How to make

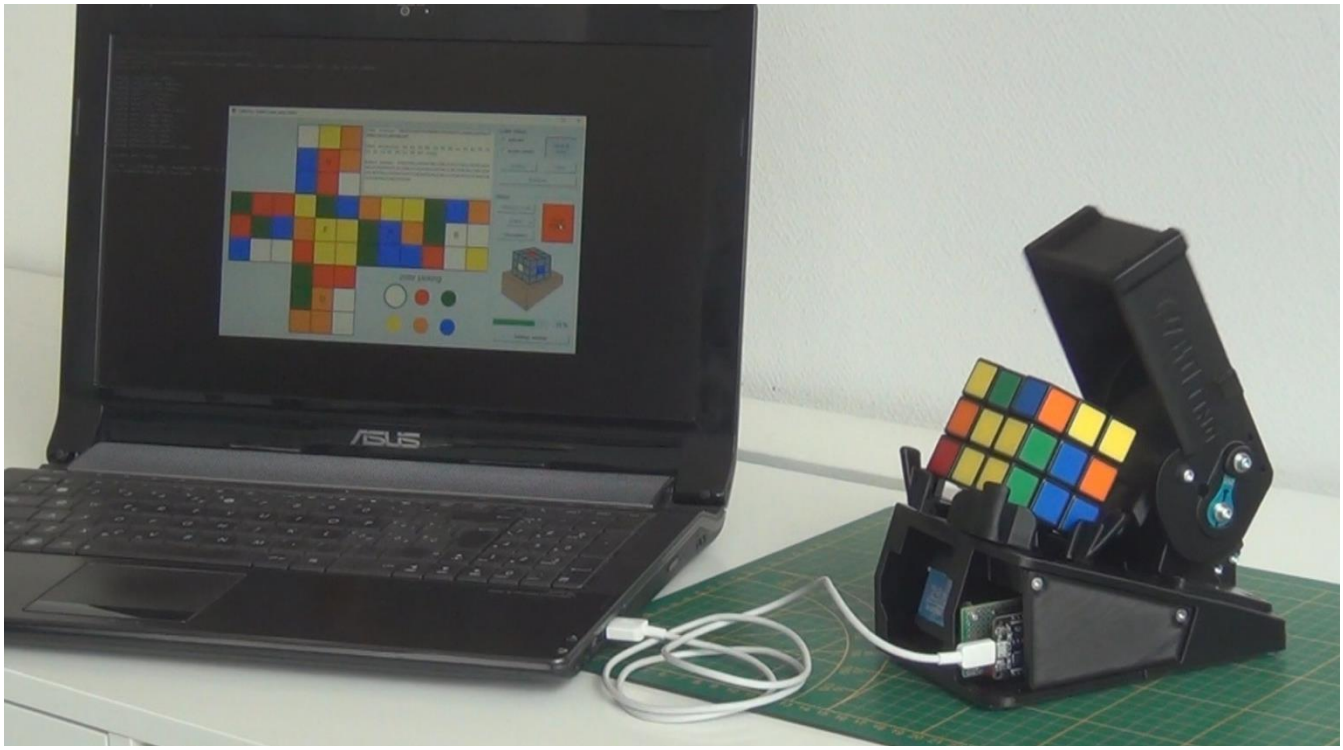
# CUBOTino: A small, simple, inexpensive Rubik's cube robot solver

https://www.instructables.com/CUBOTino-a-Small-Simple-3D-Printed-Inexpensive-Rub/

Andrea Favero, Groningen (NL)

30/06/2022 (check if a later update is available)

Robot (and script) demonstration at YouTube: https://youtu.be/ZVbVmCKwYnQ



This robot is meant to be very simple and inexpensive.

It is rather small (the base is about 160 x 100mm) and it does not require any special gripping toward the Rubik's cube.

It typically solves a scrambled cube in less than 90 seconds …. For sure not a fast robot; My previous Rubik's cube solver robot (https://youtu.be/oYRXe4NyJqs) is two time faster, yet four times more expensive 😊.

# Summary

# 1) Introduction:

To explain why I've started this project I've to shortly mention my previous Rubik's cube solver robot….
That one is based on a Raspberry pi 4B (2Gb ram) with a Picamera, it reads the cube status via the camera system, and it solves it: A full autonomous robot.
The complete process takes less than one minute, some references:

- How to make it: https://www.instructables.com/Rubik-Cube-Solver-Robot-With-Raspberry-Pi-and-Pica/
- Youtube: https://youtu.be/oYRXe4NyJqs

This robot works simply fine, I had lot of satisfaction from it, I learned a lot on different areas…..yet that robot has a clear drawback: The cost, as there are about 150euro of components.

# 2) Project scope:

Based on the introduction you probably can guess the project scope 😊
This second Rubik's cube solver robot project wants to be affordable, to attract more people and especially students into robotics and programming.
The overall idea is to build a scalable robot, based on a minimalist base version.

Project targets for this robot:

- The Base version must be as cheap as possible
- The mechanical part should be the same for all the versions
- The robot should be scalable (in automation, and consequently in complexity/materials cost)
- The robot should not require changes to the cube for gripping
- Compact design
- Fully 3D printable
- How to make it instructions and files
- Learning & Fun 😊

# 3) Robot name:

I've started this project with the idea to write and share the instructions, and along the way I thought a robot name would make the project more complete.
By combining CUBE, ROBOT and -INO, the Italian suffix for diminutives (to remark the small robot size), the chosen name is CUBOTino
By considering the Top cover, combined with the Lifter, has a "C" profile shape, then CUBOTino become:

# 4) Conclusions:

At this moment, the Base and Top versions are finished, below a high-level pros/cons summary:

Pros:

1. The new way to manoeuvre the cube has proved to be effective.
2. Robot dimensions are very small.
3. The same base mechanic works fine on these two versions.
4. Base version specific:
    a. is relatively cheap (about 30 to 40 euro of material, on early 2022, depending on where you live)
    b. GUI works simply fine
    c. Rather easy to set the robot parameters via the GUI
5. Top version specific:
    a. Raspberry Pi Zero 2 (512Mb ram) has proved to be sufficient for the computer vision, and all the required tasks….also for those not strictly needed, but nice to have.
    b. PiCamera works well despite the short distance from the cube.
    c. Cube status detection rather unsensitive to external light conditions.
    d. Despite the increased complexity, the electronic and wiring is still limited and simple.
    e. Power supply via two common 2A micro-USB phone chargers.

Cons:

1. In general, this is a slow robot; It was known since the start, yet…
2. Noise: Flipping the cube on the horizontal axis generates noise when the cube falls into the seat.
3. Base version specific:
    a. The single power supply hasn't worked well on other builds, suggesting the additional power input.
    b. Micropython documentation is somehow limited.
    c. Micropython documentation does not always correspond to real cases, for instance the PWM (on latest two official release) have much lower resolution than reported on the docs.
4. Top version specific:
    a. Connections board requires some more skills than the base version
    b. Total material cost is about 100 euro, on early 2022

# 5) Next steps:

Work out the Medium robot version, yet it won't be before winter 2022 😊

# 6) Safety:

Energize the robot only via USB ports having a class 2 insulation from the power supply net (laptops and PC normally have this safety feature).
Despite the robot mechanical force is limited, it has to be operated only under adults supervision.
If you build and use a robot, based on this information, you are accepting it is on your own risk.

# 7) Commitment:

If you read these instructions, there are chances you are interested on making this project, or to get some ideas on a sub part of it, or you're a curious person…

In any case, I hope the information provided will help you! If that's the case please consider to leave a message or a feedback or thumbs up on Youtube (https://youtu.be/ZVbVmCKwYnQ), or at the Instructable site.

In case you cannot find the solution by yourself (part that makes projects fun 😊), please drop a detailed question at the instructables site (https://www.instructables.com/CUBOTino-a-Small-Simple-3D-Printed-Inexpensive-Rub/).

I can't promise I'll be able to answer your questions, as well as I cannot commit to be fast in replying….

**Please feel free to provide your tips and feedback, on all areas: This will help me**

# 8) Models:

This project considers the robot to be scalable.
The idea is to develop three robot levels, by re-using the same mechanical part to manoeuvre the cube.

| Model | Type | Main directions | Status |
|---|---|---|---|
| **Base** | PC dependent, for cube status and cube solution | • Cube status entered on the GUI, via mouse or PC webcam<br>• Cube solution (Kociemba) generated at PC | Finalized<br>(April 2022 |
| **Medium** | PC dependent, for cube solution | • Cube status detection at the robot<br>• Cube solution (Kociemba solver) generated at PC | Stand-by, see notes below |
| **Top** | Autonomous | • Cube status detection at the robot, via a vision system<br>• Cube solution (Kociemba solver) generated at the robot | Finalized<br>(June 2022) |

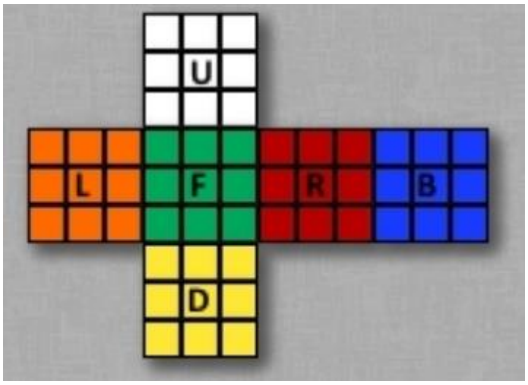
Notes:

The cube status detection method I've tried for the Medium version, is via 9 colours sensors (LDR+WS2812B leds), as explained at: https://fourboards.co.uk/rubix-cube-solving-robot
I've spent some time on this method, but the quality of my soldering was not a real success; I'm even doubting if this method really fits the overall project scope, to keep things simple.
I think the way to go is to use a ESP32-cam, in communication with the PC.

Considering the Top version is almost finished, and that I'm a bit tired with Rubik's cubes, I believe I won't (re)start the Medium version before coming winter…..
There might be good chances that someone will implement this solution better and faster that I'd do 😊

# 9) High level info (Base version):

1. To minimize the robot costs of this Base version, and to enhance DIY, the below aspects were considered:
    a. Use the PC computation power (Cube detection status, and cube solution).
    b. Use a PC USB port to communicates with the robot, via the microUSB port.
    c. Use a second microUSB connection, to energize the servos; This might be from the same PC, or a phone charger.
    d. Use two 180° servos for all the movements.
    e. Easy to find components.
    f. All parts 3D printable on a smaller printer.
2. The robot uses a ESP32 development board, flashed with micropython; This is used to "translate" the cube solution into robot movements, further than controlling servos and UART.
3. The interaction with the robot is made via a simple GUI, coded in python; Credits to Hegbert Kociemba who made available the base GUI (https://github.com/hkociemba/RubiksCube-TwophaseSolver), from which I've further built on.
4. Cube status must be entered on the GUI, by assigning the colours with the mouse to the cube sketch, or by manually presenting the six cube faces to a PC webcam; During the cube solving process, the microcontroller keeps the GUI informed on the progress, and the cube status (sketch) gets updated on the GUI.
5. Cube notations are from David Singmaster, limited to the uppercase (one "external layer rotation" at the time): https://en.wikipedia.org/wiki/Rubik%27s_Cube#Move_notation
6. Cube's orientation considers the Western colour scheme (https://ruwix.com/the-rubiks-cube/japanese-western-color-schemes/):
   The Western color sheme (also known as BOY: blue-orange-yellow) is the most used colour arrangement used not only on Rubik's Cubes but on the majority of cube-shaped twisty puzzles these days.
   Cubers who use this colour scheme usually start solving the Rubik's Cube with the white face and finish with the yellow; This colour scheme is also called **Minus Yellow** because if you add or extract yellow from any side you get its opposite.
   White + yellow = yellow
   red + yellow = orange
   blue + yellow = green




7. Cube solver uses the Hegbert Kociemba, "two-phase algorithm in its fully developed form with symmetry reduction and parallel search for different cube orientations", with an almost optimal target:
    - intro: https://www.speedsolving.com/threads/3x3x3-solver-in-python.64887/
    - Python script: https://github.com/hkociemba/RubiksCube-TwophaseSolver
8. During cube status detection via the PC webcam, the sequence URFDLB is suggested on screen for guidance.
9. This robot works with Rubik's cube size ranging from 56 to 57.5mm (those I've available); It might also work on cubes with slightly smaller size, by adjusting some parameters.
10. When rotations are express as CW, or CCW, it is meant by facing the related cube face. For the Cube_holder servo the same rule is applied: CW and CCW are meant from the motor point of view.
11. Cube's sides follow the URFDLB order, and facelets are progressively numbered according that order (sketch at side); Facelets numbers are largely used as key of the dictionaries.
12. Main parameters can be tuned via a Settings window at GUI.

| | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 0 | 1 | 2 | | | | | | |
| | | | 3 | 4 | 5 | | | | | | |
| | | | 6 | 7 | 8 | | | | | | |
| 36 | 37 | 38 | 18 | 19 | 20 | 9 | 10 | 11 | 45 | 46 | 47 |
| 39 | 40 | 41 | 21 | 22 | 23 | 12 | 13 | 14 | 48 | 49 | 50 |
| 42 | 43 | 44 | 24 | 25 | 26 | 15 | 16 | 17 | 51 | 52 | 53 |
| | | | 27 | 28 | 29 | | | | | | |
| | | | 30 | 31 | 32 | | | | | | |
| | | | 33 | 34 | 35 | | | | | | |

# 10) Construction:

The robot mechanical targets are:

1. solving a Rubik cube without changing it for special gripping
2. low cost
3. simplicity
4. compact design
5. fully 3D printable
6. limit the amount of different screw types

Construction principles:

1. The inclined cube-holder is inspired to Hans construction Tilted Twister 2.0 – LEGO Mindstorms Rubik's Cube solver – YouTube;
   This is a clever concept, as it allows to flip the cube around one of the horizontal axes by forcing a relatively small angle change (about 30 degrees, over the 20 degrees of the starting cube holder angle); Once the cube center of gravity is moved beyond the foothold, the cube falls on the following face thanks to the gravity force.
   Overall, it allows to flip the cube via a relatively small and inexpensive movement.
2. The Top-cover, combined with the cube Lifter, is the logical simplification step from my previous robot:
   - The Top-cover provides a constrainer for cube layer rotation, further than suspending sensors for the cube status detection, while keeping a compact robot construction.
   - The cube Lifter flips the cube around one of the horizontal axes.
   - Top-cover with integrated lifter is directly actuated by a servo, therefore controlled via angle.

   Overall, it allows to combine multiple functions in a relatively small space, parts quantity, and costs.
3. Cube-holder is mounted directly to a servo, therefore controlled via an angle.
4. All parts are made by 3D printing:
   - This makes possible to pursue the needed geometries, also complex shapes.
   - The biggest parts can still be printed on a relatively small plate (min plate 200x200 mm).
   - Some of the parts are split, mainly for easier, and better, 3D printing; Others are split for assembly reasons.
   - All the overhangs have been designed to enable 3D printing without support.

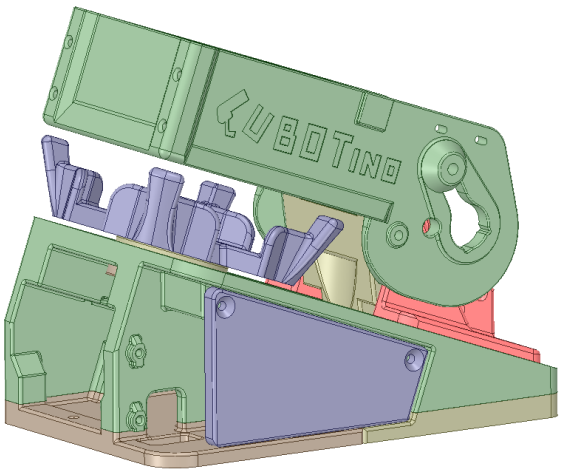
An impression of how the mechanic works https://www.youtube.com/embed/A3zE4B-gFzk?feature=oembed&autoplay=1

There are many different examples of Rubik's cube solver robot, based on two servos; I think most of them are variations from the one made by Matt's on 2014: https://hackaday.com/2014/06/28/rubiks-cube-solver-made-out-of-popsicle-sticks-and-an-arduino/
In these executions, the solution used to flip the cube on one of the horizontal axes, requires the main pivot (servo) to be placed rather far from the cube; This obviously increases a lot the overall dimensions.

# 11) 3D printed parts:

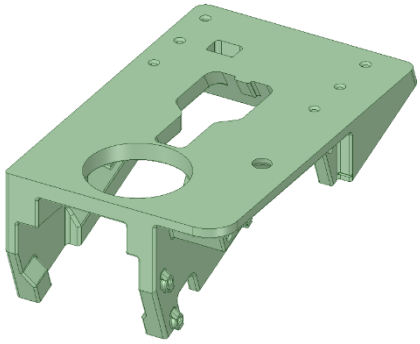
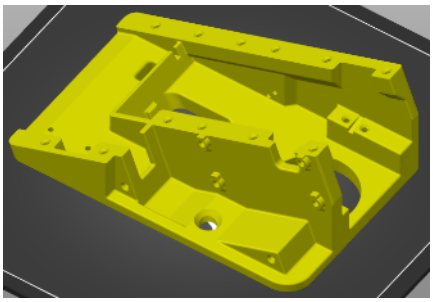
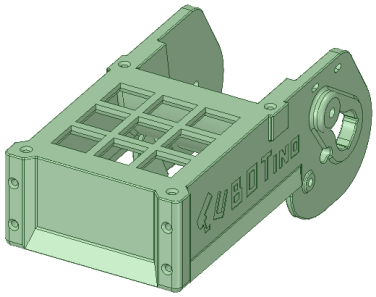
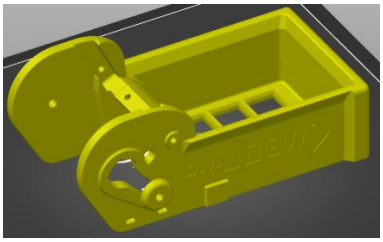
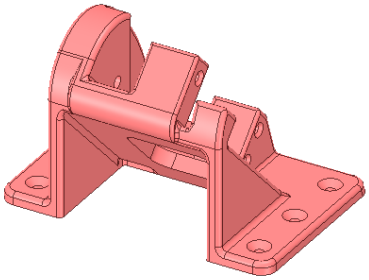
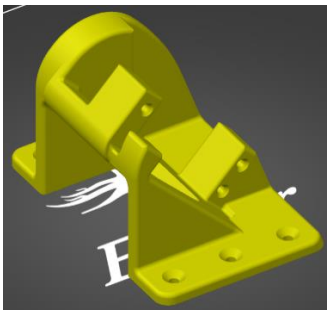
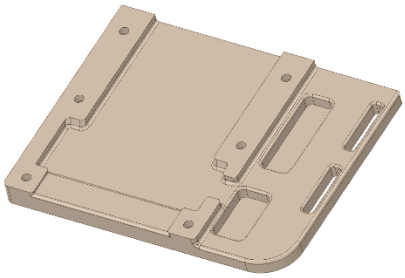
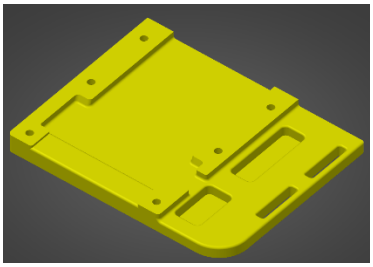
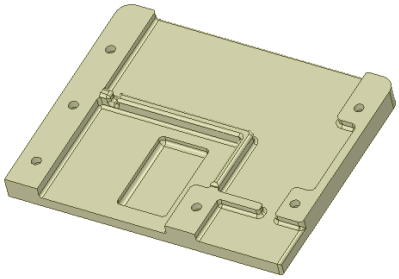
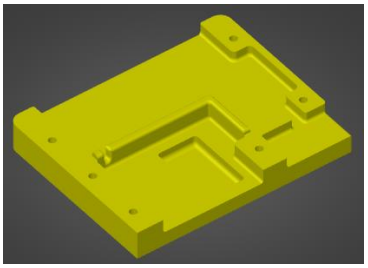
See notes below for filament quantity, and printing time …

