF version: v4.3 (release version)

Enter ESP-IDF container directory: C:\Users\Nelsen\esp \esp-idf

Enter ESP-IDF Tools directory (IDF_TOOLS_PATH): C:\Users\Nelsen\esp\.espressif

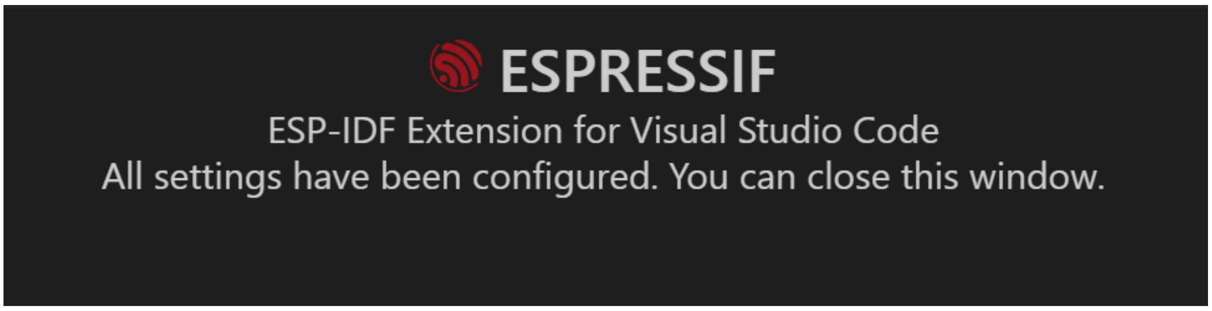
Figure 5. Choose the ESP-IDF container directory to be a path without whitespaces, like shown above.

For example, on Windows computer you may create a folder called "esp" on C: drive and choose this folder to be the directory for ESP-IDF container. For the other dropdown lists:

- Download server: we tested with Github but this choice should not matter.
- ESP-IDF version: "v4.3 (release version)".

By default, ESP-IDF Tools is put under the folder "`.espressif`" (a hidden folder, **please remember this directory**). You can leave this as is as long as the full path has no spaces. In this case, it is put under the same directory as the ESP-IDF container directory inside the "esp" folder.

Then, click "Install" and wait for the installation to finish. After everything has been installed, you should see the following window:

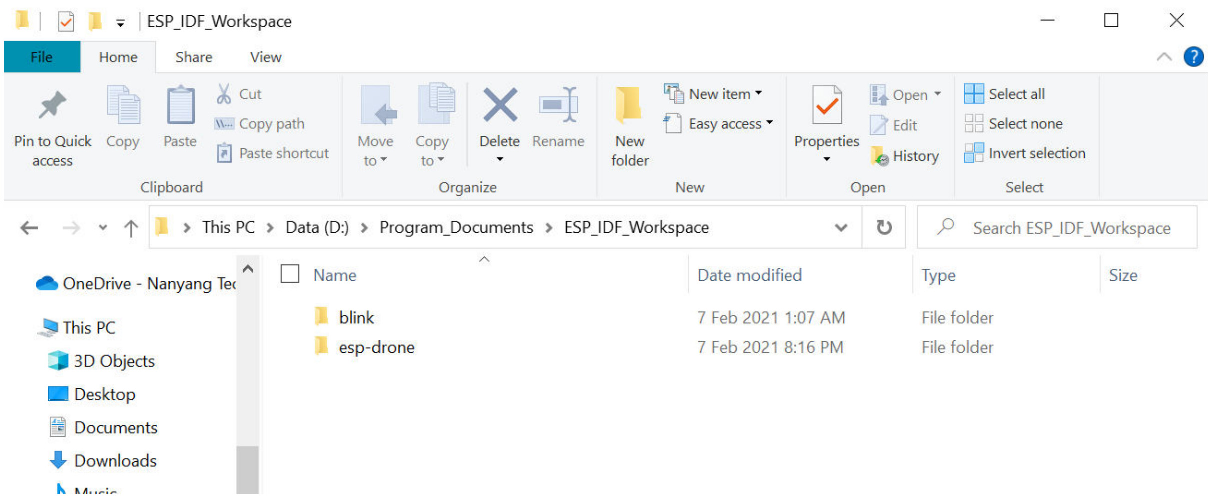

ESPRESSIF

ESP-IDF Extension for Visual Studio Code

All settings have been configured. You can close this window.

Figure 6. ESP-IDF installation window after finishing the installation process.

Next, you can optionally create another folder where you will put all your ESP-IDF projects in. You can place this folder anywhere and name it whatever you like, as long as the full path **does not contain whitespaces**. An example is **C:/espressif/workspaces**. Another example is shown below:



Name	Date modified	Type	Size
blink	7 Feb 2021 1:07 AM	File folder	
esp-drone	7 Feb 2021 8:16 PM	File folder	

Figure 7. Example of a workspace folder with two projects inside it (the full path here is **D:/Program_Documents/ESP_IDF_Workspace**). Notice that **there are no spaces in the full path**.

Testing the ESP-IDF Installation

To confirm that the ESP-IDF installation works, we will compile a built-in example project in ESP-IDF.

ESP-IDF Example Projects

To open the example project, press "F1" (or Ctrl+Shift+P) to open "Command Palette", then type "esp-idf examples", then choose "ESP-IDF: Show Examples Projects" -> "Use current ESP-IDF".

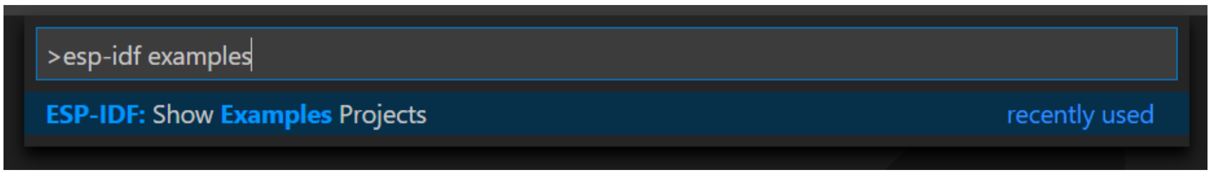
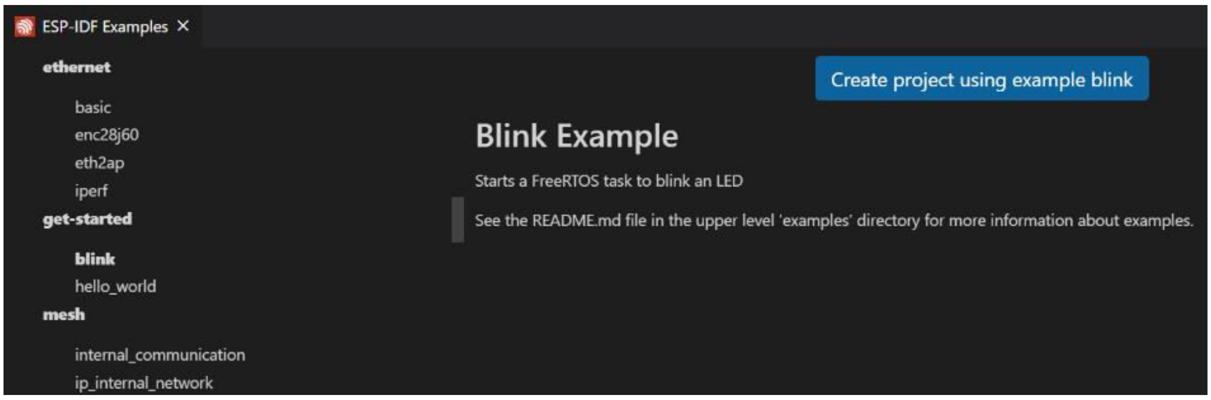


Figure 8. Accessing ESP-IDF example projects.

