How to make CUBOTino base version:

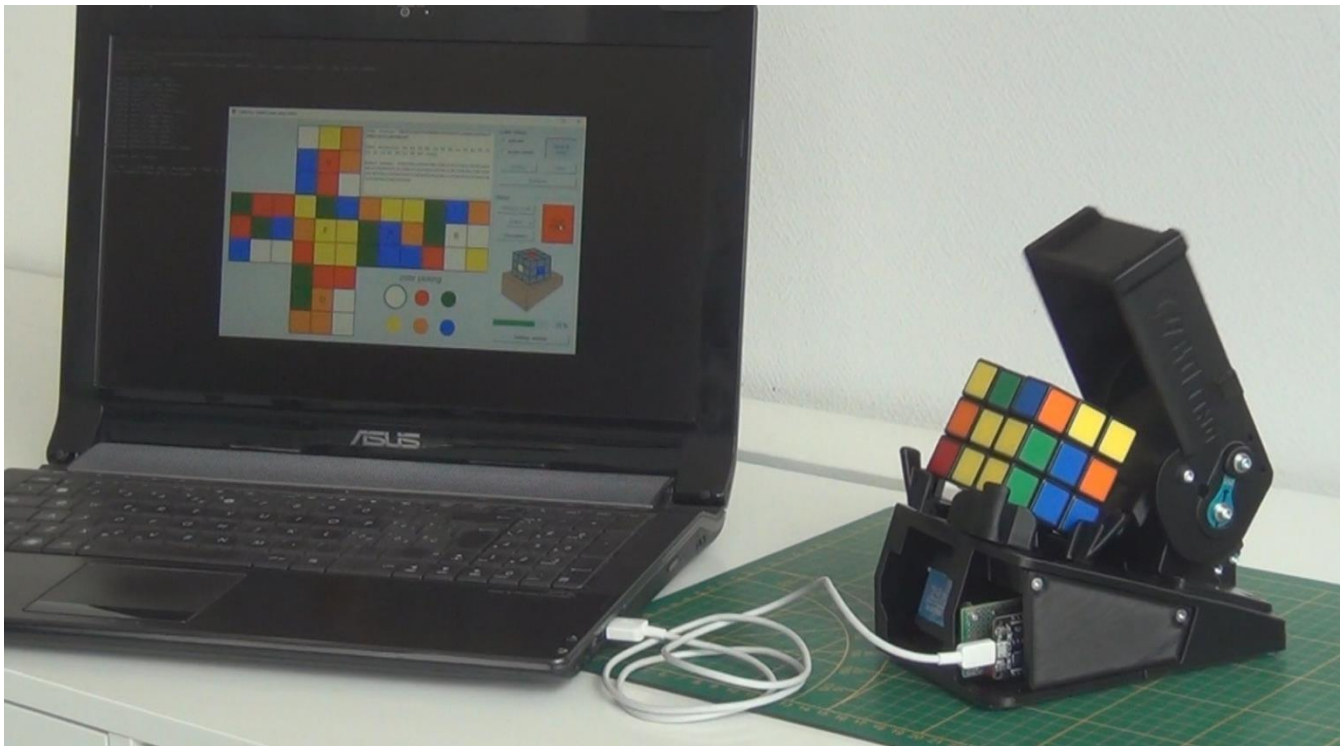
A small, simple, inexpensive Rubik's cube robot solver

https://www.instructables.com/CUBOTino-a-Small-Simple-3D-Printed-Inexpensive-Rub/
https://github.com/AndreaFavero71/CUBOTino_base_version

Andrea Favero, Groningen (NL)

V4.5 17 January 2023 (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 😊.

The Autonomous version of this small robot is visible at: https://youtu.be/dEOLhvVMcUg

1. Instructions sections

This document is divided in 6 main sections, and about 30 chapters.

The sections and chapters are ordered to facilitate Makers who want to start DOING, before having read the complete manual first 😊.

Use the Summary links to quickly reach the chapters

2. 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 must 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.

3. Manage your expectations

Be prepared the robot won't magically work right after assembling it: Tuning is expected!
This has to do with differences between each robot, in particular:

- servos
- arm positioning to the servo
- cube dimensions
- print quality

But hey, don't worry …. Other makers have successfully tuned their own Cubotino, and you will too 😊

… and now, let's start!

Summary

4. Project repository

All needed files are available in GitHub: https://github.com/AndreaFavero71/CUBOTino_base_version
It is necessary to copy (or clone) the complete project repository to your computer

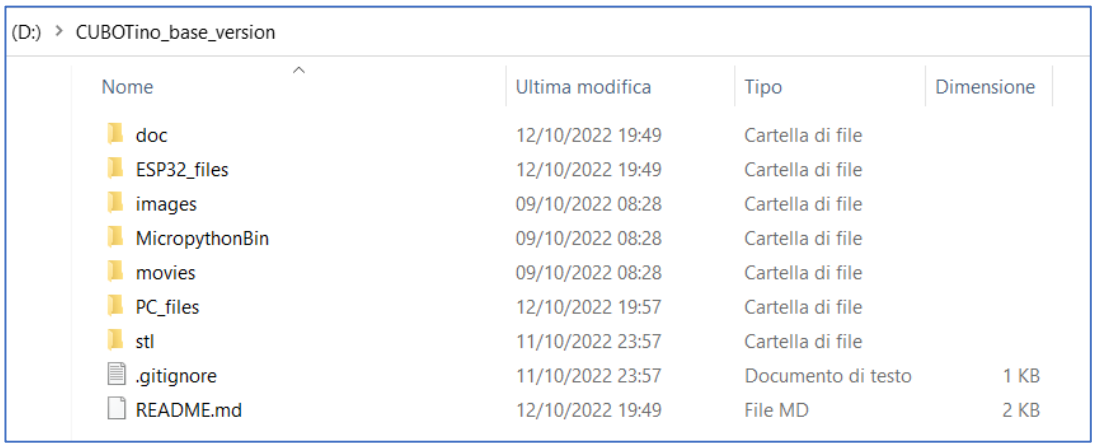
There are a couple of methods you can use, depending on your preference/skills:

1. Git method
2. Zip method

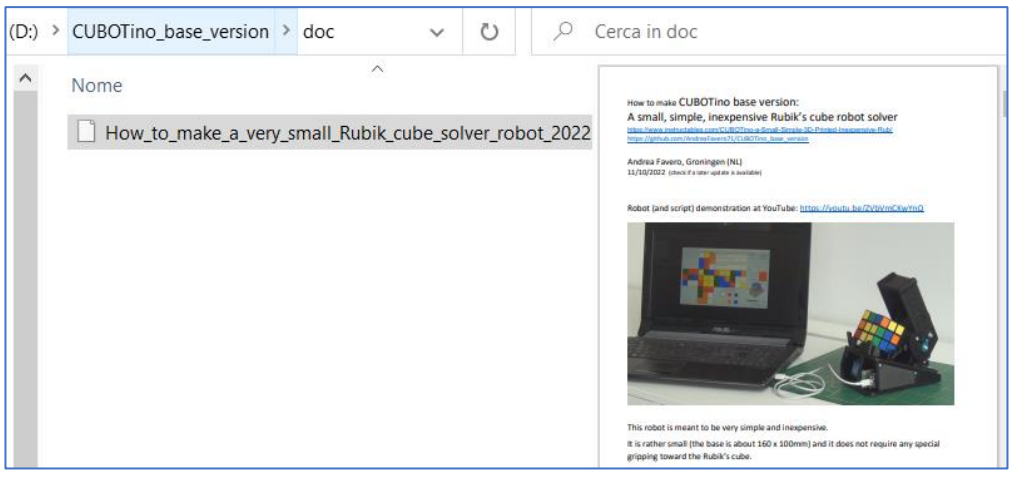
The git method makes easier to update the local repo, in case future updates will be made on the remote repository (GitHub); This method requires some little confidence with the git system.

Git method (for people being comfortable with git system)**:**

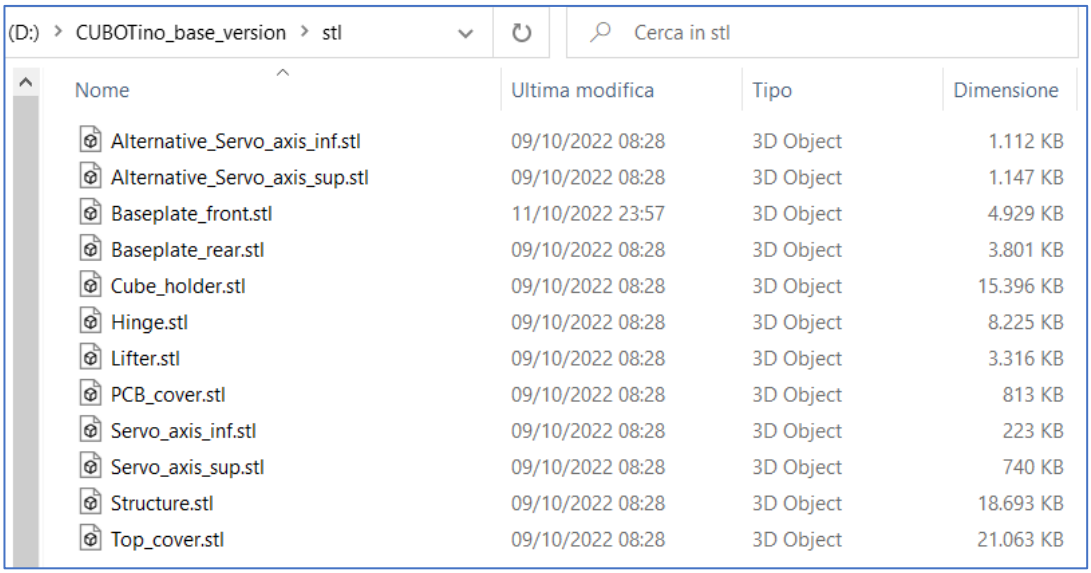
1. Ensure you have git installed in your PC
   In my case, that I use conda, I installed it via: ***conda install -c anaconda git***
2. Enter the folder where you'd like to have the CUBOTino repository.
   In my case, I wanted to install from the root of d: disk, therefore I entered ***d:*** to reach d:\>
3. Clone the repository, via ***git clone https://github.com/AndreaFavero71/CUBOTino_base_version.git***
   In short time the robot repository is cloned in your PC.
4. Enter the folder ***cd CUBOTino_base_version***. Be noted the folder name doesn't end with "-main".
5. The cloned repo contains all the needed files:



6. Under the /doc folder you'll find a copy of these instructions:



7. Under the /stl folder you'll find the stl files for the 3D prints:



How to update a local repository, originally made via git

1. From a shell enter the local repository folder, in my case: d:/CUBOTino_base_version
2. Reset your local repository: ***git reset - -hard*** (there is a space between the two "-")
3. Update the local repo: ***git pull***

Your local repo is now updated, by only using two commands !!!

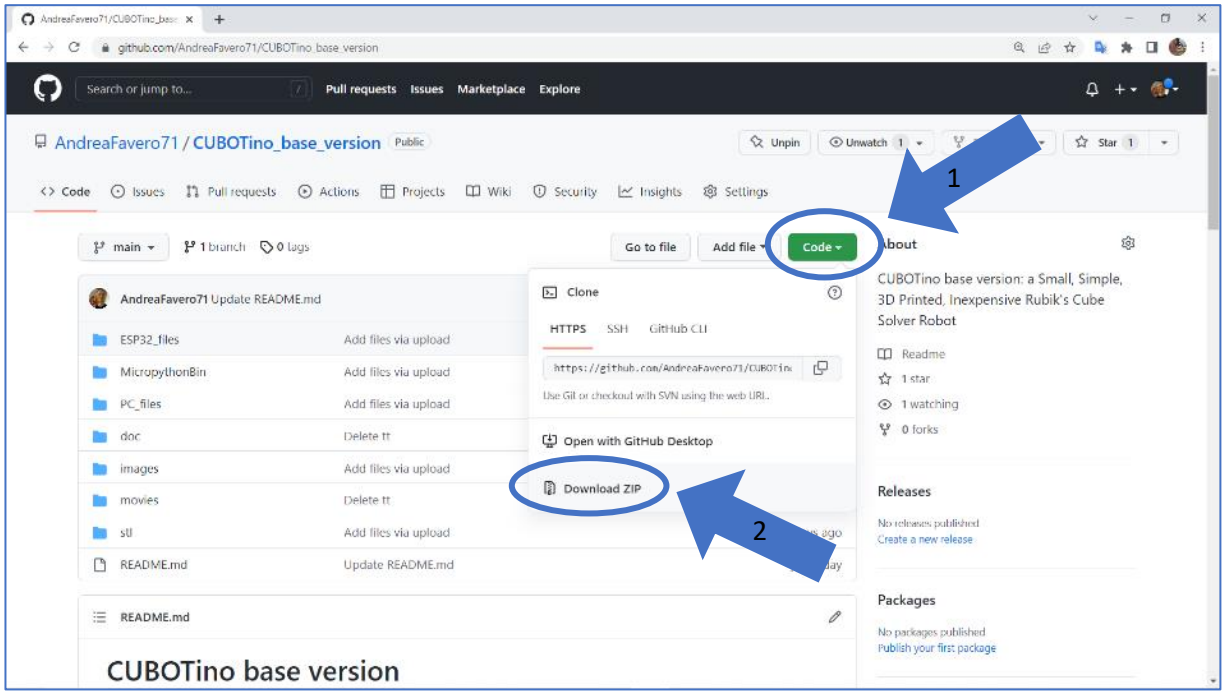
Notes:

7. Cubotino_GUI.py automatically makes backup copies of your personal servos and cam settings.
8. The repo update via git does not overwrite your "personal" files ☺, like:
    a. Servos settings
    b. Cam settings
    c. Robot logged data
    d. Cube status images
9. Be aware that other files, eventually modified by you will be overwritten

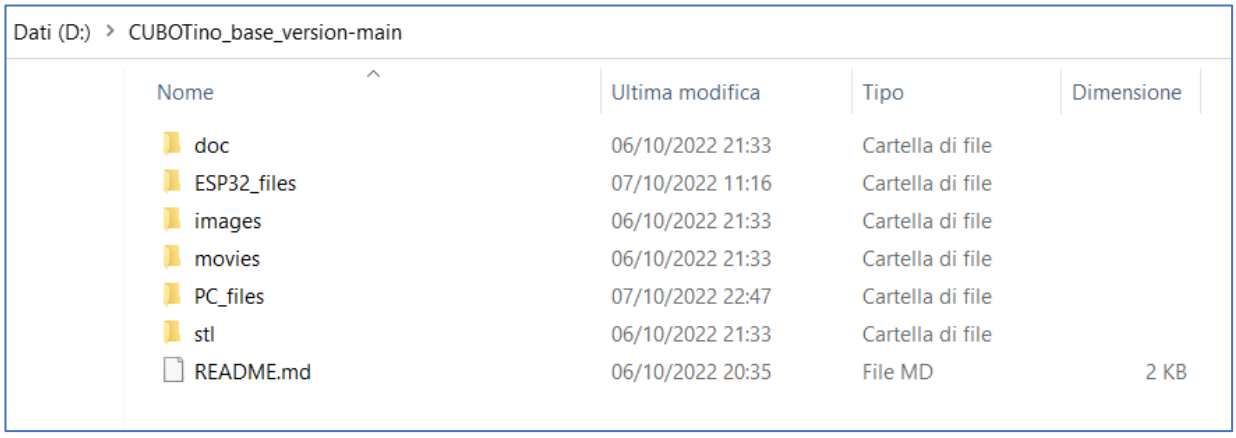
Zip Method:

This method is rather simple to create your local repo, but it will be less convenient for future updates.

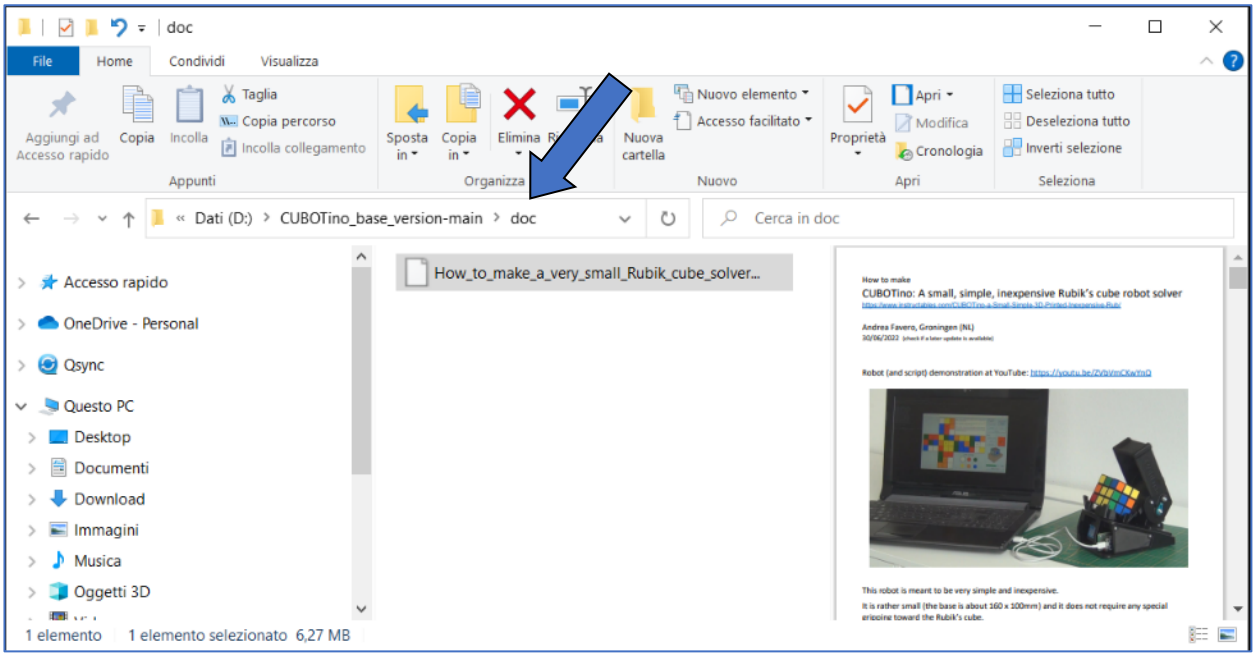
1. Open the online repo: https://github.com/AndreaFavero71/CUBOTino_base_version
2. Select Code / Download ZIP



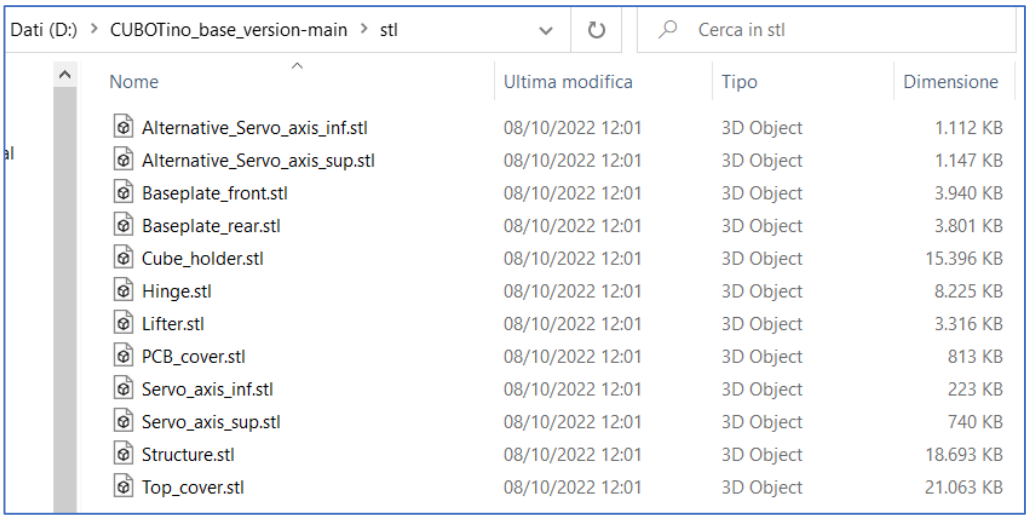
3. The ZIP file will lend in your download folder
4. Un-ZIP the complete content to a known location (in my case I chose the root of my d: drive)
5. The main folder name is **CUBOTino_base_version-main.** Be noted "-main" is added at the end when using the ZIP method; this is not as issue, as long as you are aware of that. You might also decide to rename the folder by removing the "-main" part. The folders' structure will look something like:



4. Under the /doc folder you'll find a copy of these instructions:



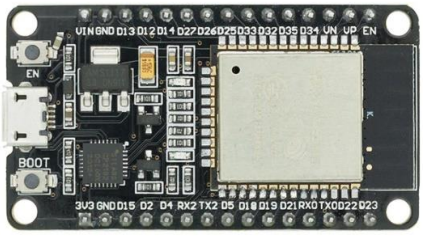
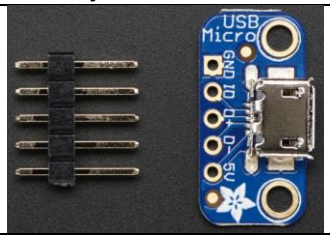
5. In the /stl folder you'll find the stl files for the 3D prints:



How to update a local repository, originally made via ZIP method

1. Before updating a repo made via the ZIP method, move below files to a location "outside" your local repo folder to prevent those files to be overwritten:
    a. Servos settings (from /PC_files and from /ESP_files)
    b. Cam settings
    c. Robot logged data
    d. Cube status images
2. Repeat the steps as per ZIP method
3. Copy back the files previously moved

4. Supplies

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". Pulse width 1 – 2 ms	https://www.aliexpress.com/item/32298149426.html?gatewayAdapt=glo2ita&spm=a2g0o.9042311.0.0.5d1e4c4d14Qjaz	25 (2 servos + 2 arms)	
2	Alternative servo (Better choice, see note below) Servo DS3225MG (**180deg** 20Kg metal, **Pulse width 500 µs to 2500µs**) and metal arm "25T"	https://www.amazon.com/ZOSKAY-DS3218-Digital-Waterproof-Control/dp/B01MU7TQV8	32 (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	

The alternative servo, proposed above, is better on: angle resolution, rotation range, and noise….

1. Servos with Pulse Width from 500 to 2500 µs have higher resolution making easier the robot tuning.
2. In addition, the embedded chip accepts Pulse Width slightly outside the range, increasing the rotation range.

Unfortunately, I discovered this model only recently, and only one of my robots have this servos type.

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. 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 printing a personalization cover for that recess.

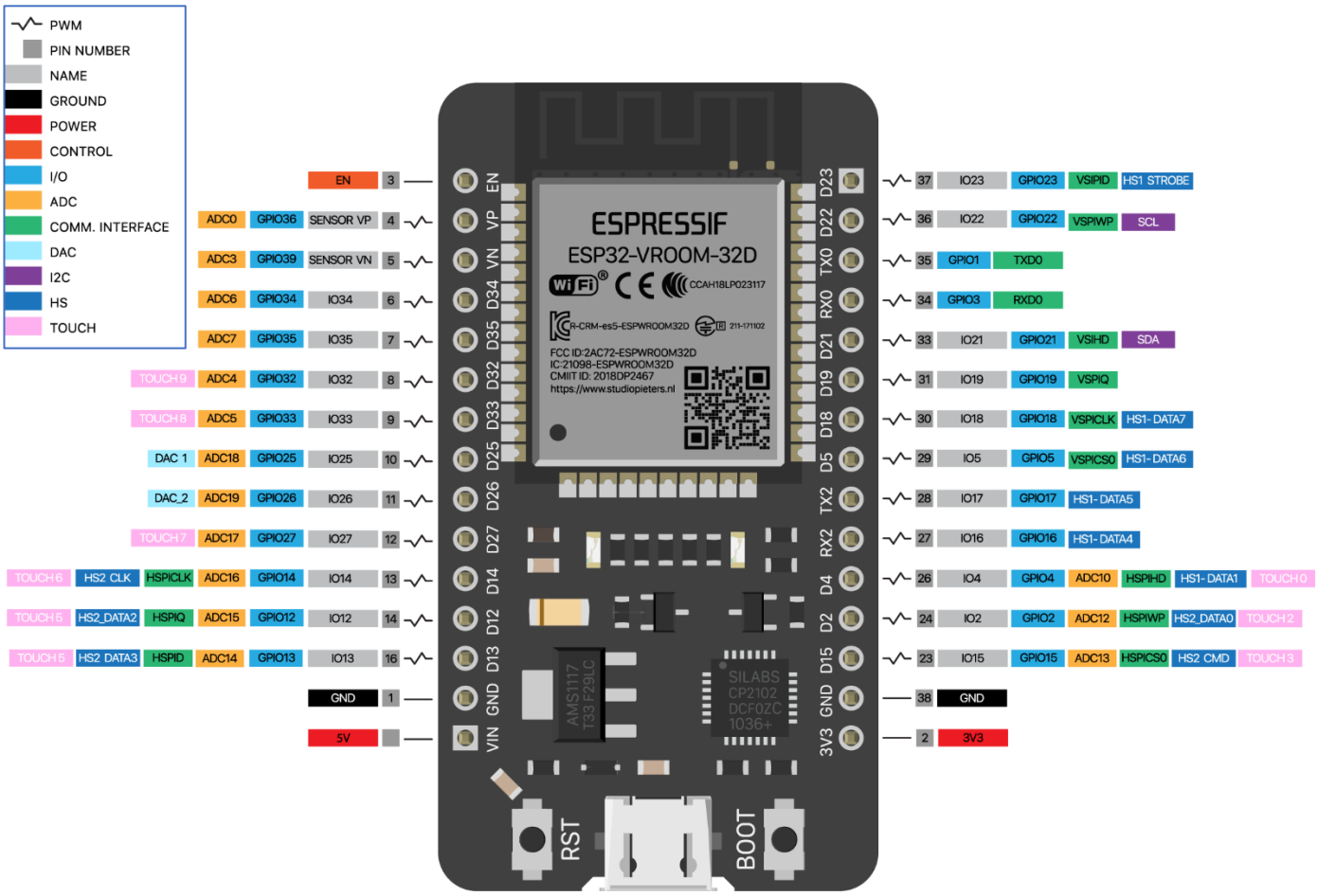
Off course some other common materials are needed:

- 1 x USB-Microusb cable with data lines, for the communication PC ← → ESP32
- 1 x USB-Microusb cable (with or without data lines), to energize the servos
- Eventual phone charger (5V 2A, not of the smart type), if the current drawn by the servos affects your PC.
- Female headers (plastic part with height ca 8.5mm)
- wires, solder and solder device, tire wraps, self-adhesive rubber feet, etc.

Do not forget a Rubik's Cube 😊

The cube size (side) should be between 56.0 and 57.5 mm.
Please give a look to suggested cube described at "A convenient cube", later in these instructions.