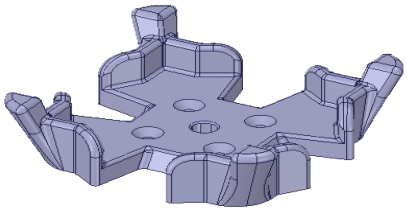
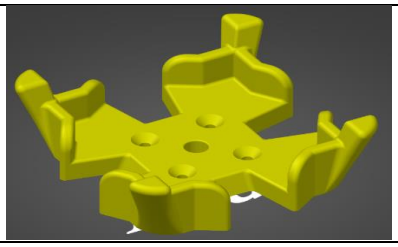
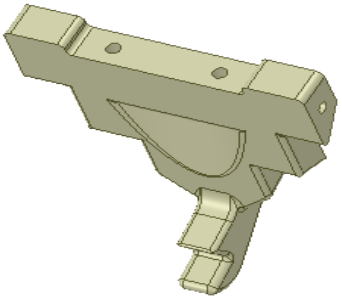
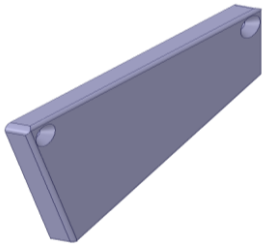
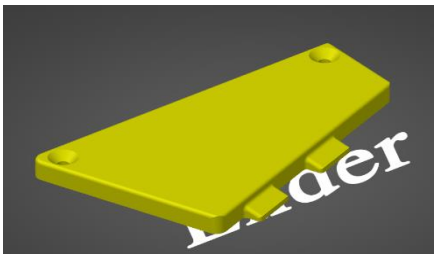
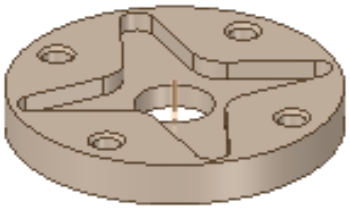
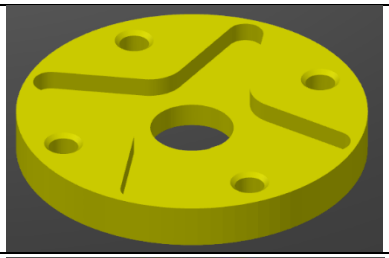
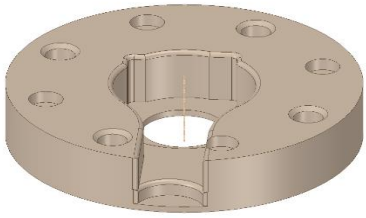
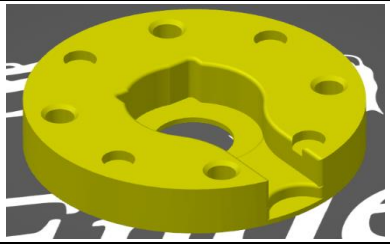
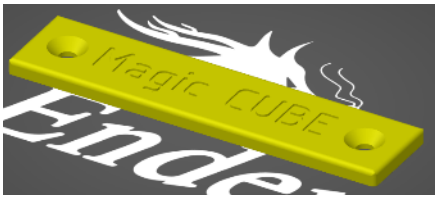
| Ref | Part | Filament | | Printing time |
|---|---|---|---|---|
| | | meters | grams | |
| 1 | Structure | 49 | 145 | 14h33m |
| **2** | Top_cover | 39,5 | 117 | 12h26m |
| **3** | Hinge | 17,2 | 51 | 5h37m |
| **4** | Baseplate front | 12,2 | 36 | 3h33m |
| **5** | Baseplate back | 12,6 | 37,3 | 3h36m |
| **6** | Cube_holder | 11,6 | 34,4 | 3h36m |
| **7** | Cube_lifter | 5,5 | 16,2 | 1h48m |
| **8** | PCB_cover | 5,1 | 15 | 1h23m |
| **9** | Servo_axis_sup (or its alternative) | 2,4 | 7,2 | 0h55m |
| **10** | Servo_axis_inf (or its alternative) | 2,4 | 7 | 0h52m |
| **11** | (optional) Personaliz_plate | 1.5 | 5 | 0h30 |
| | **TOTAL** | **158m** | **466g** | **48h20m** |



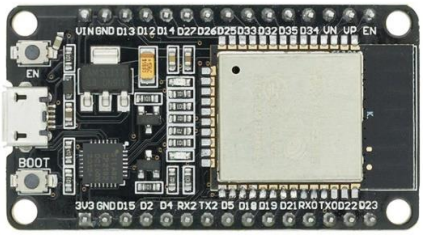
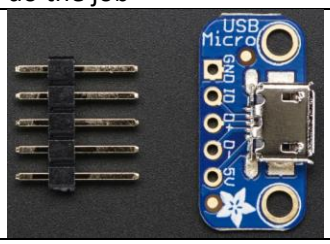

Notes:

1. The biggest part is the Structure, and it easily fits on printers with plate size of 200x200mm.
2. Filament length is based on Ø1.75mm.
3. Filament weight is based on PETG density (1.23g/mm³), and printing settings I've use on my Ender 3 printer, for accurate result:
    - 0.2mm layers
    - Low speed (between 25 to 40mm/s for the external parts and 1st layer)
    - 4 layers on vertical walls
    - 5 layers on horizontal walls
    - 30% filling
    - 8mm brim
4. The filament quantity, and printing time, in the table should be considered as an upper limit. After printing, the parts total weight is closer to 400grams than 466grams estimated by the slicer SW.
5. All parts have been designed to be printed without supporting the overhangs.
6. Some parts have been split, for easier and better 3D printing.
7. The suggested part orientation for the 3D print is showed on below Table.
8. The stl files are available on below websites:
    - https://www.instructables.com/CUBOTino-a-Small-Simple-3D-Printed-Inexpensive-Rub/
    - https://www.thingiverse.com/thing:5342368

| Part name | image | 3D print orientation |
| --- | --- | --- |
| Structure |  |  |
| Top_cover |  |  |
| Hinge |  |  |
| Baseplate front |  |  |
| Baseplate rear |  |  |

| | | |
|---|---|---|
| Cube_holder |  |  |
| Cube_lifter |  |  |
| PCB_cover |  |  |
| Servo_axis_sup (or its alternative) |  | Symmetrical |
| Servo_axis_inf |  |  |
| Alternative Servo_axis_inf |  |  |
| Personaliz_plate<br><br>Or<br><br>Personaliz_plate01 |  <br>  |  |

# 12) Bought parts:

| Q.ty | Part | link to the shop I used | Cost (euro) | Notes |
|---|---|---|---|---|
| 1 | ESP32 30 pins development board | https://www.aliexpress.com/item/32864722159.html?gatewayAdapt=glo2ita&gatewayAdapt=glo2ita&gatewayAdapt=glo2ita&spm=a2g0o.9042311.0.0.27424c4dBteSIp | 4.5 |  |
| 2 | Servo TD-8325MG (180deg 25Kg metal) and metal arm “25T” | https://www.aliexpress.com/item/32298149426.html?gatewayAdapt=glo2ita&spm=a2g0o.9042311.0.0.5d1e4c4d14Qjaz | 25 (2 servos + 2 arms) |   |
| <500g | Filament 1.75mm | | ~10 | Suggested PETG, yet other material will do the job |
| 1 | USB MICRO-B BREAKOUT BOARD | https://www.adafruit.com/product/1833 | 1.5 |   |

## Electrical small parts:

| Q.ty | Part | Notes |
|---|---|---|
| 1 | Prototype board | To make easier the ESP32 placement on the robot, as well as the connections |
| 2x15 | Female Headers | To connect the ESP32 to the support board |
| 3x3 | Male Headers | To connect the servos and the touch pad cable to the support board |
| 2 | Capacitor 16V 220uF | To prevent voltage drop when servos are activated |

## Screws:

| Quantity | Dimension | Head type |
|---|---|---|
| 1 | M4x20 | Cylindrical |
| ~ 20 | M3x12 | Cylindrical |
| ~30 | M3x12 | Conical |
| 4 | M2.5x10 | Cylindrical |

Touch pad:

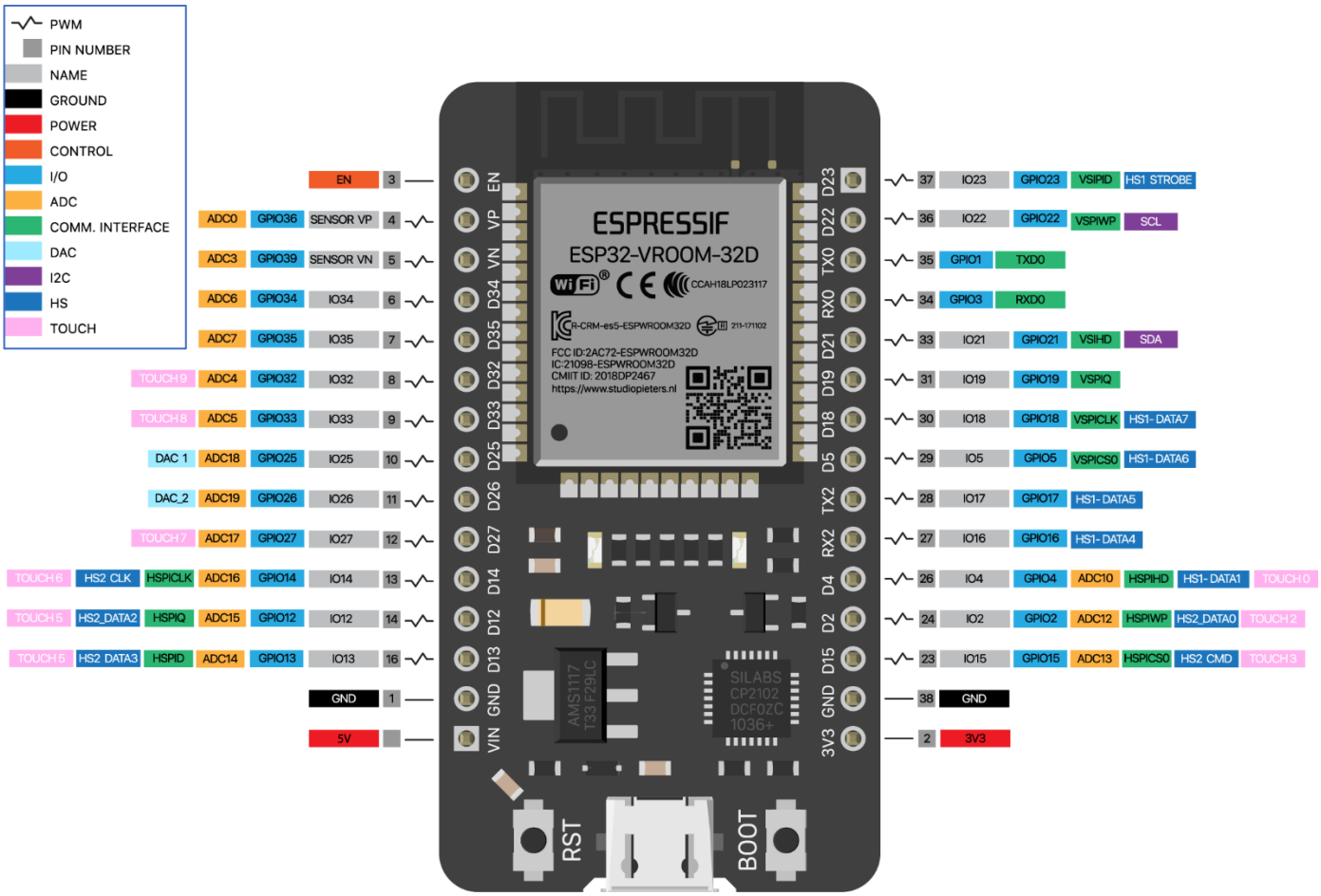
| Q.ty | Part | Notes |
|---|---|---|
| 1 | Piece of metal sheet (16x75mm, 1.5 to 2mm thickness) | Aluminium is easy to work with.<br>Alternatively, bare wire in between the two screws will also do the job |

In case you're not going to use this function, you might consider to print a personalization cover for that recess.

Off course some other common materials are needed:

- 1 x USB-Microusb cable with data lines
- 1 x USB-Microusb cable (with or without data lines)
- Eventual phone charger (5V 2A, not of the smart type), if the current drawn by the servos affects your PC.
- wires, solder and solder device, tire wraps, self-adhesive rubber feet, etc.

# 13) ESP32 dev. board (30 pin) pinout:




Used pins:

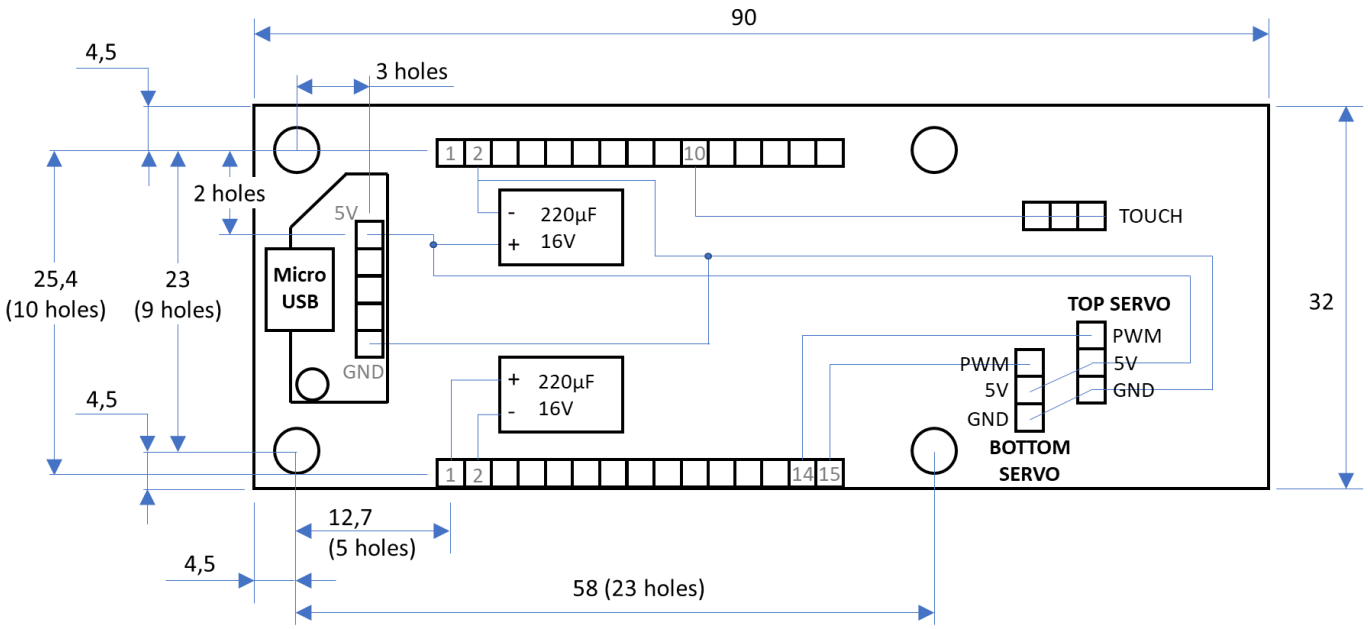
| Pin | Purpose |
|---|---|
| D22 | Cube Holder servo |
| D23 | Top cover servo |
| D32 | Touch plate for robot stop |

Notes:

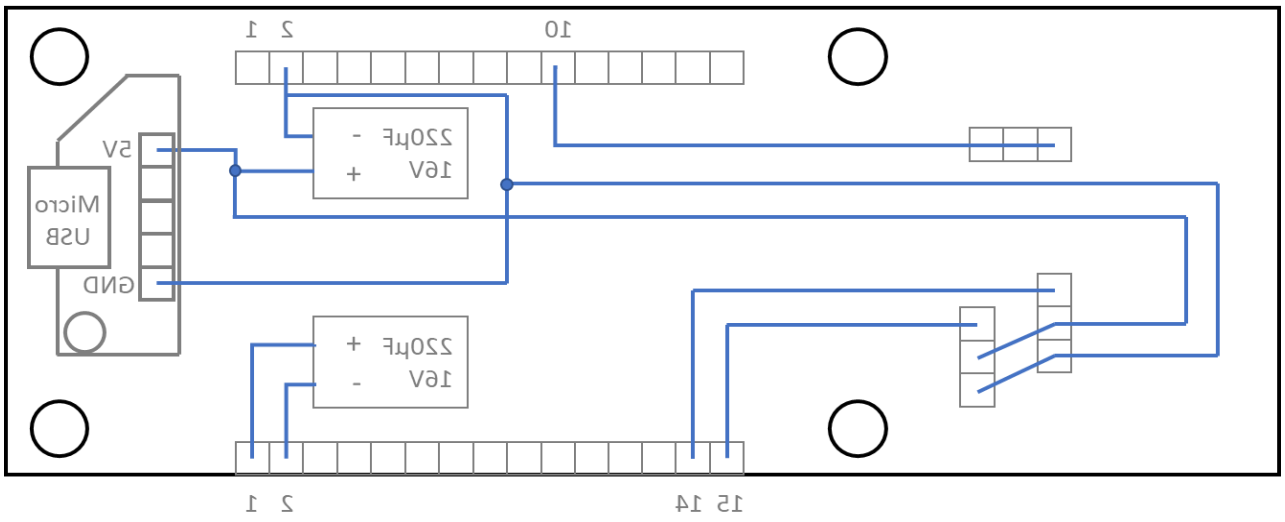
Added two capacitors (16V 220uF) at Vin and 3V3 vs GND

# 14) Connections board:

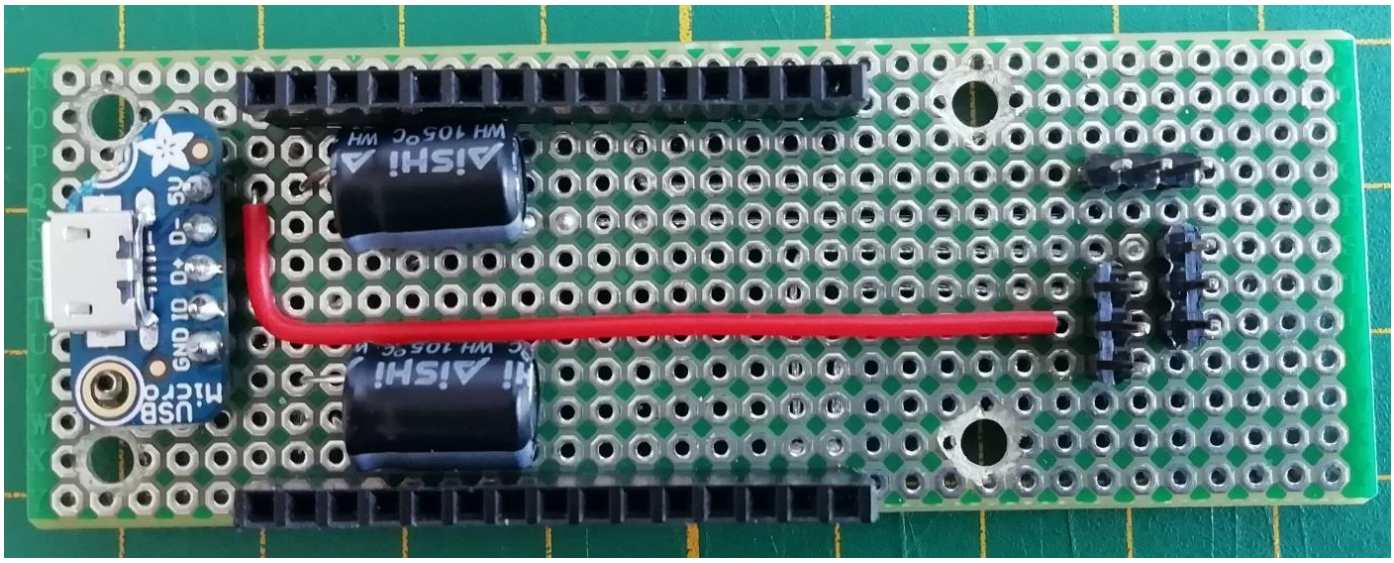
## Top view:

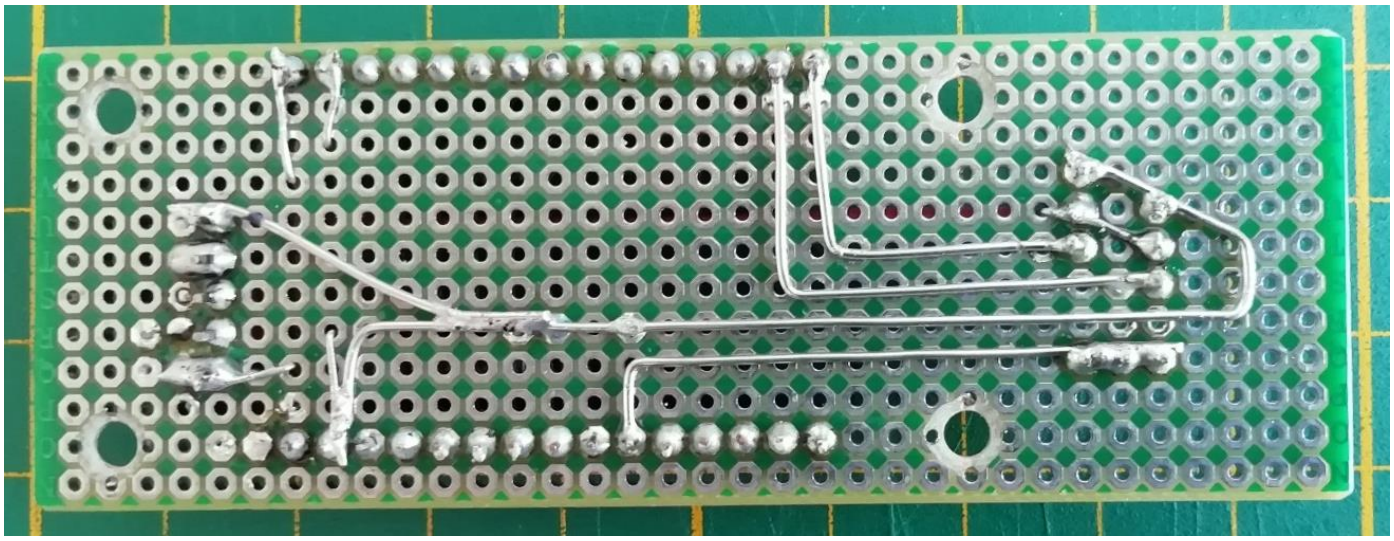


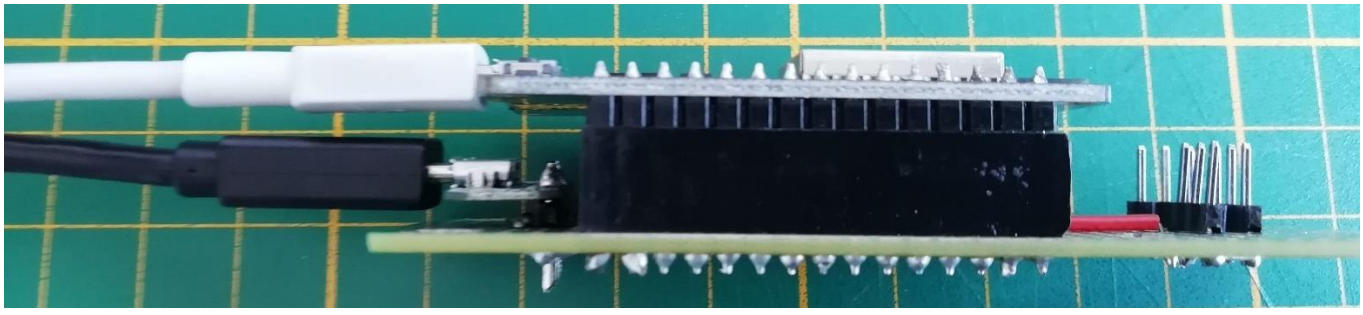

## Bottom view:



Before soldering, cut the corner of the microUSB breakout board in front of the connections board hole