We will try a simple example: the blinking LED. Find the project `blink` under `get_started` and click “Create project using example blink”.

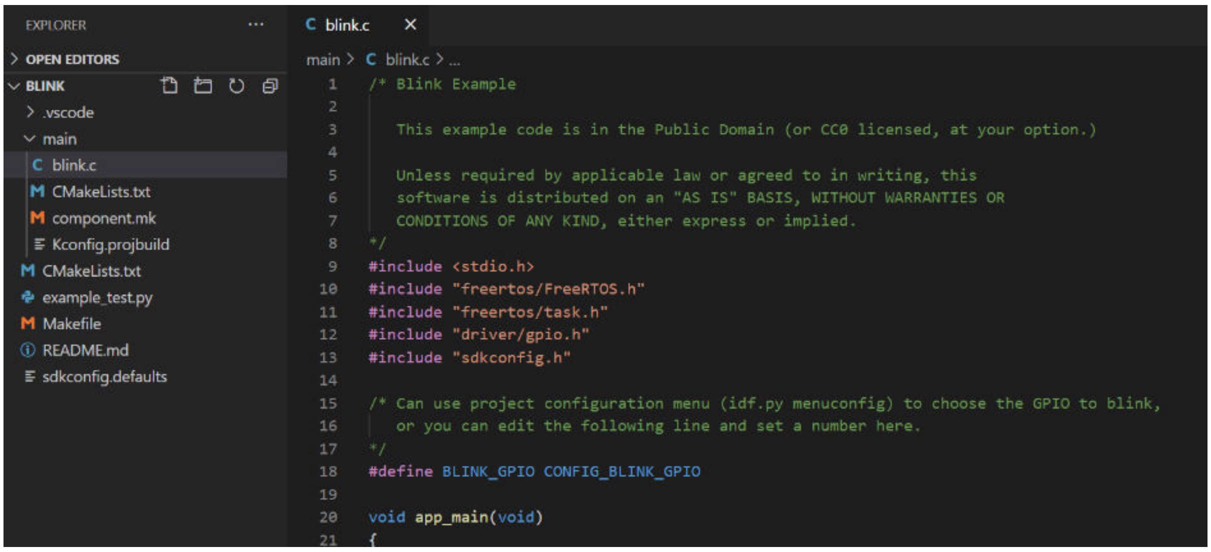


Starts a FreeRTOS task to blink an LED

See the README.md file in the upper level 'examples' directory for more information about examples.

Figure 9. Opening the `blink` example.

You will be asked to choose the folder in which you want to put the new example project in. Choose the folder that you have created previously. You should now see the new example project open automatically.



```
/* Blink Example

   This example code is in the Public Domain (or CC0 licensed, at your option.)

   Unless required by applicable law or agreed to in writing, this
   software is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR
   CONDITIONS OF ANY KIND, either express or implied.
*/
#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/gpio.h"
#include "sdkconfig.h"

/* Can use project configuration menu (idf.py menuconfig) to choose the GPIO to blink,
   or you can edit the following line and set a number here.
*/
#define BLINK_GPIO CONFIG_BLINK_GPIO

void app_main(void)
{
```

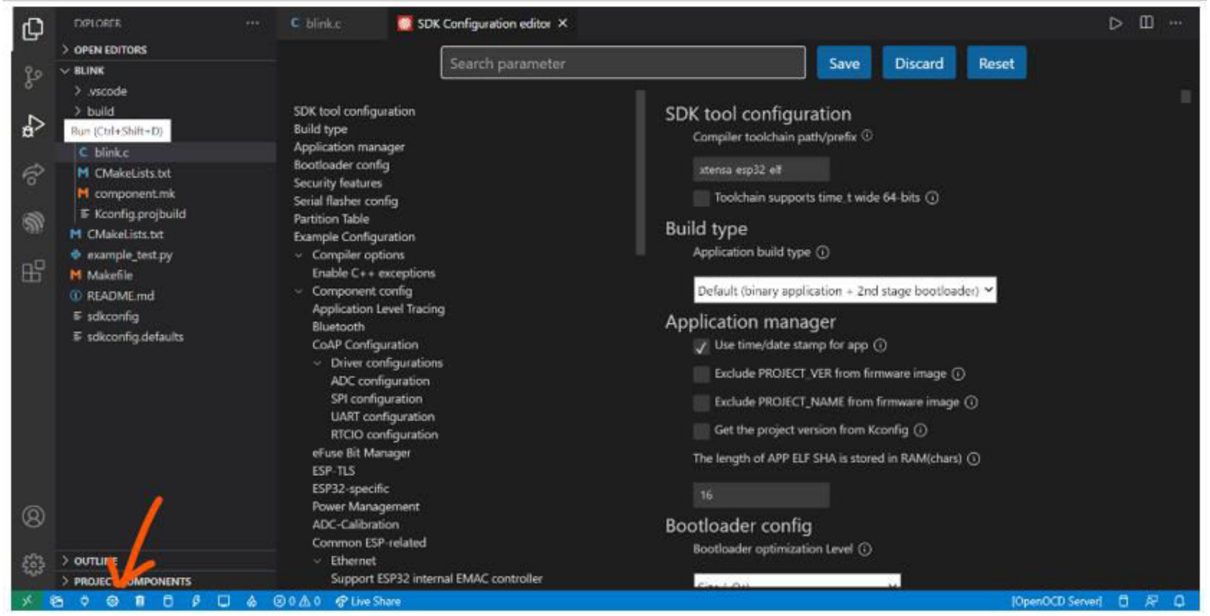
Figure 10. Example project `blink`.

The code here has been written, so we only need to set the configuration and build the project.

ESP-IDF SDK Configuration

Every ESP-IDF project has its own SDK configuration to set many aspects of the ESP32 for the project (including pins that are used in the code). For the `blink` example, the pin which the LED is attached to is configured in the SDK configuration.