5. ESP32 dev. board (30 pin) pinout



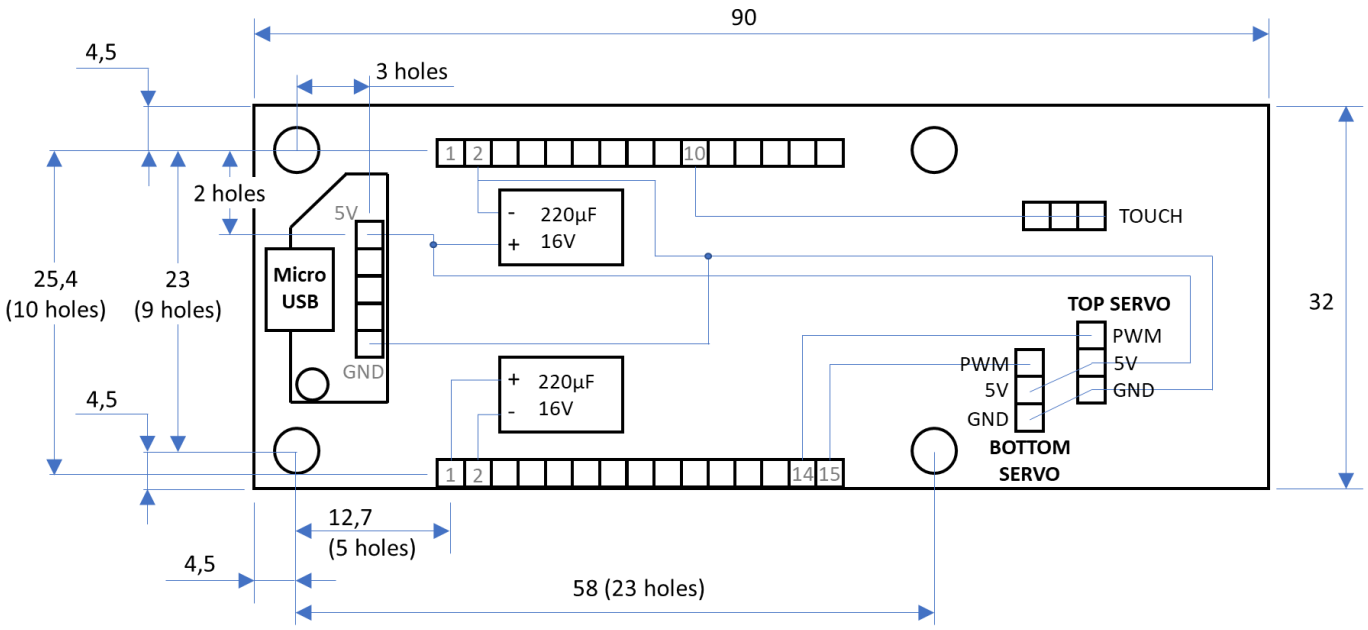
Used pins:

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

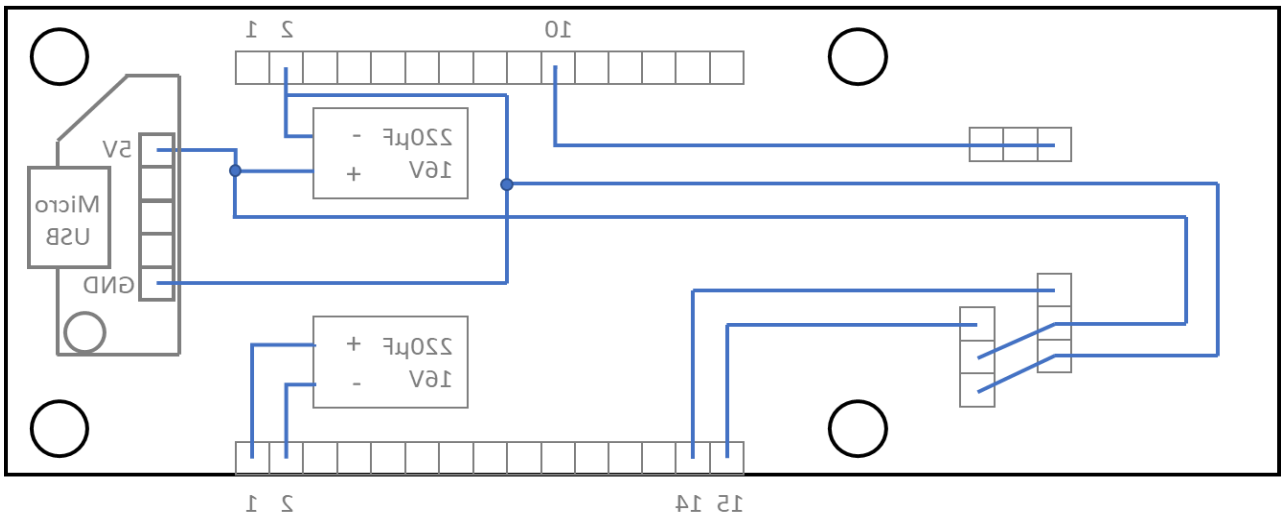
Notes:
Added two capacitors (16V 220Uf) at Vin and 3V3 vs GND

6. Connections board

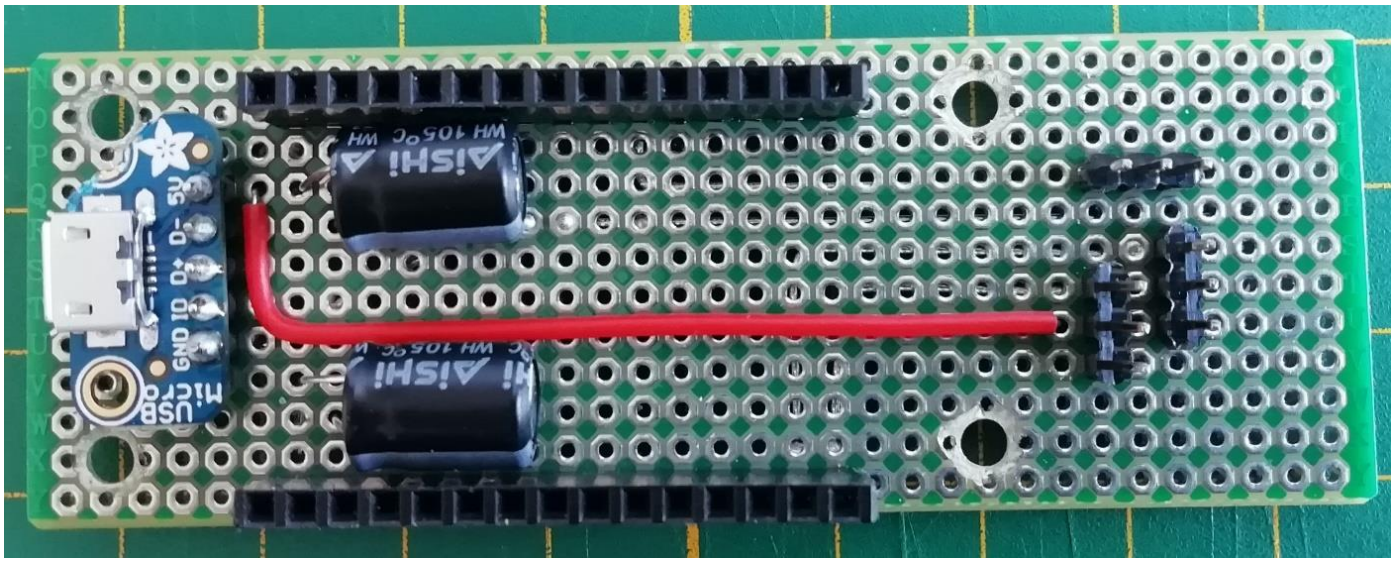
Top view:

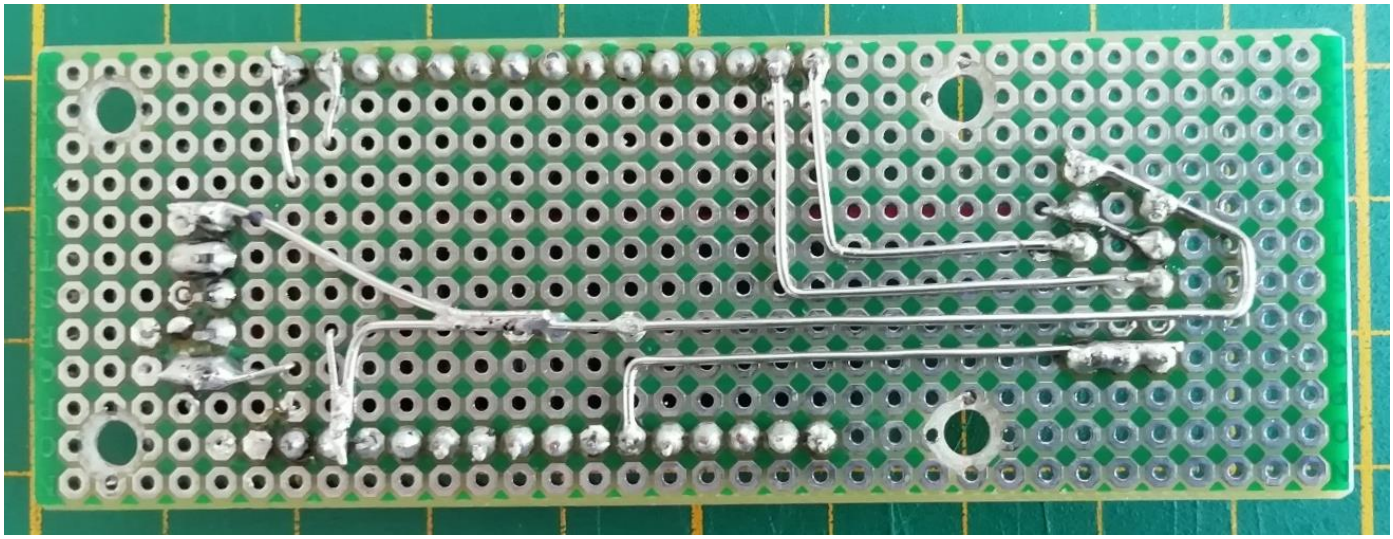


Bottom view:

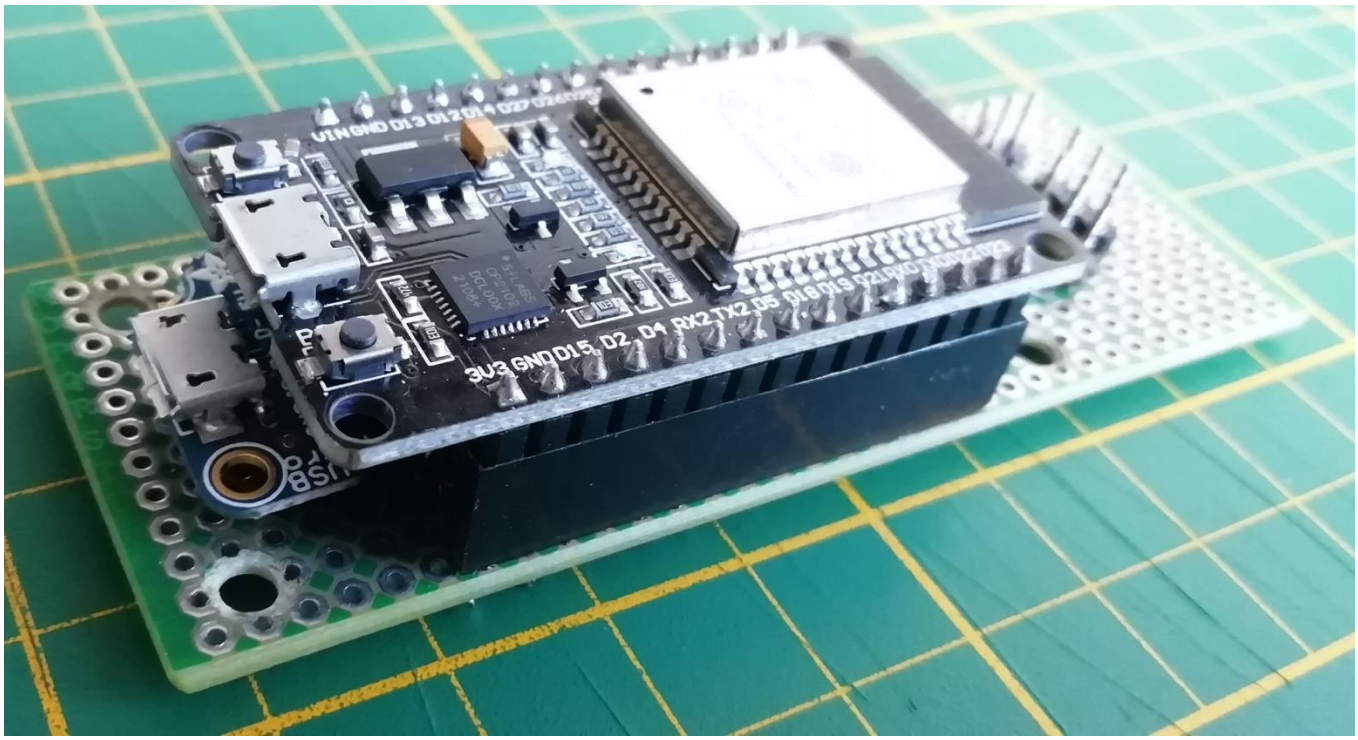


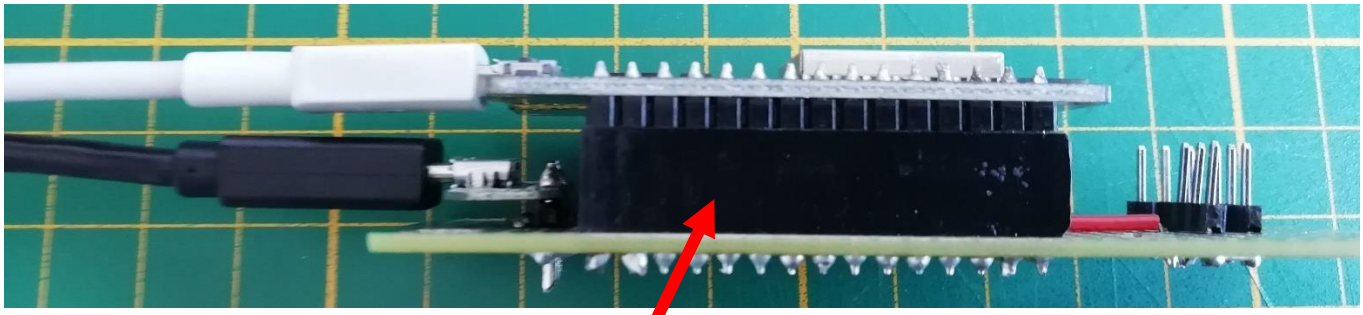
Before soldering, cut the corner of the microUSB breakout board in front of the connections board hole





Connect the ESP32:



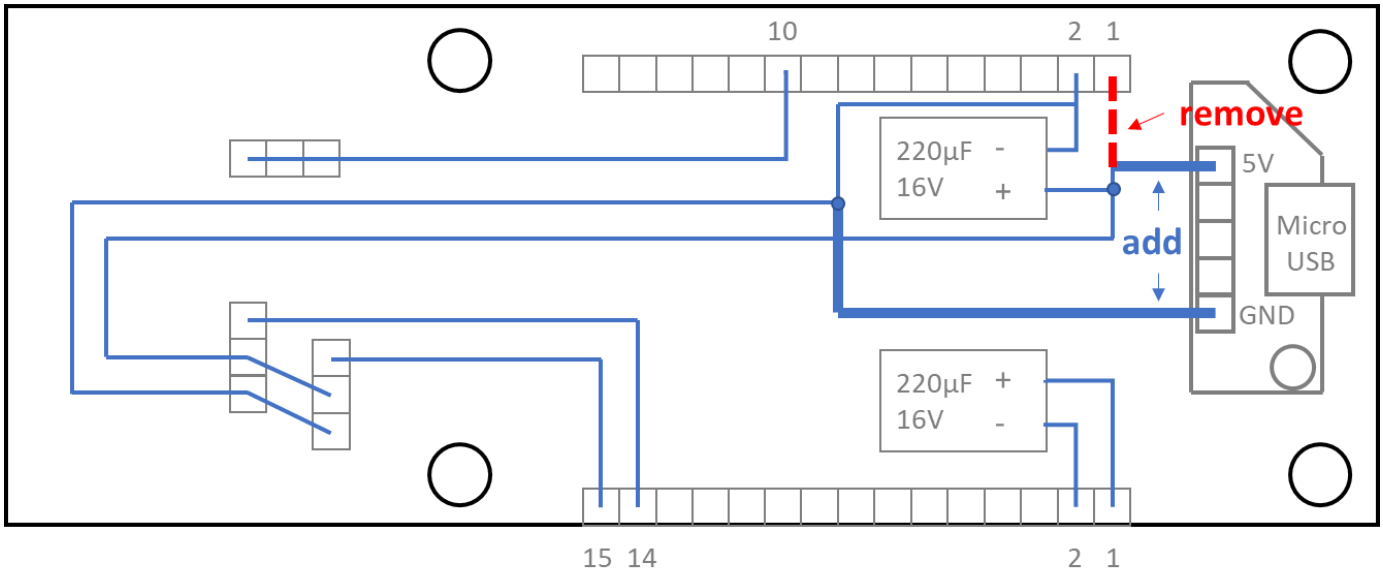


Female header: height of ca 8.5mm (plastic part)

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:

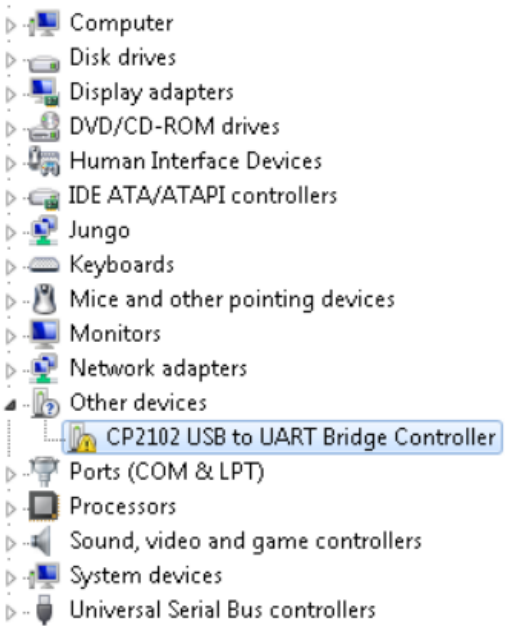


7. Setting up the ESP-32 microcontroller

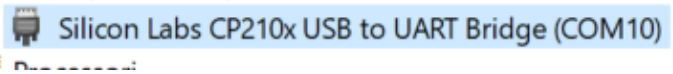
Step1: Before starting, do check if you've the driver for USB to UART communication installed in your PC:

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:



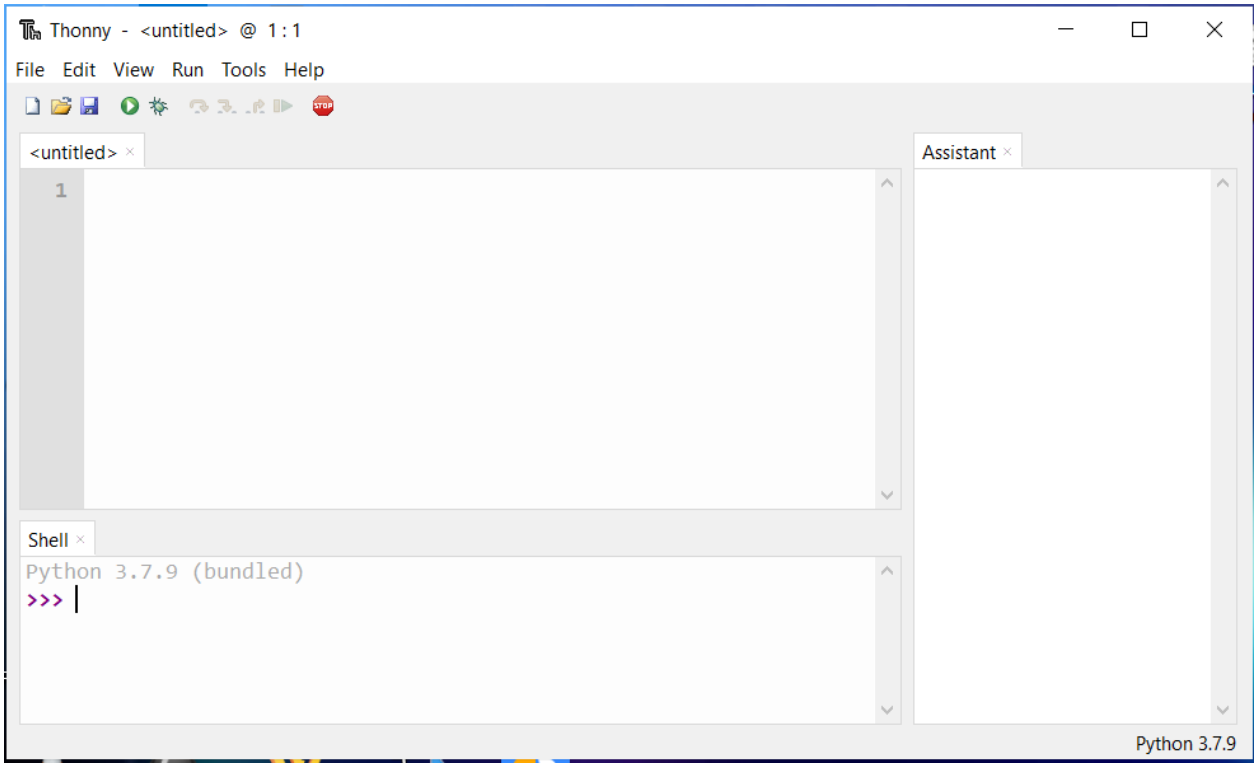
6) The COM number might be different, at it depends on the system you have

Step2: Install Thonny in your PC in case not yet available

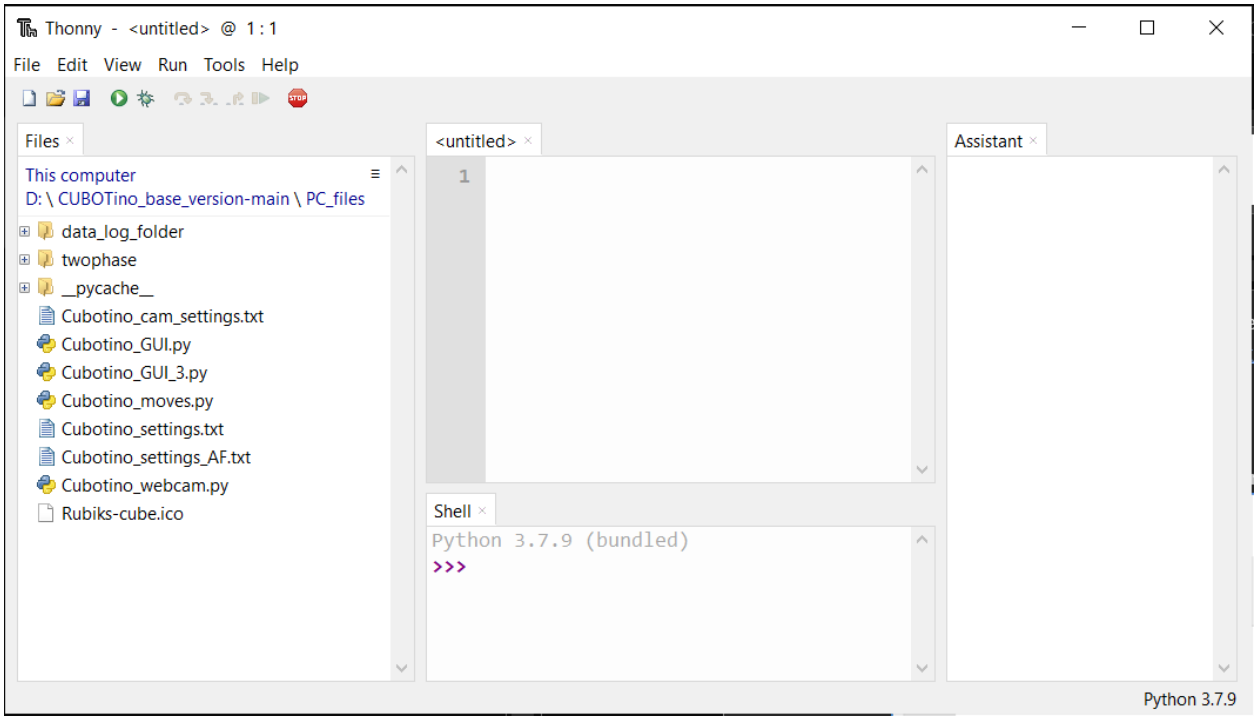
Download it from: https://thonny.org/

Step3: Flash the firmware to the ESP32

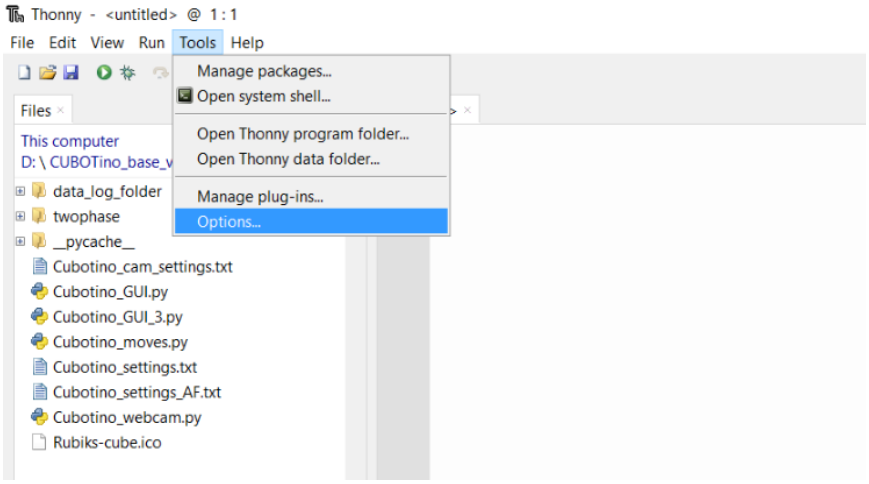
- Connect the ESP32 to your PC
- Open Thonny, that initially will appear like:



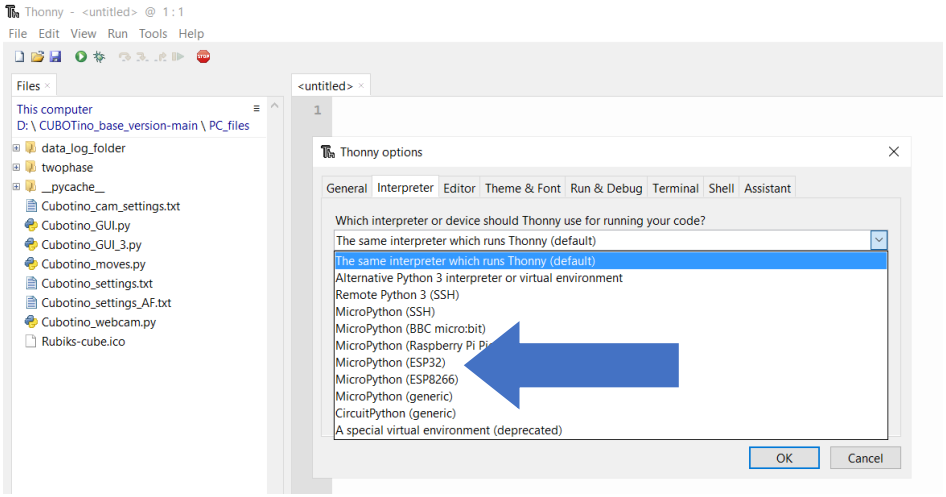
- Select View / File that shows the active folder; Navigate to the project folder in your PC:



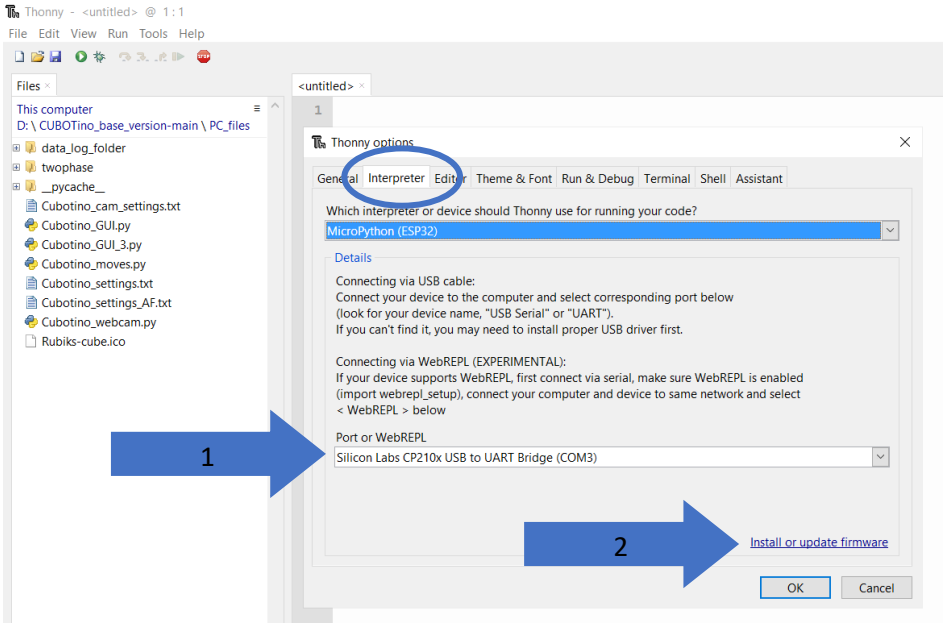
- Select Tools / Options ….