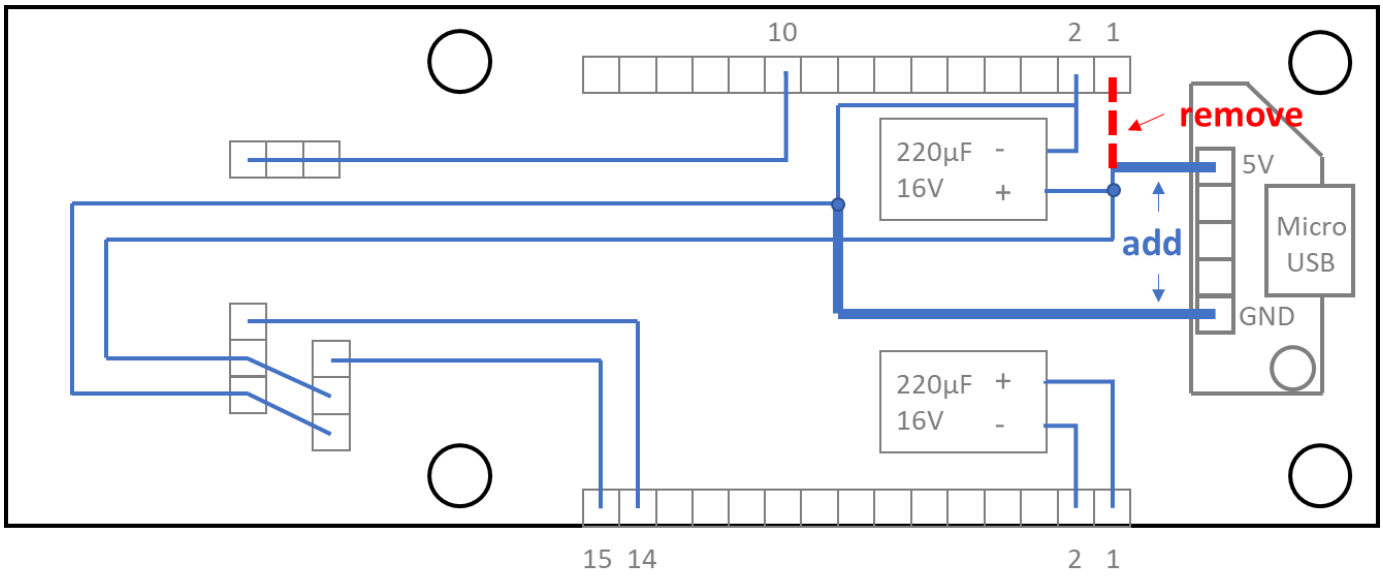





**How to update the Connection board, eventually made before 11 June 2022 (servo power supply from ESP32):**

1. Remove the (red dashed) connection reaching the Pin1 of ESP32.
2. Connect it instead to the 5V pin of the microUSB breakout board (thick blue line).
3. Connect the GND pin of the microUSB breakout board to the line from pin2 of ESP32 that goes to the GND pin of the servos connectors (thick blue line).

Bottom view:

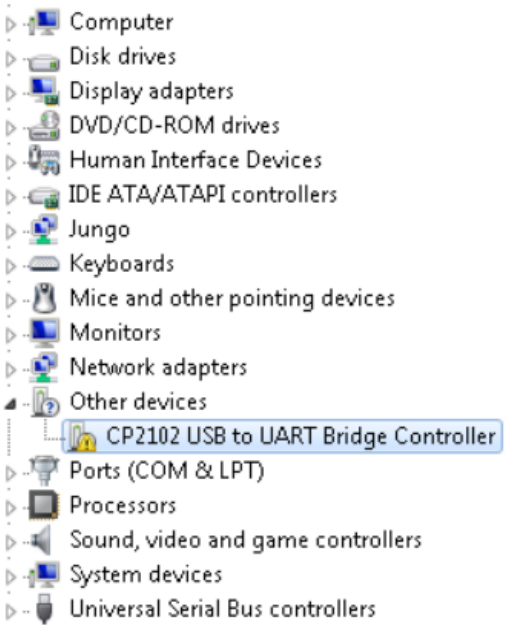


# 15) Setting up the ESP-32 microcontroller:

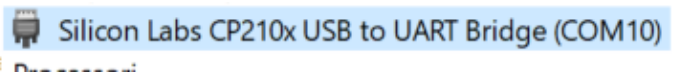
Step1: Before starting, do check if you've the driver for USB to UART communication installed:

To communicate with ESP32 board, via USB, a CP210X driver for the USB to UART communication is needed
Drivers are likely already available in your computer OS, if not:

1) Connect the ESP32 board via a USB cable
2) Go to https://www.silabs.com/developers/usb-to-uart-bridge-vcp-drivers and download the driver for you OS
3) For Windows OS, open Device manager and in case the driver is missed it should look like:




4) Right click on CP2102 device, select Update Driver Software…, navigate to the folder with unzipped driver files, confirm
5) With installed/updated drivers, the connection should look like:




6) The COM number might be different, at it depends from the system you have

Step2: Download Micropython firmare for the ESP32;

At https://micropython.org/download/, select generic ESP32 dev board


Step3: Download Micropython firmware v1.17 (2021-09-02)

(PWM function at 50Hz wasn't working on the latest release, V1.18…)


A binary file will be saved on your downloads folder

Step4: Flash the firmware to the ESP32

I've used Thonny, and it's very easy to get the ESP32 up and running with Micropython.

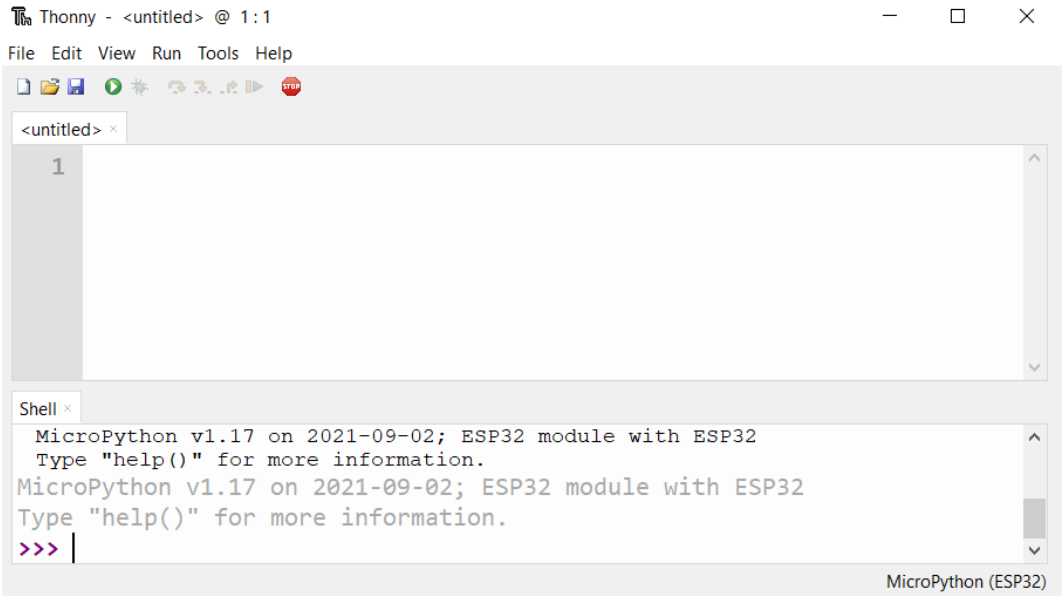
Good guide for installing micropython on ESP32: https://microcontrollerslab.com/getting-started-thonny-micropython-ide-esp32-esp8266/

If the firmware installation went well, on the REPL should appear: "MicroPython v1.17 on 2021-09-02; ESP32 module with ESP32"

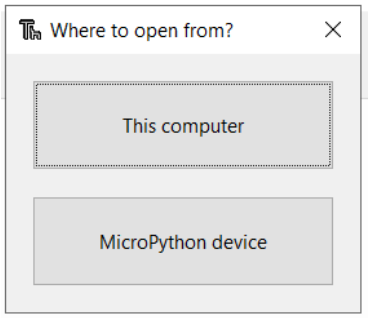
Type *esp.flash_size()* to check the flash memory, that should be around: 4194304

Step5: Save the (below listed) files on the microcontroller.

When Micropython is installed, and the board energized, by launching Thonny the below window appears:




A the File Open request below window appears, meaning Thonny can open files from both the PC and the ESP32 microntroller:




Choose *This computer*, and one by one open the below files:

- main.py
- Cubotino_moves.py
- Cubotino_servos.py
- Cubotino_settings.txt (select *all files (*.*)* option to be able to see the .txt file)

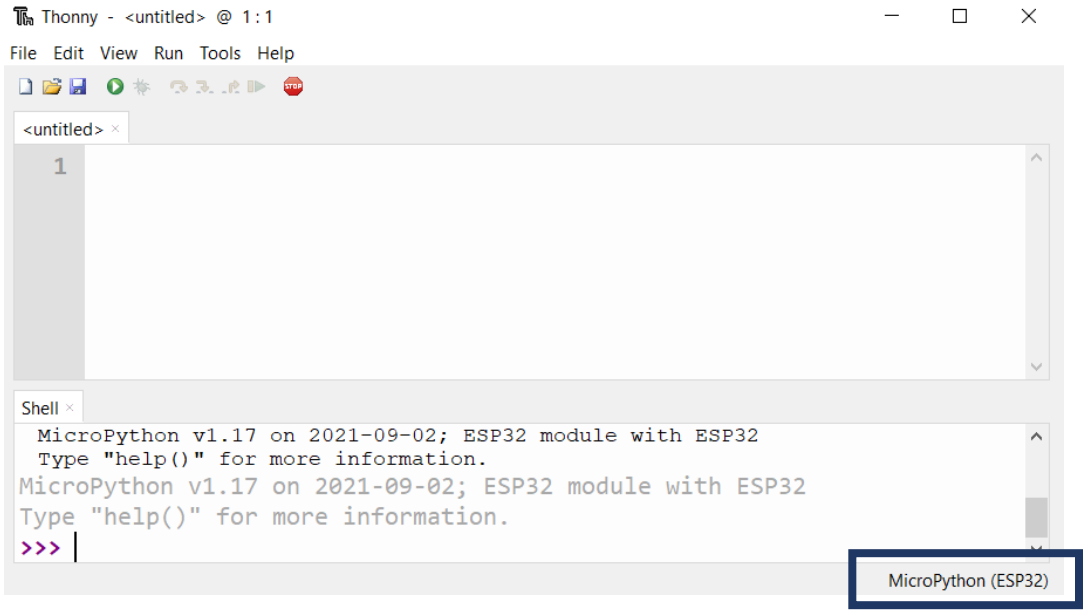
Save the above files, by using the *Save as* command, and by selecting the *MicroPython device*.

Notes:

1. files names should not be changed, as these are the reference for module import.
2. File extension ( .py and .txt) has to be manually entered when saving the files; When the file ends in ".py", the python icon will be associated:

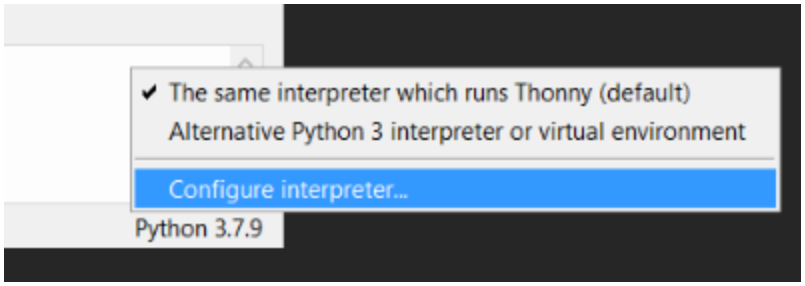
Step6 : If Thonny is your preferred IDE for Python, and you'd like to run Cubotino_GUI.py to test it, you've to change the interpreter from MicroPython back to Python.

On the Thonny bottom-right corner is it always shown which interpreter is active





In most of the cases, the interpreter for Python is the default one.
Interpreter selection can be done by left click on the bottom-rigth corner, Configure interpreter….



# 16) Files needed at PC

Python is needed for the below applications (it is suggested to make a virtual environment in Python):

| File | Purpose | Notes |
|---|---|---|
| …….. | Kociemba solver | This is made by a dozen of python files, further than table; See below. |
| Cubotino_GUI.py | GUI to interact with the robot and the other modules | |
| Cubotino_Webcam.py | Application to detect the cube status via a webcam | |
| Cubotino_moves.py | To generate the robot moves | |

Below the required libraries, that aren't part of normal Python distributions:

| Library | Main scope | Notes |
|---|---|---|
| pyserial | To manage serial communication | pyserial==3.5 |
| cv2 | For the vision part associated to the webcam usage | cv2.__version__ '4.5.3' |
| solver | Kociemba solver for the (almost optimal) cube solution | https://github.com/hkociemba/RubiksCube-TwophaseSolver<br>See Kociemba solver installation chapter |

Other files to be copied, into the folder where the Kociemba solver is located:

| Files | Main scope | Notes |
|---|---|---|
| Cubotino_webcam.py | To detect the cube status via webcam | |
| Cubotino_moves.py | To update the cube sketch during robot solving | |
| Cubotino_cam_settings.txt | Text file with robot related settings | |
| Cubotino_settings.txt | Text file with webcam related settings | |
| Rubiks-cube.ico | Icon for the GUI window | |

The project has been developed / tested with:

- Windows (W10) with Python versions:
  - 3.9.7 |packaged by conda-forge| [MSC v.1916 64 bit (AMD64)]
  - 3.8.12 [MSC v.1916 64 bit (AMD64)]
  - 3.8.13 (default, Mar 28 2022, 06:59:08) [MSC v.1916 64 bit (AMD64)]
- OpenCV cv2 ver: 4.5.1 and 4.5.3
- Thonny 3.3.13

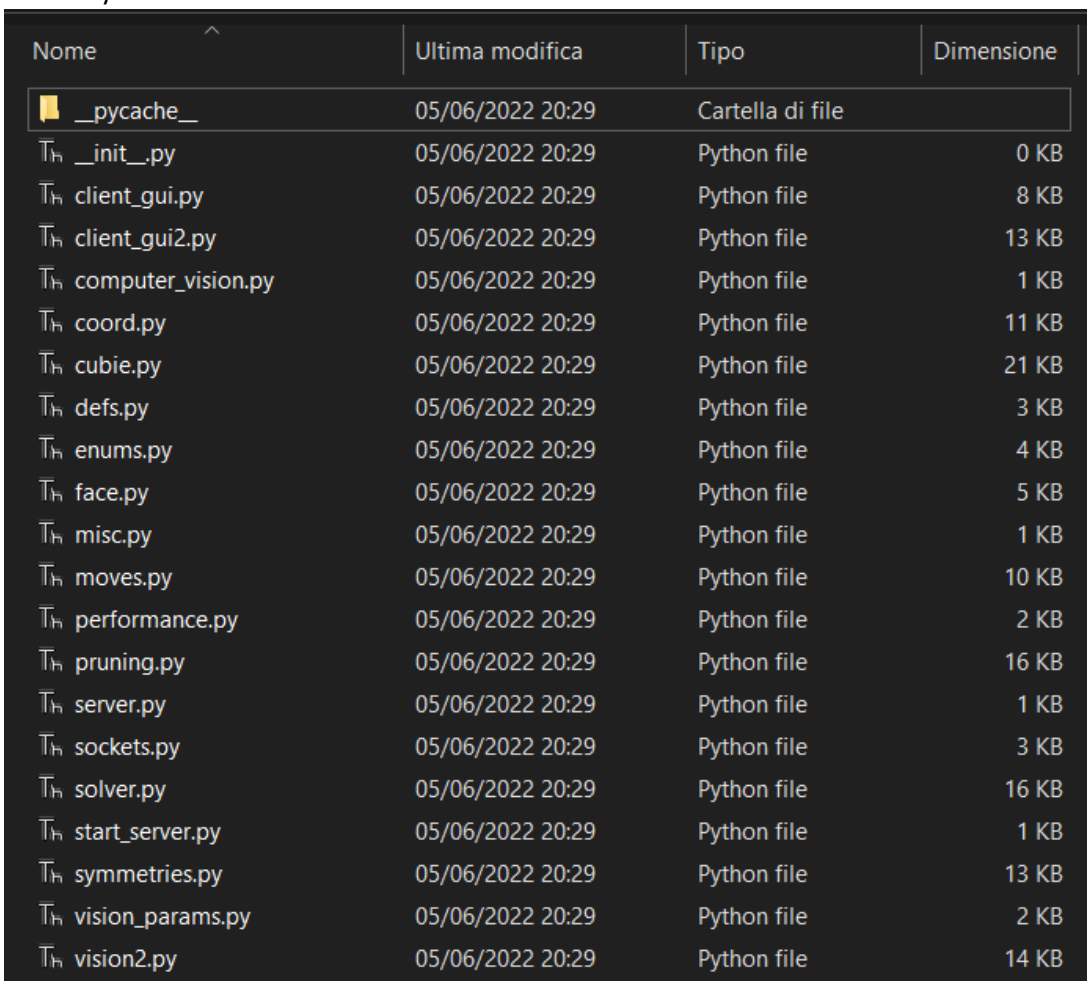
# 17) Kociemba solver installation

RubiksCube-TwophaseSolver is the great solver developed by Mr. Hergbert Kociemba (https://github.com/hkociemba/RubiksCube-TwophaseSolver)

There are few options to install the solver, and to make it working for this robot; Below a couple of possibilities. The first method uses pip command to install the solver, the second one copy the python files into the folder made for the robot.

Method A)

1) If you've made a Python venv for this project, activate that virtual environment.
2) Enter the eventual folder where you want to have the robot related files.
3) From the terminal *pip install RubikTwoPhase*
4) Below Python files will be installed:



| Nome | Ultima modifica | Tipo | Dimensione |
|---|---|---|---|
| __pycache__ | 05/06/2022 20:29 | Cartella di file | |
| __init__.py | 05/06/2022 20:29 | Python file | 0 KB |
| client_gui.py | 05/06/2022 20:29 | Python file | 8 KB |
| client_gui2.py | 05/06/2022 20:29 | Python file | 13 KB |
| computer_vision.py | 05/06/2022 20:29 | Python file | 1 KB |
| coord.py | 05/06/2022 20:29 | Python file | 11 KB |
| cubie.py | 05/06/2022 20:29 | Python file | 21 KB |
| defs.py | 05/06/2022 20:29 | Python file | 3 KB |
| enums.py | 05/06/2022 20:29 | Python file | 4 KB |
| face.py | 05/06/2022 20:29 | Python file | 5 KB |
| misc.py | 05/06/2022 20:29 | Python file | 1 KB |
| moves.py | 05/06/2022 20:29 | Python file | 10 KB |
| performance.py | 05/06/2022 20:29 | Python file | 2 KB |
| pruning.py | 05/06/2022 20:29 | Python file | 16 KB |
| server.py | 05/06/2022 20:29 | Python file | 1 KB |
| sockets.py | 05/06/2022 20:29 | Python file | 3 KB |
| solver.py | 05/06/2022 20:29 | Python file | 16 KB |
| start_server.py | 05/06/2022 20:29 | Python file | 1 KB |
| symmetries.py | 05/06/2022 20:29 | Python file | 13 KB |
| vision_params.py | 05/06/2022 20:29 | Python file | 2 KB |
| vision2.py | 05/06/2022 20:29 | Python file | 14 KB |

5) Installation folder depends on your system, yet it is expected to be in your Python Interpreter PATH. To know the PATH:
   a. From the venv run python: *python*
   b. Import a couple of libraries: *Import os, sys*
   c. Check for the PATH: *os.path.dirname(sys.executable)*

   The interpreter PATH is returned
   in my case it returns 'C:\\Users\\andre\\anaconda3', meaning the Kociemba solver is installed at 'C:\Users\andre\anaconda3\Lib\site-packages\twophase'

6) Check the solver functionality

The first time the solver is imported, it generates all the lookup tables; The time needed depends on the PC hardware, nowadays should be 30 to 60 minutes.

a. If you've made a Python venv for this project, activate that virtual environment
b. Enter the eventual folder where you want to have the robot related files
c. Launch python: *python*
d. Import the solver: *import twophase.solver as sv*

The solver will start the tables generation; These files are expected to be in 'twophase' folder, located into the folder you've chosen.