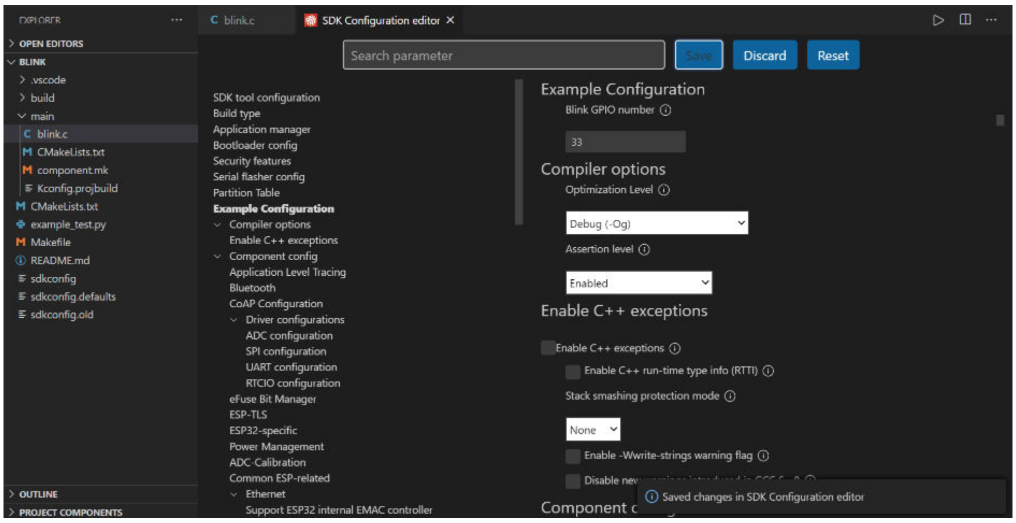
To access the SDK configuration editor, click the gear icon on the bottom left of the screen. You should then see the following window:



SDK tool configuration
Compiler toolchain path/prefix: xtensa-esp32-elf
Toolchain supports time_t wide 64-bits
Build type
Application build type: Default (binary application + 2nd stage bootloader)
Application manager
Use time/date stamp for app ✓
Exclude PROJECT_VER from firmware image
Exclude PROJECT_NAME from firmware image
Get the project version from Kconfig
The length of APP ELF SHA is stored in RAM(chars): 16
Bootloader config
Bootloader optimization Level

Figure 11. ESP-IDF SDK Configuration editor window.

Scroll down to find “Blink GPIO number” in the configuration. Change it to **33** since the red LED on our drone PCB is on that GPIO pin (you can also use GPIO 13 for the blue LED if you want to). After that, click “Save”. You should see a new `sdkconfig.old` file appear on the left[5].



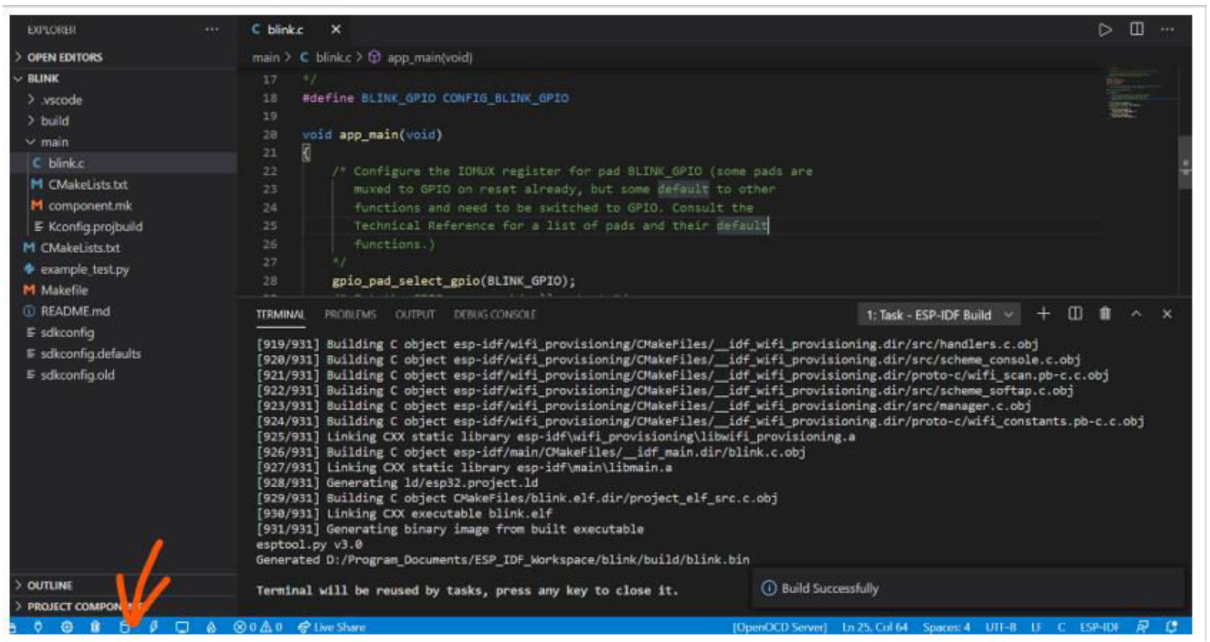
Example Configuration
Blink GPIO number: 33
Compiler options
Optimization Level: Debug (-Og)
Assertion level: Enabled
Enable C++ exceptions
Enable C++ exceptions
Enable C++ run-time type info (RTTI)
Stack smashing protection mode: None
Enable -Wwrite-strings warning flag
Saved changes in SDK Configuration editor

[5] This `sdkconfig.old` file is actually the former `sdkconfig` file. The `sdkconfig` file contains all the configuration that will be used when compiling the code. When we modify the blink GPIO number and save the configuration, a new `sdkconfig` file is generated, hence the old one is automatically renamed into `sdkconfig.old`.

Figure 12. Configure the GPIO pin number to blink to either 33 or 13 in the SDK Configuration editor.

Building the Project

After modifying and saving the configuration, we can build (compile) the project. To do so, click the cylinder icon on the bottom left of the screen (to the right of the trash can icon). Wait for the build process to finish. A successful build should look like the following:



```
*/
#define BLINK_GPIO CONFIG_BLINK_GPIO

void app_main(void)
{
    /* Configure the IOMUX register for pad BLINK_GPIO (some pads are
       muxed to GPIO on reset already, but some default to other
       functions and need to be switched to GPIO. Consult the
       Technical Reference for a list of pads and their default
       functions.)
    */
    gpio_pad_select_gpio(BLINK_GPIO);
```

```
[919/931] Building C object esp-idf/wifi_provisioning/CMakeFiles/__idf_wifi_provisioning.dir/src/handlers.c.obj
[920/931] Building C object esp-idf/wifi_provisioning/CMakeFiles/__idf_wifi_provisioning.dir/src/scheme_console.c.obj
[921/931] Building C object esp-idf/wifi_provisioning/CMakeFiles/__idf_wifi_provisioning.dir/proto-c/wifi_scan.pb-c.c.obj
[922/931] Building C object esp-idf/wifi_provisioning/CMakeFiles/__idf_wifi_provisioning.dir/src/scheme_softap.c.obj
[923/931] Building C object esp-idf/wifi_provisioning/CMakeFiles/__idf_wifi_provisioning.dir/src/manager.c.obj
[924/931] Building C object esp-idf/wifi_provisioning/CMakeFiles/__idf_wifi_provisioning.dir/proto-c/wifi_constants.pb-c.c.obj
[925/931] Linking CXX static library esp-idf\wifi_provisioning\libwifi_provisioning.a
[926/931] Building C object esp-idf/main/CMakeFiles/__idf_main.dir/blink.c.obj
[927/931] Linking CXX static library esp-idf\main\libmain.a
[928/931] Generating ld/esp32.project.ld
[929/931] Building C object CMakeFiles/blink.elf.dir/project_elf_src.c.obj
[930/931] Linking CXX executable blink.elf
[931/931] Generating binary image from built executable
esptool.py v3.0
Generated D:/Program_Documents/ESP_IDF_Workspace/blink/build/blink.bin

Terminal will be reused by tasks, press any key to close it.
```

Figure 13. Example of a successful build.

If your build is successful, great! You have installed ESP-IDF correctly. You can proceed to uploading the program to the drone PCB (see ***Uploading to the ESP32*** below).