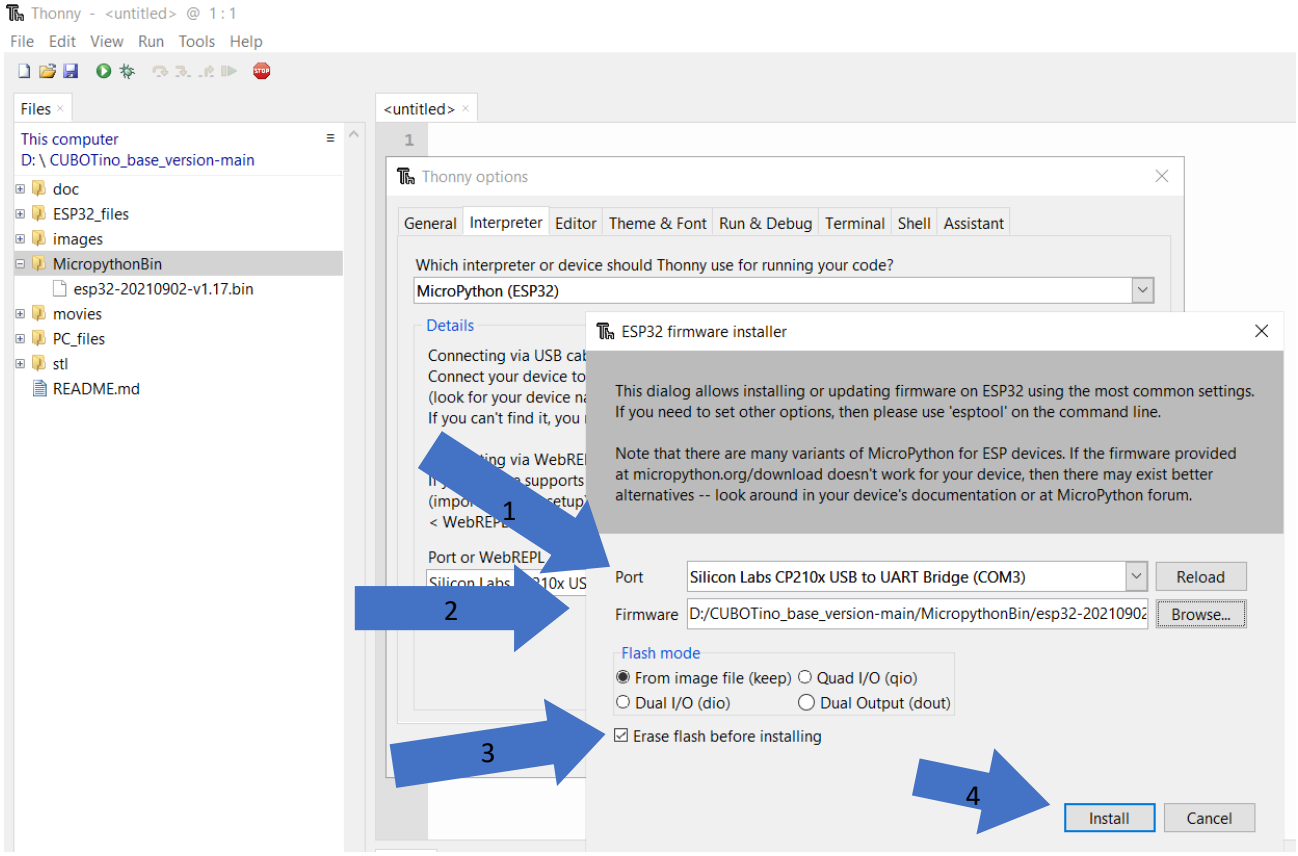
- Select the second tab (Interpreter), open the list of interpreters, and select “MicroPython (ESP32)”



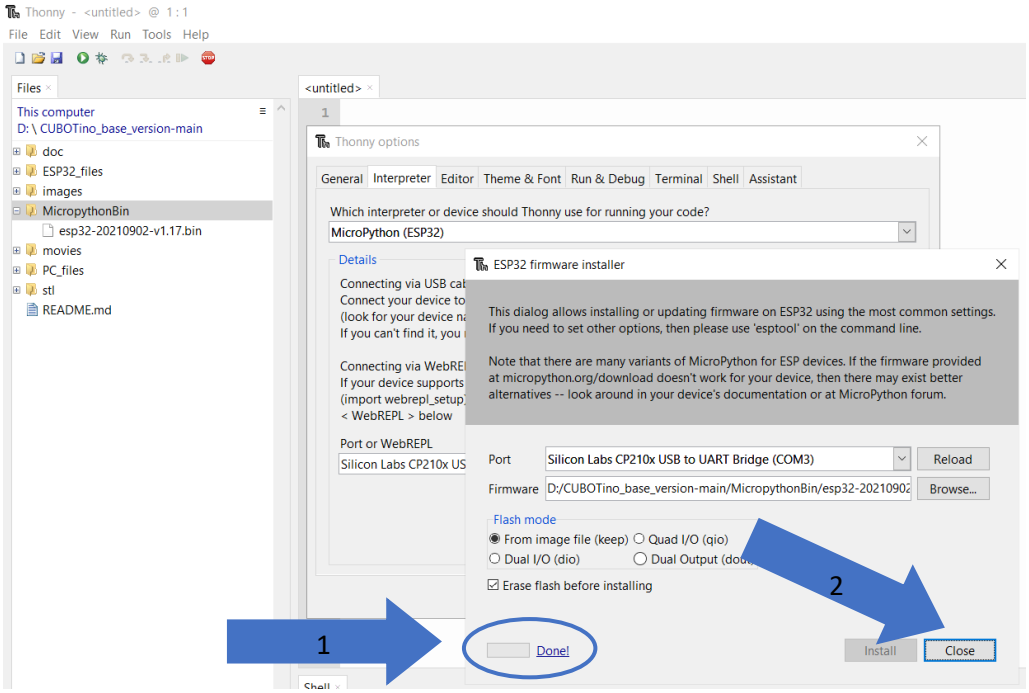
- Select the COM port the ESP32 is connected to
- Select “Install or update firmware”



- Re-select the COM port the ESP32 is connected to
- Browse for the MycroPython BIN file in your local project repo, under the folder “MycropythonBin” and select the file “esp32-20210902-v1.17.bin”
- Verify the “Erase flash before installing” is checked
- Press “Install”

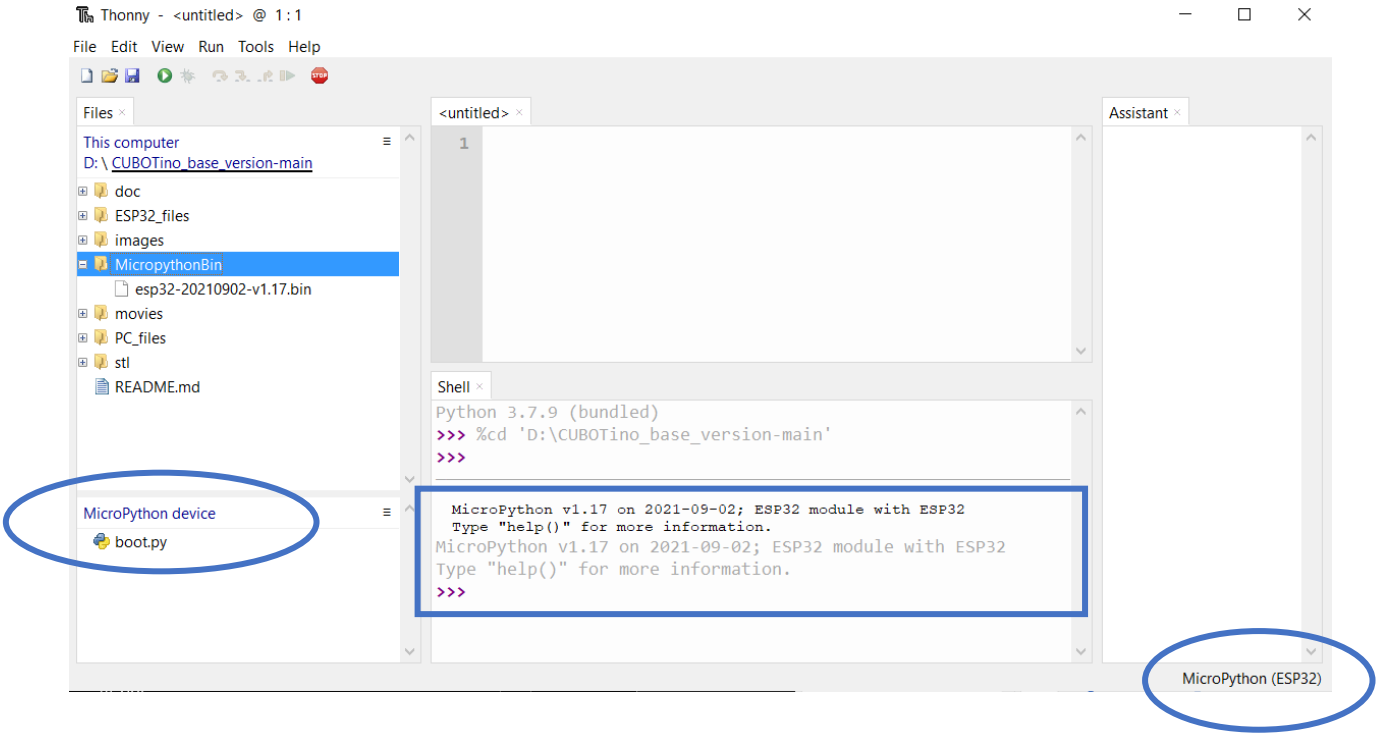


- Wait until “Done” is displayed, and close the window via the Close button



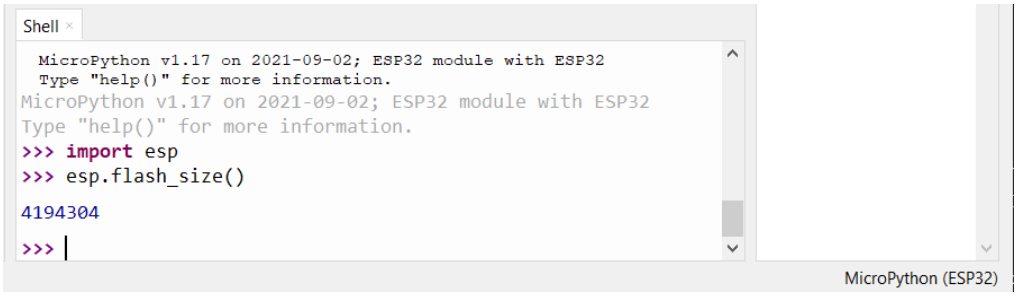
Step4: Check if MicroPython has been correctly installed to ESP32; Thonny should shows:

- "boot.py" file at the MicroPython device
- "MicroPython v1.17 on 2021-09-02; ESP32 module with ESP32" in the Shell part
- "MicroPython (ESP32)" should always appears on the bottom-right corner of Thonny (this is very useful when developing application with PC – ESP32 interaction, requiring editing files on both locations)



If you want to check the flash memory size, that should be around 4194304, type on the shell

- *Import esp*
- *esp.flash_size()*



Now MicroPython and the ESP32 have proved to work!

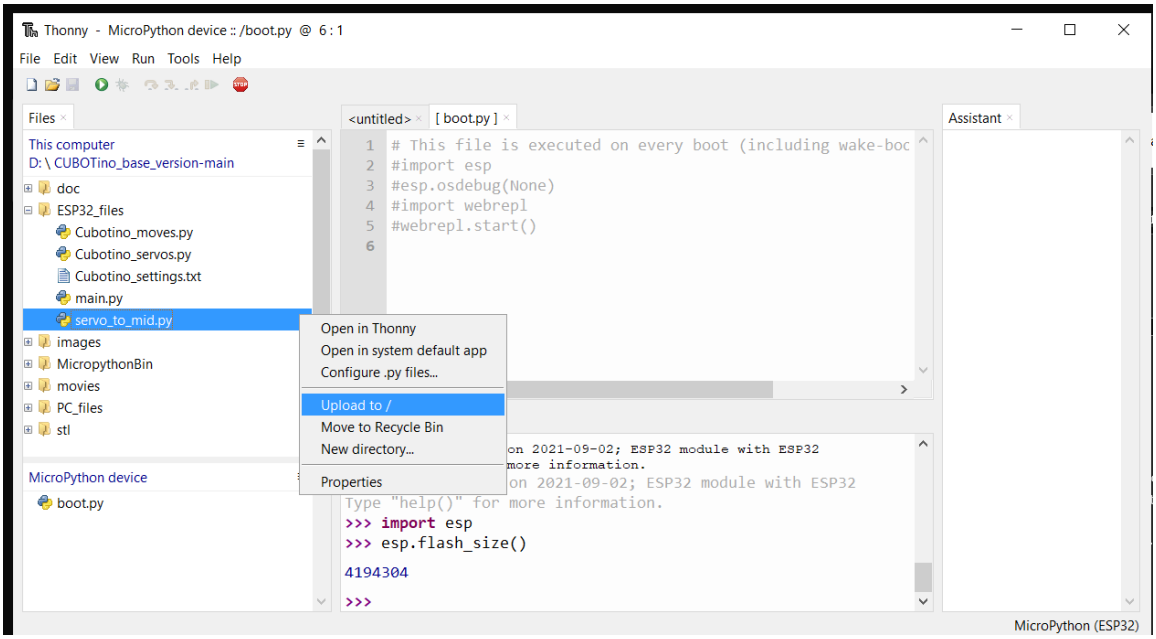
8. Servos test and set to mid position

Test objectives are:

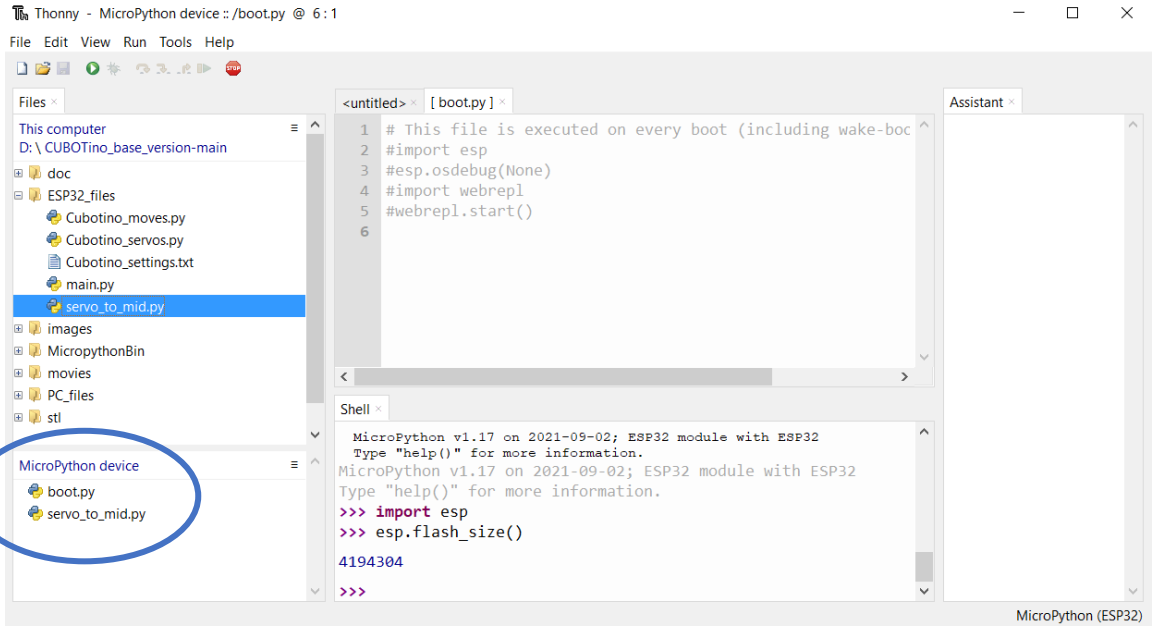
1. Check if the servos have at least 180degrees of rotation; In case no one of the two servos have about 190deg rotation, check “how to increase the servo rotation range” at troubleshooting.
2. Identify which one of the two servos have the larger rotation range; This must be identified and associated to the cube_holder.
3. Get the servos set to their mid position; Connect the arm on the one for the Upper_cover (see picture below)

ONLY one file is necessary for the servos test: “servo_to_mid.py”:

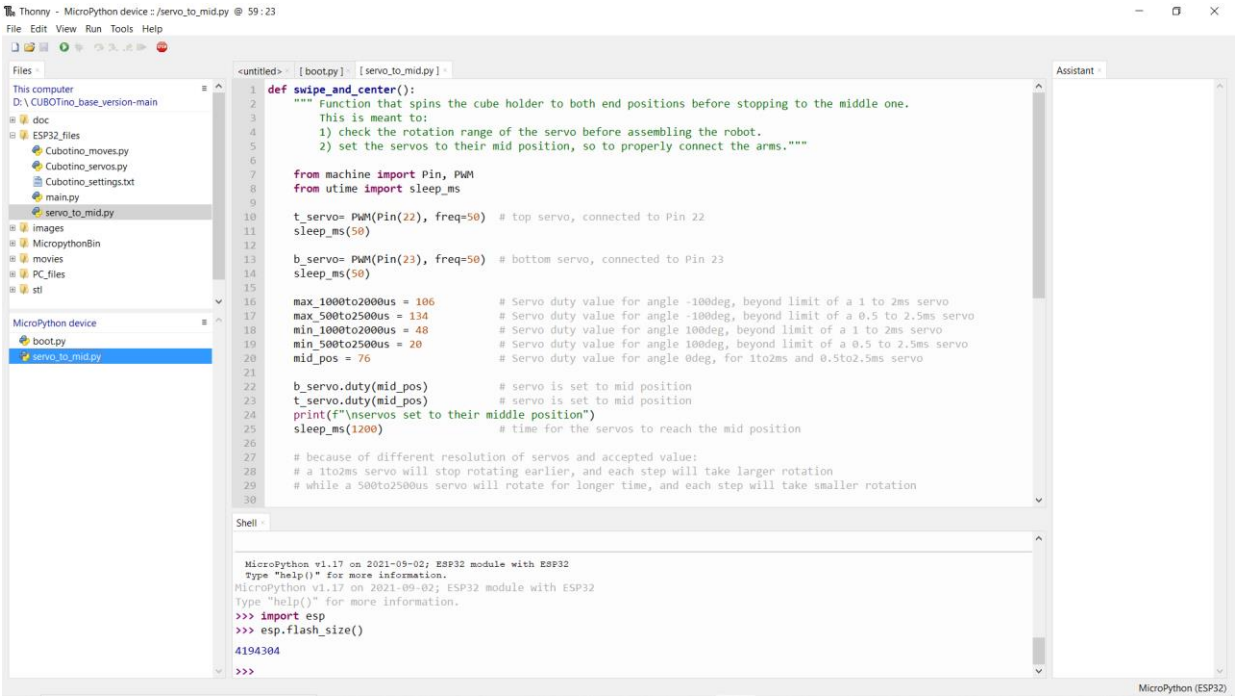
- Navigate the PC repository and open the folder /ESP32_files
- Right-click the “servo_to_mid.py” file and select “Upload to /” (means it will land to the ESP32 root)



The “servo_to_mid.py” file is now visible at “MicroPython device” files sections:

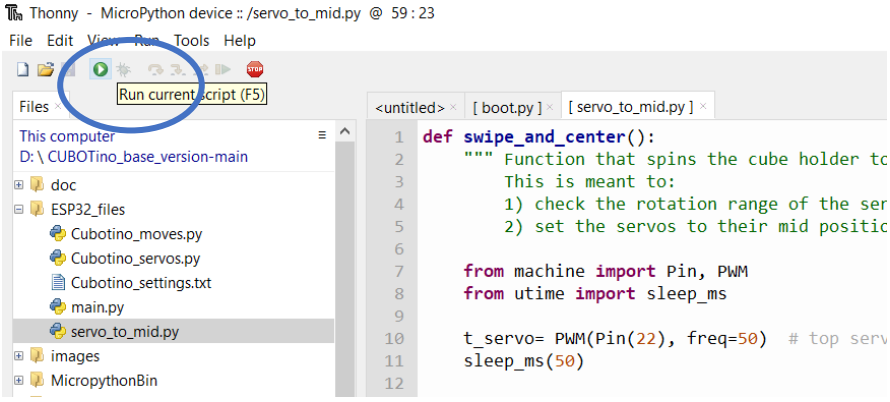


Open the “servo_to_mid.py” file in the ESP32, via a double-click:

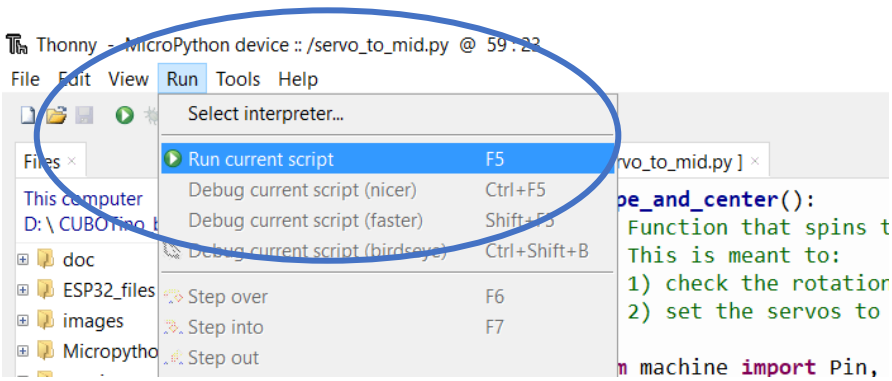


Start the test:

- Connect the power supply to the Connections_board (2nd microUSB connection)
- Connect the two servos to the Connections_board
- Connect an arm to both the servos outlet gear (no need for screws)
- Hit the “PLAY” icon, or select Run and Run current script, or press F5



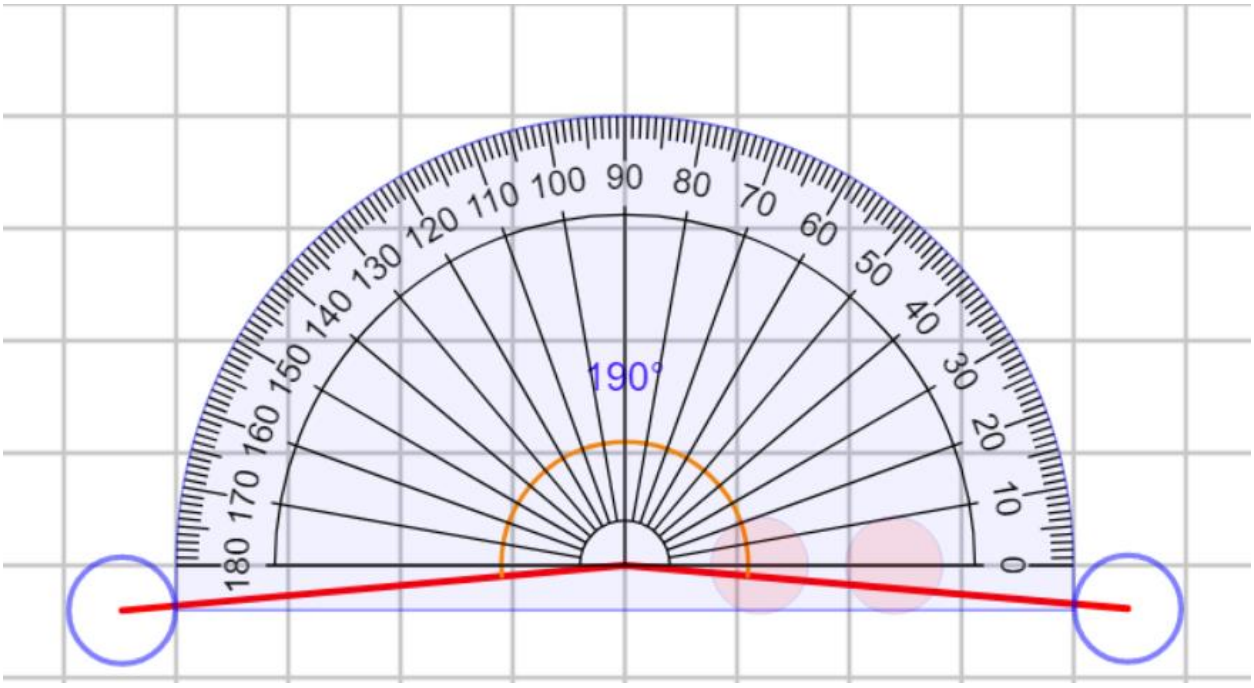
or



In about 22 seconds the servos will swipe (in steps) the full rotation range; Servos stop to their middle position at the swiping end.
Re-position the servo arm as per below picture:

Repeat the test multiple times and try to estimate if one of the two servos has about 190deg rotation, or more. The servo having the largest rotation range, and at least 190degrees, must be associated to the cube_holder.