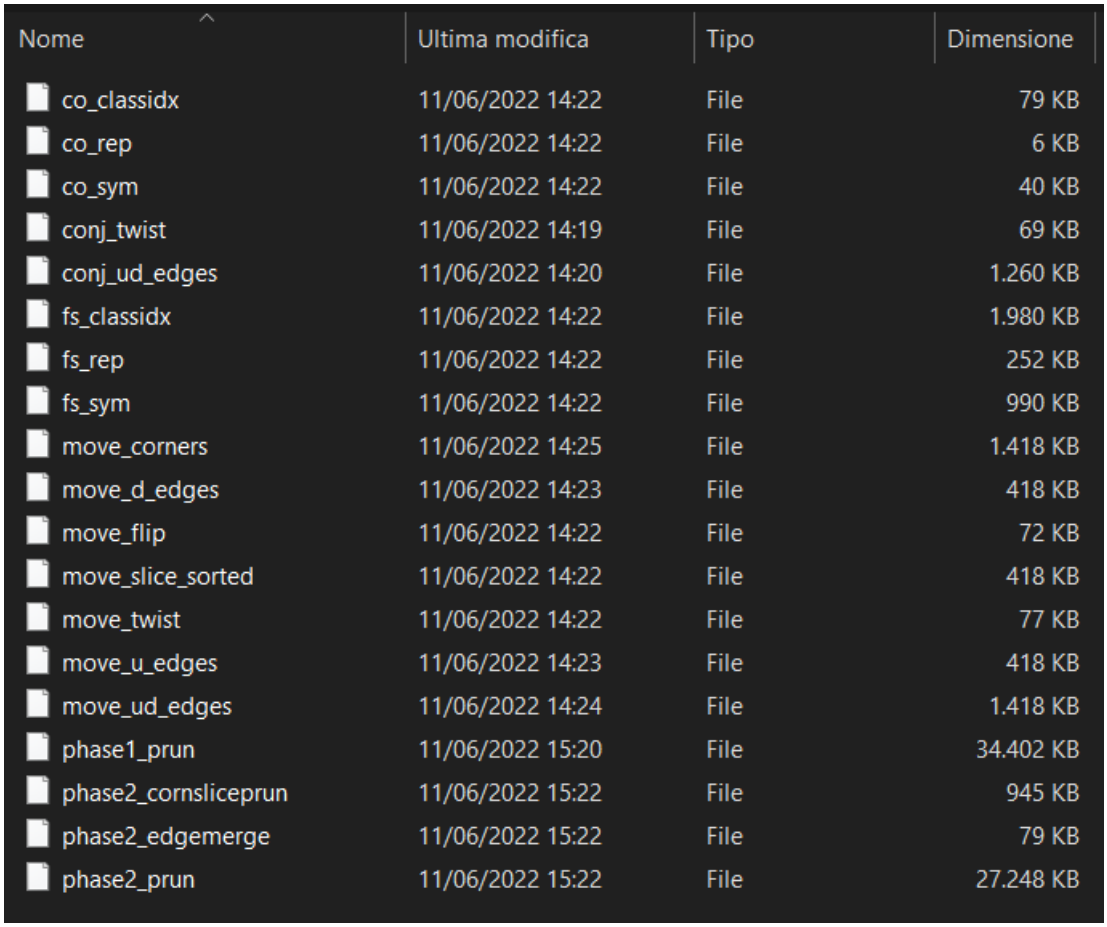
```
(base) C:\Users\andre>cd python\cube

(base) C:\Users\andre\Python\cube>pip install RubikTwophase
Collecting RubikTwophase
  Using cached RubikTwoPhase-1.0.9-py3-none-any.whl (53 kB)
Installing collected packages: RubikTwophase
Successfully installed RubikTwophase-1.0.9

(base) C:\Users\andre\Python\cube>python
Python 3.8.13 (default, Mar 28 2022, 06:59:08) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import twophase.solver  as sv
On the first run, several tables will be created. This takes about 1/2 hour or longer (depending on the hardware).
All tables are stored in C:\Users\andre\Python\cube\twophase

creating conj_twist table...
creating conj_ud_edges table...
...................
```

```
depth: 12 done: 140908410/140908410
creating phase2_prun table...
..
depth: 0 done: 1/111605760
.....................................................................
depth: 1 done: 4/111605760
.....................................................................
depth: 2 done: 14/111605760
.....................................................................
depth: 3 done: 66/111605760
.....................................................................
depth: 4 done: 351/111605760
.....................................................................
depth: 5 done: 1669/111605760
.....................................................................
depth: 6 done: 7340/111605760
.....................................................................
depth: 7 done: 33842/111605760
.....................................................................
depth: 8 done: 149309/111605760
.....................................................................
depth: 9 done: 620155/111605760
.....................................................................
depth: 10 done: 2473211/111605760
remaining unfilled entries have depth >=11
creating phase2_cornsliceprun table...
..............................................
creating phase2_edgemerge table...
..................

>>>
```

When the 'phase2_prun' table is completed (total process ca 60minutes in my case), if everything went well, the python prompt is presented.

e. Exit python: *exit()*

7) Check the presence and location of the lookup table files:
At the 'twophase' folder, located at the folder you've created for the robot, the below listed files (lookup tables for the solver) should now be available:



| Nome | Ultima modifica | Tipo | Dimensione |
|---|---|---|---|
| co_classidx | 11/06/2022 14:22 | File | 79 KB |
| co_rep | 11/06/2022 14:22 | File | 6 KB |
| co_sym | 11/06/2022 14:22 | File | 40 KB |
| conj_twist | 11/06/2022 14:19 | File | 69 KB |
| conj_ud_edges | 11/06/2022 14:20 | File | 1.260 KB |
| fs_classidx | 11/06/2022 14:22 | File | 1.980 KB |
| fs_rep | 11/06/2022 14:22 | File | 252 KB |
| fs_sym | 11/06/2022 14:22 | File | 990 KB |
| move_corners | 11/06/2022 14:25 | File | 1.418 KB |
| move_d_edges | 11/06/2022 14:23 | File | 418 KB |
| move_flip | 11/06/2022 14:22 | File | 72 KB |
| move_slice_sorted | 11/06/2022 14:22 | File | 418 KB |
| move_twist | 11/06/2022 14:22 | File | 77 KB |
| move_u_edges | 11/06/2022 14:23 | File | 418 KB |
| move_ud_edges | 11/06/2022 14:24 | File | 1.418 KB |
| phase1_prun | 11/06/2022 15:20 | File | 34.402 KB |
| phase2_cornsliceprun | 11/06/2022 15:22 | File | 945 KB |
| phase2_edgemerge | 11/06/2022 15:22 | File | 79 KB |
| phase2_prun | 11/06/2022 15:22 | File | 27.248 KB |

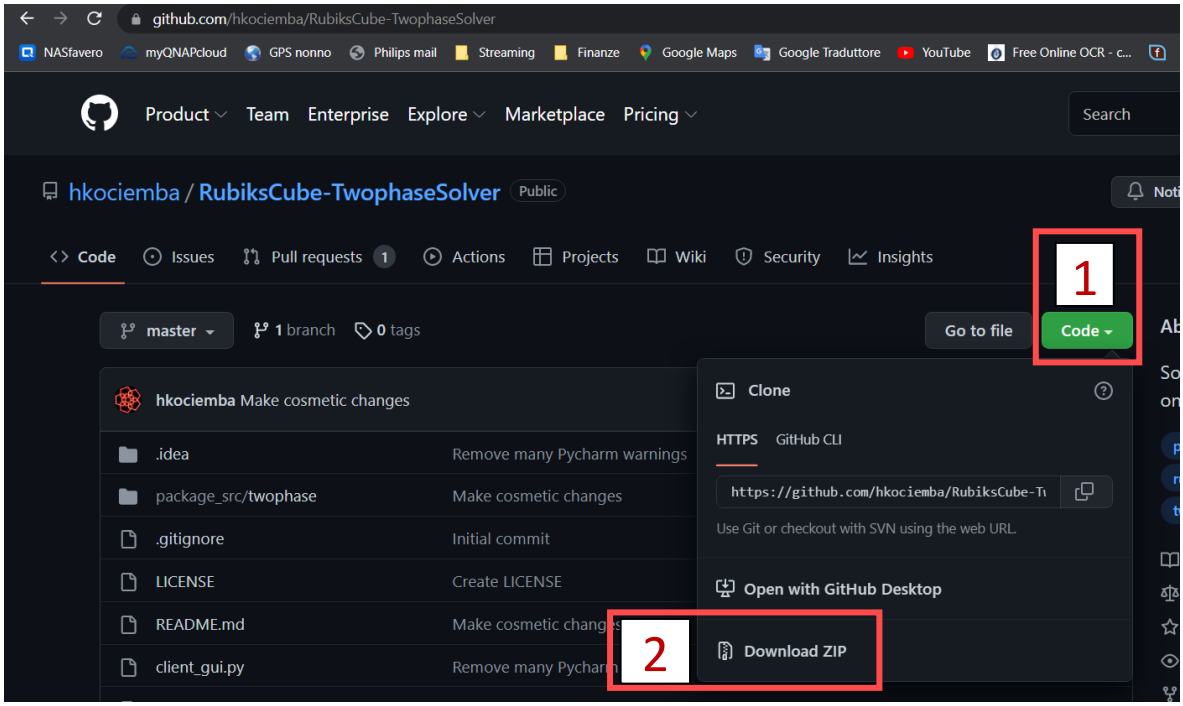
In my case these files are located at: C:\Users\andre\Python\cube\twophase

8) Run *python Cubotino_GUI.py* to check if everything works fine.
When the Kociemba solver is installed, with this first method, the below print will be made at the Cubbotino_GUI.py start:

```
Running on: Windows
Python version: 3.8.13 (default, Mar 28 2022, 06:59:08) [MSC v.1916 64 bit (AMD64)]
CV2 version: 4.5.1
webcam module AF (11 June 2022)

loading conj_twist table...
loading conj_ud_edges table...
loading flipslice sym-tables...
loading corner sym-tables...
loading move_twist table...
loading move_flip table...
loading move_slice_sorted table...
loading move_u_edges table...
loading move_d_edges table...
loading move_ud_edges table...
loading move_corners table...
loading phase1_prun table...
loading phase2_prun table...
loading phase2_cornsliceprun table...
imported the installed twophase solver
```

Method B)

1) Make/Enter the eventual folder where you want to have the robot related files.
2) Into that folder, make a sub-folder named ‘twophase’.
3) Go to https://github.com/hkociemba/RubiksCube-TwophaseSolver
4) Select *Code*, *Download ZIP*




5) Unzip the content into the ‘twophase’ folder you’ve made on step 2; This means the below python files will be located into the ‘twophase’ folder:

| Nome | Ultima modifica | Tipo | Dimensione |
|---|---|---|---|
| __init__.py | 11/06/2022 14:18 | Python file | 0 KB |
| client_gui.py | 11/06/2022 14:18 | Python file | 8 KB |
| client_gui2.py | 11/06/2022 14:18 | Python file | 13 KB |
| computer_vision.py | 11/06/2022 14:18 | Python file | 1 KB |
| coord.py | 11/06/2022 14:18 | Python file | 11 KB |
| cubie.py | 11/06/2022 14:18 | Python file | 21 KB |
| defs.py | 11/06/2022 14:18 | Python file | 3 KB |
| enums.py | 11/06/2022 14:18 | Python file | 4 KB |
| face.py | 11/06/2022 14:18 | Python file | 5 KB |
| misc.py | 11/06/2022 14:18 | Python file | 1 KB |
| moves.py | 11/06/2022 14:18 | Python file | 10 KB |
| performance.py | 11/06/2022 14:18 | Python file | 2 KB |
| pruning.py | 11/06/2022 14:18 | Python file | 16 KB |
| server.py | 11/06/2022 14:18 | Python file | 1 KB |
| sockets.py | 11/06/2022 14:18 | Python file | 3 KB |
| solver.py | 11/06/2022 14:18 | Python file | 16 KB |
| start_server.py | 11/06/2022 14:18 | Python file | 1 KB |
| symmetries.py | 11/06/2022 14:18 | Python file | 13 KB |
| vision_params.py | 11/06/2022 14:18 | Python file | 2 KB |
| vision2.py | 11/06/2022 14:18 | Python file | 14 KB |

In my case these files would be located at: C:\Users\andre\Python\cube\twophase

6) Enter the robot related files; This should be one level upper the ‘twophase’ folder.

7) Check the solver functionality:

The first time the solver is imported, it generates all the lookup tables; The time needed depends on the PC hardware, nowadays should be 30 to 60 minutes.
Launch python: *python*
Import the solver: *import twophase.solver as sv*
The solver will start the tables generation; These files are expected to be in ‘twophase’ folder, located into the folder you’ve chosen.

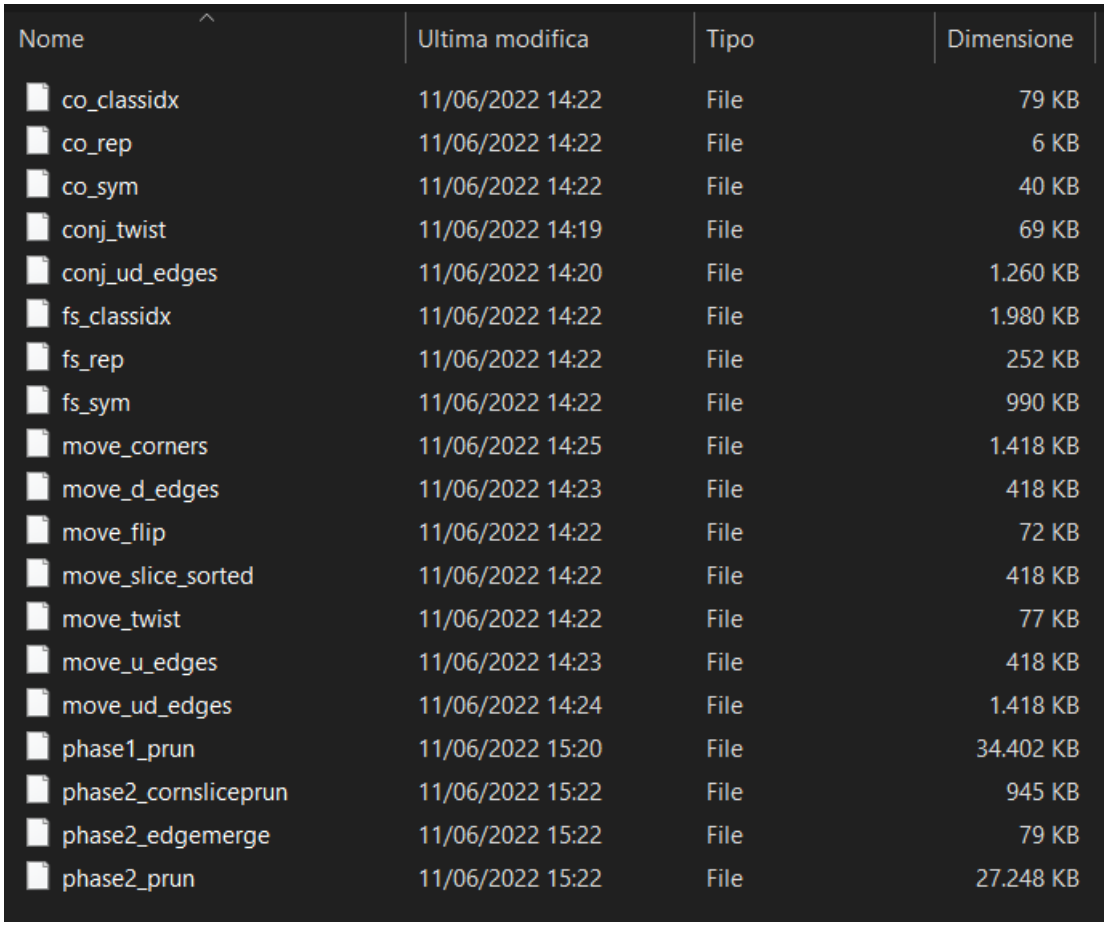
```
(base) C:\Users\andre>cd python\cube

(base) C:\Users\andre\Python\cube>pip install RubikTwophase
Collecting RubikTwophase
  Using cached RubikTwoPhase-1.0.9-py3-none-any.whl (53 kB)
Installing collected packages: RubikTwophase
Successfully installed RubikTwophase-1.0.9

(base) C:\Users\andre\Python\cube>python
Python 3.8.13 (default, Mar 28 2022, 06:59:08) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import twophase.solver  as sv
On the first run, several tables will be created. This takes about 1/2 hour or longer (depending on the hardware).
All tables are stored in C:\Users\andre\Python\cube\twophase

creating conj_twist table...
creating conj_ud_edges table...
...............
```

```
depth: 12 done: 140908410/140908410
creating phase2_prun table...
..
depth: 0 done: 1/111605760
.....................................................................
depth: 1 done: 4/111605760
.....................................................................
depth: 2 done: 14/111605760
.....................................................................
depth: 3 done: 66/111605760
.....................................................................
depth: 4 done: 351/111605760
.....................................................................
depth: 5 done: 1669/111605760
.....................................................................
depth: 6 done: 7340/111605760
.....................................................................
depth: 7 done: 33842/111605760
.....................................................................
depth: 8 done: 149309/111605760
.....................................................................
depth: 9 done: 620155/111605760
.....................................................................
depth: 10 done: 2473211/111605760
remaining unfilled entries have depth >=11
creating phase2_cornsliceprun table...
.................................................
creating phase2_edgemerge table...
.................

>>>
```

When the ‘phase2_prun’ table is completed (total process ca 60minutes in my case), if everything went well, the python prompt is presented.