(Optional) While building, you may encounter an error similar to:

```
Invalid escape sequence \o
```

If this happens, do the following modifications:

1. Find the Python script `idf.py` inside `esp-idf/tools` directory. For example, if you chose `C:/espressif` as the directory to install ESP-IDF into, find the Python script inside `C:/espressif/esp-idf/tools`.

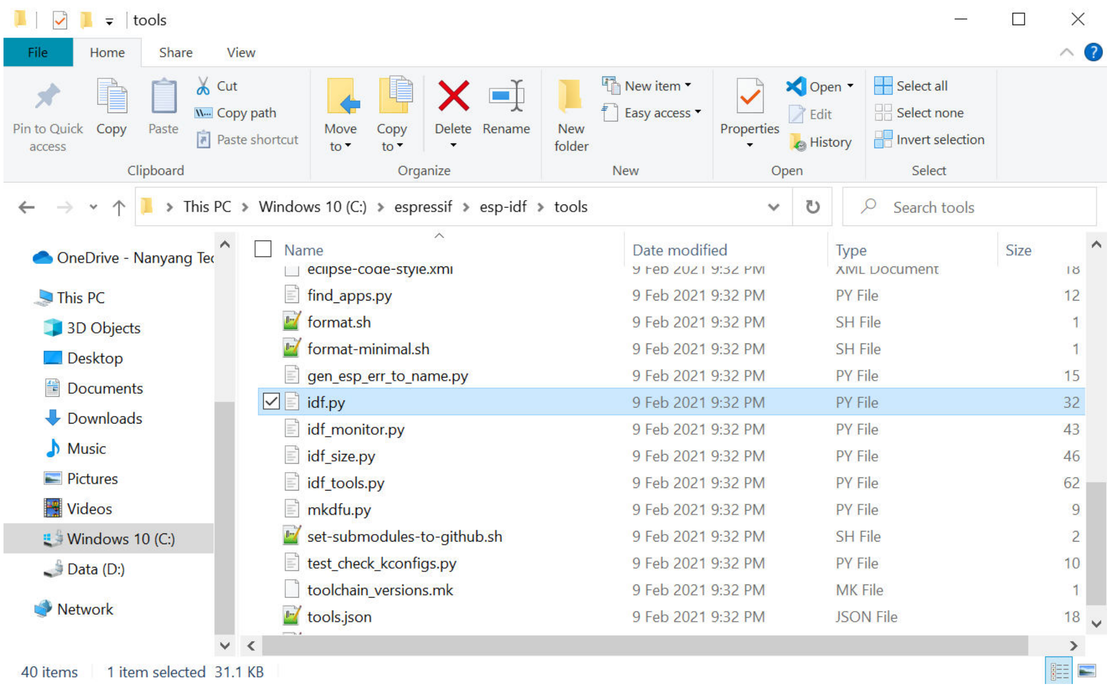
   Now, inside the Python script, change the code at line 52

   ```
   os.environ["PYTHON"] = sys.executable
   ```

   into

   ```
   import pathlib
   os.environ["PYTHON"] = pathlib.Path(sys.executable).as_posix()
   ```

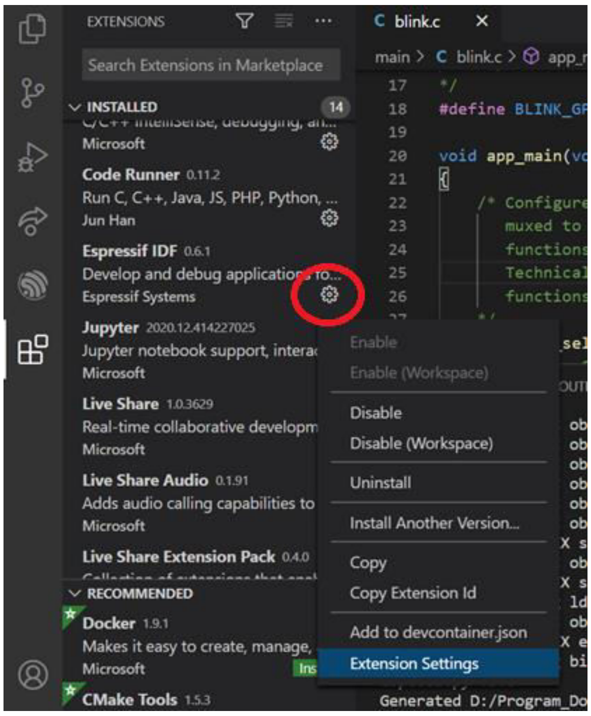
   Save the modified script (do not rename or save as).



Name	Date modified	Type	Size
eclipse-code-style.xml	9 Feb 2021 9:32 PM	XML Document	18
find_apps.py	9 Feb 2021 9:32 PM	PY File	12
format.sh	9 Feb 2021 9:32 PM	SH File	1
format-minimal.sh	9 Feb 2021 9:32 PM	SH File	1
gen_esp_err_to_name.py	9 Feb 2021 9:32 PM	PY File	15
idf.py	9 Feb 2021 9:32 PM	PY File	32
idf_monitor.py	9 Feb 2021 9:32 PM	PY File	43
idf_size.py	9 Feb 2021 9:32 PM	PY File	46
idf_tools.py	9 Feb 2021 9:32 PM	PY File	62
mkdfu.py	9 Feb 2021 9:32 PM	PY File	9
set-submodules-to-github.sh	9 Feb 2021 9:32 PM	SH File	2
test_check_kconfigs.py	9 Feb 2021 9:32 PM	PY File	10
toolchain_versions.mk	9 Feb 2021 9:32 PM	MK File	1
tools.json	9 Feb 2021 9:32 PM	JSON File	18

Figure 14. Example of where to find `idf.py` (if you installed ESP-IDF in C:/espressif).

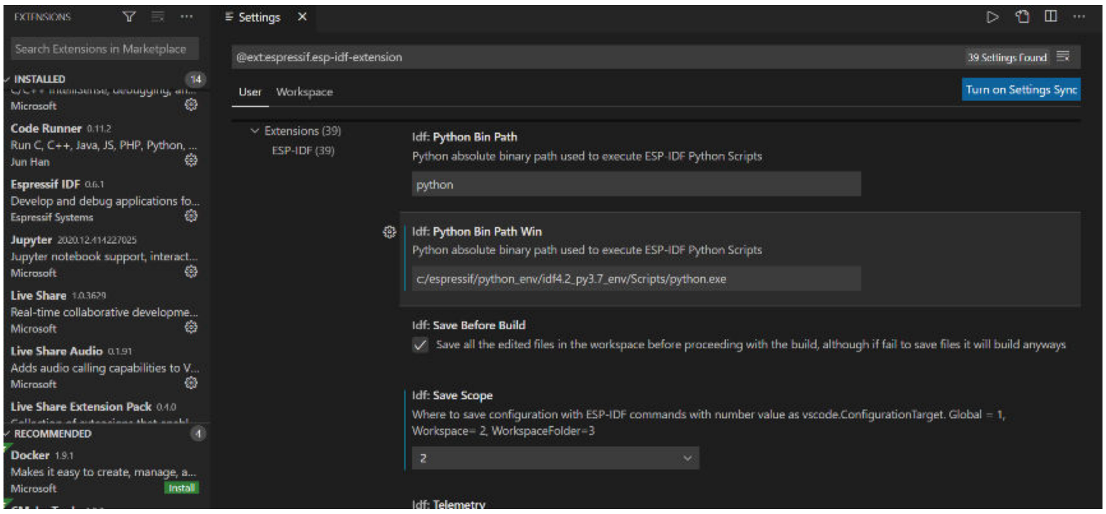
2. Open the ESP-IDF extension setting window. To do so, click the gear icon to the right of "Espressif IDF" under Extensions. Then, click "Extension Settings".



```
*/
#define BLINK_GP
void app_main(vo
{
    /* Configure
       muxed to
       functions
       Technical
       functions
```

Figure 15. Accessing extension settings.