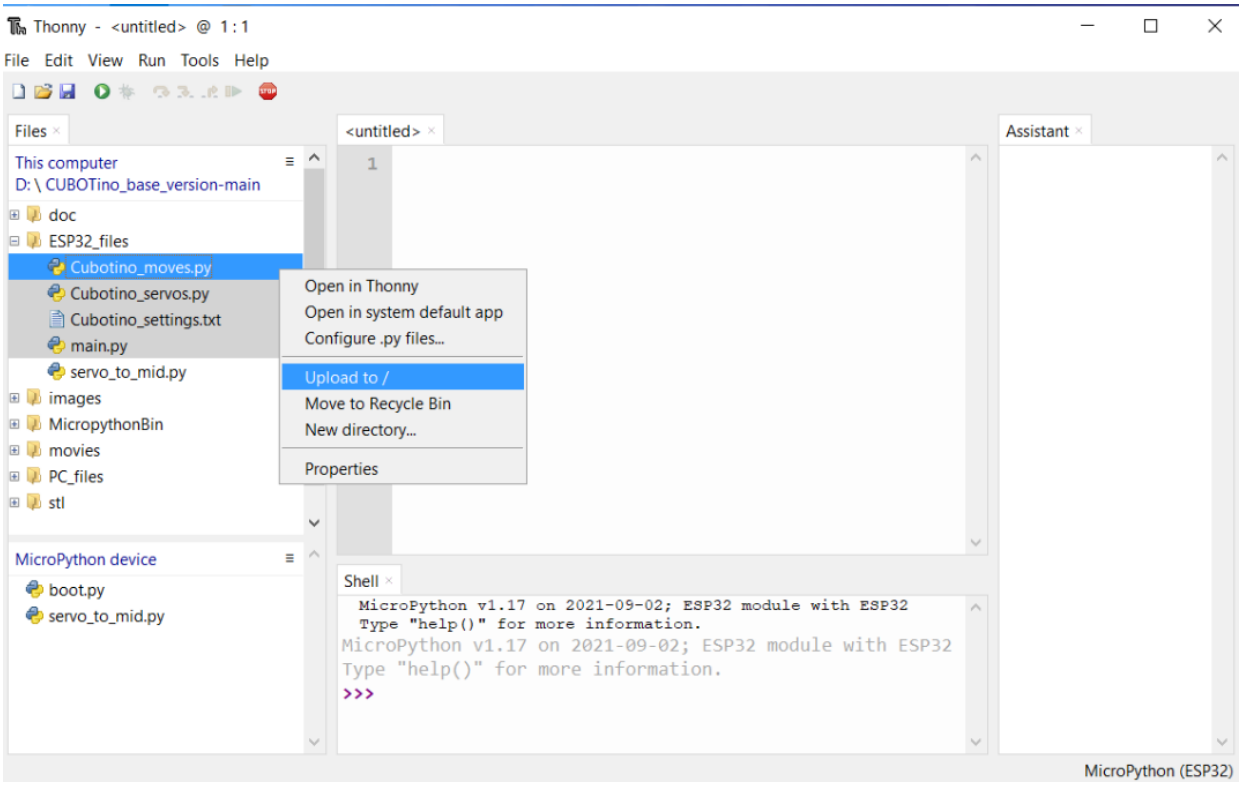
The angle can be evaluated by printing a protractor:



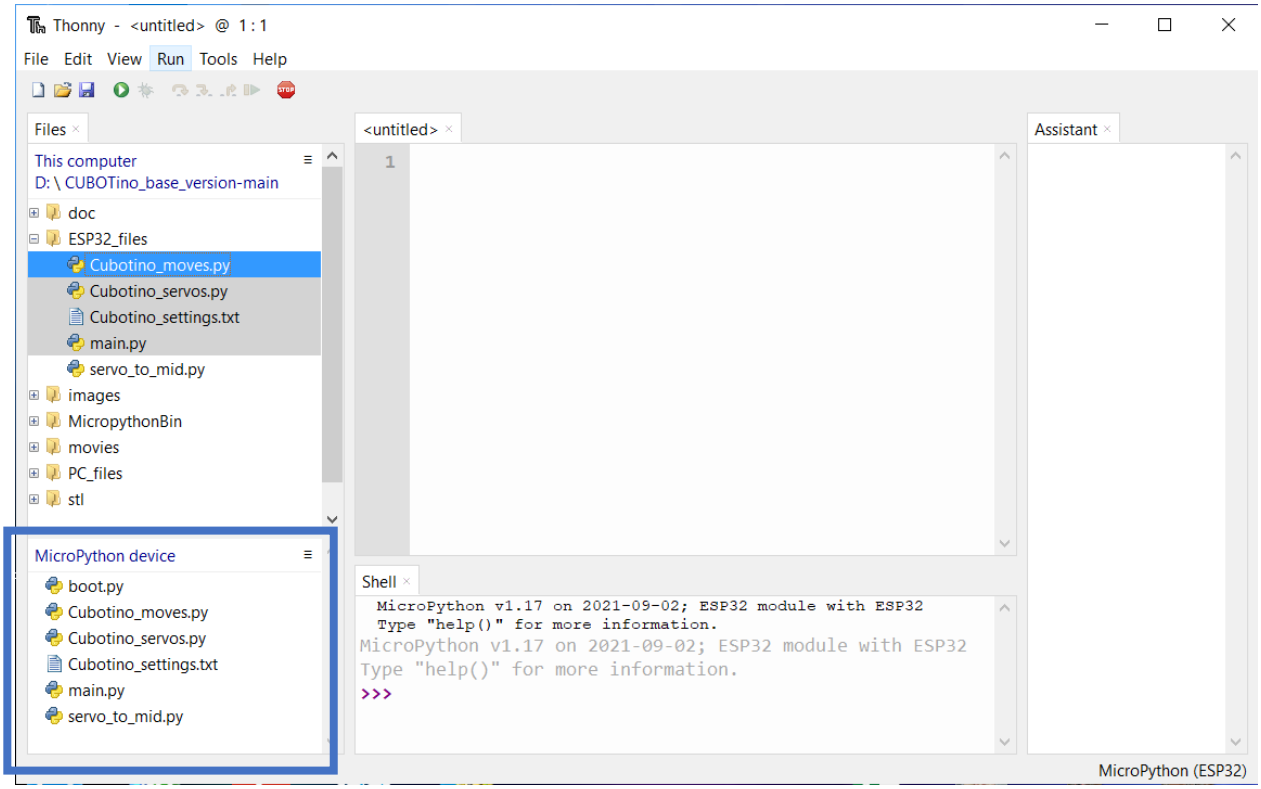
There also are online transparent protractors to apply over a picture, also apps for the phone…. Here it comes to your creativity!

9. Upload all files to the ESP32

- Navigate the PC repo to the ESP32_files
- Select all the files but “servo_to_mid.py”
- Right-click and select the “Upload to /“



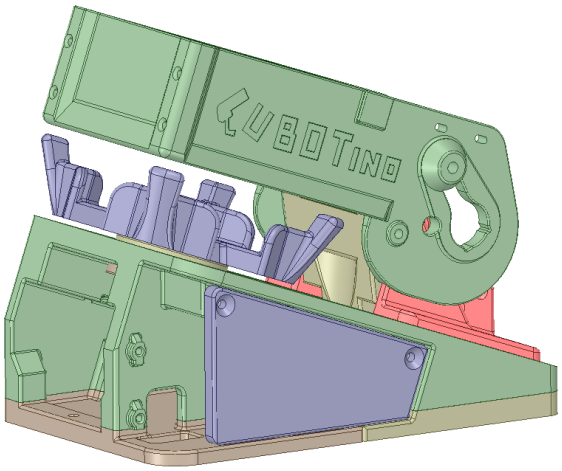
- The selected files lend to the ESP32 root



10. 3D printed parts

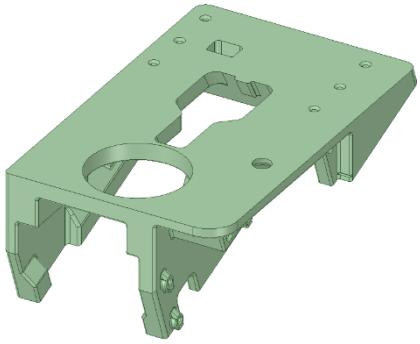
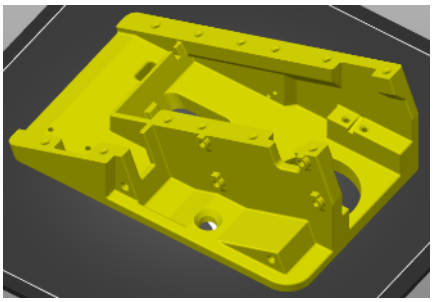
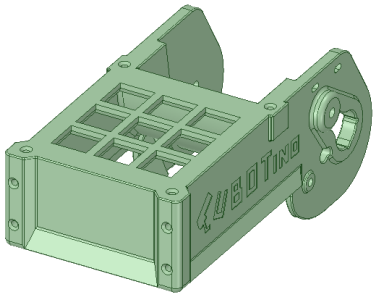
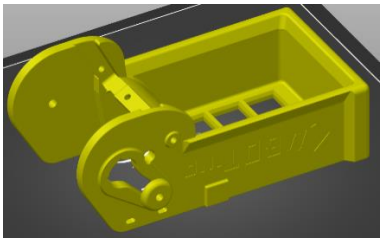
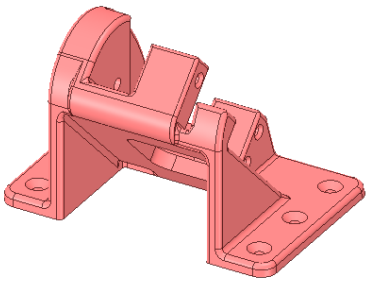
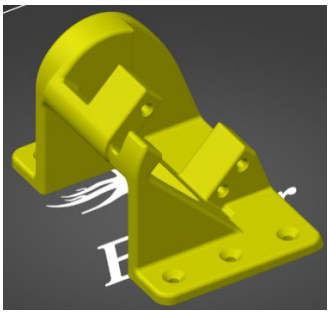
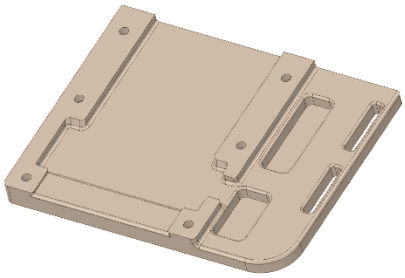
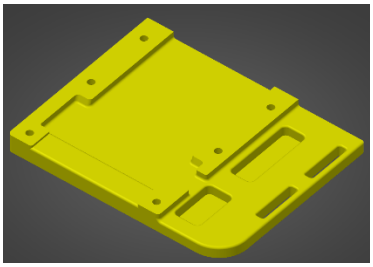
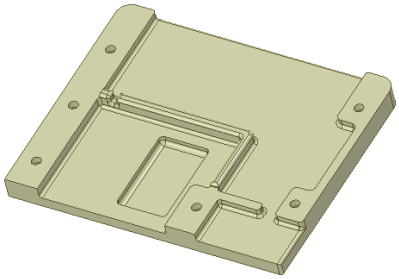
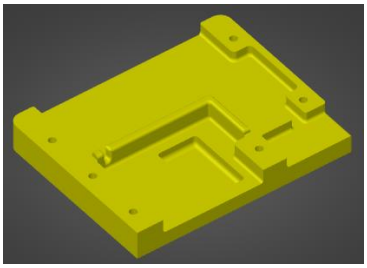
See notes below for filament quantity, and printing time …

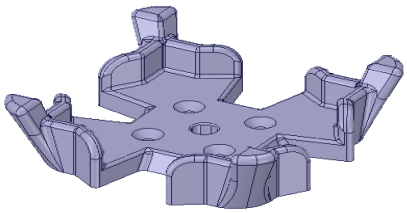
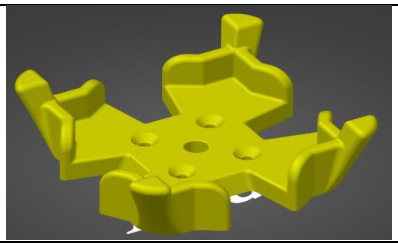
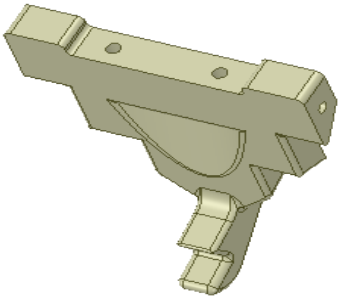
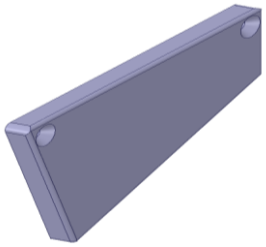
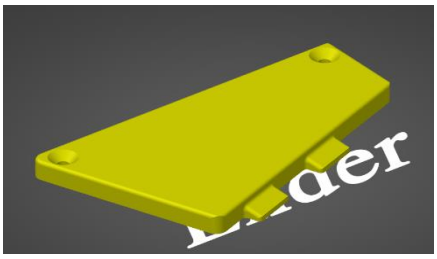
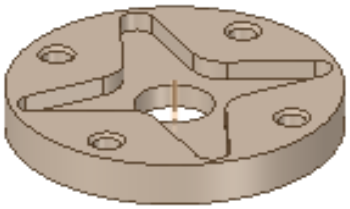
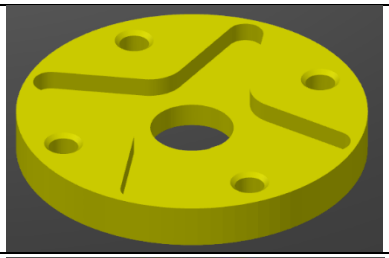
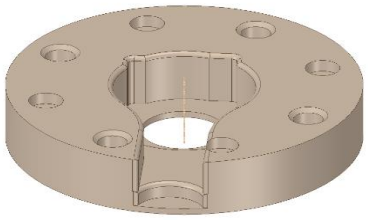
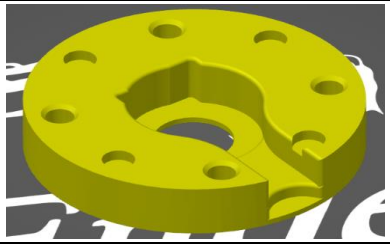
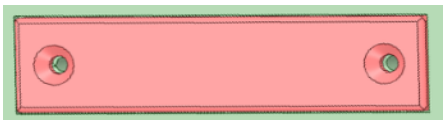
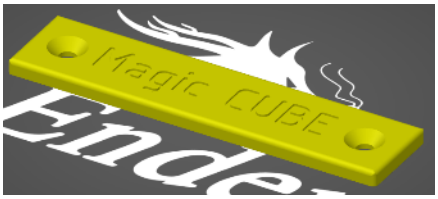
Ref	Part	Filament		Printing time
		meters	grams	
1	Structure	49	145	14h33m
2	Top_cover	39,5	117	12h26m
3	Hinge	17,2	51	5h37m
4	Baseplate front	12,2	36	3h33m
5	Baseplate back	12,6	37,3	3h36m
6	Cube_holder	11,6	34,4	3h36m
7	Cube_lifter	5,5	16,2	1h48m
8	PCB_cover	5,1	15	1h23m
9	Servo_axis_sup (or its alternative)	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:
   1.1 0.2mm layers
   2.1 Low speed (between 25 to 40mm/s for the external parts and 1st layer)
   3.1 4 layers on vertical walls
   4.1 5 layers on horizontal walls
   5.1 30% filling
   6.1 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 at the /stl folder in your downloaded repository

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 Or Personaliz_plate01	 	

11. Assembly steps

Before assembling the robot:

- Make the ESP32 connections board
- Setup the ESP32 microcontroller
- Position the two servos output gear to their middle position (if not done yet, check Servos test above)

Assembly order (for the details see Assembly details, at the next chapter):

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

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

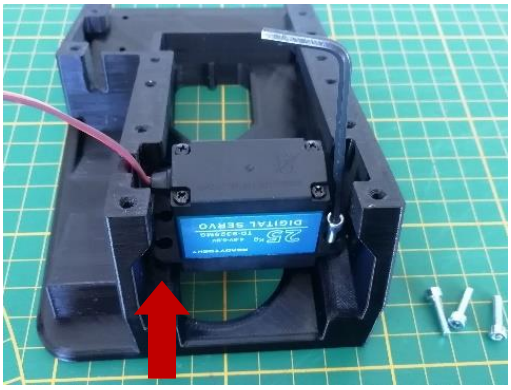
12. Assembly details

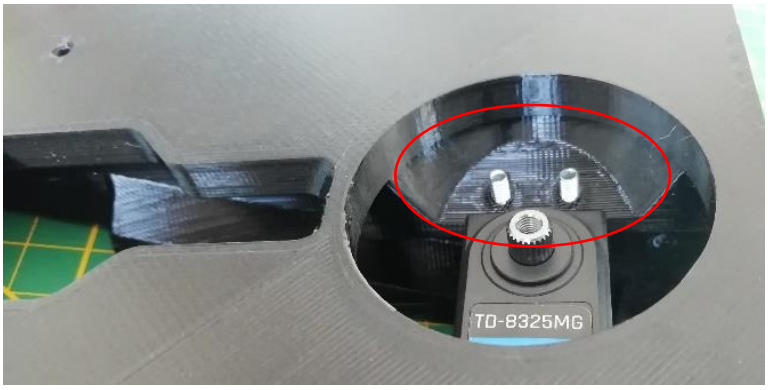
Step4 (Mount the bottom servo to the structure):
4x M3x12mm cylindrical head
Couple of washers

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

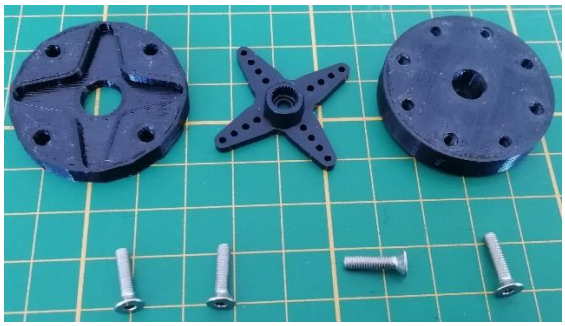
Couple of notes:

- 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.

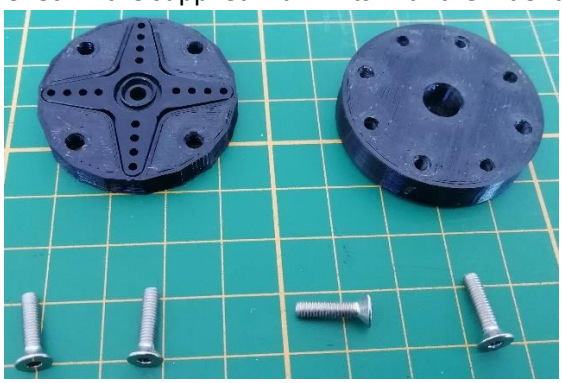




Step5 (sandwich Servo_axis_inf / servo arm / Servo_axis_inf):
4x M3x12mm conical head

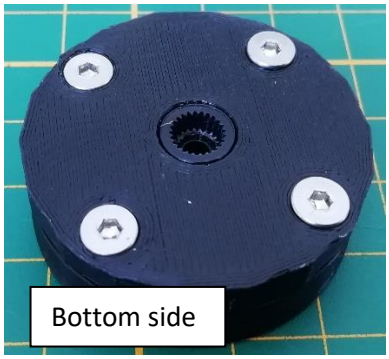


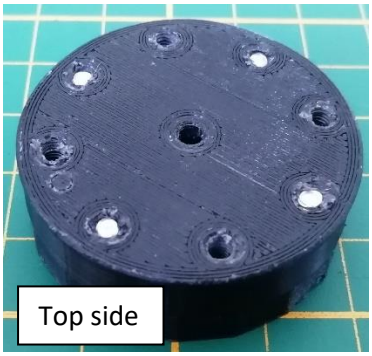
Check if the supplied X arm fits with the indentation of Servo_axis_inf part:



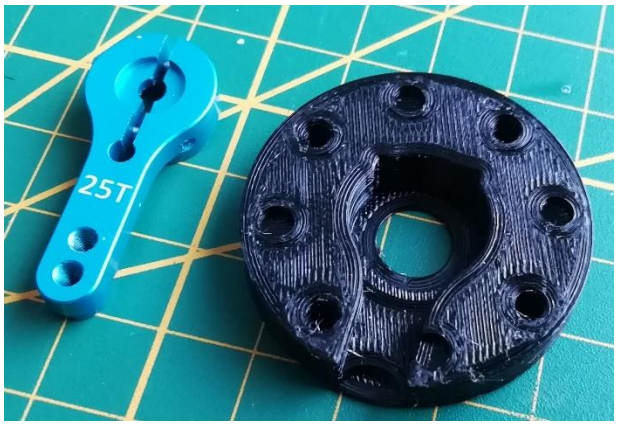
If you don't have a X arm to make it fit, please print the alternative parts (Servo_axis_inf and Servo_axis_sup) designed to fit the aluminium "T25" arm (arm has to be reduced in length).

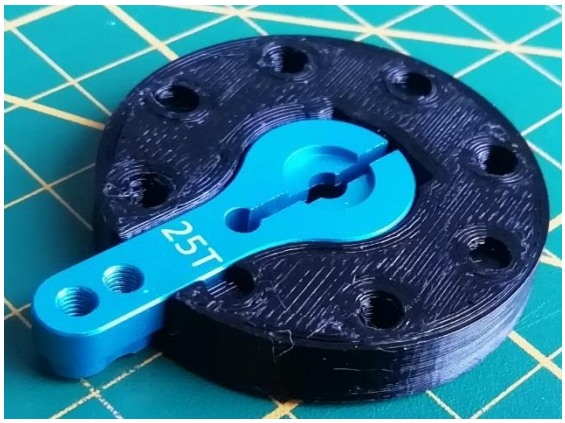
Make the sandwich





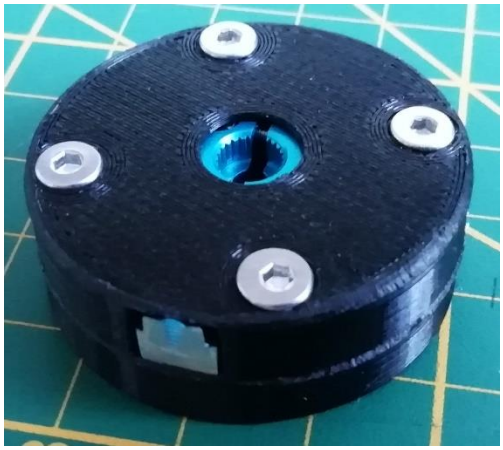
Step5a Alternative servo axis assembly:





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



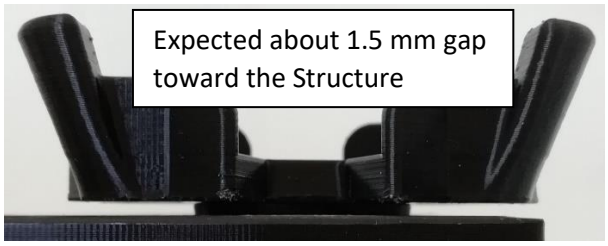
Step7 (Assemble the Cube_holder assembly to the bottom servo):

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

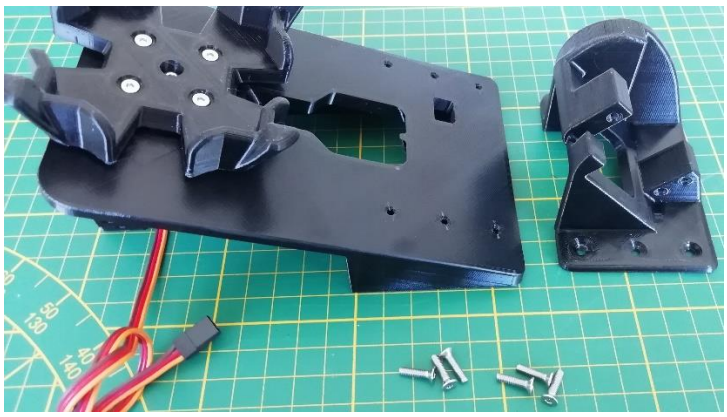

1x M3x12mm cylindrical head

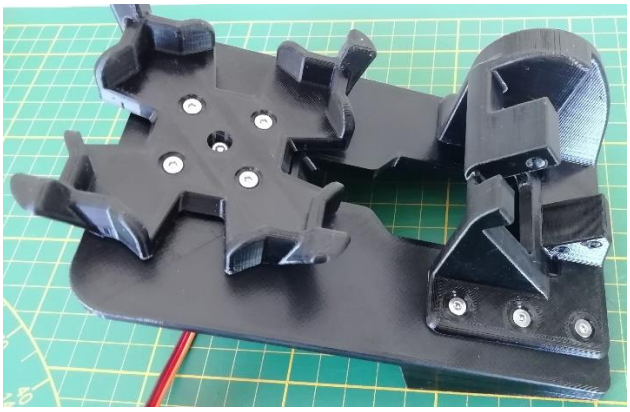




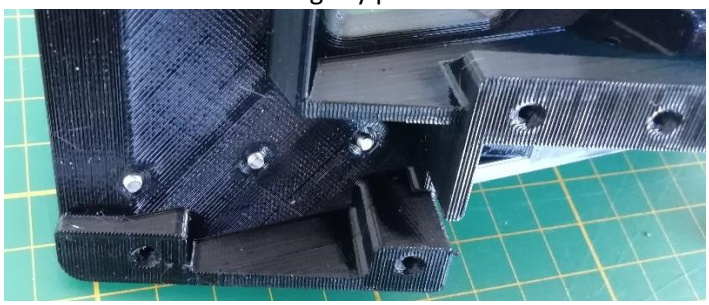
Step8 (Assemble the Hinge to the Structure):

6x M3x12mm conical head





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



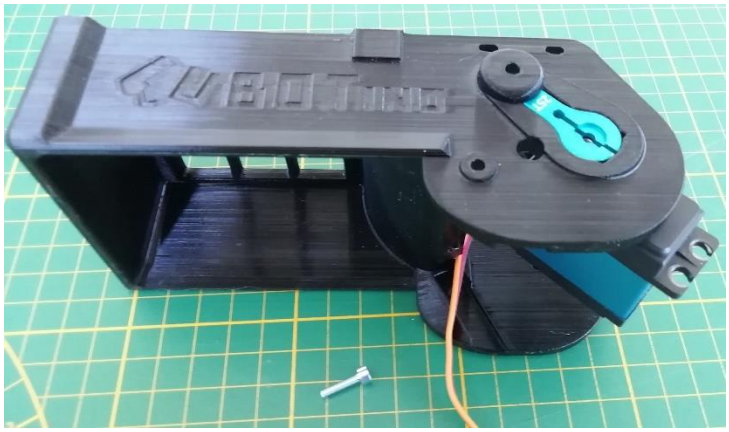
Step9 (Assemble the "T25" servo arm to the upper servo):
Place the "T25" arm along the main Servo axis (Servo should be prepared upfront with the gear at middle angle).
Close the two arm tiny screws

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

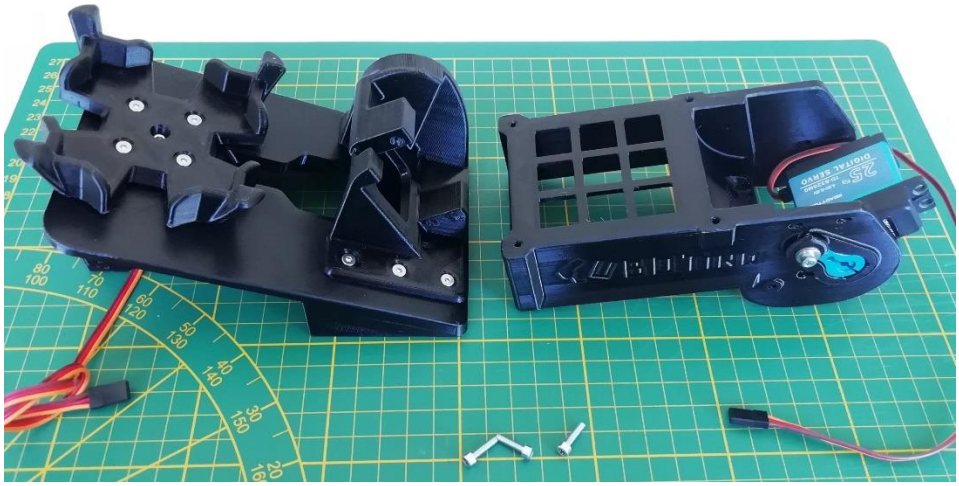
Rotate the servo by about 45deg, to facilitate next steps

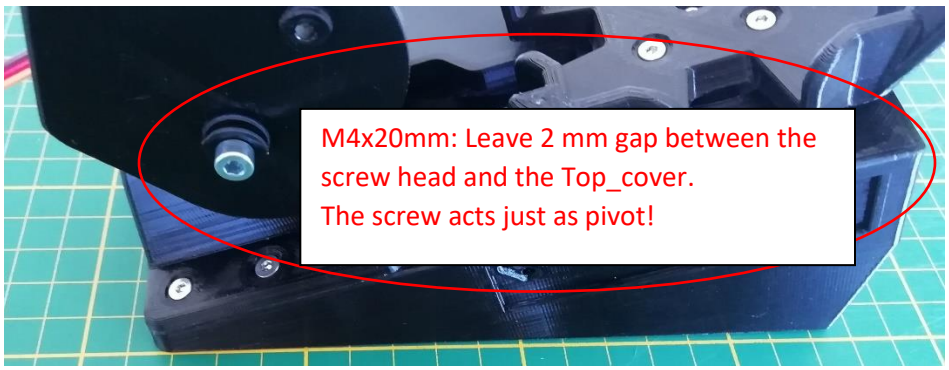
Step11 (Assemble the Top_cover assembly to the Hinge):