8) Exit python: *exit()*

9) Check the presence and location of the lookup table files:
At the 'twophase' folder, located at the folder you've created for the robot, the below listed files (lookup tables for the solver) should now be available together with the python files listed in step 5 above:



| Nome | Ultima modifica | Tipo | Dimensione |
|---|---|---|---|
| co_classidx | 11/06/2022 14:22 | File | 79 KB |
| co_rep | 11/06/2022 14:22 | File | 6 KB |
| co_sym | 11/06/2022 14:22 | File | 40 KB |
| conj_twist | 11/06/2022 14:19 | File | 69 KB |
| conj_ud_edges | 11/06/2022 14:20 | File | 1.260 KB |
| fs_classidx | 11/06/2022 14:22 | File | 1.980 KB |
| fs_rep | 11/06/2022 14:22 | File | 252 KB |
| fs_sym | 11/06/2022 14:22 | File | 990 KB |
| move_corners | 11/06/2022 14:25 | File | 1.418 KB |
| move_d_edges | 11/06/2022 14:23 | File | 418 KB |
| move_flip | 11/06/2022 14:22 | File | 72 KB |
| move_slice_sorted | 11/06/2022 14:22 | File | 418 KB |
| move_twist | 11/06/2022 14:22 | File | 77 KB |
| move_u_edges | 11/06/2022 14:23 | File | 418 KB |
| move_ud_edges | 11/06/2022 14:24 | File | 1.418 KB |
| phase1_prun | 11/06/2022 15:20 | File | 34.402 KB |
| phase2_cornsliceprun | 11/06/2022 15:22 | File | 945 KB |
| phase2_edgemerge | 11/06/2022 15:22 | File | 79 KB |
| phase2_prun | 11/06/2022 15:22 | File | 27.248 KB |

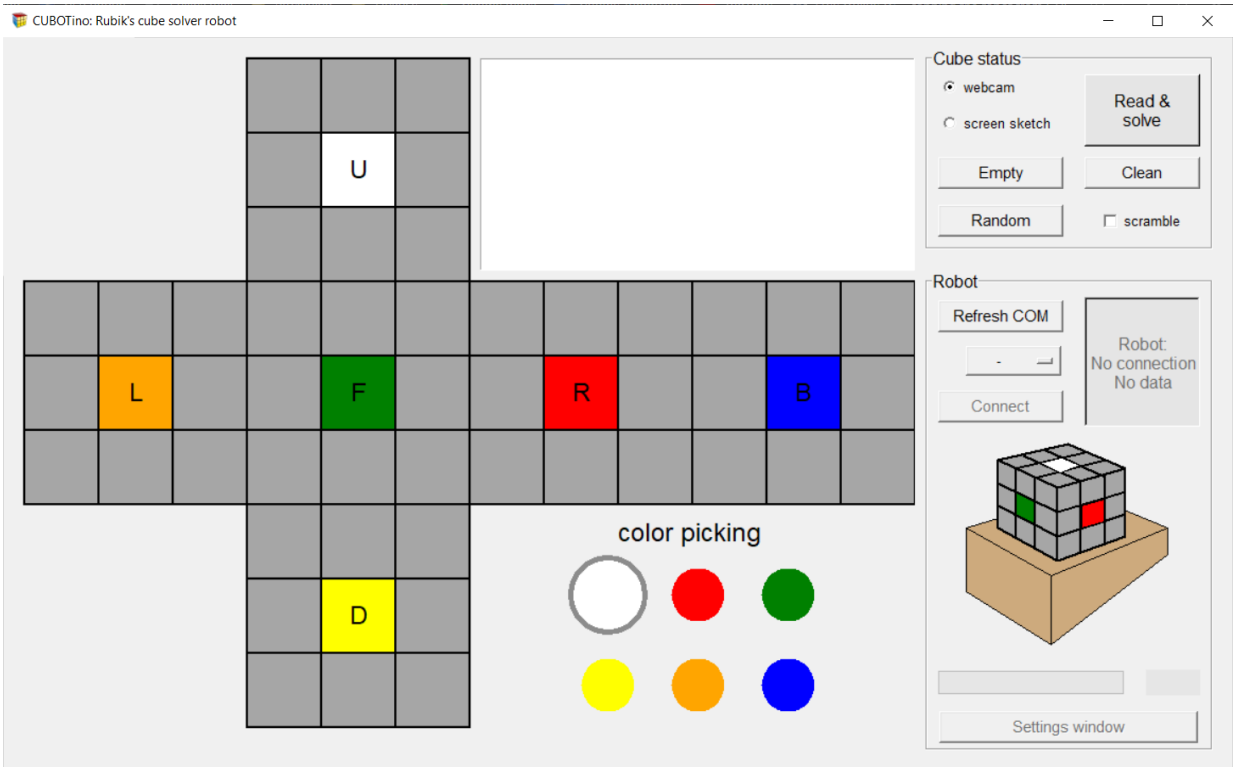
In my case these files would be located at: C:\Users\andre\Python\cube\twophase

10) Run *python Cubotino_GUI.py* to check if everything works fine.
When the Kociemba solver is installed (copied) with this second method, the below print will ibe made at the Cubbotino_GUI.py start:

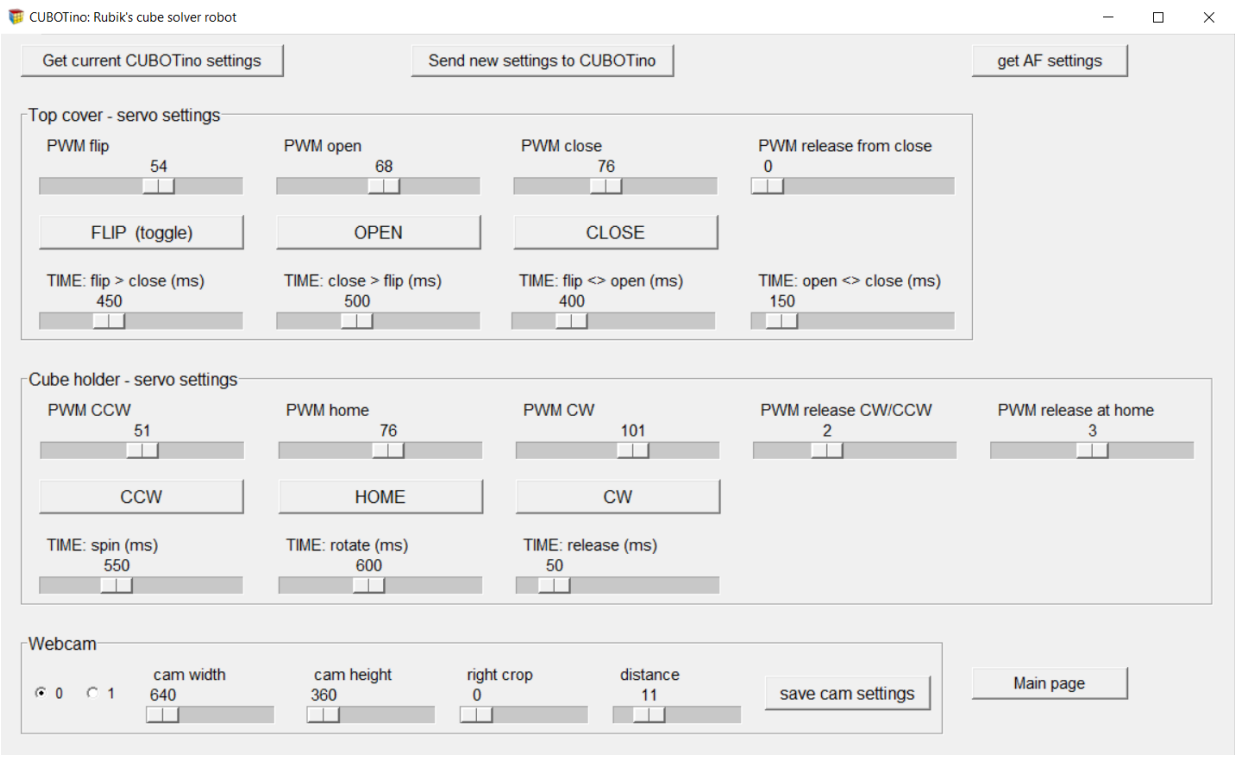
```
Running on: Windows
Python version: 3.8.13 (default, Mar 28 2022, 06:59:08) [MSC v.1916 64 bit (AMD64)]
CV2 version: 4.5.1
webcam module AF (11 June 2022)

loading conj_twist table...
loading conj_ud_edges table...
loading flipslice sym-tables...
loading corner sym-tables...
loading move_twist table...
loading move_flip table...
loading move_slice_sorted table...
loading move_u_edges table...
loading move_d_edges table...
loading move_ud_edges table...
loading move_corners table...
loading phase1_prun table...
loading phase2_prun table...
loading phase2_cornsliceprun table...
imported the copied twophase solver
```

# 18) GUI

The GUI has two windows, “Main page” and “Settings windows”
At GUI launch (Cubotino_GUI.py), the Main window appears, as per below:




When the serial communication is established, the “Settings windows” button is activated, and via such button the Settings window can be reached:

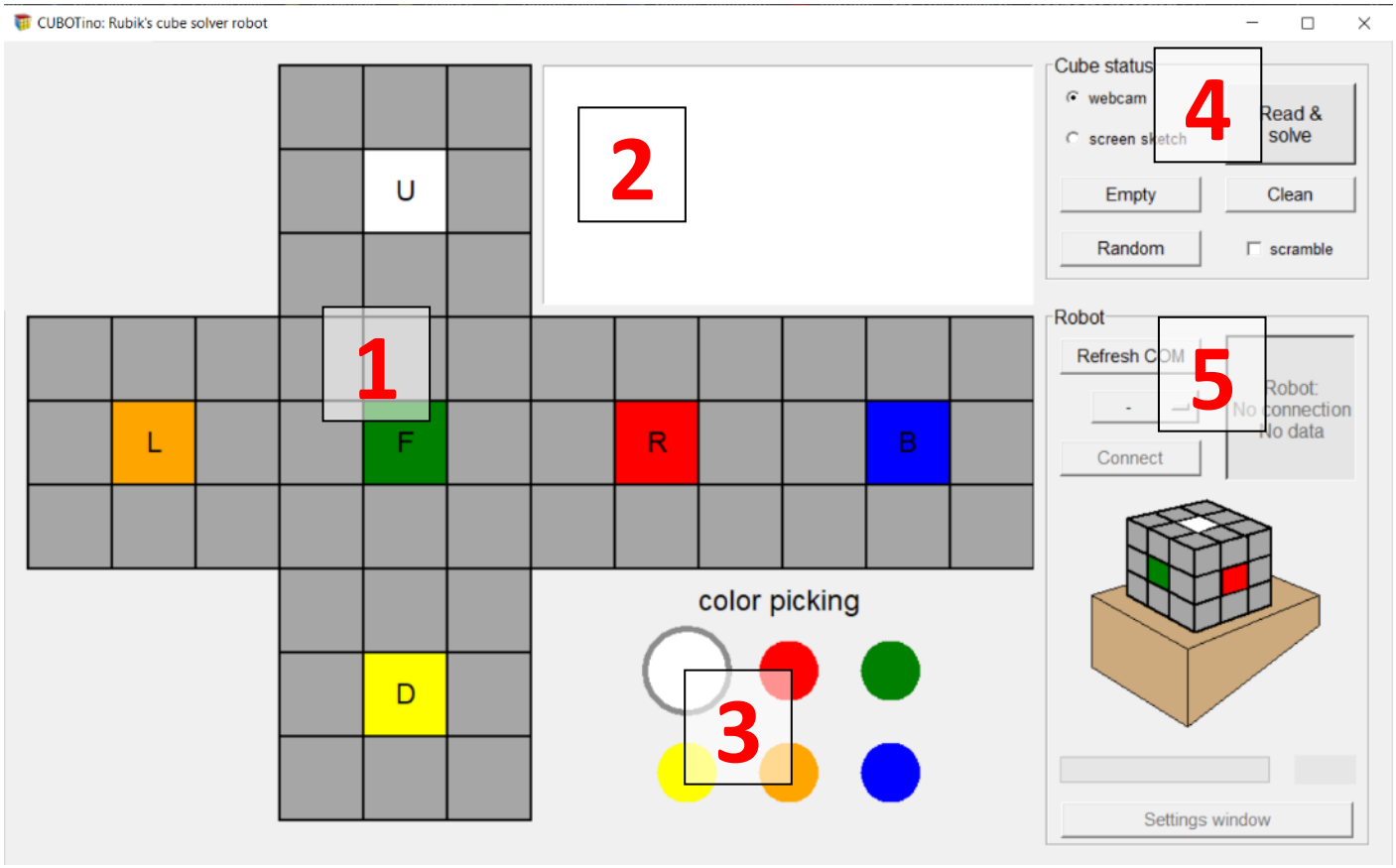


Each window is somehow divided in areas:

On the *Main window* there are the below areas:

| Area | Name | Functions |
|---|---|---|
| 1 | Cube sketch | - Used to generate the initial cube status<br>- Feedback the cube status during the robot solving |
| 2 | Text message window | - Provides text feedback and info |
| 3 | Colour picking | - Allows to select colour to be assigned to the sketch, via mouse left button |
| 4 | Cube status interface | - Interface with buttons and settings for the cube status detection part. |
| 5 | Robot interface | - Interface to interact with the robot |

Notes:
Some buttons are activated only when some pre-conditions are fulfilled; For instance the *Settings window* button requires to have the serial communication up and running with the robot.

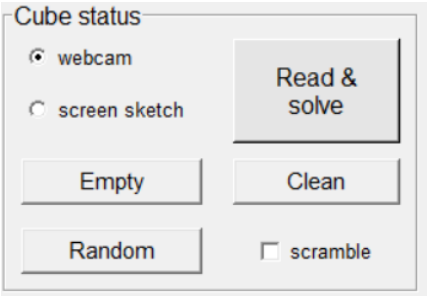



Area 1 and 3:

- To fill the cube sketch, a color can be picked on the palette, with the left mouse button, and applied on the facelets by pressing again the left mouse button; Colors can also be entered, or adjusted, via the mouse scroll wheel when the pointer is over a facelet.
- It is possible to change the faces center colours as well (not when the pointer is not over the face letter).

The Area2 is just informative

Area 4, Cube status interface:

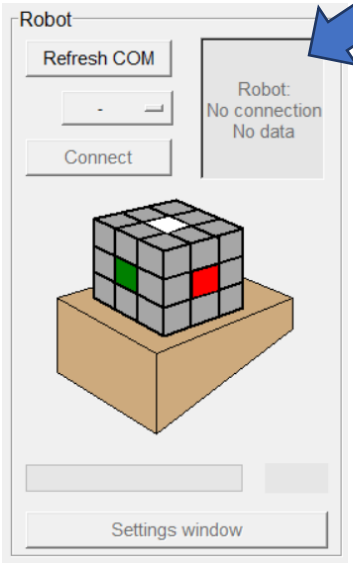



It's possible to select the cube status source, *webcam* or *screen sketch*.
The *Read & solve* button sends, to the Kociemba solver, the cube status depicted on the sketch (or greyed out when *scramble* is selected); In case of coherent cube status (and no *scramble*), on the Text window will be printed:

- Cube status
- Cube solution string, and number of manoeuvres
- Robot solution string

The *Random* button generates random cube status on the screen sketch; Select *scramble*, before requesting a random cube, to be keep "hidden' the cube status, when the robot is used as a cube scrambler.
*Read & solve* reads the cube status, also when greyed out for cube scrambling, and return both the cube manoeuvres and the robot solution string.

Area 5, Robot interface:




The large *Robot* button turns active only when the robot is connected and there is a cube solution; It always provides the status feedback:

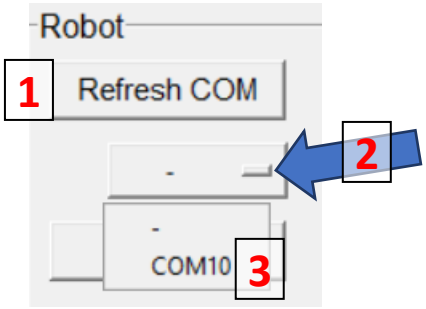
- No connection, No data (No data means there isn't a robot solution to send)
- Connected, No data
- Send data to robot
- STOP ROBOT

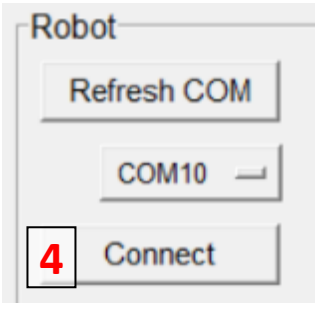
The CUBOTino sketch [illegible] the center facelets colours, as guidance to drop the cube on the robot according to the orientation used while detecting the status.

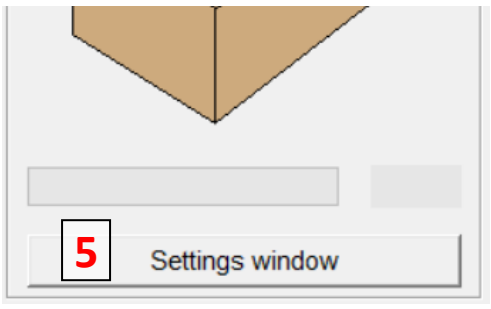
The progress bar is upd[illegible]n real time, according to the progress feedback sent by the robot at each move.

The *Settings window* b[illegible]ets activated when the communication with ESP32 is established.

The *Refresh COM* button updates the connected COM ports, that will populate the drop-down menu; The COM ports list will be displayed by pressing the little rectangle, see picture below



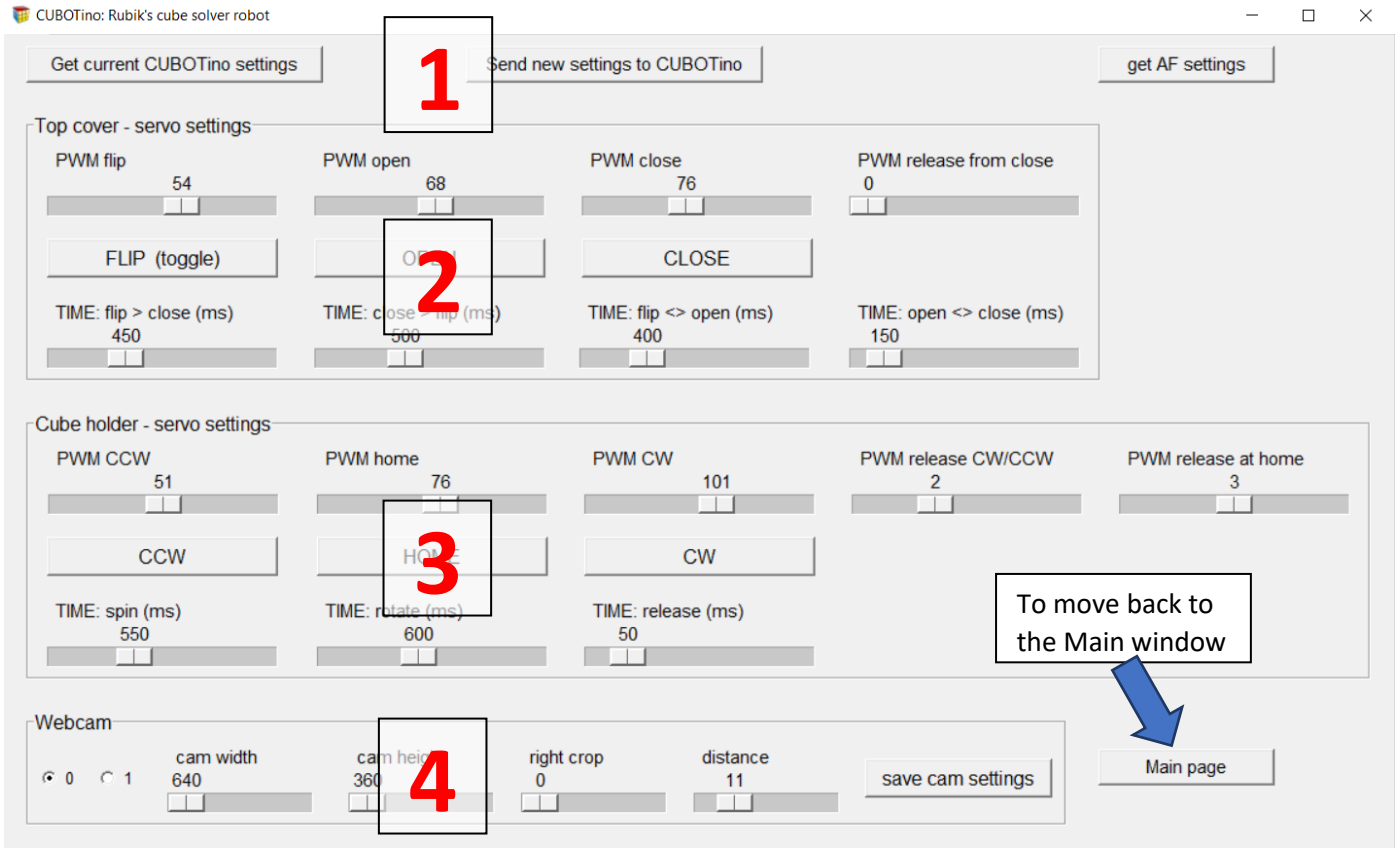

When a COM port has been selected, the drop-down menu closes and the *Connect* button becomes active.
When the Connect button is pressed, in case the communication with ESP32 goes well the *Settings window* button gets activated.
The *Settings window* button activates the GUI dedicated window for the robot and webcam settings.

On the *Settings window* there are the below areas:

| Area | Name | Functions |
|---|---|---|
| 1 | Receive and sending robot setting | - Retrieve the settings that are stored at the micro-controller<br>- Send the new settings to the micro-controller<br>- Retrieve the setting of my robot, as reference |
| 2 | Top cover servo | - Settings for the Top cover (and lifter)<br>- Buttons allow to test the specific positions, and some of the associated timers |
| 3 | Cube holder servo | - Settings for the Cube holder<br>- Buttons allow to test the specific positions, and some of the associated timers |
| 4 | Webcam | - Settings for the webcam |

Notes:

1. The changes eventually made on the servo related sliders, will be effective at the robot only after pressing *Send new settings to CUBOTino* button.
2. By pressing the servo related buttons (FLIP, OPEN, CLOSE, CCW, HOME, CW) it's possible to test the PWM and servos positions.
3. Not all the timers can be tested via test buttons associated functions, yet most of them 😊




Area1, Receive and sending robot setting:
*Get current CUBOTino settings* button, to receive the servo settings stored in the ESP32 microcontroller.
*Send new settings to CUBOTino* button, to send the sliders servo settings to the ESP32 microcontroller.
*Get AF settings* button, to receive the servo settings of my CUBOTino, just as reference (hard coded settings).

Area2, Top cover – servo settings:

| Name | Type | Function |
|---|---|---|
| PWM flip | slider | Set the flip position, meaning the Lifter reaches a height of ca two cube layers |
| PWM open | slider | Set the open position, meaning both the Top cover and the Lifter should be out of the way for the cube/cube Holder rotation |
| PWM close | slider | Set the close position, meaning the Top cover constrains top & mid cube layers. |
| PWM release from close | slider | Set the release position from close, meaning the Top cover slightly re-opens from the Close position. This might be useful if the close position is also used to flatten the cube (i.e. after flipping) by setting a close angle forcing the Top cover against the top cube face. |
| FLIP (toggle) | button | Tests the Flip angle, associated to the PWM flip. It toggles between Flip and Open position |
| OPEN | button | Tests the Open angle, associated to the PWM Open. |
| CLOSE | button | Tests the Close angle, associated to the PWM Open, as well as the release from close angle. |
| Time: flip → close | slider | Time (in ms) for Top cover to move from flip to close position |
| Time: close → flip | slider | Time (in ms) for Top cover to move from close to flip position |
| Time: flip < > open | slider | Time (in ms) for Top cover to move from flip to open position, and viceversa |
| Time: open < > close | slider | Time (in ms) for Top cover to move from open to close position, and viceversa |

Notes:

1. For the PWM smaller values means the Top cover rotates closer to the cube
2. Servo motors don't provide feedback of their positions: Timers are used to wait the needed rotation time before moving to the next action; Note the servos rotation speed is affected by the power supply.