Then, find "Idf: Python Bin Path Win" and replace all backslashes (\) with forward slashes (/).


EXTENSIONS
Search Extensions in Marketplace
INSTALLED 14
Microsoft
Code Runner 0.11.2
Run C, C++, Java, JS, PHP, Python, ...
Jun Han
Espressif IDF 0.6.1
Develop and debug applications fo...
Espressif Systems
Jupyter 2020.12.414227025
Jupyter notebook support, interact...
Microsoft
Live Share 1.0.3629
Real-time collaborative developme...
Microsoft
Live Share Audio 0.1.91
Adds audio calling capabilities to V...
Microsoft
Live Share Extension Pack 0.4.0
RECOMMENDED 4
Docker 1.9.1
Makes it easy to create, manage, a...
Microsoft
Install
Settings
39 Settings Found
User
Workspace
Turn on Settings Sync
Extensions (39)
ESP-IDF (39)

@ext:espressif.esp-idf-extension

Idf: Python Bin Path
Python absolute binary path used to execute ESP-IDF Python Scripts

python

Idf: Python Bin Path Win
Python absolute binary path used to execute ESP-IDF Python Scripts

c:/espressif/python_env/idf4.2_py3.7_env/Scripts/python.exe

Idf: Save Before Build
✓ Save all the edited files in the workspace before proceeding with the build, although if fail to save files it will build anyways

Idf: Save Scope
Where to save configuration with ESP-IDF commands with number value as vscode.ConfigurationTarget. Global = 1, Workspace= 2, WorkspaceFolder=3

2

Idf: Telemetry

Figure 16. Python Bin Path Win under Extension Settings.

Preparing the Software for the Drone

The espdrone Project

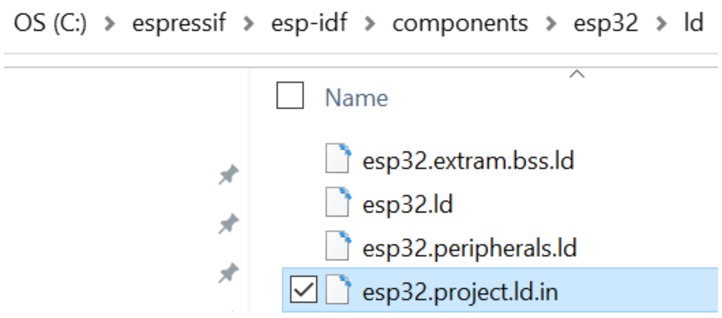
In the following weeks, we will base our developments on the open source `espdrone` project for development with the drone PCB. We have made the necessary adjustments for our hardware on the project and put the repository for this project on Github:

https://github.com/NelsenEW/espdrone-nh

You can put this inside the folder that you have previously created.

To use this project, two project link scripts must be modified.

1. Go to "`C:\Users\Username\esp\esp-idf\components\esp32\ld`" and find the script `esp32.project.ld.in`.



OS (C:) › espressif › esp-idf › components › esp32 › ld

Name

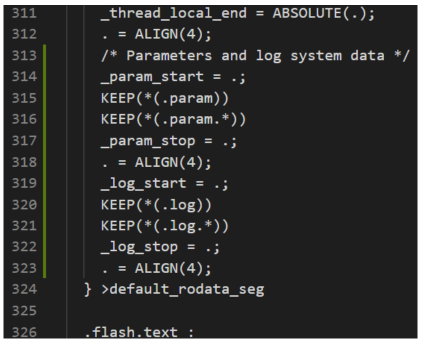
esp32.extram.bss.ld
esp32.ld
esp32.peripherals.ld
esp32.project.ld.in

Figure 17. ESP32 project link script.

Right click on the script and click "Open with Code" to open the script. Then, add the following lines before `} >default_rodata_seg`:

```
/* Parameters and log system data */
 _param_start = .;
 KEEP(*(.param))
 KEEP(*(.param.*))
 _param_stop = .;
 . = ALIGN(4);
 _log_start = .;
 KEEP(*(.log))
 KEEP(*(.log.*))
 _log_stop = .;
 . = ALIGN(4);
```

The newly added lines (highlighted in green) should look as follow:



```
    _thread_local_end = ABSOLUTE(.);
    . = ALIGN(4);
    /* Parameters and log system data */
    _param_start = .;
    KEEP(*(.param))
    KEEP(*(.param.*))
    _param_stop = .;
    . = ALIGN(4);
    _log_start = .;
    KEEP(*(.log))
    KEEP(*(.log.*))
    _log_stop = .;
    . = ALIGN(4);
  } >default_rodata_seg

  .flash.text :
```

Figure 18. ESP32 project link script modification.

2. Similarly, go to "`C:\Users\Username\esp\esp-idf\components\esp32s2\ld`" and find the script `esp32s2.project.ld.in`.

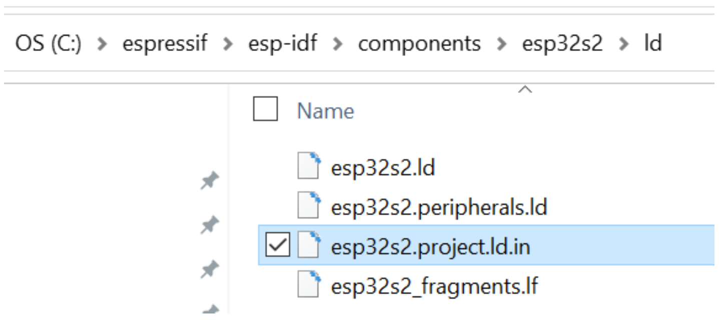
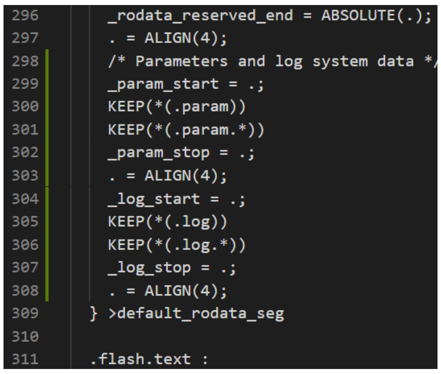


Figure 19. ESP32S2 Project link script.