1x M4x20mm cylindrical head

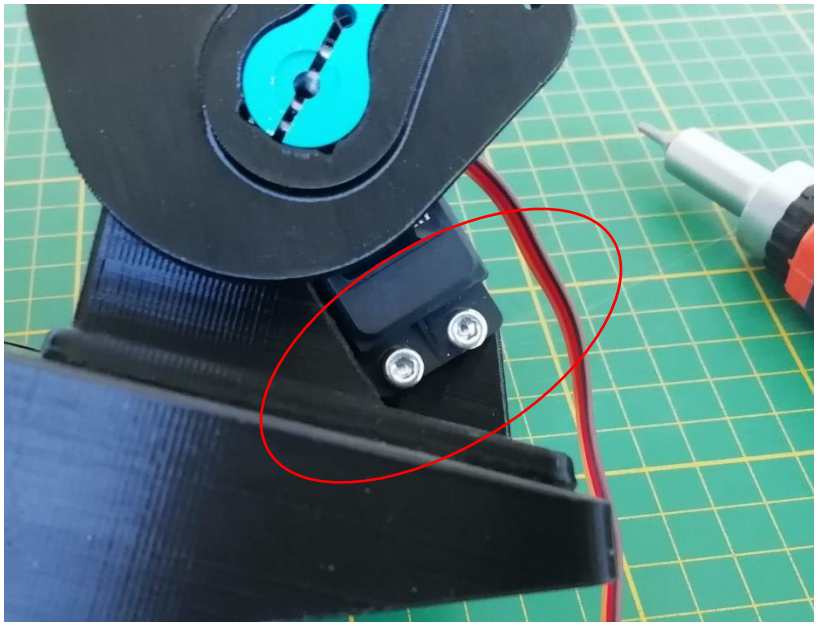
3x M3x12mm cylindrical head

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



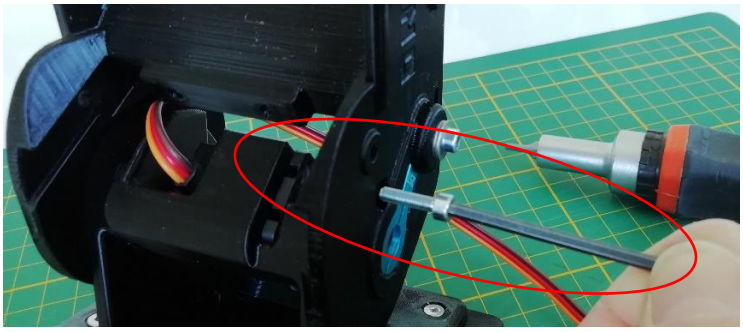


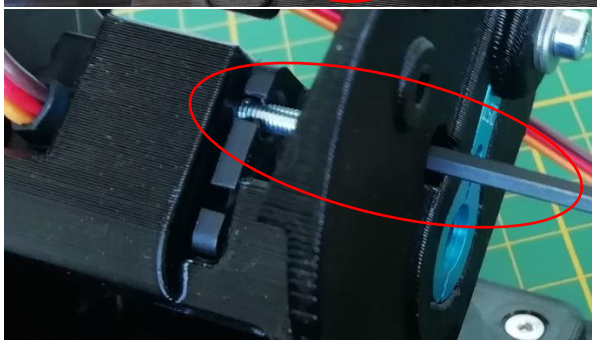
Do not fully tighten the two M3x12mm, until also the third screw is positioned



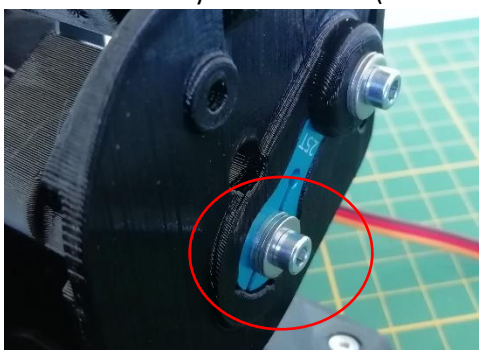
Step12 (Complete the top servo assembly to the Hinge):

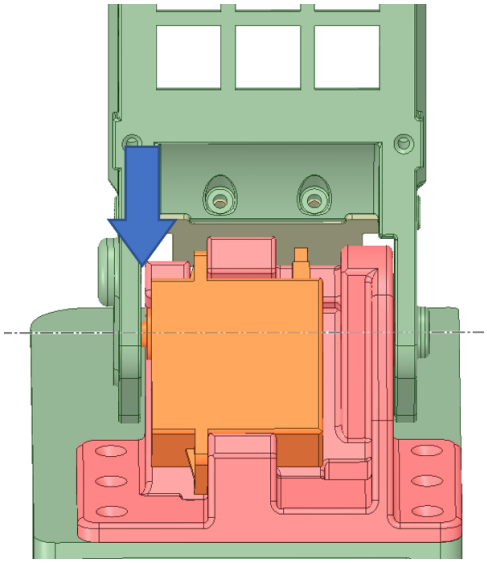
Rotate the Top_cover to have the hole facing the third screw accessible





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

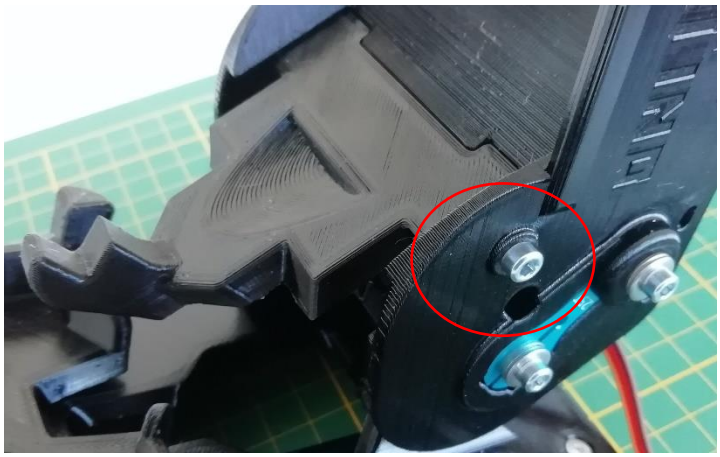




Verify gap presence in between the Top_cover and the Hinge, at the servo output gear side (arrow at the side).
In case there is little or no gap, unscrew the M3 screws of the servo, and place some little spacers (0.5 to max 1mm) in between the servo and the Hinge (preferably close to the screws locations);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

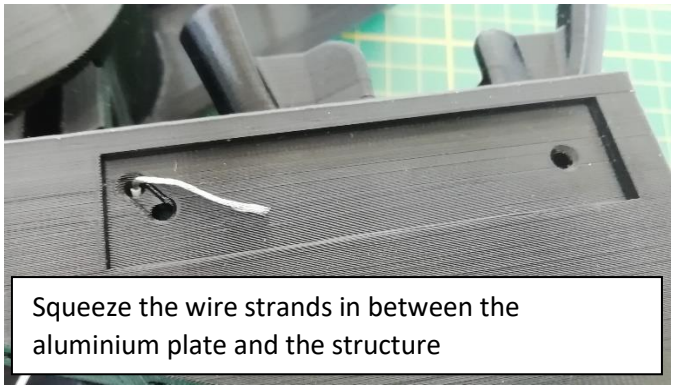
Step13 (Assemble the Lifter to the Top_cover):
4x M3x12mm cylindrical head

Slide the Lifter into the Top_cover slots

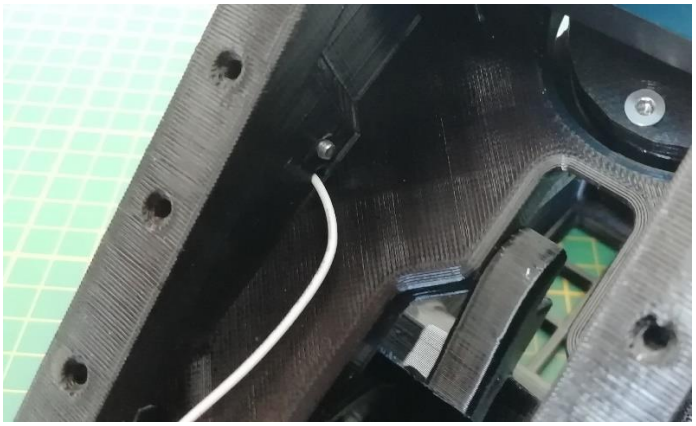


Step14 (Assemble the “STOP” touch plate, and related cable):

2x M3x12mm conical head

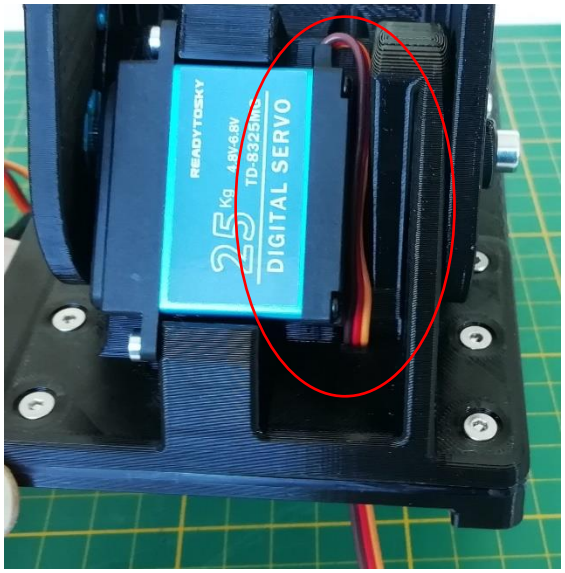


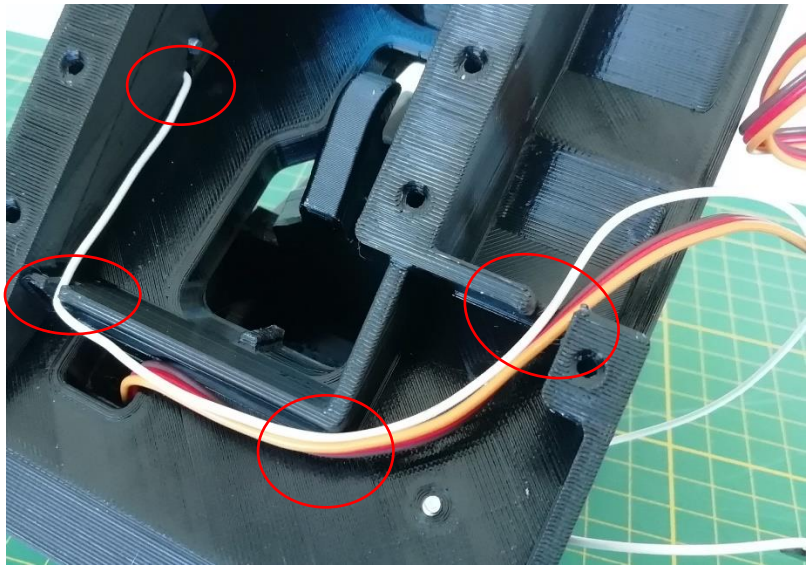
Squeeze the wire strands in between the aluminium plate and the structure

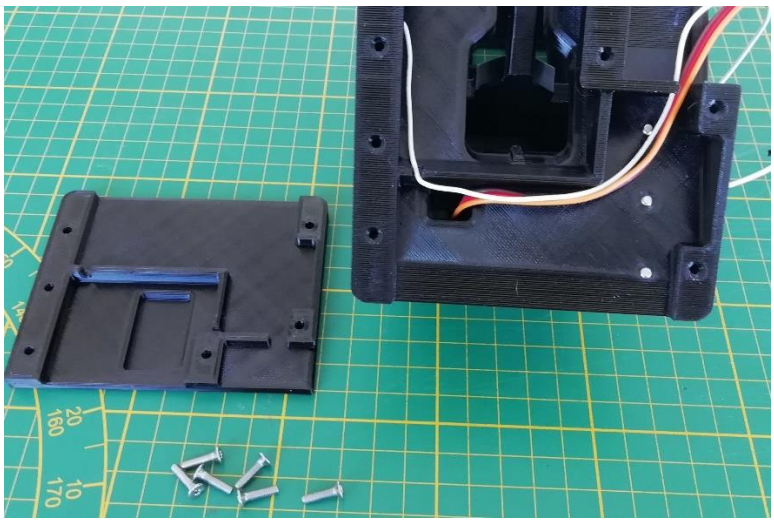


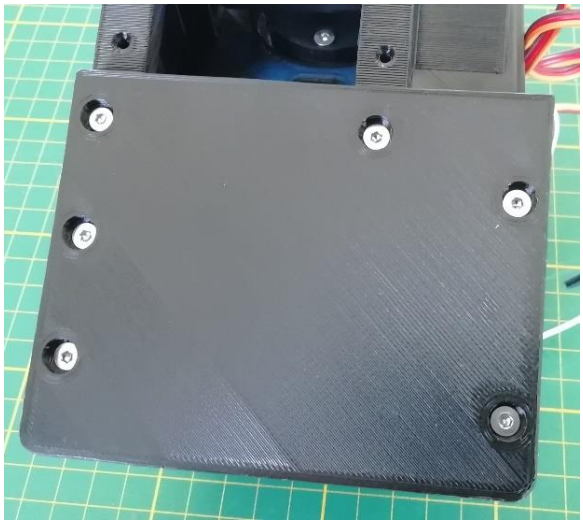
Step15 (Position the cables, and assemble the Baseplate_rear):
6x M3x12mm conical head (4 screws at corners are sufficient)

Dress the cables and close the Baseplate_rear

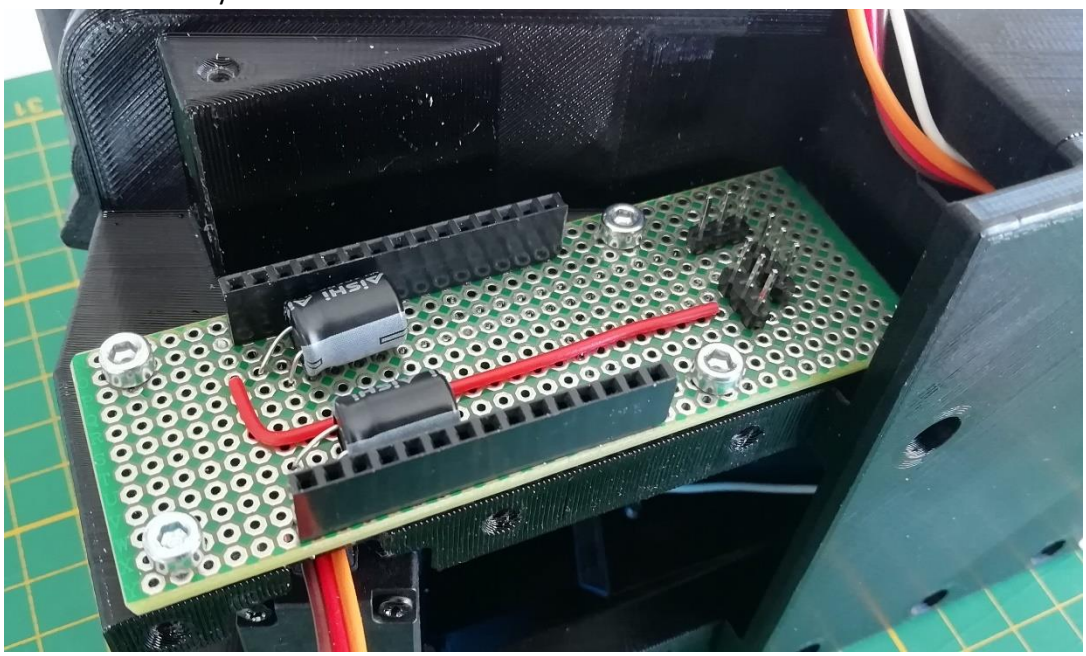




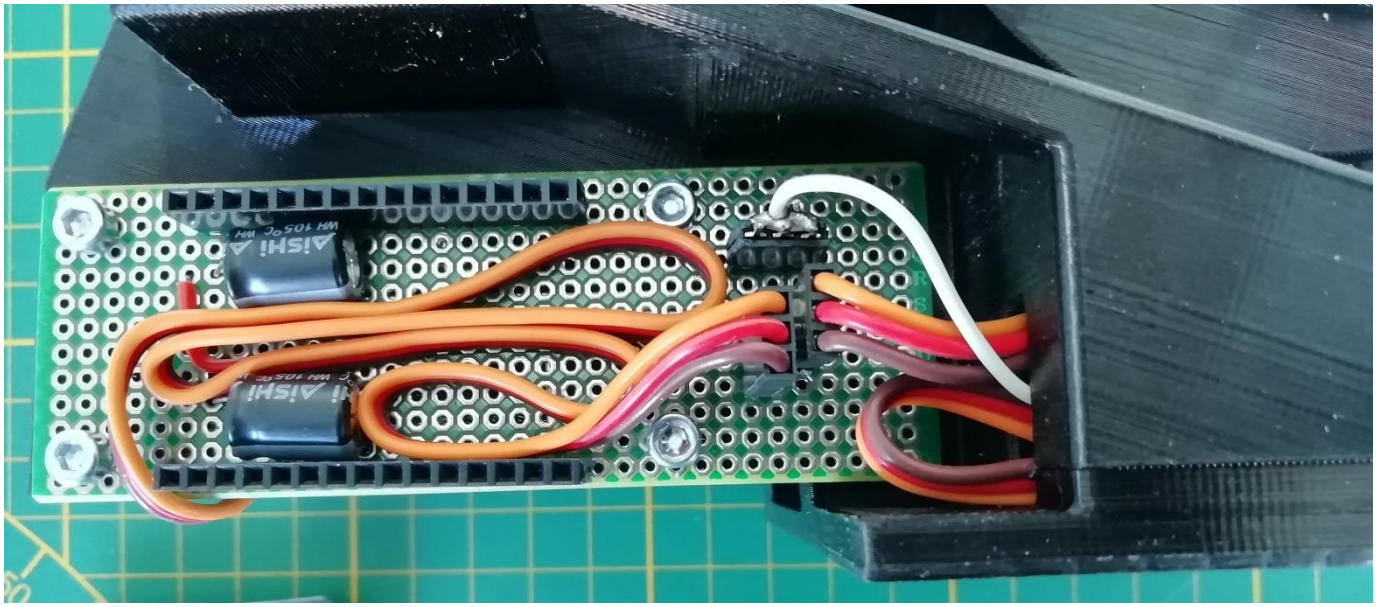




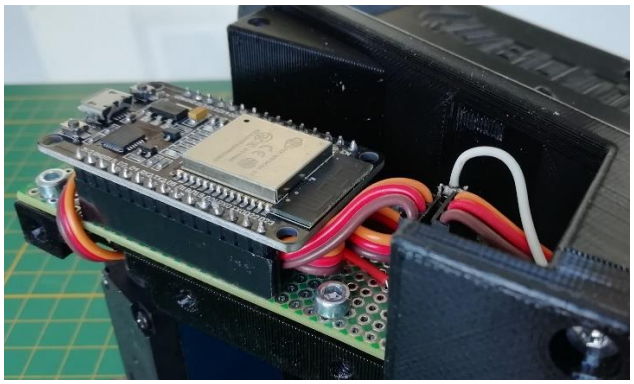
Step16 (Fix the ESP32 connections board to the Structure)
4x M2.5x10mm cylindrical head

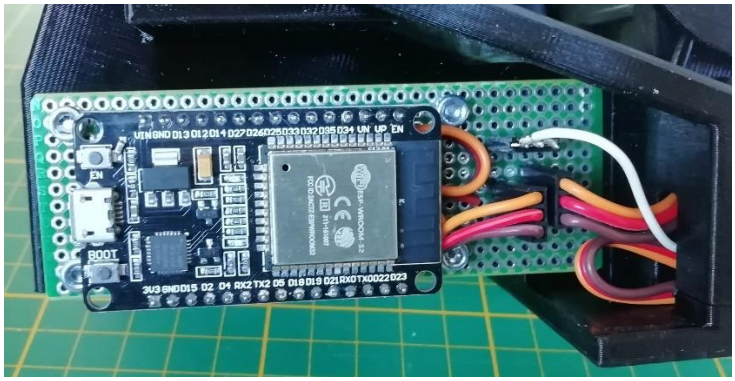


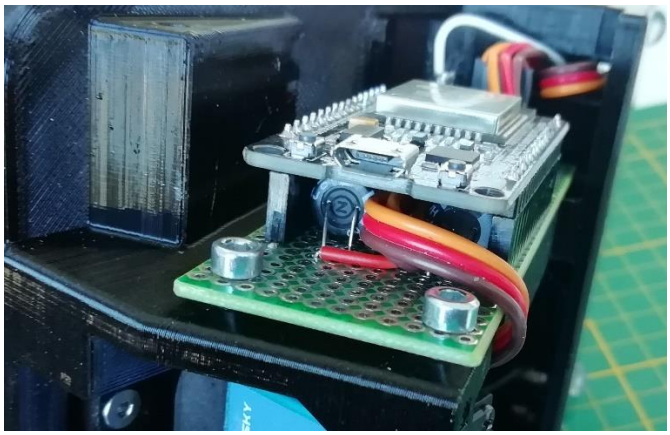
Step17 (Connects servo and touch pad)
Dress the wires and connect
In general the brown wire of servo connector is the ground, therefore to be oriented toward the bottom



Step18 (Insert the ESP32 dev board):

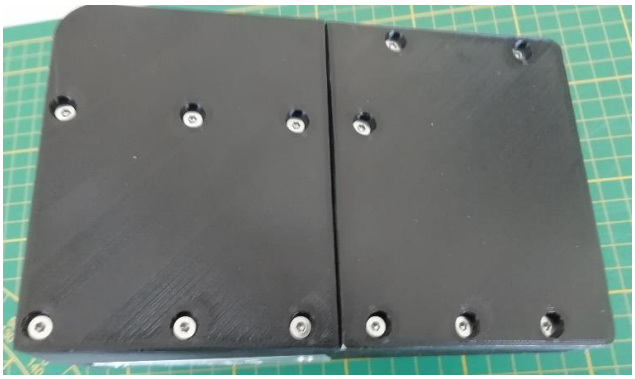






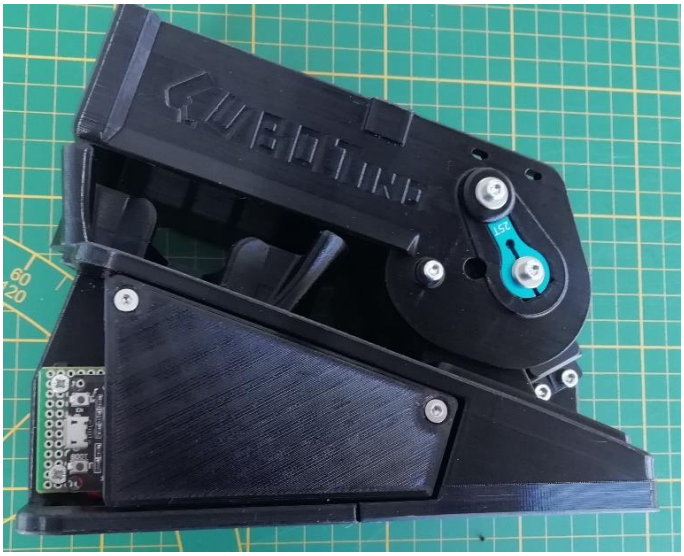
Step19 (Assemble the Baseplate_front to the Structure):

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 be 3D printed as cover; 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 😊.

13. Libraries and files needed at PC

1. Python is the (interpreted) programming language used for the GUI and the files at /PC_files folder; This means Python is simply needed. In my case I've used Python 3.7 and later versions.

2. Make a virtual environment for all the needed libraries
   It will be convenient to name the venv with a meaningful name, in my case I've used cubotino
   (On my PC I've Conda installed, so my method to create a virtual environment is: From the Anaconda Prompt "conda create -n cubotino python=3.9")

3. Activate the virtual environment; in my case I've to tape "conda activate cubotino"
   The activate venv will be displayed between parentheses
   From→ (base) C:\Users\andre>***conda activate cubotino***
   To → (cubotino) C:\Users\andre>

4. Enter the project folder /PC_files
   (cubotino) C:\Users\andre>***d:*** (in my case I've to change disk driver, d: 😊)
   (cubotino) D:\>***cd CUBOTino_base_version-main/PC_files***

5. Install the three needed libraries:
   - ***pip install opencv-python*** (necessary for the computer vision part)
   - ***pip install pyserial*** (necessary for the communication with the ESP32)
   - ***pip install RubikTwoPhase*** (necessary for close to optimum cube solution. Strongly suggested version 1.1.1 on macOS and Ubuntu)

6. Check the presence of the installed libraries, by typing ***pip list***
   Numpy gets installed via the opencv-python installation; A few other libraries are installed by default

```
(cubotino) D:\CUBOTino_base_version-main\PC_files>pip list
Package       Version
------------- ---------
certifi       2022.9.24
numpy         1.23.3
opencv-python 4.6.0.66
pip           22.2.2
pyserial      3.5
RubikTwoPhase 1.0.9
setuptools    63.4.1
wheel         0.37.1
wincertstore  0.2

(cubotino) D:\CUBOTino_base_version-main\PC_files>
```

7. Check the function of the installed libraries
   a. Enter python, by typing ***python***
   b. Import numpy, by typing import ***numpy***
   c. Import cv2 (the 44penCV library) by typing ***import cv2***
   d. Import the solver, by typing ***import twophase.solver***
      The solver should only show "loading", as the tables are already copy into the repo at /PC_files/twophase

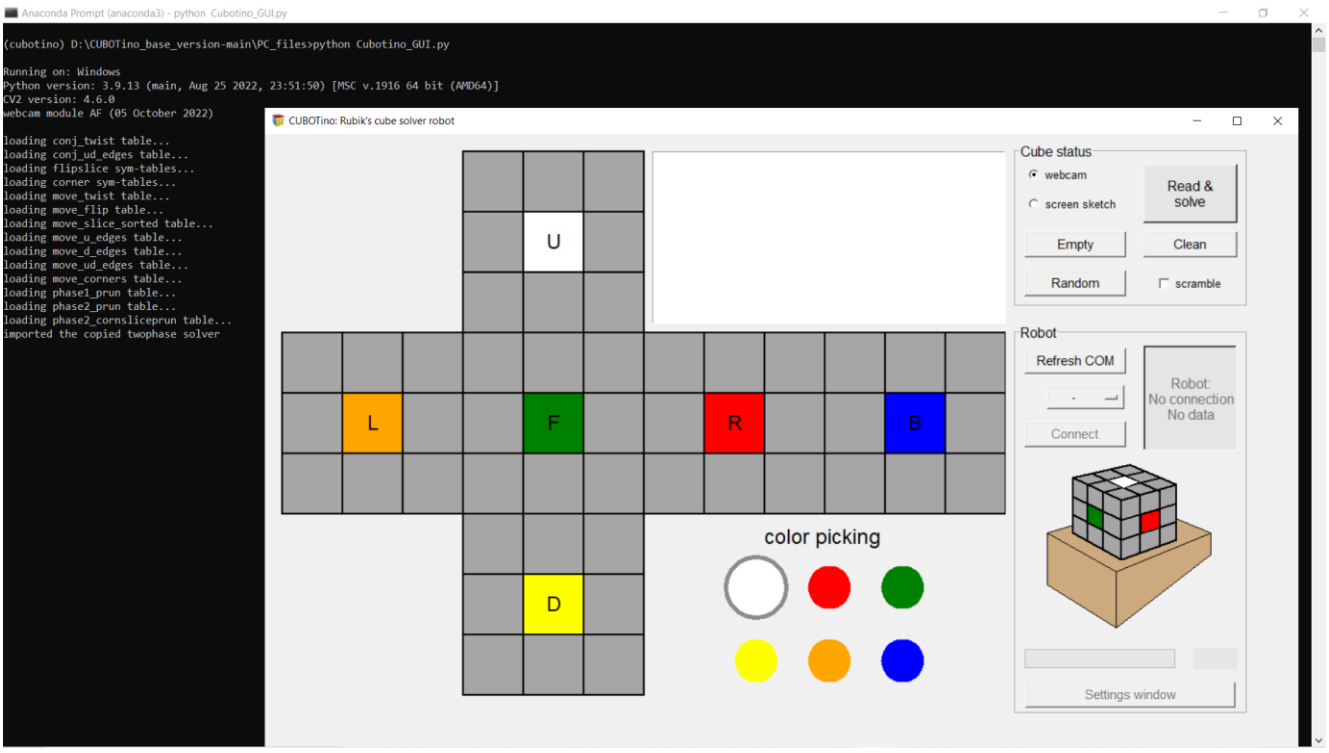
```
(cubotino) D:\CUBOTino_base_version-main\PC_files>python
Python 3.9.13 (main, Aug 25 2022, 23:51:50) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import numpy
>>> numpy.version.version
'1.23.3'
>>> import cv2
>>> cv2.__version__
'4.6.0'
>>> import twophase.solver
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...
>>> exit()

(cubotino) D:\CUBOTino_base_version-main\PC_files>
```

> "*EOFError: read() didn't return enough bytes*" error might be returned on macOS or Ubuntu.
>
> To solve the problem update RubikTwoPhase to version 1.1.1

   e. Exit python by typing exit()

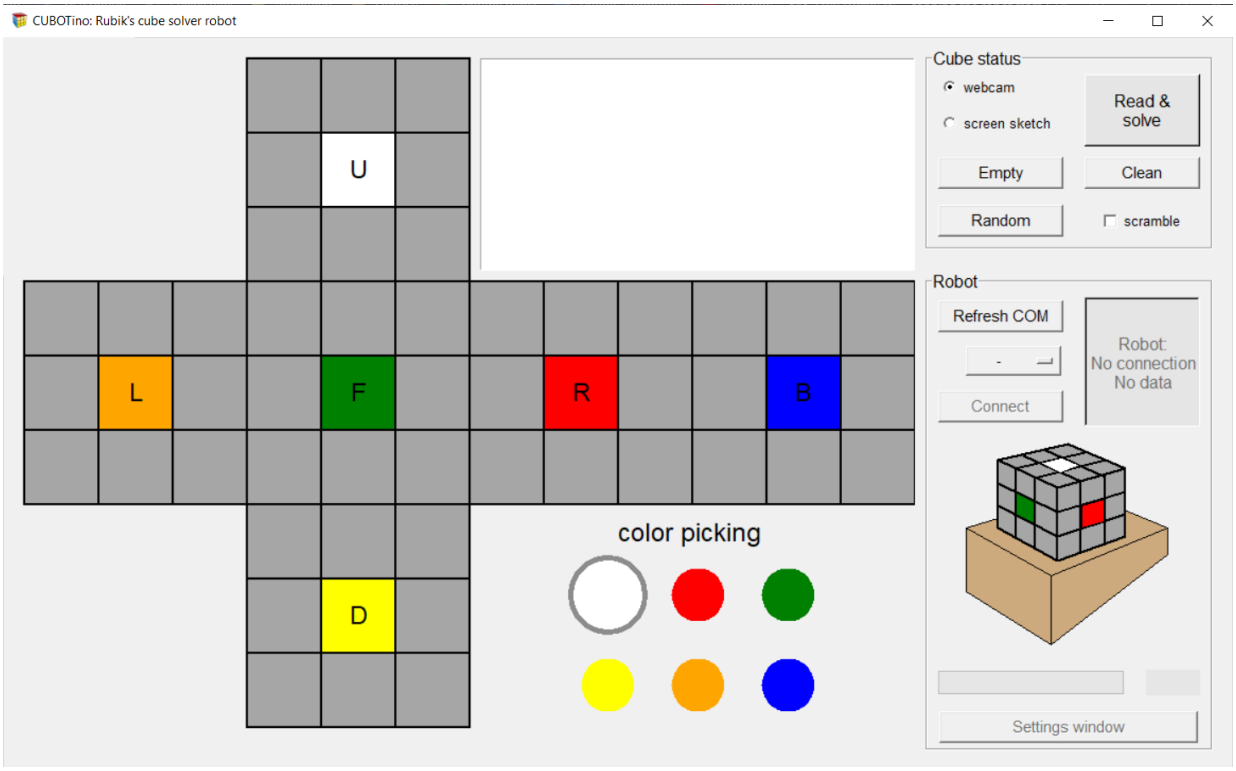
   If no error messages, the PC is set; If errors, try to re-install the related library.