Area3, Cube holder servo-settings:

| Name | Type | Function |
|---|---|---|
| PWM CCW | slider | Set the CCW position, meaning the cube Holder rotates slightly more than 90° CCW from home (reference from the servo point of view) |
| PWM home | slider | Set the home position, meaning the cube Holder centered to the Lifter path |
| PWM CW | slider | Set the CW position, meaning the cube Holder rotates slightly more than 90° CW from home (reference from the servo point of view) |
| PWM release CCW/CW | slider | Set the release position from CCW and CW, meaning the cube Holder rotates slightly back (toward home) to release tension. |
| PWM release home | slider | Set the release position from home, meaning the cube Holder makes a slightly larger rotation before rotating back home. This is needed for proper cube layers alignment, further than tension release |
| CCW | button | Tests the CCW angle, associated to the PWM CCW, and the release angle |
| HOME | button | Tests the Home angle, associated to the PWM home, as well as the release |
| CW | button | Tests the CW angle, associated to the PWM CW, and the release angle |
| Time: spin | slider | Time (in ms) for the cube Holder to spin 90°, with open Top cover |
| Time: rotate | slider | Time (in ms) for the cube Holder to rotate 90°, with close Top cover |
| Time: release | slider | Time (in ms) for the cube Holder to rotate back at CCW, CW and home to release tension |

Area4, Webcam settings:

| Name | Type | Function |
|---|---|---|
| 0 1 | Radio-button | Set the webcam number that will be used by python CV module. If only an integrated webcam (laptop) it will be 0, differently just test. |
| Cam width | slider | Set the webcam width (in pixels) for the python CV module. Typically, the set value will be rounded to the closest acceptable value of the webcam |
| Cam height | slider | Set the webcam height (in pixels) for the python CV module. Typically, the set value will be rounded to the closest acceptable value of the webcam |
| Cam crop | slider | Set the vertical bandwidth (in pixels) o be cropped at the right side of the frame. This is useful when the webcam has a 16:9 ratio |
| Distance | slider | The smaller the value the closer the cube must be presented to the webcam.<br>This is used to filters out small squares: The minimum facelet side is calculated by dividing the frame width by value set with this slider |

Notes:

1. The smaller the frame dimension, the faster the facelets detection time is.

# 19) Data logged

Each time the robot solves a cube, or when it get stopped, the below data is logged in a text file:

| Column name | Info |
|---|---|
| Date | Date and time (yyyymmdd_hhmmss), i.e. 20220308_213439 |
| CubeStatusEnteringMethod | “screen sketch” or webcam |
| CubeStatus | i.e. RLFDUUDBBLFRURRBDDRDDFFLURULDRFDFLUFDLBRLRFLBUBFUBBLBU |
| CubeSolution | i.e. D2 L2 F2 R3 F2 L1 D2 F2 R3 U2 F1 L1 F3 R1 B2 F3 D3 L2 F2 (19f) |
| RobotoMoves | i.e. R1S3R1S3S3F1R1S3R1F1R1S3R1S3F3S1R3S3F1R1S3R1S3F3R1S3<br>F1R1S3R1F3R1S3R1S3S1F3R3S3F1R1S3R1F3R1S3F3R1S3S1F1R3F1R1<br>S3S3F3R1S3R1F2S1R3S1F3R3S3F3R1S3R1F3R1S3R1S3 |
| TotCubotinoMoves | i.e. 97 |
| EndingReason | Solved, scrambled, stopped (if solving or scrambling are interrupted) |
| RobotTime(s) | i.e. 77.2 (this is reported also when the robot gets stopped) |

Notes:

1. The folder *data_log_folder* is made from the folder where Cubotino_GUI.py is running.
2. The logged data is saved in the *Cubotino_log.txt* file.
3. Text file uses tab as separator.

Purpose for the data logging is mainly fun, yet it might be useful for debug or statistics

# 20) Assembly steps:

Before assembling the robot:

- Make the ESP32 connections board
- Setup the ESP32 microcontroller
- Install all the files and libraries
- Position the two servos output gear to their middle position (see Tuning chapter)

Assembly order (for the details see Assembly details, toward the document end):

4. Screw the bottom servo to the structure
5. Prepare the sandwich Servo_axis_inf / servo lever / Servo_axis_inf
6. Assemble the Cube_holder to the Servo_axis assembly
7. Assemble the Cube_holder assembly to the bottom servo
8. Assemble the Hinge to the Structure
9. Assemble the “T25” servo arm to the upper servo.
10. Insert the Top_servo assembly into the Top_cover slot
11. Assemble the Top_cover assembly to the Hinge
12. Complete the top servo assembly to the Hinge
13. Assemble the Lifter to the Top_cover
14. Assemble the “STOP” touch plate, and related cable, to the Structure
15. Position the cables, and assemble the Baseplate_rear
16. Fix the ESP32 connection board to the Structure
17. Connect the servos and the touch plate to the ESP connection boards
18. Insert the ESP32 dev board to the ESP connection board
19. Assemble the Baseplate_front to the Structure
20. Stick self-adhesive rubber feet to the baseplates
21. Assemble the PCB cover
22. Optional: Personalization cover instead of the STOP plate

**Tools necessary:** Allen keys 3mm, 2.5mm and 2mm

# 21) How to operate the robot:

1. Robot → Connect the power supply (5V 2A) to the microUSB for servos.
2. GUI , Robot → Launch the GUI and connect the robot to an USB port of your PC (no obliged order on energizing the robot / launching the GUI).
3. Robot → When the robot gets energized:
    1. The red LED on ESP32 dev board will light up
    2. The ESP32 boots, in few seconds
    3. Servos settings are loaded, and servos are “initialized”:
        a. Top cover is moved to the open position
        b. Cube holder is moved to the home position
4. GUI → Refresh the COM ports on the GUI.
5. GUI → Select the used COM from the drop-down menu.
6. GUI → Connect the robot; If the communication works correctly, then the blue LED of ESP32 lights up.
7. GUI → Generate the cube status, via the cube sketch or via the webcam:
    - If the cube status is coherent, a cube solution is shown on the Text message window and the GUI Robot button background colour changes to green.
    - Cube solution can be sent to robot.
8. GUI → Start the cube solving process, via *Send data to robot* button:
    - The blue LED at ESP32 dev board will flash during the cube solving period.
    - CUBOTino starts solving the cube.
    - Progress is feedback to the GUI, that keeps updated the Cube sketch and the progress bar.
    - When the robot completes all the moves, the solving time is displayed on the Text message window.
9. GUI, Robot → Stopping the robot:
    - The GUI *Robot* button changes background colour to red when the robot is solving the cube; by pressing the *STOP ROBOT* button the robot stops almost immediately (the latest move is completed, and the servo are right after set to the initialization positions).
    - The robot can also be stopped by touching the touch plate.
    - After stopping the robot, is still possible to *Read & solve* the cube status on screen (after selecting this option); Double check if the cube is oriented as per the sketch, is so *Send data to robot* to complete the solving process.
    - If *Disconnect* button is pressed, during the cube solving process, the robot stops and the serial communication is discontinued.
10. Robot → The robot can be unplugged at any moment (no shut down procedure needed).
11. GUI → The GUI can be closed at any moment; The cube status on screen sketch is never saved.

# 22) Tuning:

**1. Reference angles for servos:**

The servos I bought, have 180-degrees of rotation, that is more than sufficient for the (lifter and Upper-cover) angles of this robot; I don't suggest buying 270-degrees servo as this will further affect the angle resolution.
The point is that the connection between the servo arms, and the servo's outlet gear, have many possible positions (I believe there are 25 teeth).
This means the reference angles set on Cubotino_settings.txt file, are working fine on my robot, but not necessarily the best choice on other systems.
These parameters must be tuned on your system, and the Settings windows GUI is the convenient tool for this.

**2. Setting servos positions:**

Servos are controlled on angle, via a PWM signal
The PWM resolution with Micropython V1.17 is rather poor at 50Hz (the frequency used by servos); The possible range is in unit from ca 50 to 100, meaning a 3.6° resolution for a 180° servo.
Despite the low resolution, this robot is quite forgiven, but the positions (angle=PWM) must be properly chosen.
With the servos I bought (link on the material list), and Micropython V1.17, the servos move for PWM values from 51 to 101 included (these are the values for the cube Holder CCW and CW, on my CUBOTino); With this range the servos make slightly more than 180°, that is simply perfect for this application.

Before assembling the robot, the servos rotation range must be checked:

1. In case only one servo rotates 180° or slightly more, use this servo for the cube holder.
2. Search the min and max PWM values that makes the servos gear moving (connect an arm to better check).
3. Set both the servos on their mid range position (mid PWM value), prior to the robot assembly.
4. The Settings windows at GUI can be used for these tests.

Note: In case your servo makes too much less than 180° rotation, to get the robot working properly, there are tutorials in internet on how to increase the rotation angle, by adding some resistors in series with the servo potentiometer (servo must be opened for this eventual change).

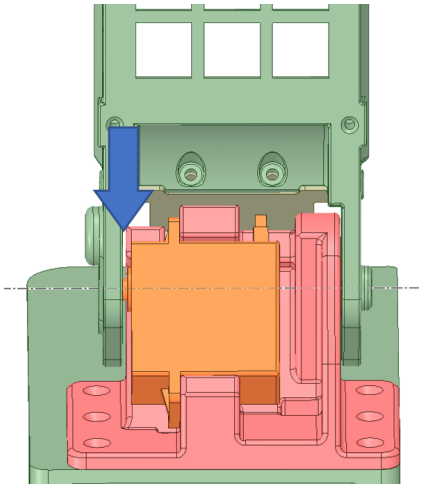
**3. Timers for servos:**

The servos don't provide feedback when they have completed the requested angular rotation; For this reason it's necessary to set appropriate waiting time to allows the servo to complete the action.
It will be convenient to use larger delays at the beginning, and progressively reduce them.

# 23) Troubleshooting

Some of the below aspects were encountered during the robot development, other are hypothetical

1. Servos don't move smoothly
    - Don't use jumper wires, or use quality jumper wires
    - Don't use bread boards, make the connections board instead.
    - Add the capacitors, to prevent voltage drops when servos are activated.
    - Use a 5V 2A phove charger for the servos.
    - Use a 20 to 25Kg/cm servo.
    - Minimize Top_cover rotation friction:
        i. Ensure the hole for the M4 screw (pivot) has some gap on the Top_cover hole (Ø4.1 to Ø4.3mm).
        ii. Rub some candle wax on the screw.
        iii. Ensure the M4 screw head is not pushing toward the Top_cover; 1mm gas is suggested
        iv. Ensure gap presence between Top_cover inner surface and the Hinge at the servo gear outlet side, as per below arrow:



In case your robot has little or no gap:

1) Unscrew the M3 screws holding the top servo, and place some little spacers (0.5 max 1.0mm), in between the servo and the Hinge (possibly close to the screws location); Tighten again the screws.
2) Rub some candle wax on the Hinge surface toward the Top-cover and the Top_cover inner surface toward the Hinge.

2. Connection from the GUI to the robot fails:
    - Ensure the ESP32 microcontroller isn't still connected to Thonny or an IDE.

3. Connection from the IDE to the robot fails:
    - Ensure the ESP32 microcontroller isn't still connected to the GUI.

4. Bottom cube layer doesn't align nicely:
    - Verify if the cube Holder makes an extra rotation, at both CCW and CW directions, before stopping; If this doesn't happens:
        i. Increase the timers, as too small time don't give sufficient time to the servo to make the stroke visible when testing the cube holder position.
        ii. Adapt the PWM release CCW/CW value.
        iii. Place the PWM release CCW/CW at zero, and test if the CCW and CW position have a slightly overstroke from the 90°. If this is not the case, check if the other servo has a larger rotation range. If still not the case, check in internet how to (slightly) increase the servo rotation angle (additional resistors must be soldered into the servo)
    - Verify if the cube Holder makes an extra rotation, before stopping home; If this doesn't happen, adapt the PWM release home value.

5. The Top_cover isn't intended to keep pushing the cube when it's in the close position; In case the cube layers don't align nicely, by playing with the cube_Holder settings, it's possible to use the Top_cover to level the cube. By lower the Top_cover close position to have a little interference with the cube, will improve the cube layer alignment in particular after flipping the cube. In this case it will be convenient to set one or few units on *PWM release from close setting.* Via this setting is possible to release the tension between the Top_cover and the cube, after pressing it, to allow the cube_Holder to rotate with less effort.

6. Webcam app reads twice the same cube face: After one face has been detected, there are 3 seconds time in which the facelets are ignored, to give the time to move the cube to another face. A solution is to move the cube apart as soon as a face has been detected, and presented it again to the webcam once rotated to the next face.

7. Webcam app is slow: Speed is largely affected by the number of pixels under analysis. On the *Settings window* at GUI, it's possible to reduce the frame size (width, height, crop) to gain speed.

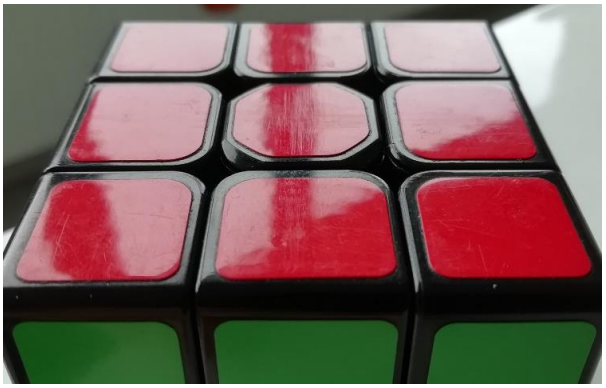
8. Webcam app ends with incoherent cube status statement: There are few reasons:

    - The cube orientation isn't correct; The cube faces order should be URFDLB, and oriented to correctly read the numbers 0 to 53 as per the sketch at "High level info" chapter.
    - The cube interpretation counts the same colour more than 9 times; This happens when the light reflection changes between faces. To solve this issue, it's recommended to have light coming from the side or to use a cube with less glossy facelets.
    - In case the cube has some prints (i.e. brand) , typically on the white center, it is suggested to carefully scratch out.

9. Cube's facelet and light reflection (webcam cube status detection):
   Detection of edges, as well as colours, can be largely affected by light reflection made by the facelets.
   I have two cubes available, one with in-moulded coloured facelets, and the other with glossy stickers.
   On the cube with plastic facelet, I made the surface matt by using a fine grit sandpaper (grit 1000); This makes the cube status detection much more unsensitive to the light situations.

Cube with in-moulded coloured facelet, that I've made matt with sandpaper (grit 1000)

Cube with glossy stickers (after taking this picture I made matt these facelets too)



10. If Cubotino_GUI.py file doesn't open as expected, by using Thonny, please check if the Interpreter is still set to work with MicroPython instead of Python.

11. If the ESP32 doesn't react, for instance the servos don't move when trying to set their positions, check the following:

    - ESP32 blue led isn't switched ON (serial communication isn't working):
        i. Check if the main.py file has been copied to the ESP32
        ii. Note that files at ESP32 have to be entered with explicit file extension (python icon will be associated to the .py files)
        iii. Try a different cable, as some cables are just meant for power supply, and don't have the data lines

    - If the ESP32 blue led is switched ON (serial communication working, between the PC and the ESP32):
        i. Check if the Cubotino_servos.py file has been copied to the ESP32
        ii. double check servo connections

12. If "ModuleNotFoundError: No module named 'scipy' "error is returned on PC, when running Cubotino_GUI.py there are a couple of possible solutions
    - If you've copied the Cubotino_webcam.py until 13/05/2022 you might decide to install the scipy library (see https://docs.scipy.org/doc/scipy/getting_started.html for the installation)

    Or differently
    - copy Cubotino_webcam.py file that is available since 13/5/2022 (end of AMS day time); In this version the calculations previously made by scipy.spatial distance are now done with Numpy library (part of the normal Python distribution, and largely used on this project).

# 24) Robot solver algorithm:

On this chapter it's explain the approach used to convert the cube solution manoeuvres into robot moves; This part is embedded in the Cubotino_moves.py file.

It is clear this robot has very limited degrees of freedom, as it can only rotate the bottom face (from -90° to +90°), farther than flipping the cube around the L-R horizontal axis; This obviously requires an algorithm that prevents additional cube movements to those (many) that are strictly necessary.

The Kociemba solver provides a string with the rotations to be applied on the 6 faces, like U2 F1 R3 etc (I will refer to these three moves as example on the below explanation).
The precondition for the cube solution is that the cube orientation doesn't change, meaning the U (upper) side remains up oriented and the F (front) side remains front oriented during the solving process; This pre-condition is clearly not fulfilled by the robot.

The robot solver follows instead the below approach:

1. All the 18 possible cube moves (U1, U2, U3, ….. , B1, B2, B3) the solver can return, are used as keys in a dictionary; There are 18 sets of robot movements (hard coded) associated to these keys. These robot movements consider the cube as ideally positioned: U side facing up and F side facing front.
2. When the robot moves the cube, its orientation is tracked per each applied movement (i.e. after the first U2 move, to follow above example).
3. When the next move must be applied, F1 in our example, the robot solver simply swaps the requested move (F1) to an adapted move; The adapted move reflects the real F side location at that moment in time. Based on the above example, the F1 move will be done by using the servo sequence associated to B1, simply because the F side is located at B side at that moment in time.
4. Above points are considered when the cube solution string is parsed, to generate a string with all the servo movements.

# 25) Colour's detection strategy (when using the webcam):

1. The vision system is used to detect the cube's facelets edges (contours), more or less as explained at https://medium.com/swlh/how-i-made-a-rubiks-cube-color-extractor-in-c-551cceba80f0
2. Average BGR is calculated for the areas defined by the contours, for the 54 facelets, and stored in a dictionary; Average HSV (of a small area at each contour's centers) is also calculated and stored on a second dictionary.

**Note**: On a 3x3x3 Rubik's cube, the 6 center's facelets have useful properties:

   a. These facelets don't move (fix facelets number)
   b. These facelets have (obviously) 6 different colours
   c. Opposite faces have known colours couples, white-yellow, red-orange, green-blue (Western colour code).

   This means we can make use of these 6 facelets as colour reference

3. The average HSV, detected on the 6 centers, is used to determine which colour is located on the 6 centers:
   a. White facelet is the one having the largest V-S delta (difference between Value, or Brightness, and Saturation), while the yellow one is located at opposite face.
   b. Remaining 4 centers are evaluated according to their Hue, and the Hue at opposite face.
   c. Orange has very low Hue, and red should be very high (almost 180); Depending on light condition, the red's Hue could "overflow" and resulting very low (few units). The red is expected to be much higher than Orange, unless it overflow … in this case both red and orange are rather small with red smaller than orange.
   d. Out of the two remaining centers, blue is the one with highest Hue, and consequently the green is also known.
4. Based on previous step, the 6 cube colours (at least their centers) have a known average HVS and therefore an average BGR colour; This also informs on the cube orientation (colours) as placed on the cube-holder.
5. Facelets colour interpretation is made, by using two methods, via a tentative approach:
   a. The first method compares the average RGB colour of each facelet, in comparison with the one at the 6 centers, and the colour decision is based on the smallest colour distance. The Euclidian distance of RGB per each facelet if calculated toward the 6 centers.
   b. In the second method the Hue value of each coloured (non-white) facelet are compared to the Hue of the 5 reference centers; White facelets are retrieved according to 3 parameters (Hue, Saturation, Value), in comparison to the white center HSV.

   First method is in general better than the second one, yet the second one "wins" when there is lot of light; The second method is only used (called) when the first one fails.
   As result both methods are used, to get reliable cube status detection under different light situations.

# 26) Credits:

- to Mr. Kociemba, that further than developing the two-phase-algorithm solver, he also wrote a python version of it; Credits to him also for the simple GUI he made available, from which I've further build the one for this robot.
- Hans Andersson, with his Tilted Twister (Tilted Twister 2.0) Lego robot, so inspiring: Very simple yet effective mechanic concept.
- to Jacques, who triggered me on the colours sensors direction.
- to Richard, for his keen check of this document and precious feedbacks.
- all the people who have provided feedbacks.