Add the same lines to the script at the same location:



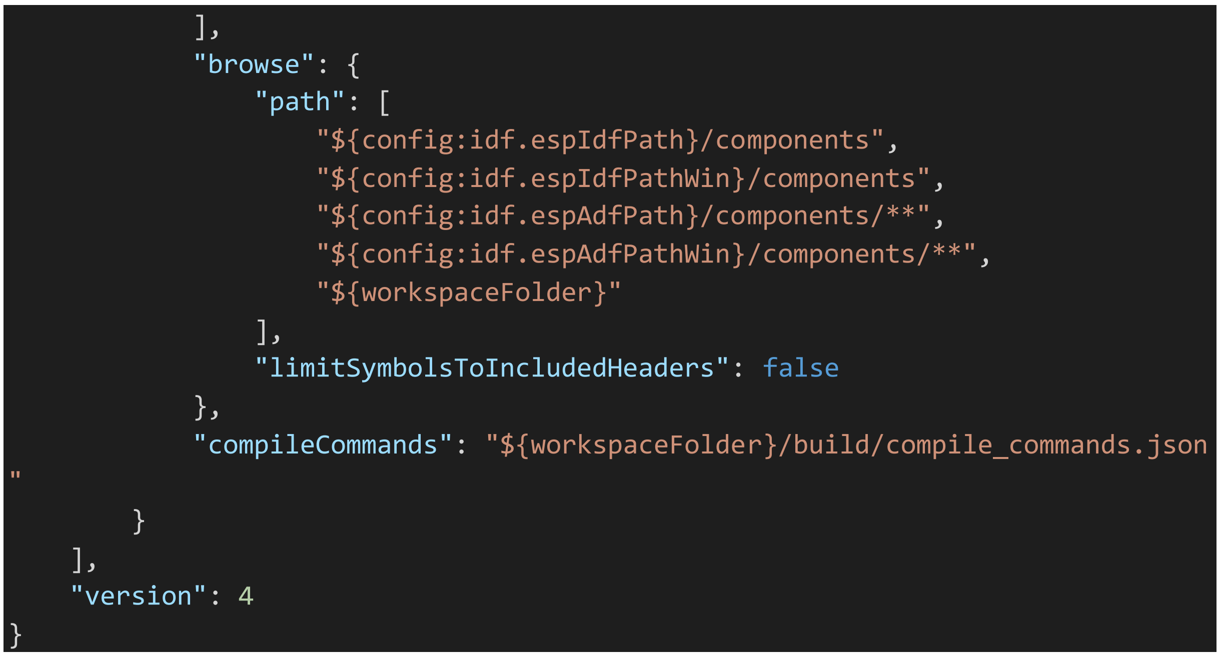
```
        _rodata_reserved_end = ABSOLUTE(.);
        . = ALIGN(4);
        /* Parameters and log system data */
        _param_start = .;
        KEEP(*(.param))
        KEEP(*(.param.*))
        _param_stop = .;
        . = ALIGN(4);
        _log_start = .;
        KEEP(*(.log))
        KEEP(*(.log.*))
        _log_stop = .;
        . = ALIGN(4);
    } >default_rodata_seg

    .flash.text :
```

Figure 20. ESP32S2 Project link script modification.

Testing the `espdrone-nh` Workspace

Once you have downloaded the `espdrone-nh` project, open VS Code and click "Open Folder…" on the menu bar and choose `espdrone-nh`. Go to "`.vscode`" inside the `espdrone-nh` directory. You should add the following:

```
{
    "configurations": [
        {
            "name": "ESP-IDF",
            "compilerPath": "C:\\Users\\Nelsen\\esp\\.espressif\\tools\\xtensa-esp32-elf\\esp-2020r3-8.4.0\\xtensa-esp32-elf\\bin\\xtensa-esp32-elf-gcc.exe",
            "cStandard": "c11",
            "cppStandard": "c++17",
            "includePath": [
                "${config:idf.espIdfPath}/components/**",
                "${config:idf.espIdfPathWin}/components/**",
                "${config:idf.espAdfPath}/components/**",
                "${config:idf.espAdfPathWin}/components/**",
                "${workspaceFolder}/**"
```

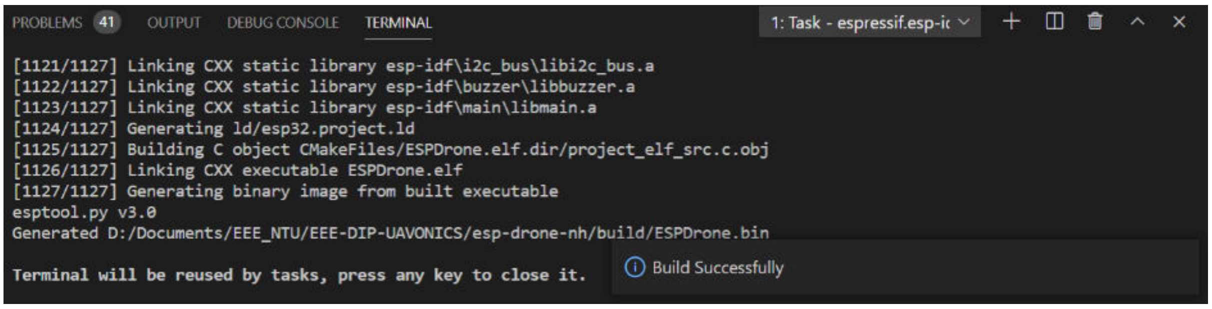


```
            ],
            "browse": {
                "path": [
                    "${config:idf.espIdfPath}/components",
                    "${config:idf.espIdfPathWin}/components",
                    "${config:idf.espAdfPath}/components/**",
                    "${config:idf.espAdfPathWin}/components/**",
                    "${workspaceFolder}"
                ],
                "limitSymbolsToIncludedHeaders": false
            },
            "compileCommands": "${workspaceFolder}/build/compile_commands.json
"
        }
    ],
    "version": 4
}
```

Figure 21. Visual Studio Code with `espdrone-nh c_cpp_properties.json`.

Don't forget to change the compilerPath! Compiler path is the path to `.espressif` directory followed by `tools\xtensa-esp32-elf\esp-2020r3-8.4.0\xtensa-esp32-elf\bin\xtensa-esp32-elf-gcc.exe`.

To test the project, we can try building the workspace by clicking the cylinder icon on the bottom left of the screen (to the right of the trash can icon). Wait for the build process to finish. A successful build should look like the following:



```
[1121/1127] Linking CXX static library esp-idf\i2c_bus\libi2c_bus.a
[1122/1127] Linking CXX static library esp-idf\buzzer\libbuzzer.a
[1123/1127] Linking CXX static library esp-idf\main\libmain.a
[1124/1127] Generating ld/esp32.project.ld
[1125/1127] Building C object CMakeFiles/ESPDrone.elf.dir/project_elf_src.c.obj
[1126/1127] Linking CXX executable ESPDrone.elf
[1127/1127] Generating binary image from built executable
esptool.py v3.0
Generated D:/Documents/EEE_NTU/EEE-DIP-UAVONICS/esp-drone-nh/build/ESPDrone.bin

Terminal will be reused by tasks, press any key to close it.
```

Figure 22. `espdrone-nh` successfully built.