8. Start the GUI application, by typing ***python Cubotino_GUI.py***
   The GUI application shows up this way:

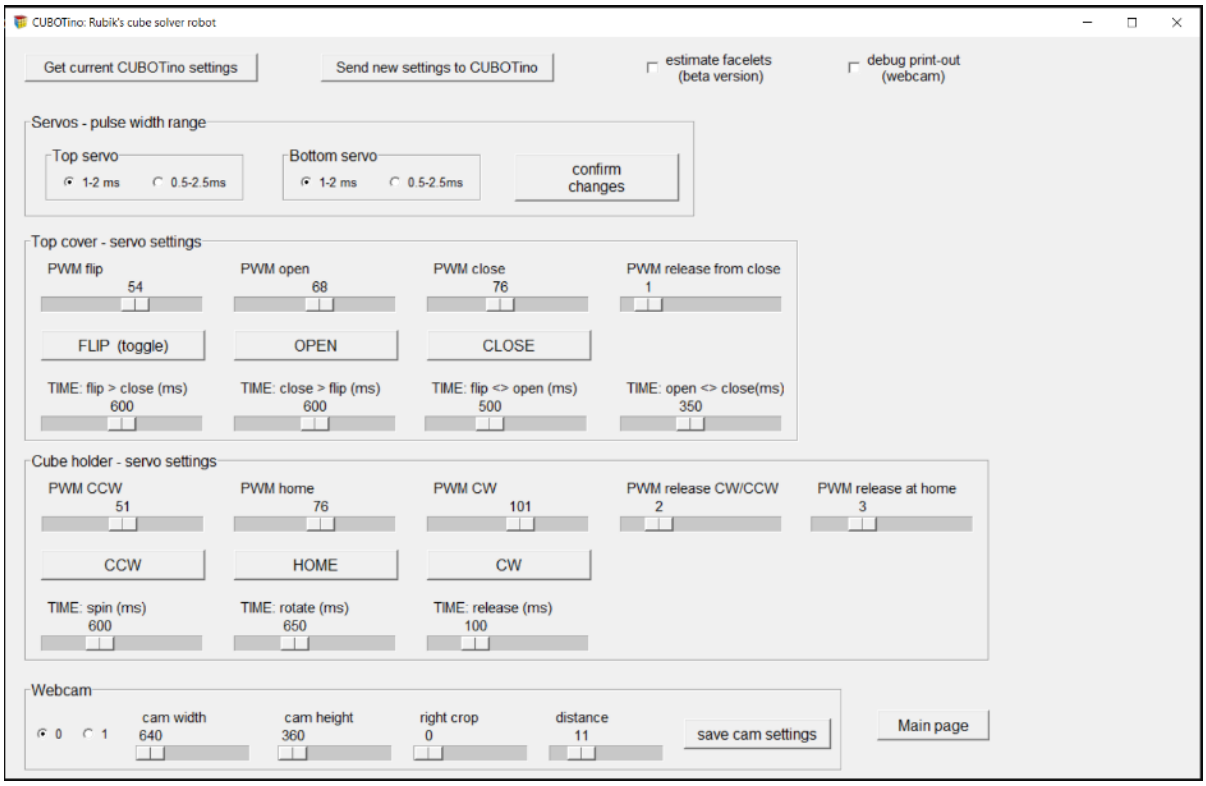


14. GUI

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

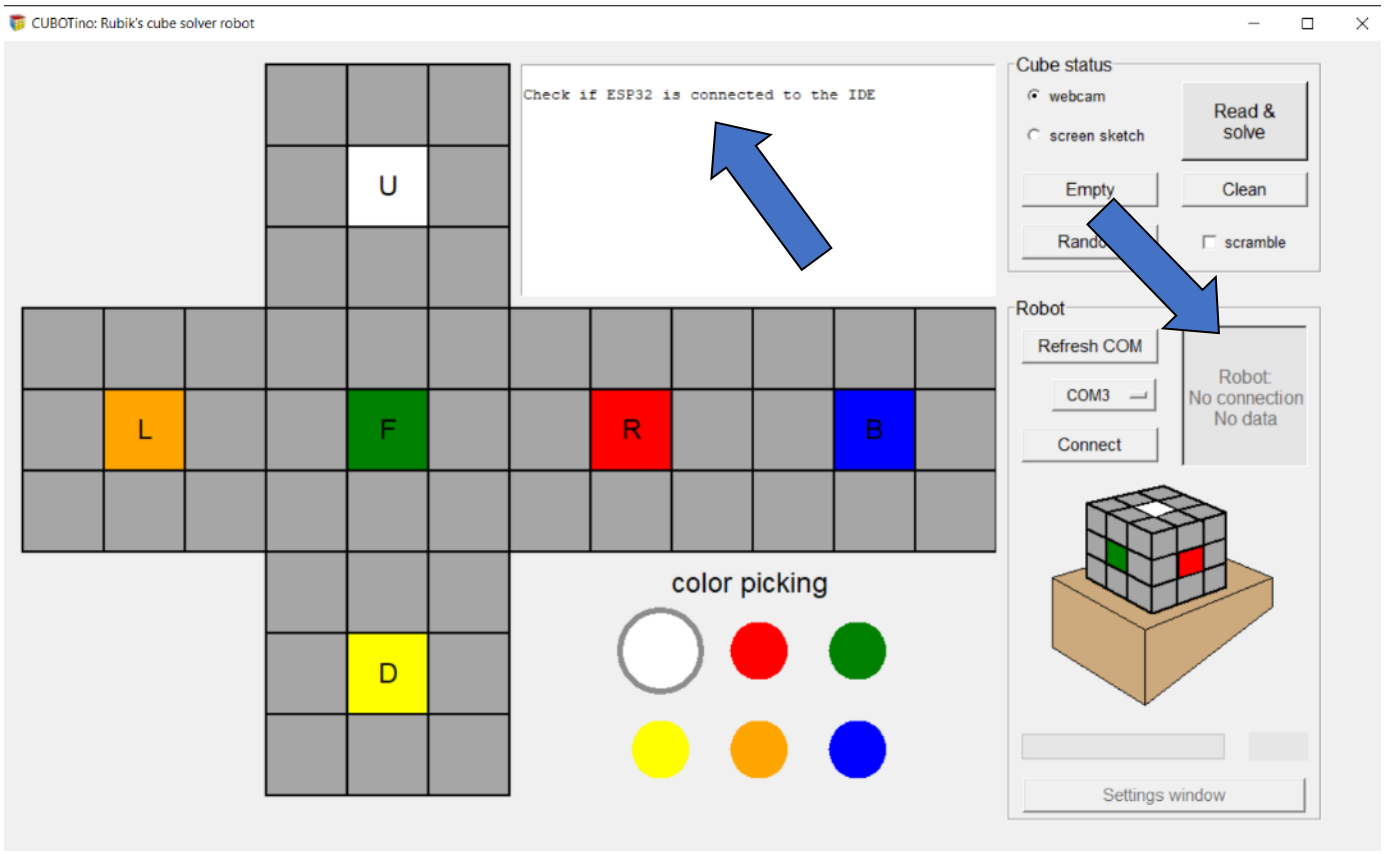


When the serial communication is established, the “Settings windows” button is activated, and via such button the Settings window can be reached:



When the connection to the robot fails there are a few feedback visible:

- The Robot button indicates “No connection”
- The informative window suggests “check if the ESP32 is connect to an IDE” (i.e. Thonny); If that’s the case, close the IDE, and press again the Connect button

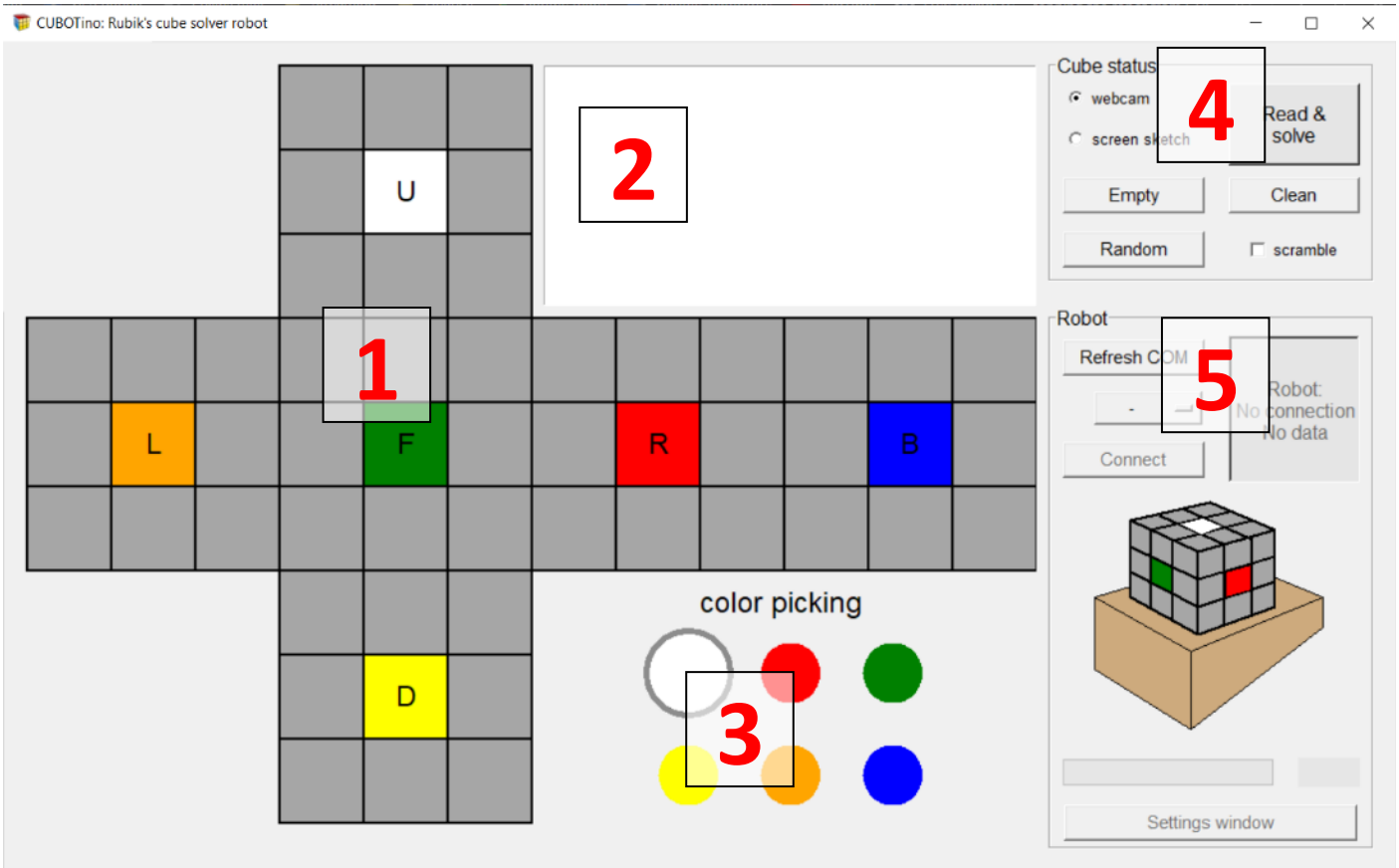


Each window is somehow divided in areas:

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

Area	Name	Functions
1	Cube sketch	- Used to generate the initial cube status - 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.

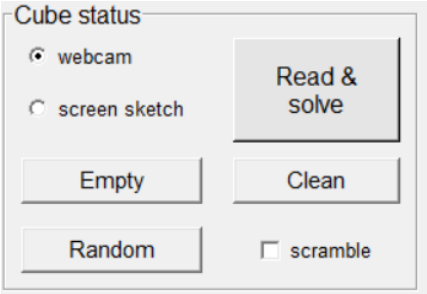


Area 1 and 3:

- To fill the cube sketch, pick a colour on the palette, with the left mouse button, and applied on the facelets by pressing again the left mouse button; Colours 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).

Area2 is just informative

Area 4, Cube status interface:

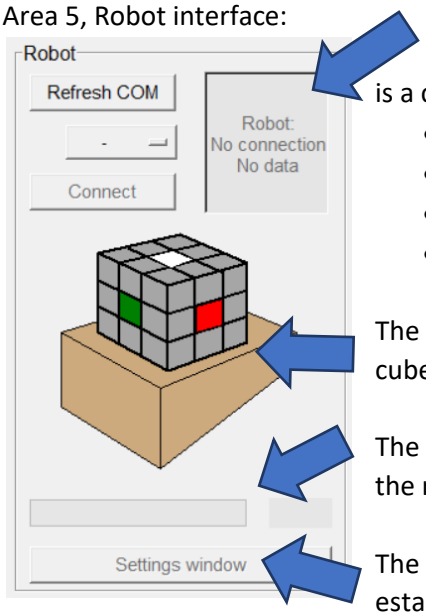


It's possible to select the cube status source, *webcam* or *screen sketch*.
The *Read & solve* button sends, to the Kociemba solver, the cube status depicted on the sketch (or greyed out when *scramble* is selected); In case of coherent cube status (and no *scramble*), on the Text window will be printed:

- 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:

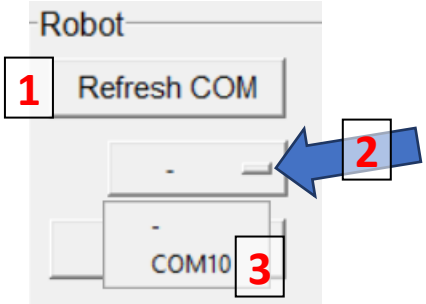
- No connection, No data (No data means there isn't a robot solution to send)
- Connected, No data
- Send data to robot
- STOP ROBOT

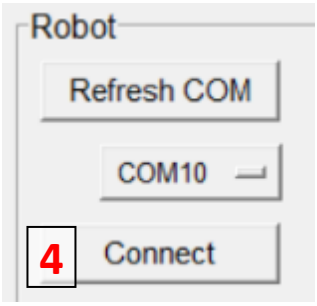
The CUBOTino sketch updates the center facelets colours, as guidance to drop the cube on the robot according to the orientation used while detecting the status.

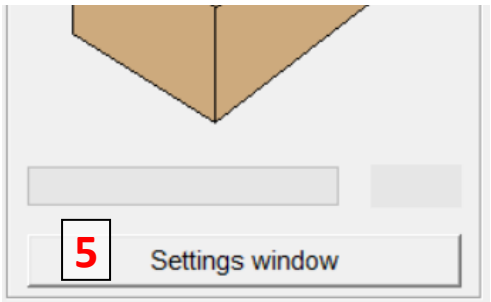
The progress bar is updated in real time, according to the progress feedback sent by the robot at each move.

The *Settings window* button gets 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







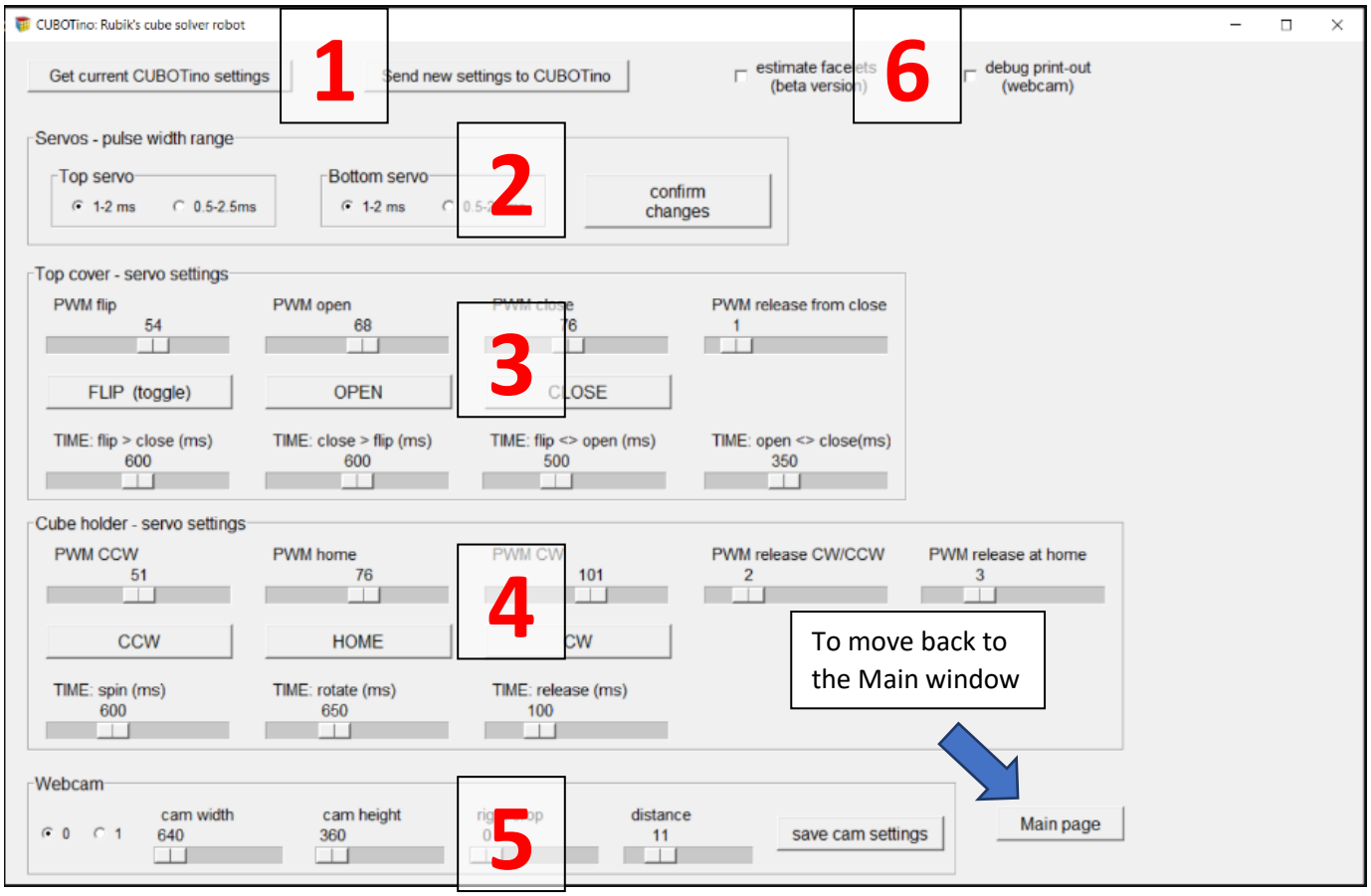
When a COM port has been selected, the drop-down menu closes and the *Connect* button becomes active.
When the Connect button is pressed, in case the communication with ESP32 goes well, the *Settings window* button gets activated.
The *Settings window* button activates the GUI dedicated window for the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

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

Area	Name	Functions
1	Receive and send robot setting	- Retrieve the settings that are stored at the micro-controller - Send the new settings to the micro-controller - Retrieve the setting of my robot, as reference
2	Set the servos pulse width range	depending on the servo, some of them are controlled with a PWM having a pulse width from 1 to 2ms while others span from 0.5 to 2.5ms
3	Top cover servo	- Settings for the Top cover (and lifter) - Buttons allow to test the specific positions, and some of the associated timers
4	Cube holder servo	- Settings for the Cube holder - Buttons allow to test the specific positions, and some of the associated timers
5	Webcam	- Settings for the webcam
6	Options	- Estimate facelets and debug print-out

Notes:

1. Changes eventually made to the servos pulse width take effect if followed by “Confirm changes” button press.
2. Changes eventually made on the servo related sliders, will be effective at the robot only after pressing *Send new settings to CUBOTino* button.
3. By pressing the servo related buttons (FLIP, OPEN, CLOSE, CCW, HOME, CW) it’s possible to test the PWM and servos positions.
4. Not all the timers can be tested via test buttons associated functions, yet most of them ☺



Areas description

Area1

This allows to send new settings to the robot

When a slider is changed, the data is only graphical: To make it effective it must be sent to the robot.

Area2

This part has been introduced in October 2022, because of the feedback from some Makers about ordering or receiving servos with different Pulse Width range.

Each time a setting is changed, from 1-2ms to 0.5-2.5ms or the other way around, all the PWM sliders on screen are updated; The graphical update maintains the equivalent servo angle, by updating the cursor value as well as the slider range.

Changes on the Pulse Width ranges will only take effect once the "Confirm changes" button is pressed, and transmitted to the robot only when pressed *Send new settings to CUBOTino* button.

Area3, 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.

Area4, 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

Area5, 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. 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: The smaller the frame dimension, the faster the facelets detection time is.

Area6, Options:

Estimate facelets (beta version): When selected, it activates the estimation of latest facelets (position), based on the first 7 facelets effectively detected via the edge detection.
This helps detecting cubes having a (small) logo, or when the light conditions aren't optimum.
Under "difficult" conditions this method is faster yet it's less reliable.

Debug print-out (webcam): Some information is print to the terminal when the webcam is used for the cube status detection.
This feature is mostly meant to make visible some of the information on used to define the cube status via the image acquisition

Note: These options (check buttons) are always off when the GUI starts.

15. Tuning:

Tuning process is inevitable, as all servos are slightly different from the others, as well as the coupling of the arm to the servo's outlet gear is not unique.
Before start tuning the robot, take 5 minutes to read this chapter 😊

1. 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.

2. Reference angles for servos:

The servos I bought, have 180-degrees of rotation, that is more than sufficient for the (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.

3. Setting servos positions:

Servos are controlled on angle, via a PWM signal
Micropython V1.17 with ESP32 board control the PWM in 2^10 units
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 having 1-2ms Pulse Width.
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 supplies list), 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. Set both the servos on their mid range position (mid PWM value), prior to the robot assembly
2. In case only one servo rotates about 190° or more, use this servo for the cube holder.
3. Search the min and max PWM values that makes the servos gear moving (connect an arm to better check). This can be done via the GUI setting windows, once the robot is assembled.

Note: In case your servo makes just 180° rotation, to get the robot working properly, it's necessary to increase the rotation range, up to about 190°. Search the "how to" on Troubleshooting.

4. Testing and checking the servo via GUI setting window:

The servos tuning process is substantially based on trial and errors process

To minimize the errors, it is convenient to

1. be aware the bottom servo (cube_holder) has some gap toward the cube and the cube has gap toward the Upper_cover. To get proper cube layers alignment, extra rotation is needed. The extra rotations are called “release”, because after the extra-rotation the servo comes back to release the tension.
2. slow down the robot, by setting the 7 timers to the max, this will be needed for later
3. the CW and CCW are notations from the servos point of view, meaning the opposite from the user POV.
4. values to set the CW and CCW should be the values the servos are responsive; You’ll find difficult to set the release values if CW and CCW are set to values beyond the capability of the servo to make extra rotations.
5. start with the bottom servo (cube_holder), as the top one is rather easy to set.