# 27) Revisions

| Rev | Date | Notes |
|---|---|---|
| 0 | 06/04/2022 | First release |
| 1 | 09/04/2022 | Added the scrambling function; New files at PC:<br>• Cubotino_GUI.py<br>• Cubotino_moves.py<br><br>Improved servos (re)initialization, after cube solving and robot stooping; New files at ESP32:<br>• main.py<br>• Cubotino_servos.py<br>• Cubotino_moves.py |
| 1.1 | 09/05/2022 | Added some notes on this document:<br>• Info to type the file extension when saving them in ESP32<br>• Info of the need to install "scipy" module, for the webcam cube status detection |
| 1.2 | 11/05/2022 | Added some info at troubleshooting |
| 1.3 | 13/05/2022 | Modified:<br>• Cubotino_webcam.py file to work without "scipy" library; Not anymore needed to install library.<br>Updated (accordingly):<br>• Files needed at PC on this doc<br>• Troubleshooting |
| 1.4 | 22/05/2022 | Modified:<br>• Cubotino_webcam.py<br>An error was preventing the HSV color detection to work when "debug" was set True.<br>Updated:<br>• Doc revision |
| 1.5 | 23/05/2022 | Modified:<br>• Cubotino_webcam.py<br>A quote error was preventing script from working at all.<br>Updated:<br>• Doc revision |

| | | |
|---|---|---|
| 2.0 | 04/06/2022 | Modified:<br>• Hinge.stl, Lifter.slt<br>Increased the gap between Top_Cover and Hinge, at the side of servo output gear<br>Added:<br>• Personaliz_plate<br>Updated:<br>• Models table, conclusions nd next steps, to include the Top version (now available).<br>• Doc revision |
| 3.0 | 11/06/2022 | Modified:<br>• Supplies list, by adding a microUSB breakout board<br>• Connections board, by splitting the power supply for servos from the ESP32<br>• Info related to the two power supply, and related How to use the robot<br>• Cubotino_GUI.py and Cubotino_webcam.py to tentatively import Kociemba solver from multiple installation locations, and to allow window dimension adjustement<br>• Structure.stl for (better) compatibility with the Top_version<br>Added:<br>• Kociemba installation chapter<br>Updated:<br>• Troubleshooting, with more details for the Top_cover < -- > Hinge gaps and friction<br>• Doc revision |
| 3.1 | 30/06/2022 | Modified Lifter.stl to increase the clearance toward the Hinge |

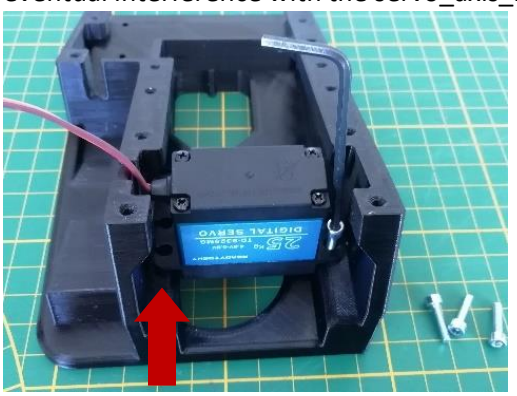
# 28) Assembly details:

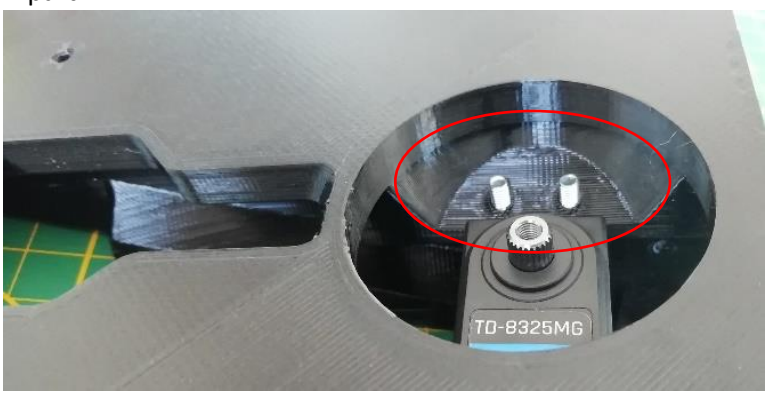
**Step4** (Mount the bottom servo to the structure):
4x M3x12mm cylindrical head
Couple of washers

To reach these screws it's necessary to use a narrow Allen key:

Couple of notes:

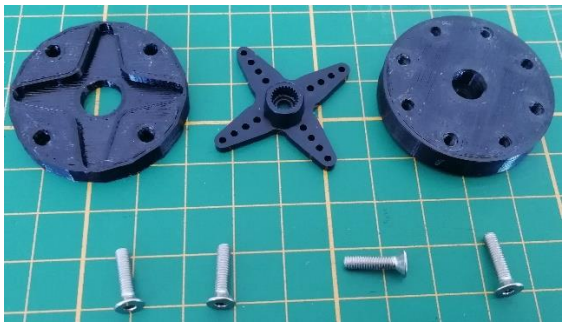
- Before tightening the four screws, checks if the servo output gear is well centred to the Structure hole.
- To limit the protrusion of the below two screws, add a couple of washers to these screws; This to prevent eventual interference with the servo_axis_inf part.



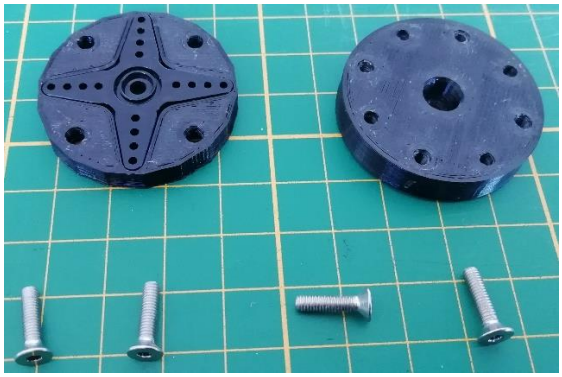


**Step5** (sandwich Servo_axis_inf / servo arm / Servo_axis_inf):

4x  M3x12mm conical head

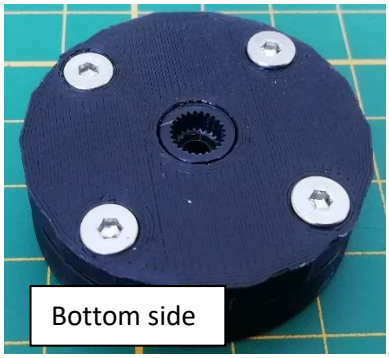


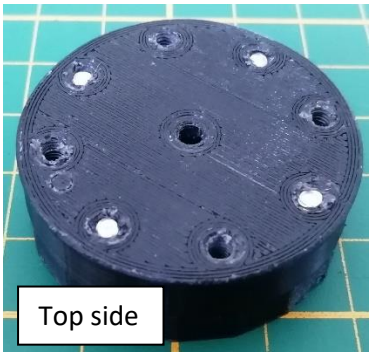
Check if the supplied X arm fits with the indentation of Servo_axis_inf part:



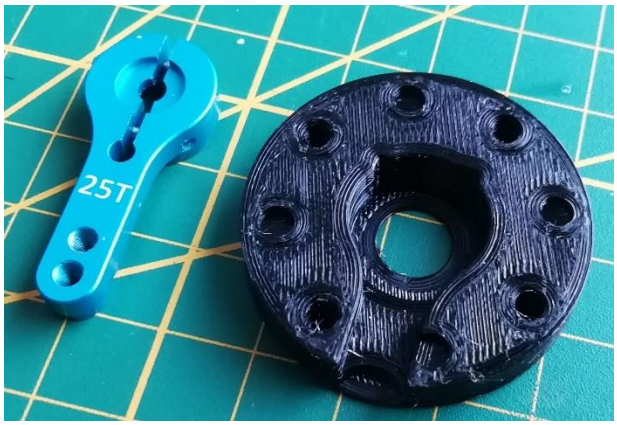
If you don't have a X arm to make it fit, please print the alternative parts (Servo_axis_inf and Servo_axis_sup) designed to fit the aluminium "T25" arm (arm has to be reduced in length).

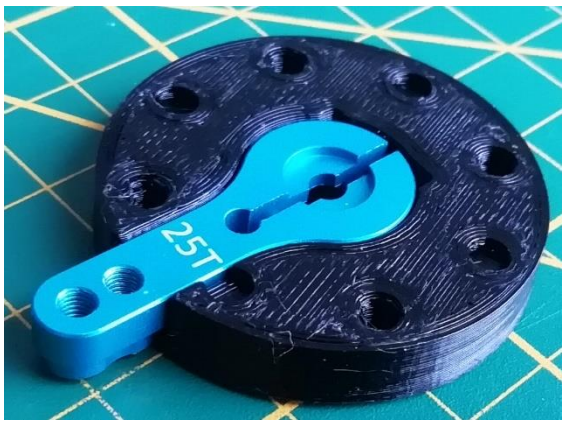
Make the sandwich



**Step5a** Alternative servo axis assembly:

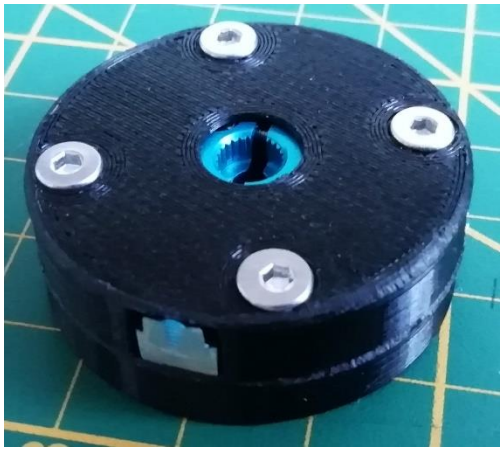



Cut the protruding part of the “T25” arm
Adjust its screws to be able to enter the servo outlet gear



Make the sandwich as per Step5
4x  M3x12mm conical head



**Step6** (Assemble the Cube_holder to Servo_axis assembly):

4x  M3x12mm conical head



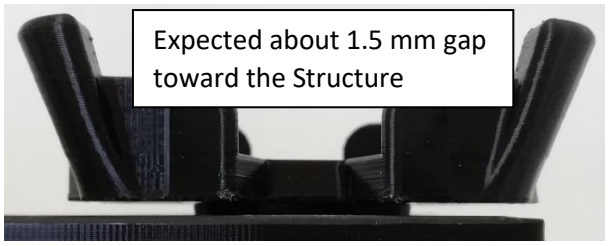
**Step7** (Assemble the Cube_holder assembly to the bottom servo):

Try to not rotate the Servo output gear during this step: Gently try to feel the teeth coupling, and if the Cube_holder is not well aligned, retract it, rotate the Cube_holder by about 90 degrees and check again.

Servo output, and Servo arm, have even number of teeth: For sure there is one good coupling

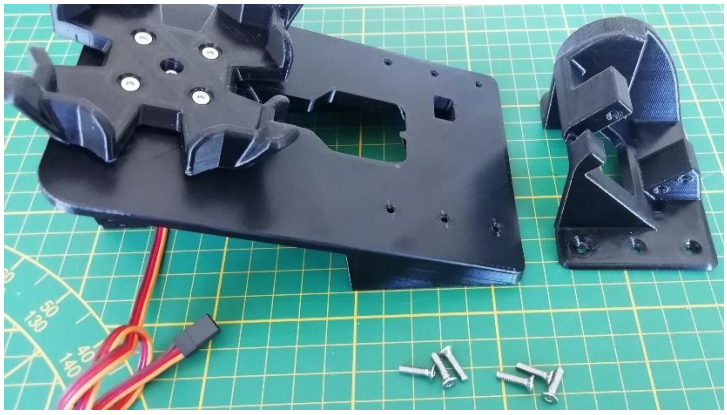

1x  M3x12mm cylindrical head

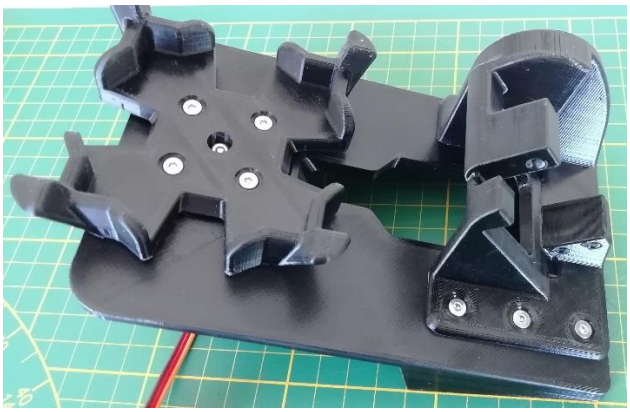




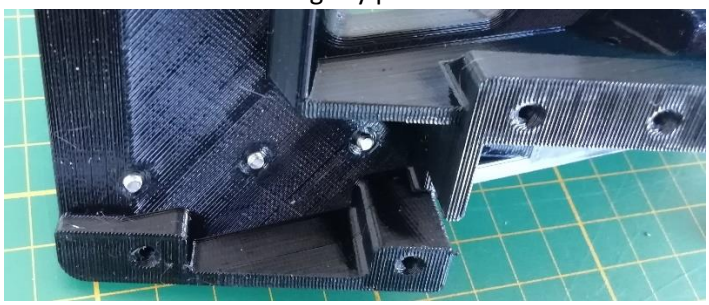
**Step8** (Assemble the Hinge to the Structure):

6x  M3x12mm conical head





Note: three screws will slightly protrude underneath the Structure; If screws of 12mm then this won't be a problem.



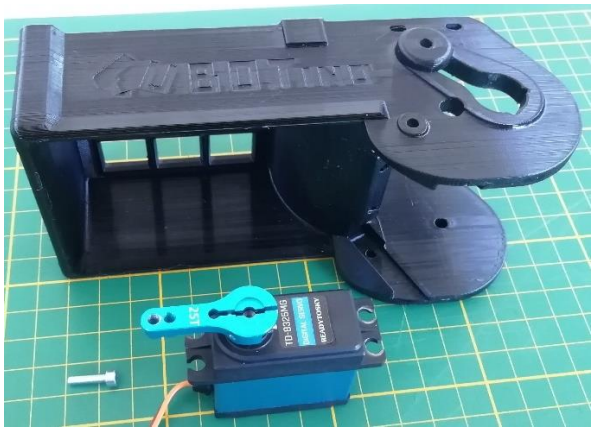
**Step9** (Assemble the "T25" servo arm to the upper servo):
Place the "T25" arm along the main Servo axis (Servo should be prepared upfront with the gear at middle angle).
Close the two arm tiny screws

**Step10** (Insert the Top_servo assembly into the Top_cover slot):

1x  M3x12mm cylindrical head

The slot for the “T25” arm might be tight; Remove eventual excess of material if needed

Ensure the hole for the M4 screw doesn’t not constrain the screw (it should have Ø4.1 to Ø4.3mm)



Note: The screw should not protrude from the Structure plane; Add some washers under the screw head if needed

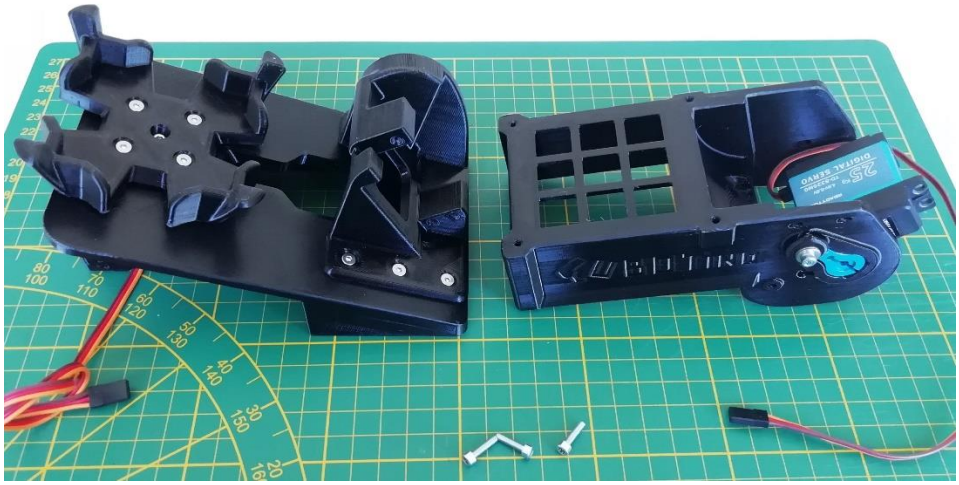
Rotate the servo by about 45deg, to facilitate next steps

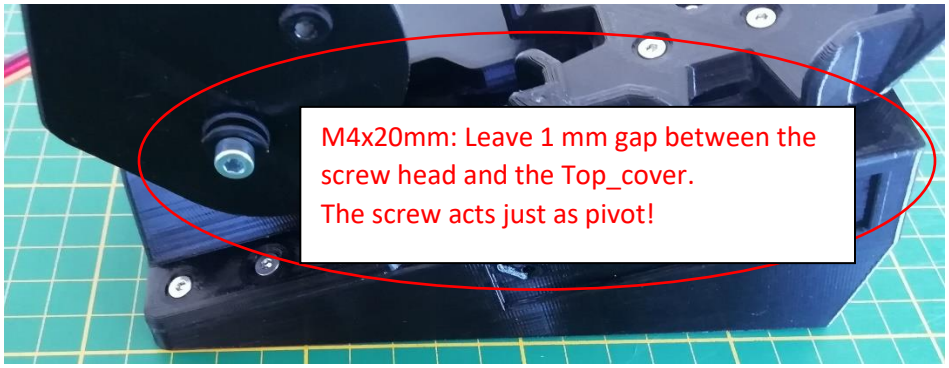
**Step11** (Assemble the Top_cover assembly to the Hinge):

1x  M4x20mm cylindrical head

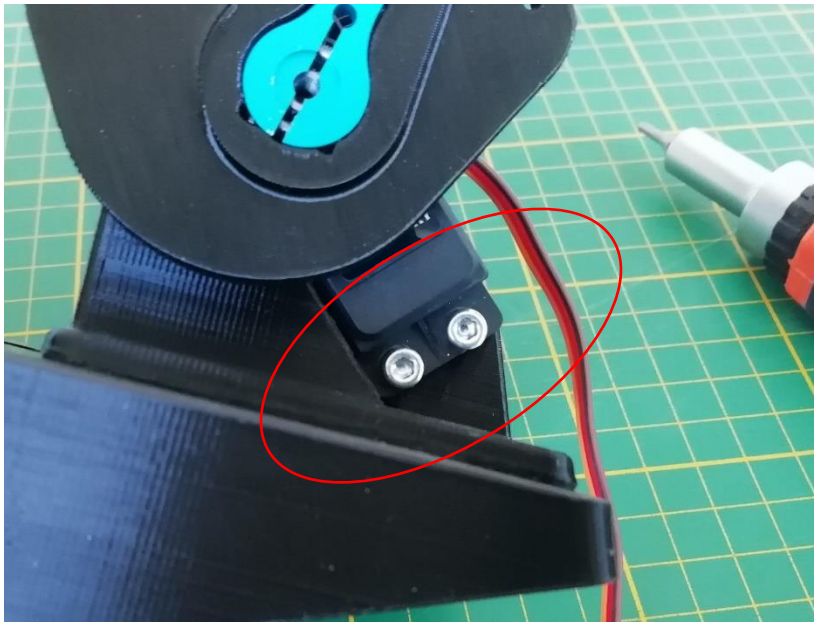
3x  M3x12mm cylindrical head

Ensure the hole for the M4 screw doesn't not constrain the screw (it should have Ø4.1 to Ø4.3mm)



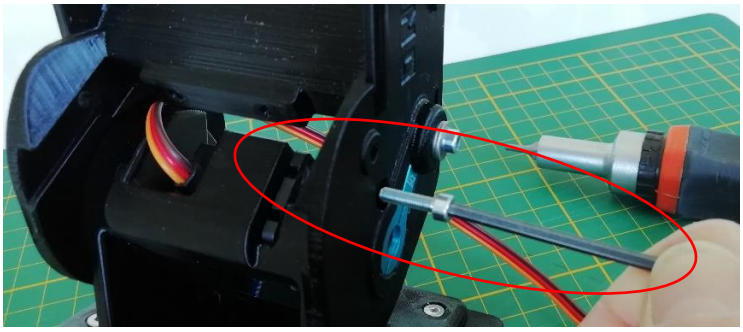



Do not fully tighten the two M3x12mm, until also the third screw is positioned

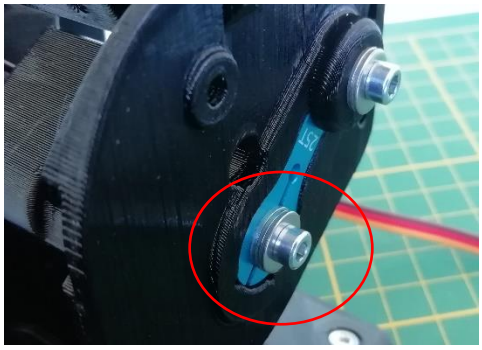


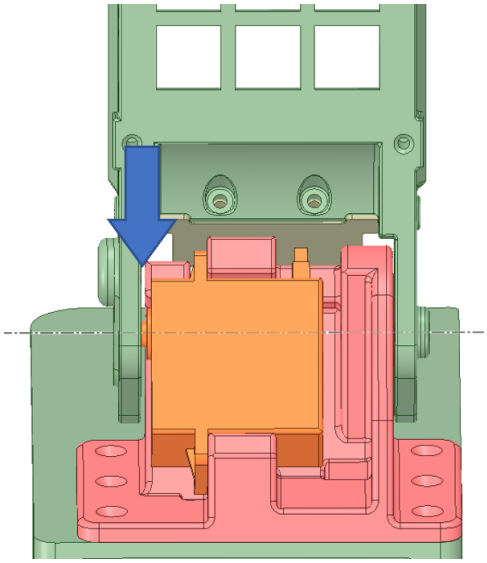
**Step12** (Complete the top servo assembly to the Hinge):

Rotate the Top_cover to have the hole facing the third screw accessible



1x M3x12mm cylindrical head (add washers if the screws doesn't push on the "T25"Arm)





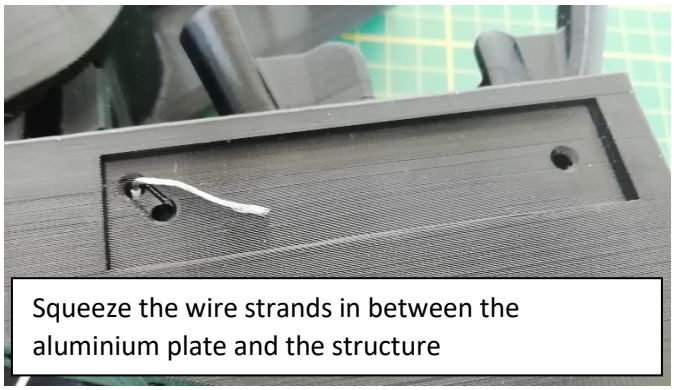
Verify gap presence in between the Top_cover and the Hinge, at the servo output gear side (arrow at the side).
In case there is little or no gap, unscrew the M3 screws of the servo, and place some little spacers (0.5 to max 1mm) in between the servo and the Hinge (preferably close to the screws locations);
Tighten the M3 screws and re-check the gap between the Top_cover and the Hinge.