For future developments with this repository, you can refer to the documentation:

https://github.com/NelsenEW/espdrone-nh/blob/main/docs/esp-drone-docs.pdf

Installing and Setting Up `espdrone-lib-python` and `espdrone-client`

Before setting up, install these on your computer if you have not already:

1. **Python 3.6 only!** You can install **Python 3.6** using Anaconda, although you do not have to. Link: https://www.python.org/downloads/release/python-368/

We will use `espdrone-client` (`edclient`) to interface with the drone (e.g. read sensor readings). To use it, clone it (as well as `espdrone-lib-python`, which it uses as backend) from Github:

```
git clone https://github.com/NelsenEW/espdrone-clients

git clone https://github.com/NelsenEW/espdrone-lib-python
```

Then, navigate to the folder `espdrone-lib-python` (use the `cd` command) and install the required packages:

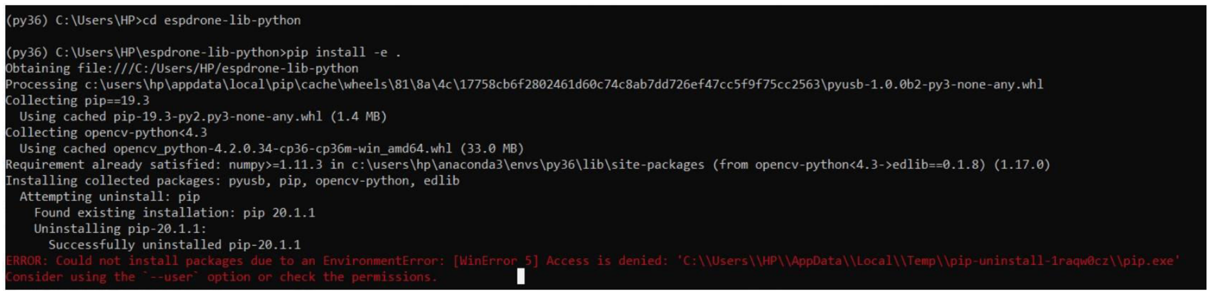
To install the dependencies of the `espdrone-lib-python` package, do the following command

```
pip install -r requirements.txt
```

Then, install the `espdrone-lib-python` in the editable mode:

```
pip install -e .
```

If you encounter the problem below:



```
(py36) C:\Users\HP>cd espdrone-lib-python

(py36) C:\Users\HP\espdrone-lib-python>pip install -e .
Obtaining file:///C:/Users/HP/espdrone-lib-python
Processing c:\users\hp\appdata\local\pip\cache\wheels\81\8a\4c\17758cb6f2802461d60c74c8ab7dd726ef47cc5f9f75cc2563\pyusb-1.0.0b2-py3-none-any.whl
Collecting pip==19.3
  Using cached pip-19.3-py2.py3-none-any.whl (1.4 MB)
Collecting opencv-python<4.3
  Using cached opencv_python-4.2.0.34-cp36-cp36m-win_amd64.whl (33.0 MB)
Requirement already satisfied: numpy>=1.11.3 in c:\users\hp\anaconda3\envs\py36\lib\site-packages (from opencv-python<4.3->edlib==0.1.8) (1.17.0)
Installing collected packages: pyusb, pip, opencv-python, edlib
  Attempting uninstall: pip
    Found existing installation: pip 20.1.1
    Uninstalling pip-20.1.1:
      Successfully uninstalled pip-20.1.1
ERROR: Could not install packages due to an EnvironmentError: [WinError 5] Access is denied: 'C:\\Users\\HP\\AppData\\Local\\Temp\\pip-uninstall-1raqw0cz\\pip.exe'
Consider using the `--user` option or check the permissions.
```

Try to re-run:

```
pip install -e .
```

The next step is to install the `espdrone-client`, to do that navigate to `espdrone-client`.

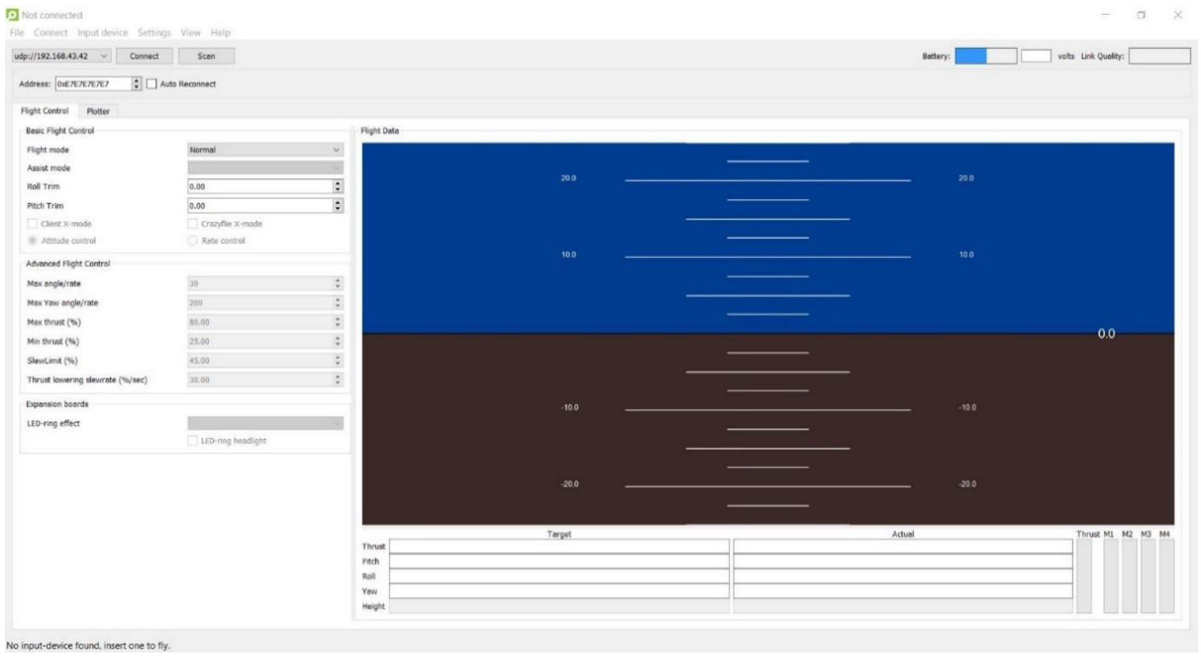
Then, install the `espdrone-client` in the editable mode:

```
pip install -e .
```

Finally, inside the same directory, launch the GUI interface (`edclient`):

```
edclient
```

You should then be greated with the following window:



Not connected

udp://192.168.43.42

Address: 0xE7E7E7E7E7 Auto Reconnect

Basic Flight Control

Flight mode	Normal
Assist mode	
Roll Trim	0.00
Pitch Trim	0.00
Client X-mode	Crazyflie X-mode
Attitude control	Rate control

Advanced Flight Control

Max angle/rate	30
Max Yaw angle/rate	200
Max thrust (%)	80.00
Min thrust (%)	25.00
SlewLimit (%)	45.00
Thrust lowering slewrate (%/sec)	30.00

Expansion boards

LED-ring effect	
	LED-ring headlight

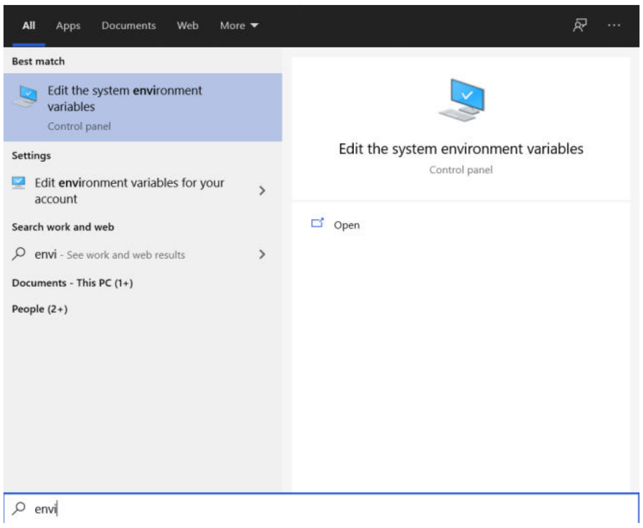
No input-device found, insert one to fly.