Cube_holder (bottom servo) angles:

There are 3 (+2) defined angles (most of time mentioned as positions in this instructions):

a. CCW: position to spin or rotate the Cube_holder >90° CCW from Home (Direction according to the motor point of view)

b. CW: position to spin or rotate the Cube_holder >90° CW from Home (Direction according to the motor point of view)

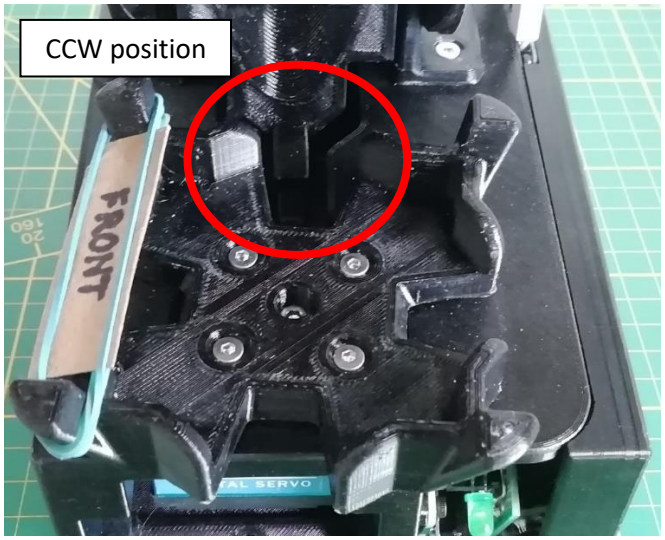
c. Home: mid position between CCW and CW

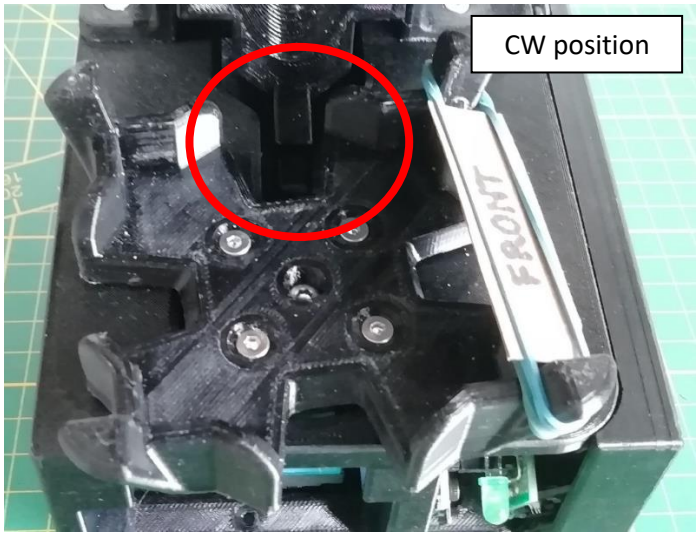
d. Rel from CW CCW: position(s) to release cube tension from Top_Cover at CW and CCW

e. Rel from home: position(s) to release cube tension from Top_Cover at Home

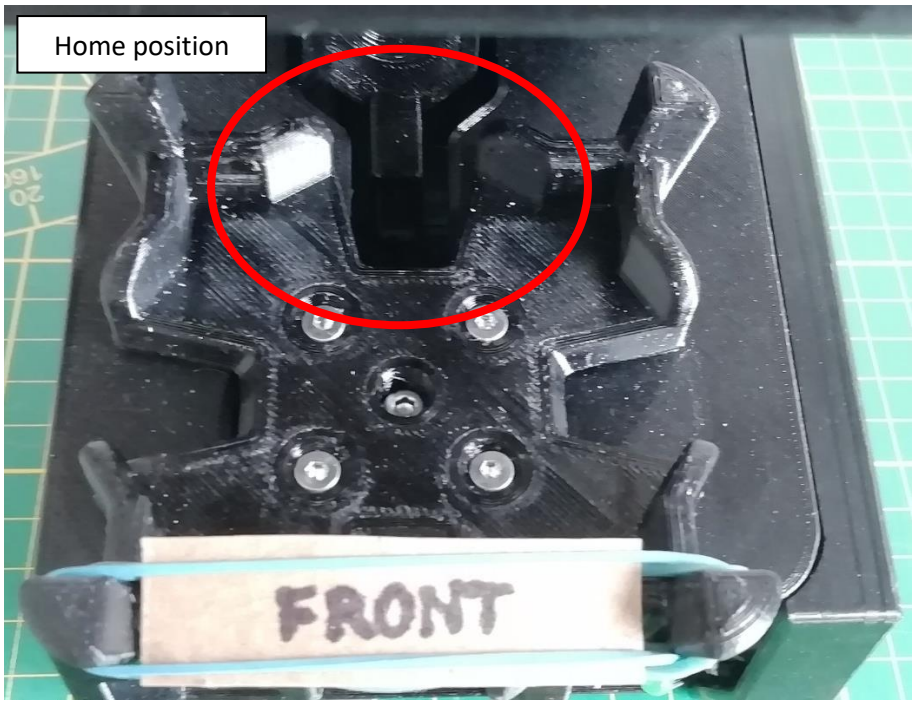
Be noted the CCW and CW angles are slightly more than 90° apart from the Home position

This is needed in order to turn ~90° to the cube, after recovering the radial plays: There is play in between the cube and the Cube_holder, and again there is play between the cube and the Top_cover





The Home position must be well centered.

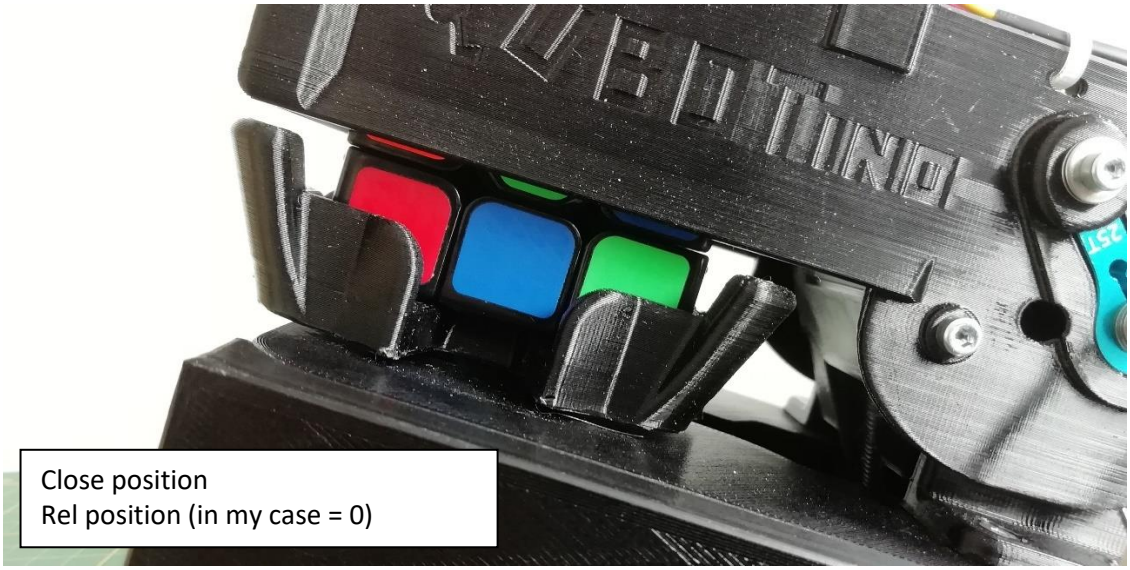


Upper_cover (top servo) angles:

There are 4 defined angles (most of time mentioned as positions):

a. Close: position to constrain the top and mid cube layers
b. Release: position to release tension, from cube, at Close position
c. Open: position without interferences with the cube and Cube_holder
d. Flip: position for the Lifter to flip the cube (about 2 cube layers height)

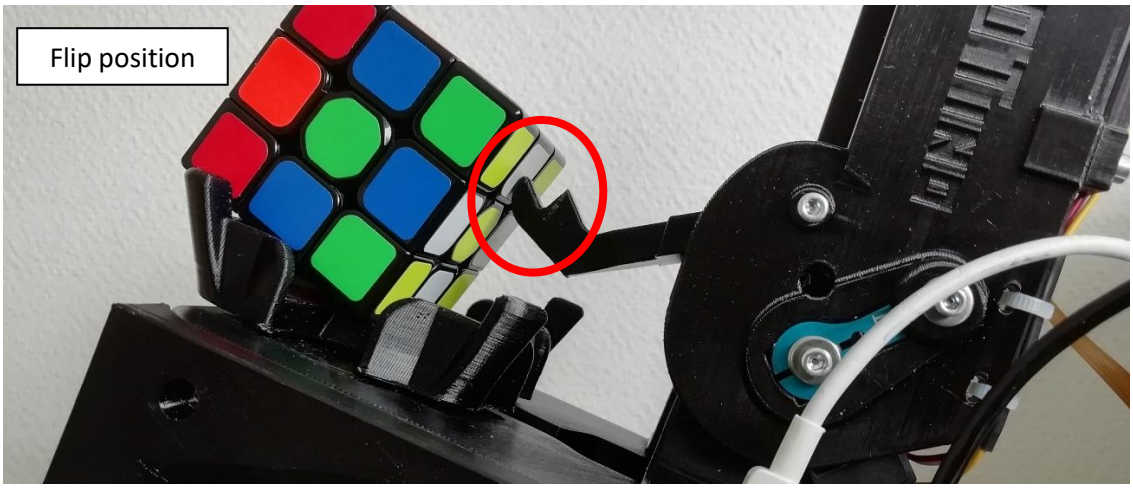
Close position, to constrain the top and mid cube layers



Release position, to release tension, from cube, at Close position (when 0 is used, the Upper_cover keeps the Close position)

Open position, without interferences with the rotation of the cube and the Cube_holder

Flip position, for the Lifter to flip the cube (about 2 cube layers height)
The Upper_cover forms an angle of almost 90° from the Structure



5. **Check the servo with a cube, still via the GUI setting window:**
   Once the positions are all sets, you can verify how these positions interact with the cube, by using the buttons at the GUI Settings window:
   - Position a cube on the cube_holder
   - Close the Upper_cover
   - Force the cube_holder to the three positions
   - Per each position check if the 1st cube layer is well aligned with the second one.

   For the Home position, make the check by coming either from CW and from CCW !
   If this looks promising, you can move to the next part.

6. **Testing and checking the servo by solving a random cube:**
   - Place a cube on the cube_holder
   - At the GUI generate a random cube to be solved
   - Send the data to the robot and observe if the cube manoeuvring looks good

   In case the cube seems to randomly misalign, it means that probably only one setting requires improvements; To detect which one, it's convenient to organize for proper observations.
   Label the front, left and right side of the cube holder (L, F, R) and take some video with the phone.
   Watch the movies to see which movement determines the misalignment.
   When a misalignment occurs, the movement right before is the one that requires improvement.

Home position is **OK**. This is the case when home is reached from either CW and CCW:	CW position (L is in front) is **NOT OK**. This happens by rotating from home to CW:

16. Troubleshooting

Some of the below aspects were encountered during the robot development, other were reported by other Makers, and some are hypothetical

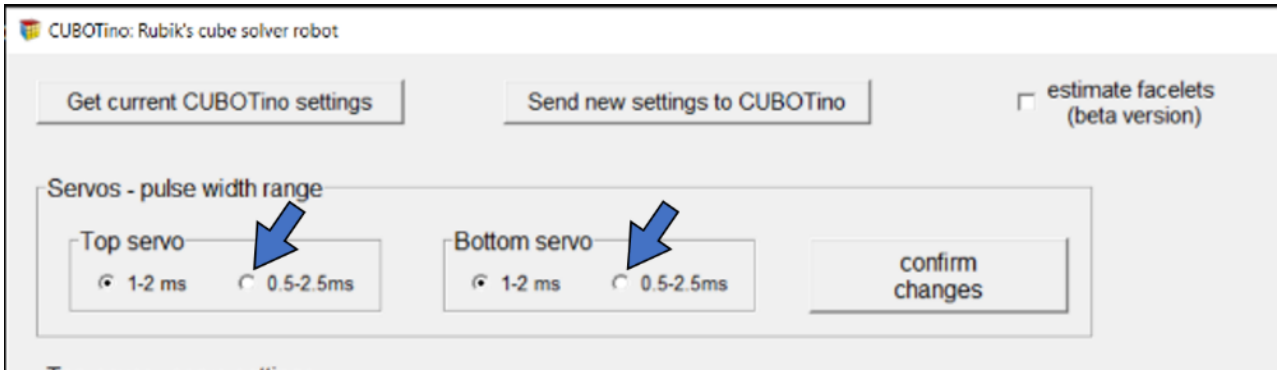
Below aspects are covered on next slides

1. Servos have too little rotation ranges:
   - Rotation range > 180deg when tested with “servo_to_mid.py”, and much less with Cubotino_GUI.py
   - Rotation range < 180deg when tested with “servo_to_mid.py”
2. Servos don’t rotate smoothly
3. Communication failure between GUI and robot
4. Communication failure between IDE and robot
5. Bottom cube layer doesn’t align nicely
6. Top_cover usage to improve cube layers alignment
7. Webcam app reads the same cube face twice
8. Webcam app is slow
9. Webcam app ends with “incoherent cube status” statement
10. Cube’s facelet and light reflection (webcam cube status detection)
11. Cubotino_GUI.py file doesn’t open with Thonny
12. ESP32 doesn’t react
13. “*ModuleNotFoundError*” error on the terminal
14. “*EOFError: read() didn't return enough bytes*” error on the terminal

1. **Servos have too little rotation ranges:**
   - **Case rotation range > 180deg when tested with "servo_to_mid.py", and much less when trying to set servos angles via the sliders at settings windows of Cubotino_GUI.py**

In this case is well likely the servos having Pulse Width range 500µs to 2500µs, and the setting is still left on the default 1ms to 2ms.
To solve the problem:

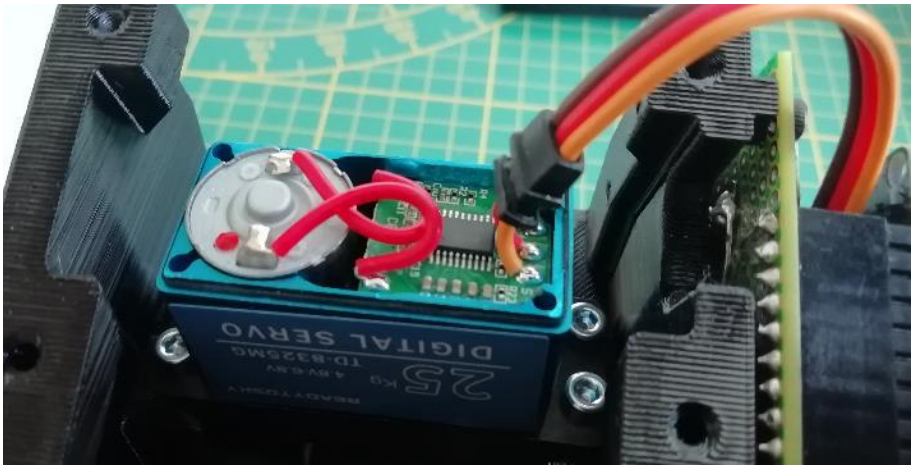
1) Ensure you have the latest revision of the repository (at least V4.3 from 12th November 2022).
2) Select pulse width range "0.5-2.5ms" for the related servo.
3) Press "confirm changes" to save the setting at the ESP32 and PC



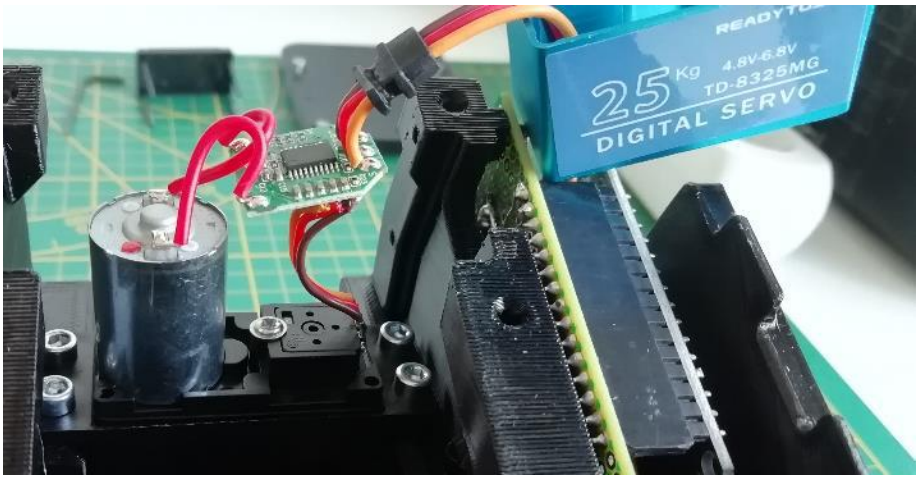
- **Case rotation range < 180deg when tested with "servo_to_mid.py**

Before changing the servos, ensure you have verified the rotation range is not limited by the Pulse Width range (see previous point).
It's possible to increase the rotation angle, by adding one or two resistors in series with the servo potentiometer (servo must be opened for this change). In my case I had 1 servo (out of 8 used so far) with insufficient rotation range.

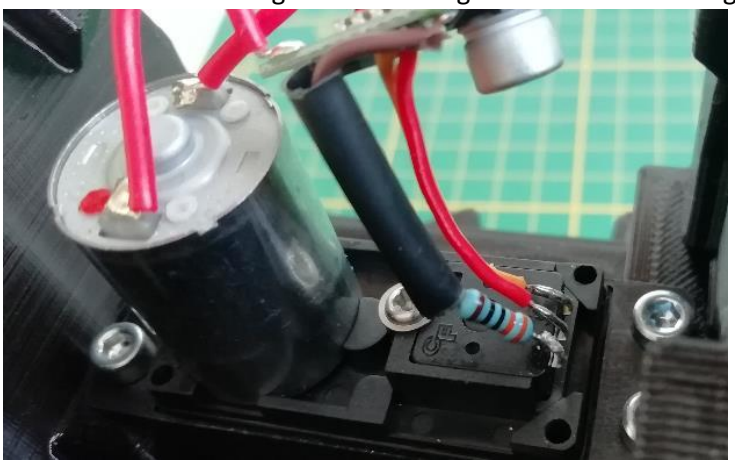
1. Open the servo (4 very long screws)



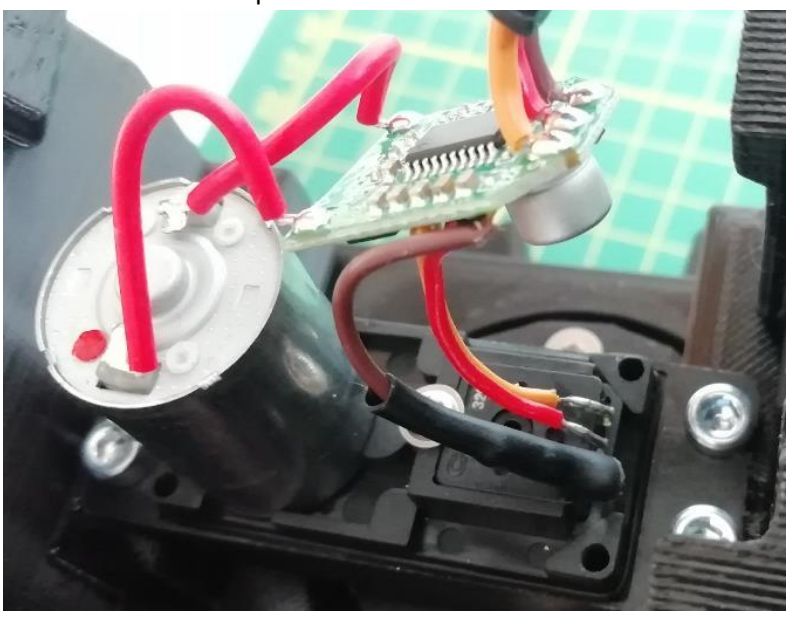
3. Slide out the pcb, and afterward slide the case out of the way.



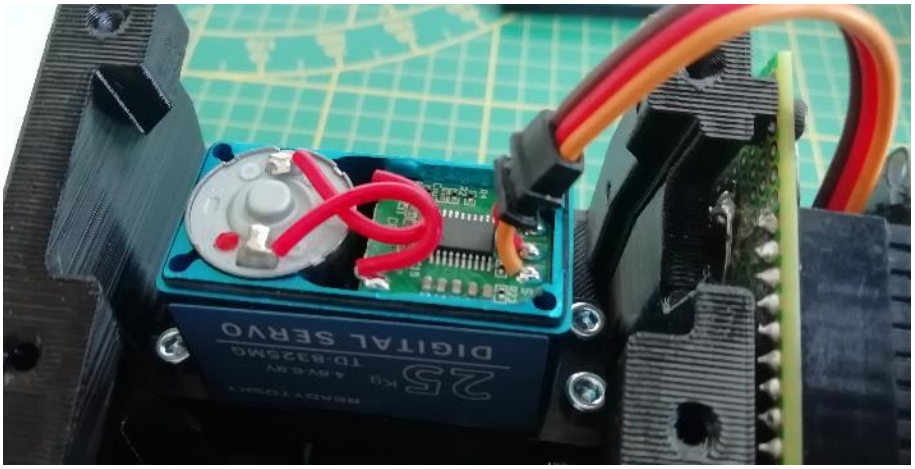
4. Solder one resistor to one external pin of the potentiometer (or one resistor per each of the two external potentiometer pins to keep the same PWM value for the mid position).
   I've added 330ohm to gain about 5 degrees on a servo having pulse width from 1 to 2 ms.



5. Protect the soldered parts with a sleeve



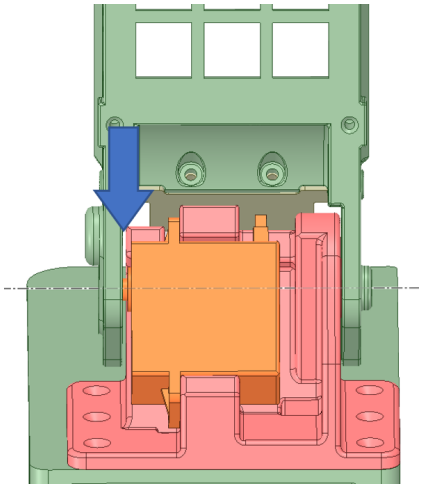
7. Close the servo, with the attention to insert the potentiometer rod into the servo output gear.



8. Place the cover and close the 4 screws

2. 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 phone 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; 2mm gap 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.

3. Communication failure between GUI and robot:

- Ensure the ESP32 microcontroller is not connected to Thonny or an IDE.

4. Communication failure between IDE and robot:

- Ensure the ESP32 microcontroller is not connected to the Cubotino_GUI.

5. **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 and/or increase the rotation time slider.
    - If you've tried all the suggestions, and still the cube doesn't align well, consider buying the suggested cube; This cube has magnets, so it tends to align nicely even when the robot is not well tuned.

6. **Top_cover usage to improve cube layers alignment**
   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.

7. **Webcam app reads the same cube face twice**
   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 presenting it again to the webcam once rotated to the next face.

8. **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.

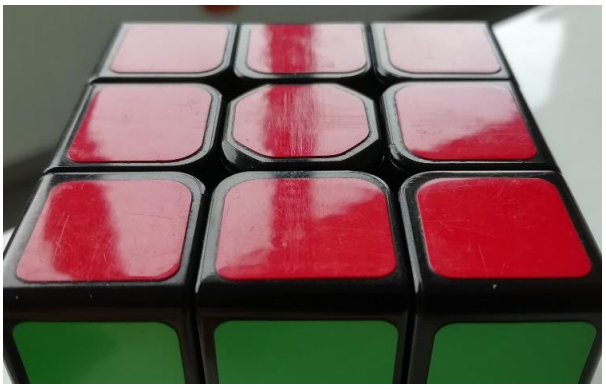
9. **Webcam app ends with "incoherent cube status" statement**
   There are a few reasons leading to this result:
    - The cube orientation isn't correct; The cube faces order should be URFDLB, and oriented to correctly read the numbers 0 to 53 as per the sketch at "High level info" chapter.
    - The 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.

10. 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)



11. Cubotino_GUI.py file doesn't open with Thonny

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.

12. ESP32 doesn't react

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

13. "ModuleNotFoundError" error on the terminal

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; 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).

14. "EOFError: read() didn't return enough bytes" error on the terminal

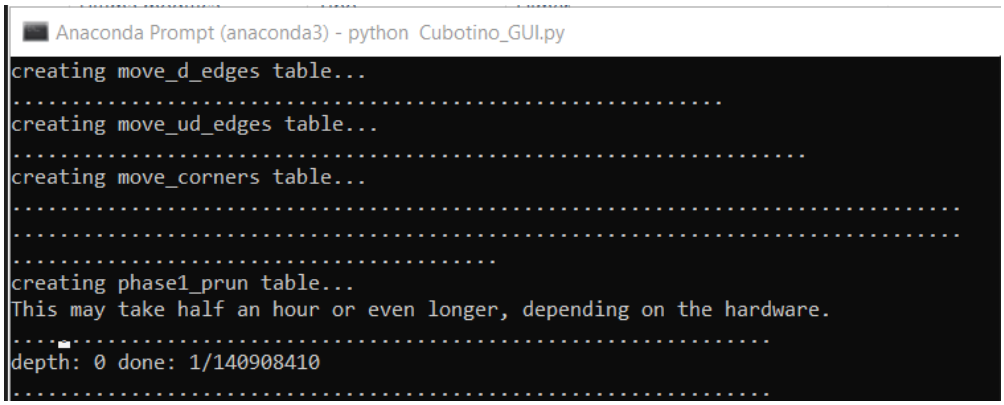
This error is returned on PC running macOS or Ubuntu, in case the RubikTwoPhase library is V1.0.9 or earlier versions are used in combination with the (/twophase) files downloaded from this project repository.
To solve this problem:

a. Update RubikTwoPhase to V1.1.1 (released on 16th November 2022).

Alternative, and longer solution option (at least 30 minutes, on my PC ca 50 minutes)

b. An **alternative** way to solve the problem is to let RubikTwoPhase library generating the look up tables directly on your PC.
   1) Erase all the files under /twophase folder
   2) Activate the venv environment
   3) Enter the project / PC_files folder
   4) Enter Python, by typing: ***python***
   5) From the python prompt (>>>) type: ***import twophase.solver***

   Now the library generates the look up tables; Initial part looks like below snippet:



   After about 30 to 60 minutes the python prompt (>>>) is returned
   6) Exit Python, by typing: ***exit()***

c. Check if the problem is solved:
   1) From the venv environment, enter python, by typing: ***python***
   2) Import the solver, by typing: ***import twophase.solver***

   The solver should show "loading", as the tables are available into folder …/PC_files/twophase

```
>>> import twophase.solver
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...
>>> exit()

(cubotino) D:\CUBOTino_base_version-main\PC_files>
```

   3) Exit python by typing ***exit()***

17. 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).

   Some parameters can be passed as argument when launching the GUI:

Argument	type	range	Description and notes
--debug			Prints to terminal settings, variables, and info
- -delay	int	2 to 30	When webcam is used: Delay in secs to start detecting the facelets after a cube face change. Countdown is showed on screen. Spacebar to jump into detection mode. 10 secs as default if this argument is not passed
--estimate			When webcam is used: Estimates the position of latest two facelets, of each cube face.

   Note: arguments can be combined
   Example: *python Cubotino_GUI.py --debug --estimate --delay 20*
   in this case the debug prints to the terminal are activated, the estimation of latest 2 facelets are activated and the facelets detection start after 20 seconds

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.

10. 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.
11. Robot → The robot can be unplugged at any moment (no shut down procedure needed).
12. GUI → The GUI can be closed at any moment; The cube status on screen sketch is never saved.