**Step13** (Assemble the Lifter to the Top_cover):
4x  M3x12mm cylindrical head

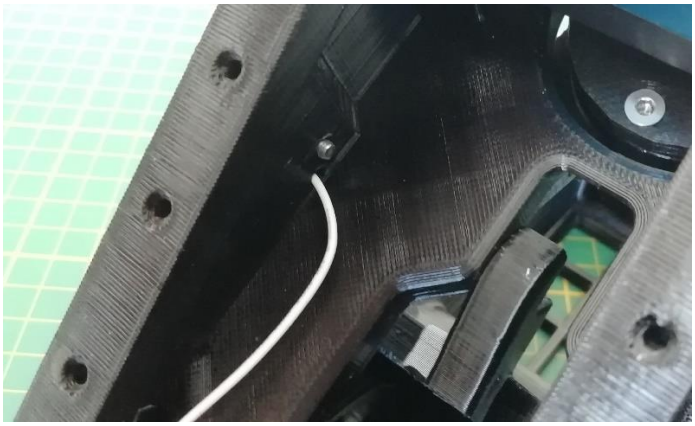
Slide the Lifter into the Top_cover slots

**Step14** (Assemble the “STOP” touch plate, and related cable):

2x  M3x12mm conical head

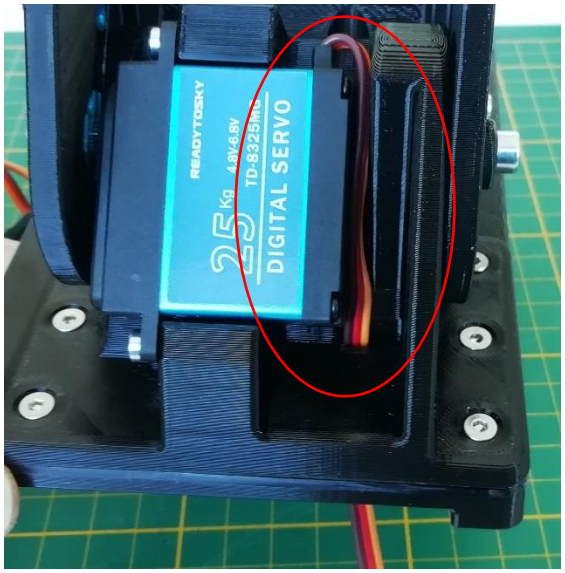



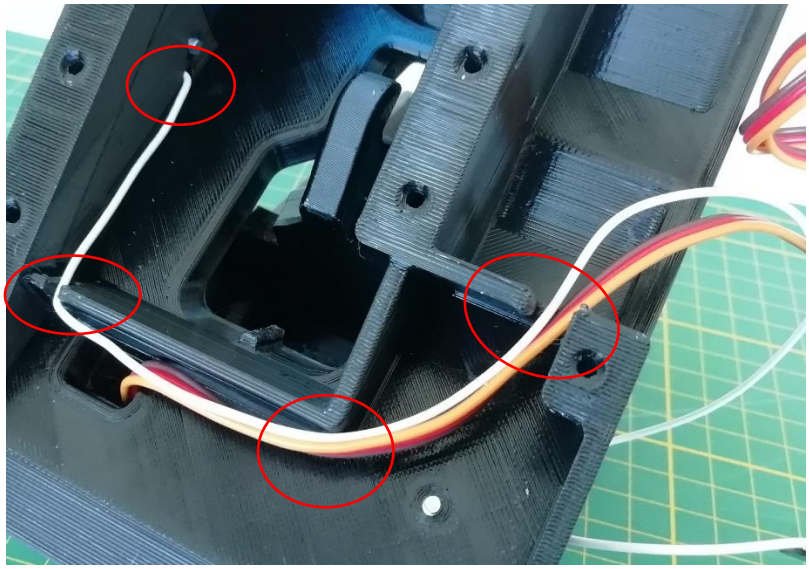
Squeeze the wire strands in between the aluminium plate and the structure

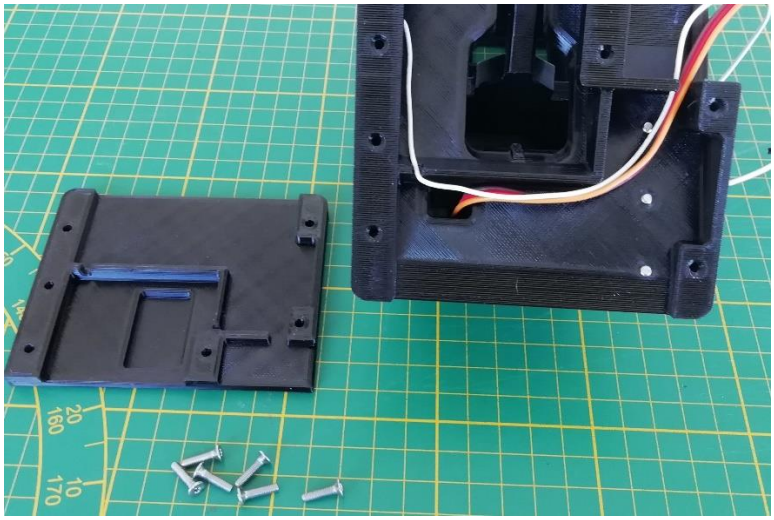



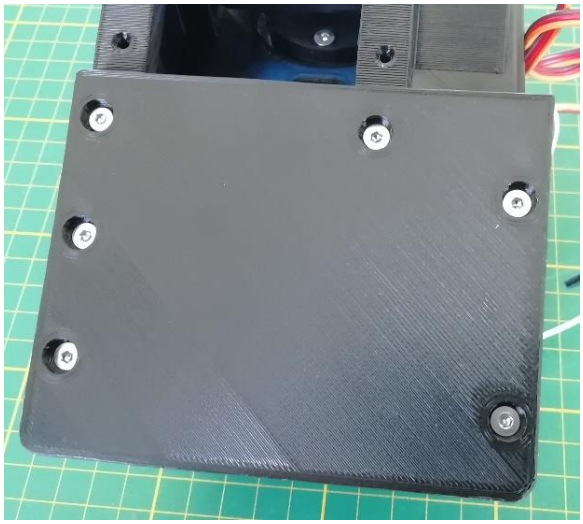
**Step15** (Position the cables, and assemble the Baseplate_rear):
6x  M3x12mm conical head (4 screws at corners are sufficient)

Dress the cables and close the Baseplate_rear



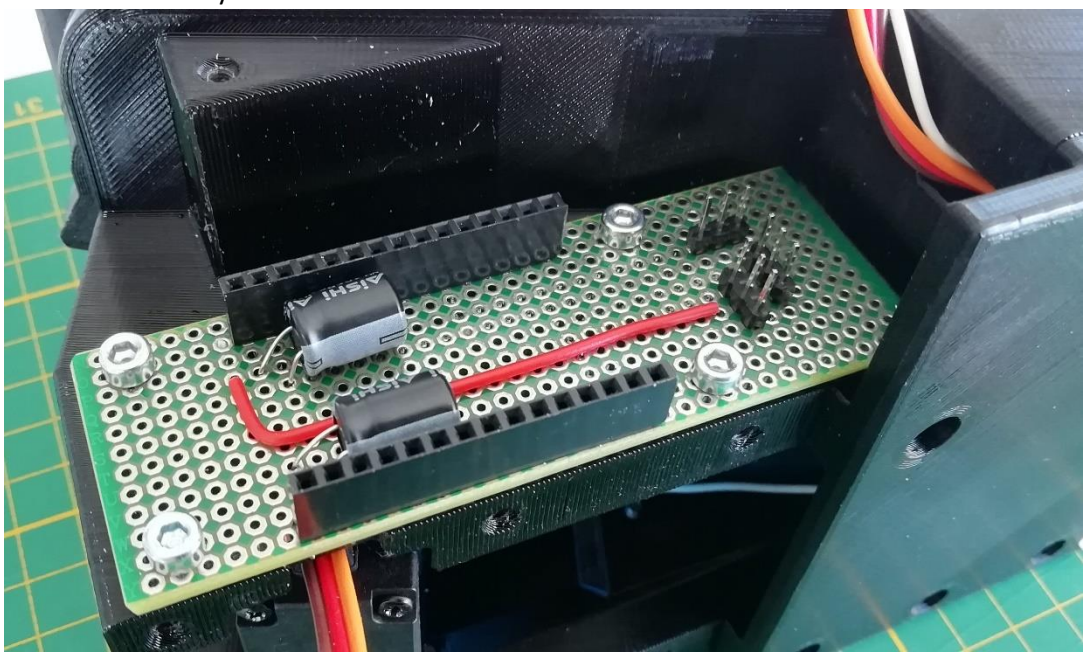







**Step16** (Fix the ESP32 connection board to the Structure)

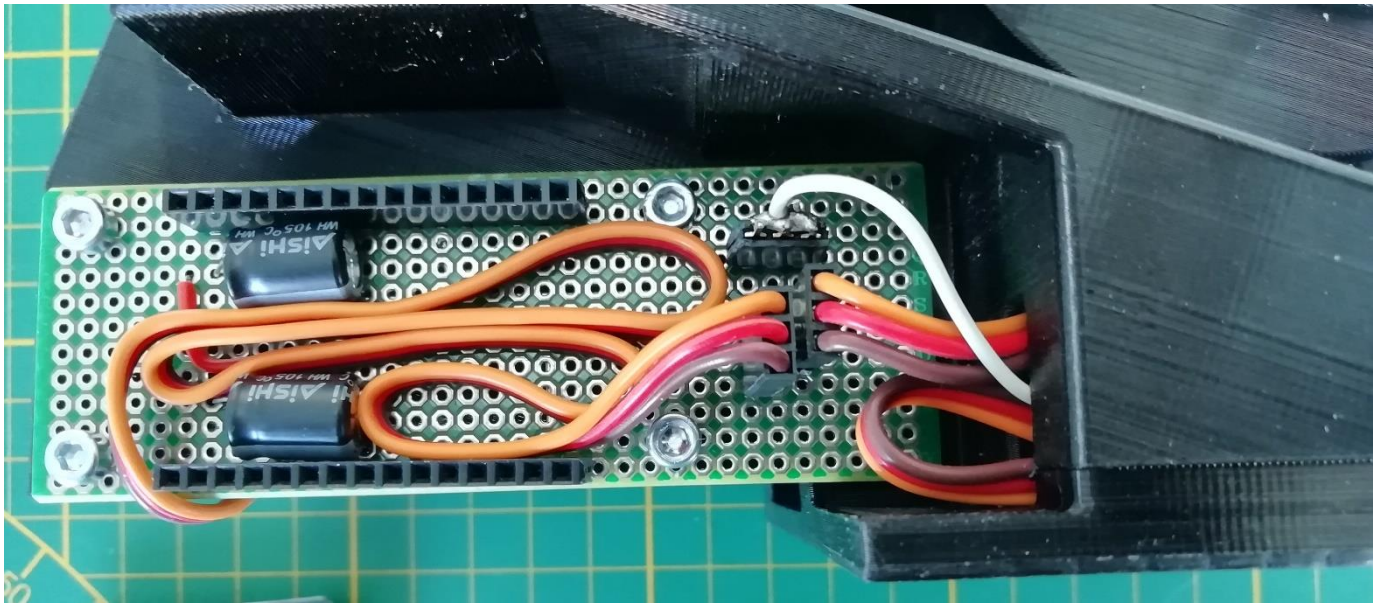
4x  M2.5x10mm cylindrical head



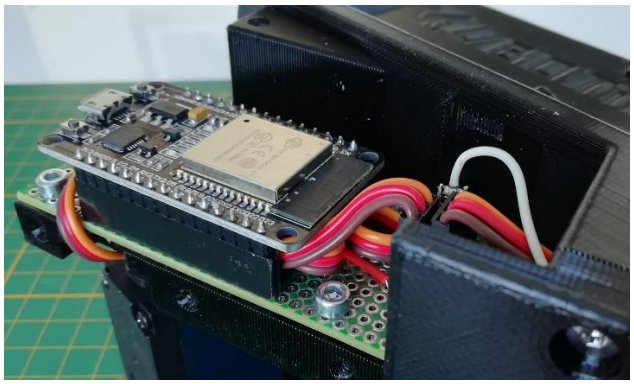
**Step17** (Connects servo and touch pad)

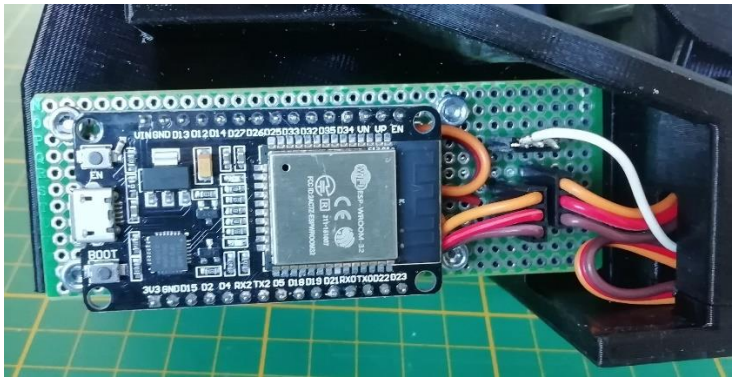
Dress the wires and connect

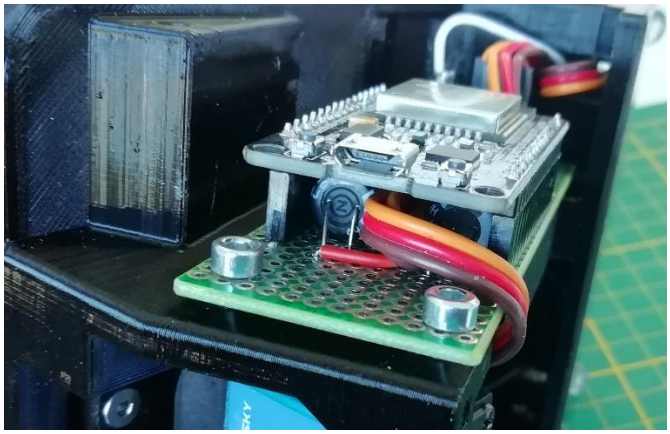
In general the brown wire of servo connector is the ground, therefore to be oriented toward the bottom



**Step18** (Insert the ESP32 dev board):

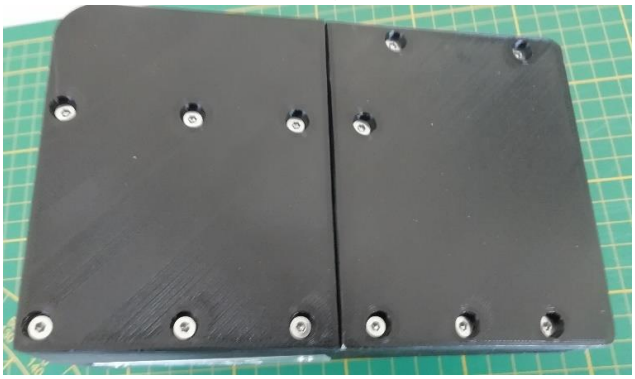






**Step19** (Assemble the Baseplate_front to the Structure):

6x  M3x12mm conical head (at the bottom, 4 screws at corners are sufficient)



**Step20** (Stick self-adhesive rubber feet to the baseplates):
4x  Self adhesive rubber feet
Rubber feet at the back should be placed as close as possible to the corners

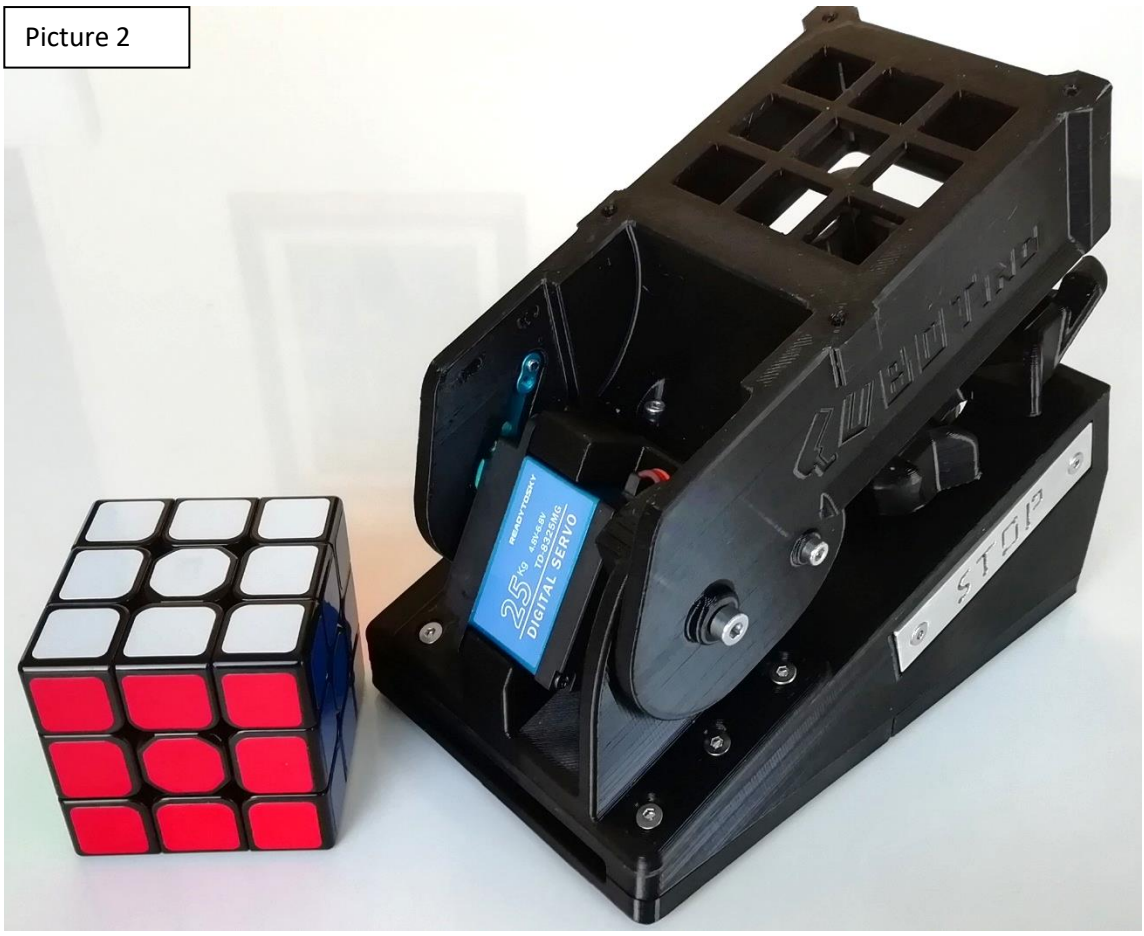
**Step21** (Assemble the PCB cover):
2x  M3x12mm conical head

**Step22** (Personalization, if you don't use the Stop plate):
The Structure has a recess, meant to hold a metal plate working as a Stop touch sensor.
In case you aren't interested in that function, you might consider printing a cover for the recess….
I made available two alternative stl files of a plate to 3D printed as cover such a recess; One plate is neutral, the second one has 'Magic CUBE' words engraved.
Of course, you might consider using the neutral one as baseline to personalize your own Cubotino 😊.

## 29) Collection of robot's pictures:

Picture 1


Picture 2



Picture 3