Figure 23. The `edclient` GUI.

Additional Setup (for Windows)

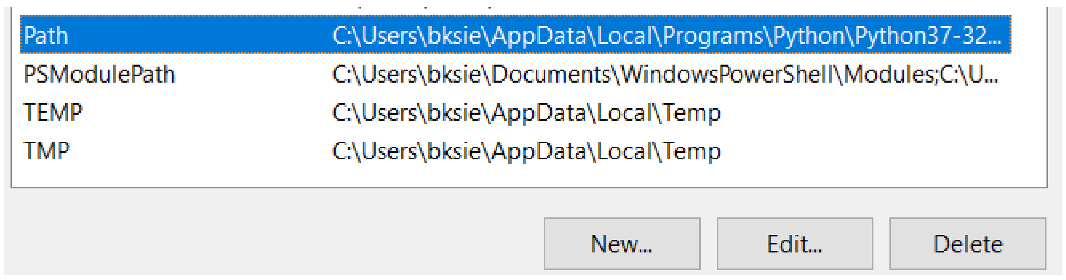
Up to now, you need to navigate to the `espdrone-clients` directory inside command prompt (CMD) in order to run `edclient`. To call `edclient` in CMD without needing to navigate to that directory, you can add a new Path pointing to it in system environment variables. You only need to do this if Python is not added to your system Path variable.

1. Open the “Edit system environment variables” window by typing “path” or “environment” in the Windows search box.



All
Apps
Documents
Web
More
Best match
Edit the system environment variables
Control panel
Settings
Edit environment variables for your account
Search work and web
envi - See work and web results
Documents - This PC (1+)
People (2+)
Edit the system environment variables
Control panel
Open
envi

2. On that window, click Path and press Edit…

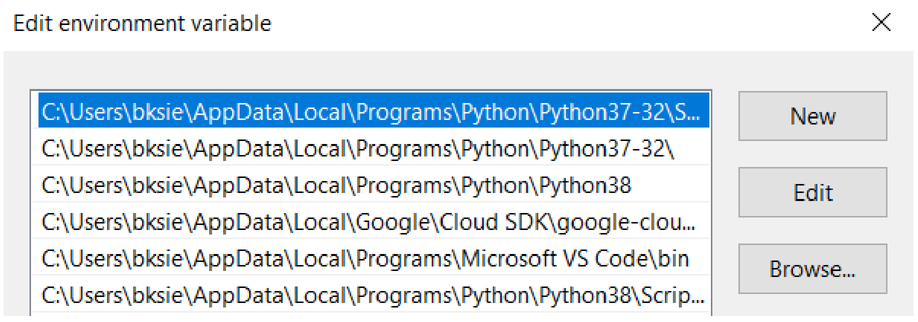


Path	C:\Users\bksie\AppData\Local\Programs\Python\Python37-32…
PSModulePath	C:\Users\bksie\Documents\WindowsPowerShell\Modules;C:\U…
TEMP	C:\Users\bksie\AppData\Local\Temp
TMP	C:\Users\bksie\AppData\Local\Temp

3. Create a new path pointing to python scripts, e.g.

   C:\Users\< your username >\AppData\Local\Programs\Python\Python36\Scripts

   or to other directories where Python stores its scripts at.



Edit environment variable

```
C:\Users\bksie\AppData\Local\Programs\Python\Python37-32\S…
C:\Users\bksie\AppData\Local\Programs\Python\Python37-32\
C:\Users\bksie\AppData\Local\Programs\Python\Python38
C:\Users\bksie\AppData\Local\Google\Cloud SDK\google-clou…
C:\Users\bksie\AppData\Local\Programs\Microsoft VS Code\bin
C:\Users\bksie\AppData\Local\Programs\Python\Python38\Scrip…
```

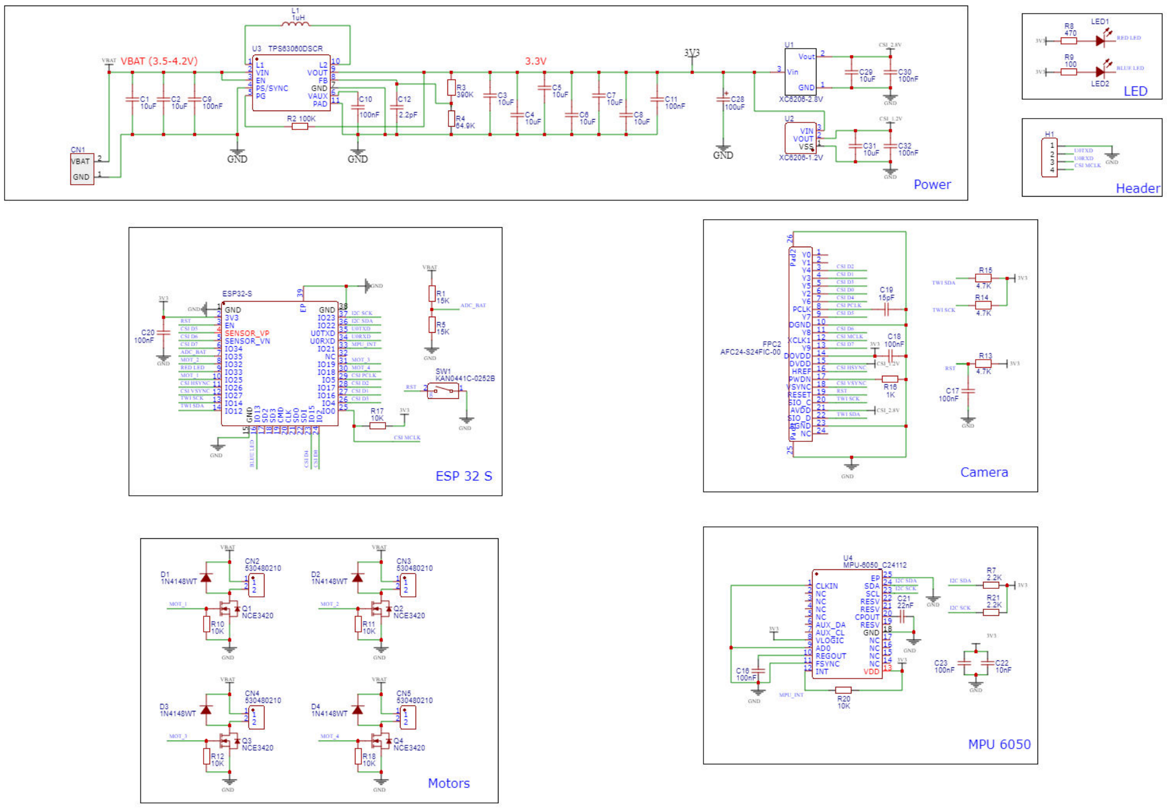
References:

https://docs.espressif.com/projects/espressif-esp-drone/en/latest/gettingstarted.html

https://github.com/NelsenEW/espdrone-lib-python

Appendix

Appendix A – Hardware Schematics of the ESP-Drone



VBAT (3.5-4.2V)
3.3V
TPS63060DSCR
XC6206-2.8V
XC6206-1.2V
Power
LED
Header
ESP32-S
ESP 32 S
AFC24-S24FIC-00
Camera
NCE3420
Motors
MPU-6050_C24112
MPU 6050