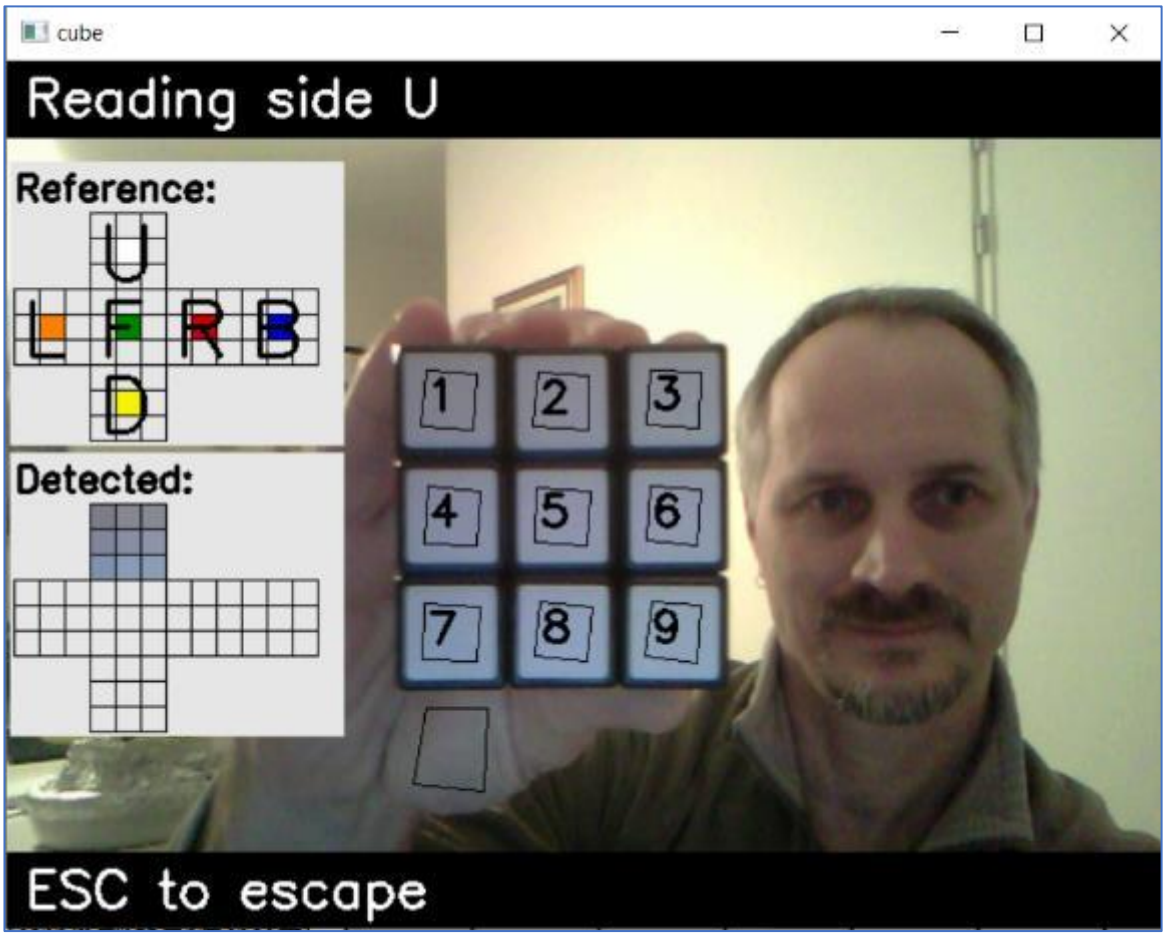
18. Detect the cube status via webcam

It's possible to detect the cube status by presenting the cube to a webcam

- The cube faces must follow a given order: U R F D L B
- The order is suggested on the screen
- Video tutorial showing how to manoeuvre the cube in the /movie folder of your local repo or YouTube https://youtu.be/udr6tryxA_Y

On the Reference sketch the suggested colours are according to the Western colour scheme; This is used as guidance, by considering this is the most diffused cube type.
Non-Western colours scheme cubes can be also used, therefore only the face letters must be followed

When all the 9 facelets are detected, that face will populate the "Detected" sketch; On below example the U (upper) face was under detection:

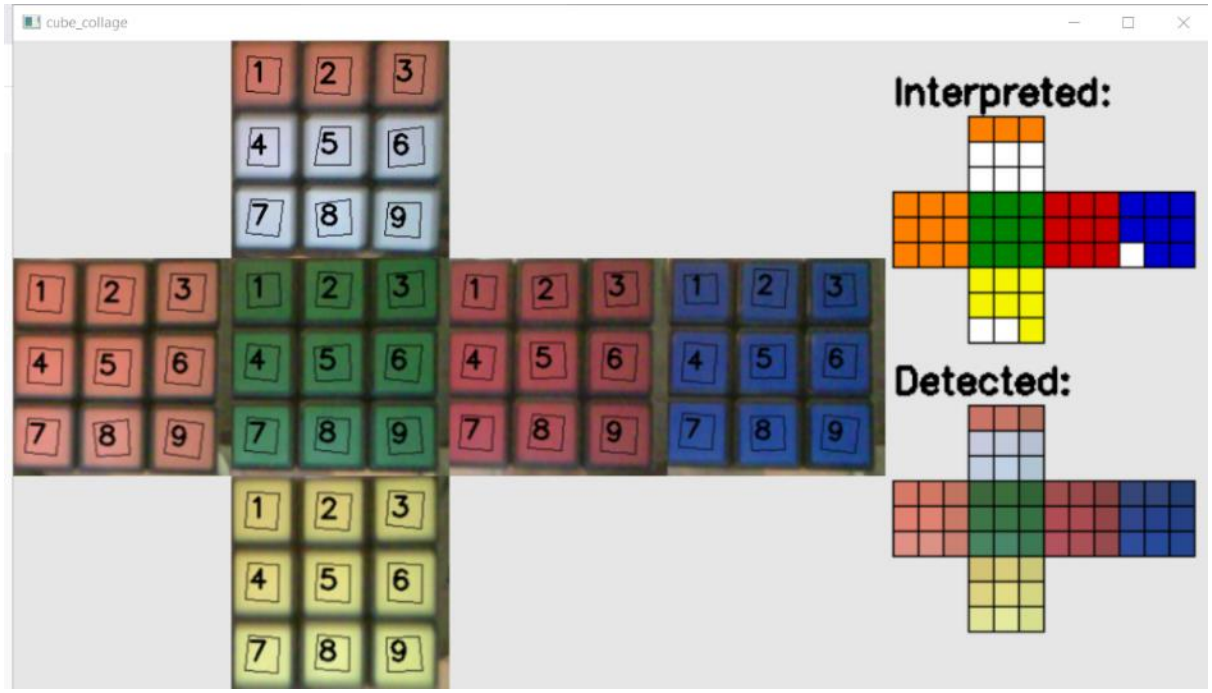


Be noted a contour was detected on my hand (below the facelet 7), yet correctly filtered out.

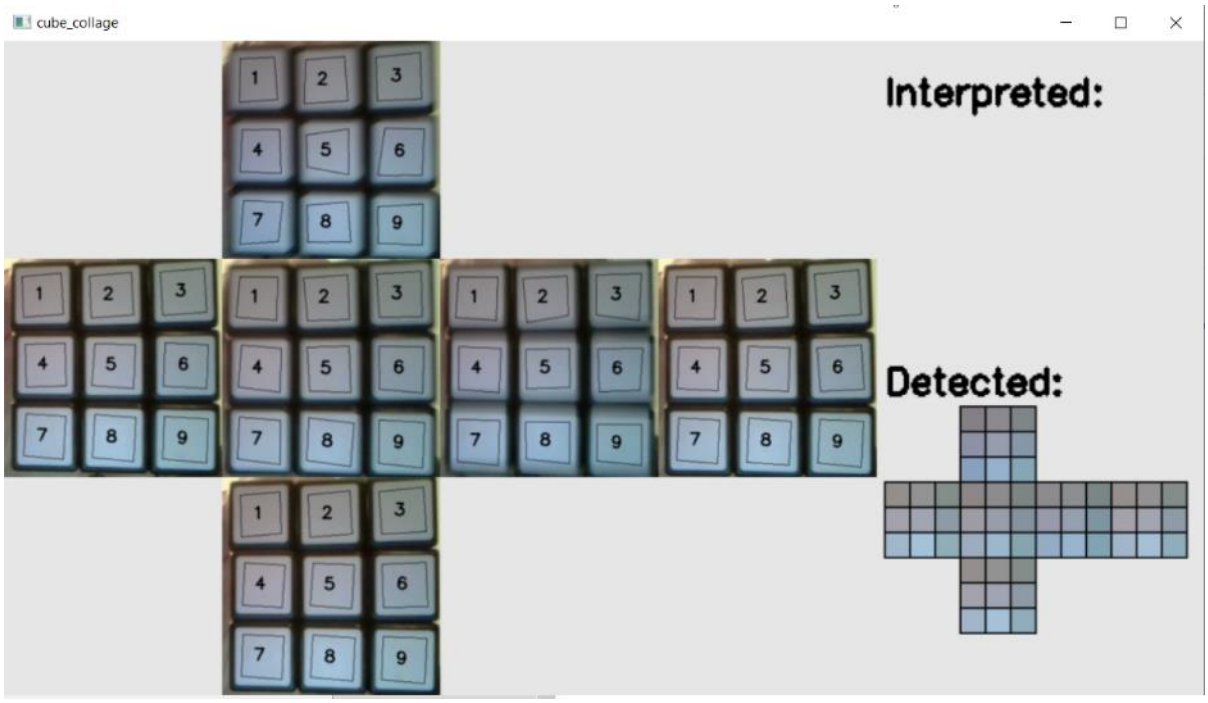
When the option "Estimate facelets" is selected, there will be the facelet number visible without the contour.

In case the 54 Detected facelets don't provide a coherent cube status, a "cube collage" remains on screen for about 10 seconds; There are a couple of cases:

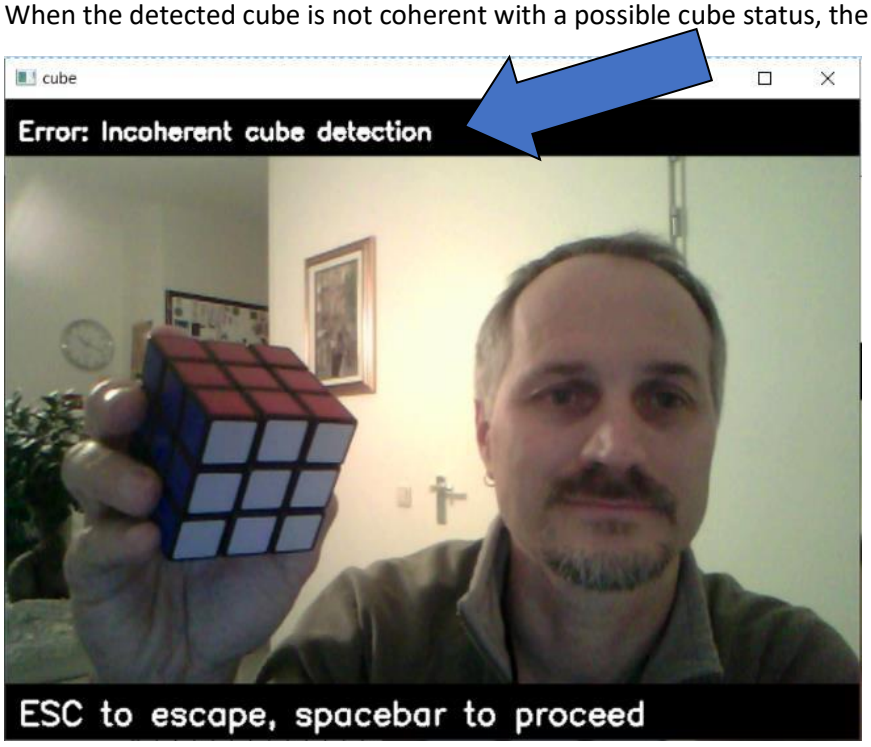
- If the 6 centers have been "correctly" detected, the cube_collage will also show the cube interpretation leading to the failure:



- If the 6 centers haven't been "correctly" detected, the cube_collage won't show the cube interpretation leading to the failure:

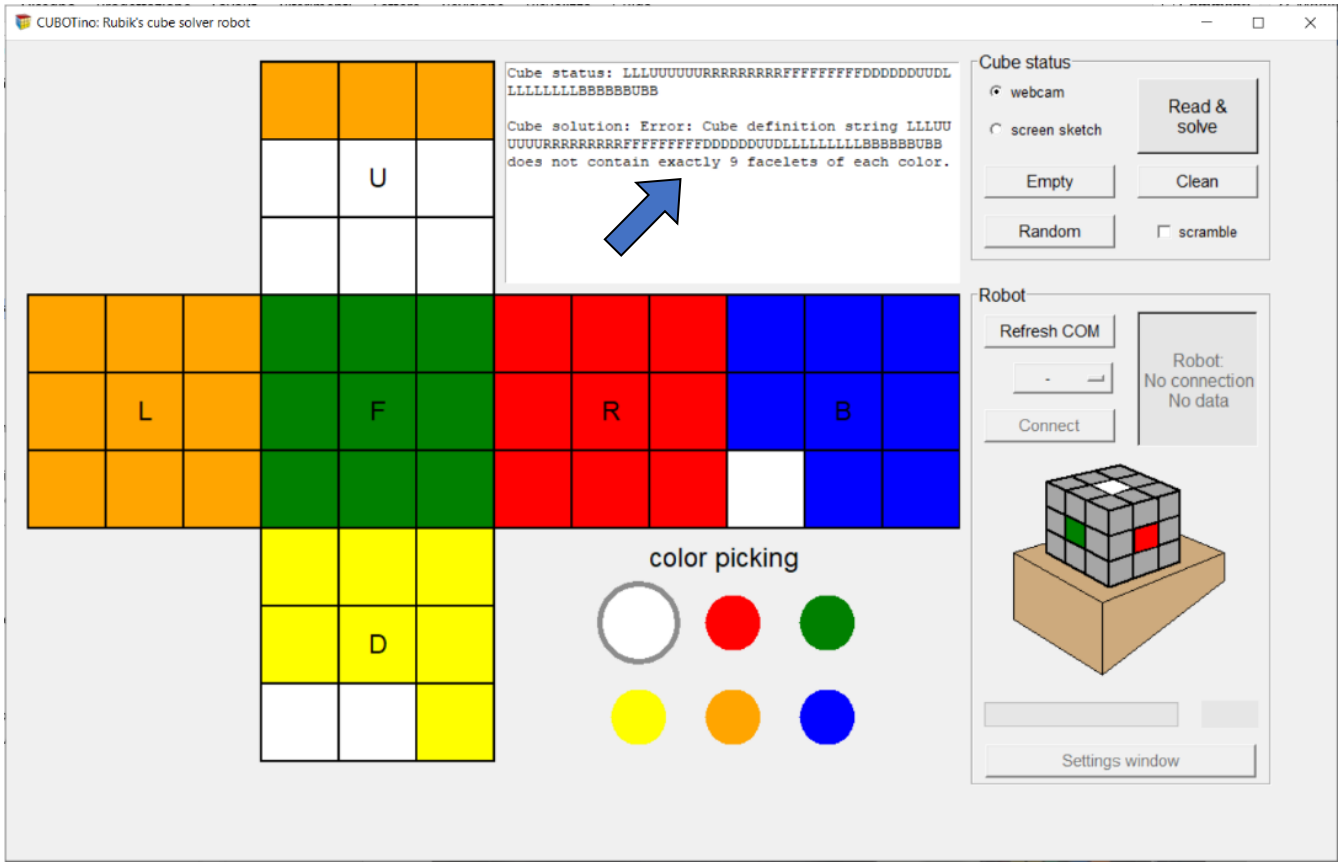


When the detected cube is not coherent with a possible cube status, the GUI inform about an Error:



- By pressing the spacebar, the cube detection process starts again from the U face.
- By pressing the Esc button, the cube status detection via webcam is interrupted, and the main GUI page is shown.

Returning to the main GUI, after an incoherent cube detection error, some info will be provide on the error reason:



19. Project background

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.

20. 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 😊

21. 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:

22. 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 • Cube solution (Kociemba) generated at PC	Finalized (April 2022
Medium	PC dependent, for cube solution	• Cube status detection at the robot • Cube solution (Kociemba solver) generated at PC	Stand-by, see notes below
Top	Autonomous	• Cube status detection at the robot, via a vision system • Cube solution (Kociemba solver) generated at the robot	Finalized (June 2022)

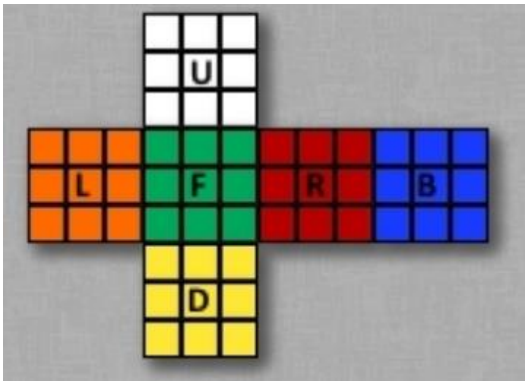
Notes:

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 😊

23. 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						

24. 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 from 2008 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://youtu.be/A3zE4B-gFzk

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 robot dimensions.

25. Robot solver algorithm

On this chapter it's explained 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 (… and only on one flipping direction); 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.

26. Computer vision part

From https://en.wikipedia.org/wiki/Computer_vision, computer vision is an interdisciplinary scientific field that deals with how computers can gain high-level understanding from digital images or videos. From the perspective of engineering, it seeks to understand and automate tasks that the human visual system can do.

In this CUBOTino Base Version, the computer vision part takes place outside the robot, by combining the below elements:

- **Your PC** (the computer)
- **OpenCV** (an open source library for computer vision; From https://en.wikipedia.org/wiki/OpenCV: OpenCV is a library of programming functions mainly aimed at real-time computer vision).
- **Your PC webcam** (the camera)

In which the python script '**Cubotino_GUI.py'** is responsible for the interaction with these elements.

The computer vision is used by the Cubotino_webcam.py to detect the position of the cube facelets; Once the positions of the 9 facelets are known, the colour from those facelets is retrieved and memorized.

Below aspects, are presented on the next pages:

A. Image analysis
B. Contour analysis
C. Colour retrieved
D. Is all this really needed?

Colours detection strategy is described on a dedicated chapter, as in my case it has proved to be the most challenging part of the project.

A. Image analysis:

The approach uses a similar technic as explained at https://medium.com/swlh/how-i-made-a-rubiks-cube-colour-extractor-in-c-551cceba80f0

1. The camera image is converted to grayscale: *gray = cv2.COLOUR_BGR2GRAY(frame….*
2. The grayscale image is filtered with a low pass filter to reduce noise: *blurred = cv2.GaussianBlur(gray, …*
3. The de-noised grayscale image is analysed with a Canny filter; This function transforms the image to binary, assigning 1 (white) the pixels detected as edges: *canny = cv2.Canny(blurred….*
4. The binary image is analysed with Dilate, a morphological operation, aimed to join eventual interruptions of the thin edges returned by the Canny filter: *dilated = cv2.dilate(canny,….*

   The edges are now thicker (or much thicker) according to the kernel definition. Having Thicker edges is a way to reduce the quantity of edges, and gain speed.
5. The "Dilated" binary image, is analysed with Erode, a morphological operation that works opposite of Dilate: *eroded = cv2.erode(dilated….*
6. The Eroded binary image is now used to find contours: *cv2.findContours(image, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)*

B. Contour analysis:

Despite the image preparation, it is very common to get many more, and unwanted, contours; This requires filtering out the contours not having the potential to be a facelet.

1. To facilitate the contours selection, it is convenient to approximate them (those with more than 4 vertices).
2. From the approximated contours, those not having 4 vertices are discharged.
3. The remaining contours are ordered CW, to have the first vertex on top-left.
4. The approximated and ordered contours are then evaluated on
   - *a)* Area, that should be within pre-defined thresholds (linked to the camera resolution set)
   - *b)* Max area deviation, from the median one
   - *c)* Max sides length difference, from a pre-defined threshold (how much is look like a square)
   - *d)* Max diagonals length difference, from a pre-defined threshold (how much it looks like a rhombus)
   - *e)* Max inclination from the horizon (only on one facelet, the top-left after ordering the facelets)
   - f) Max distance from the central one; This step includes ordering the 9 contours, according to their center coordinates.
   - *g)* Quantity of contours left, after discharging those not ok

5. When "Estimate facelets" is not selected, the algorithm waits for the first 9 contours passing through this process.
   When "Estimate facelets" is selected, once there are 7 facelets detected, the algorithm estimates the position of the remaining two.
   The detected and estimated contours are used to generate masks; These masks are applied on the coloured image, as guidance for the facelets position.

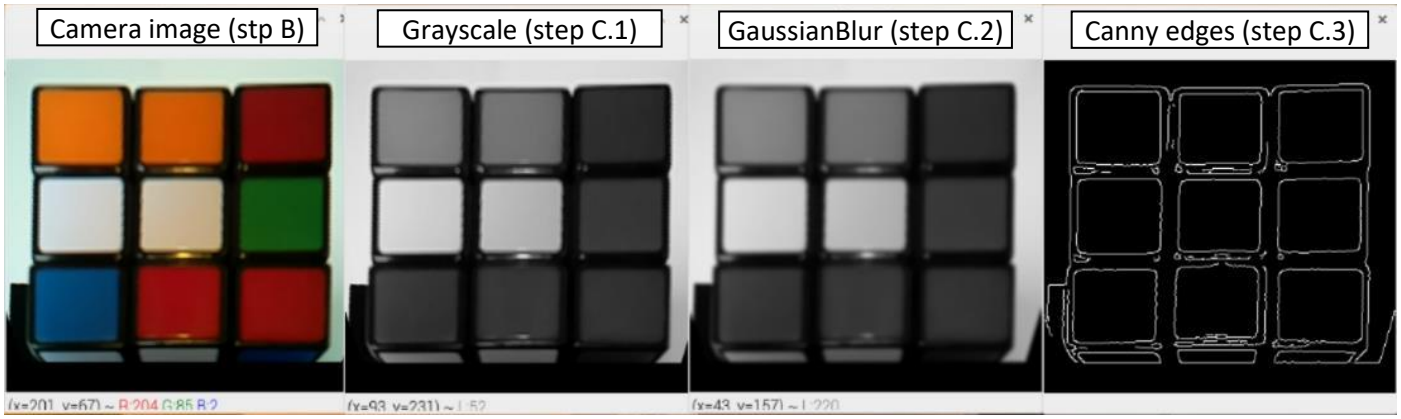
6. The 'detected and accepted' contours are plot over the coloured image, and numbered from 1 to 9 as visual feedback; The 'estimated and accepted' are only numbered (no contours are drawn).

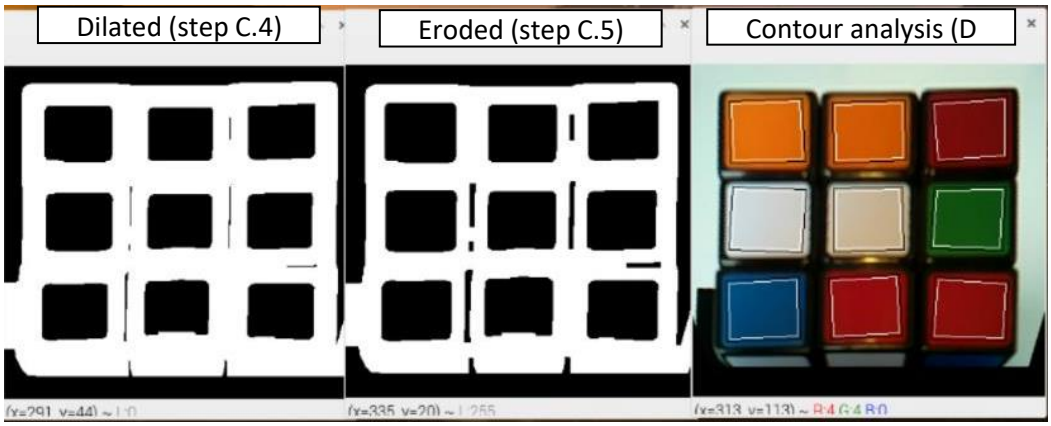
C. Colours retrieved:

On each facelet, 2 main info are retrieved:

1. Average BGR, for a portion of the facelet around the detected contour center.
2. Average HSV, based on the average BGR.

Below how a cube face looks like along the image manipulation:





The described image analysis process is repeated for the 6 cube faces.

D. Is all this really needed?

The clear benefit of this method is on the fact the user isn't obliged on precise cube positioning to the webcam.

Another reason I made choice is that I wanted to learn computer vision back on 2021.

There might be easier ways to reach the same goal, i.e. drawing a fix mask on screen, and ask the user to press a button when the cube appears aligned with the mask: Up to you to experiment 😊

27. Colour's detection strategy (when using the webcam)

Some of the strategies used to detect the cube status aren't described on papers, at least not that I could find on free websites; In particular:

- Collect all the 54 facelets colours info prior to assign them to a reference face colour.
- Switch to an alternative colour analysis when the primary approach doesn't deliver a coherent cube status.
- Dynamic colour reference.

- Cube facelets location are detected via as described in the computer vision chapter.

  Based on the identified contours:

  a. The outer one, in black on below picture, shows the simplified contour retrieved by the edge analysis; This analysis is used to find the 9 facelets per each cube, and to know the contour center coordinates.
  b. The inner one, in white on below picture, depicts a smaller square area centred on the outer contour; This smaller area is used to:
     i. Calculate the BGR average value, used for the colour interpretation according to the 1st method (BGR colour distance)
     ii. calculate the HSV average colours, used for colour interpretation according to the 2nd method (Hue value).

- Properties of the faces center facelet:

  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

- 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 overflows … 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.

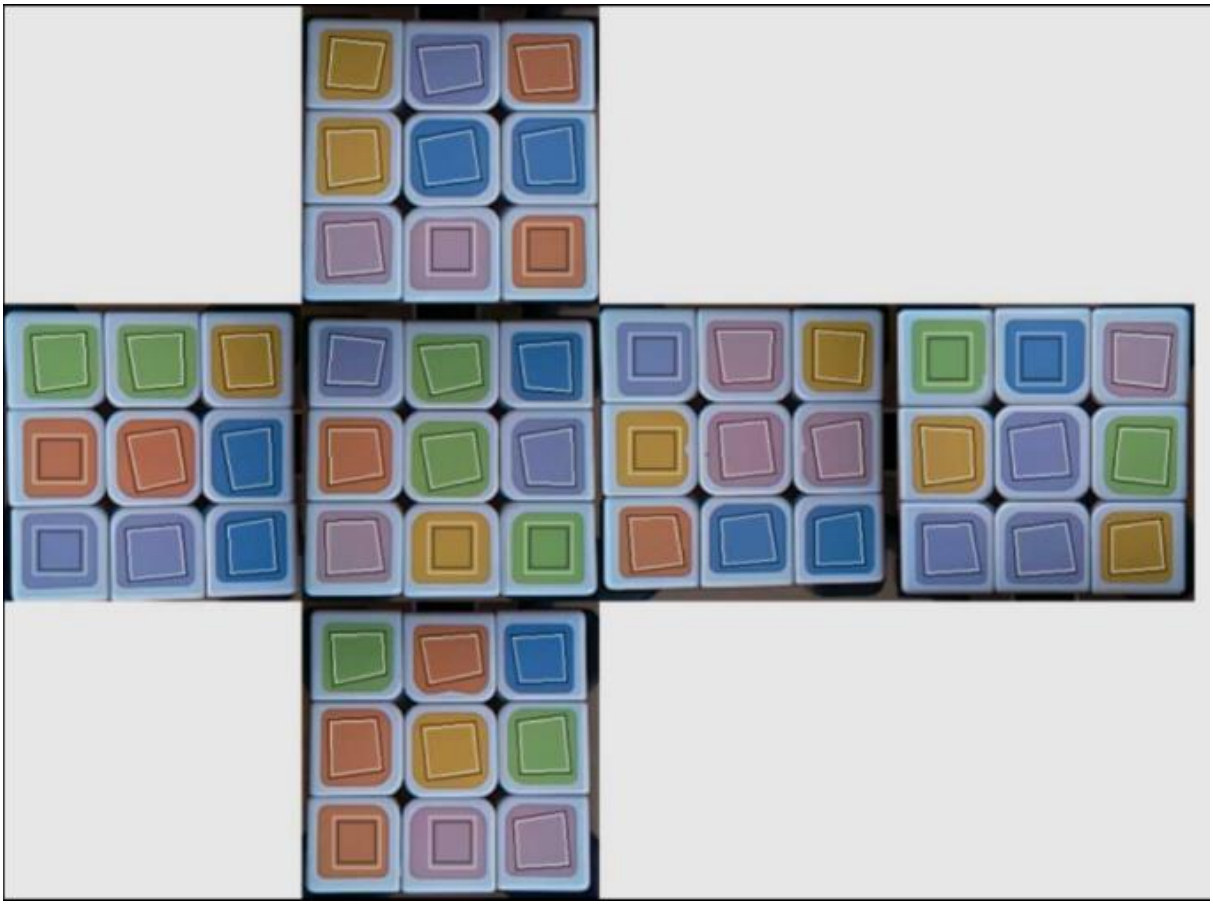
- Based on previous step, the 6 cube colours (at least their centers) have a known average HVS and therefore an average BGR colour; This also informs on the cube orientation (colours) as placed on the cube-holder.

- 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.

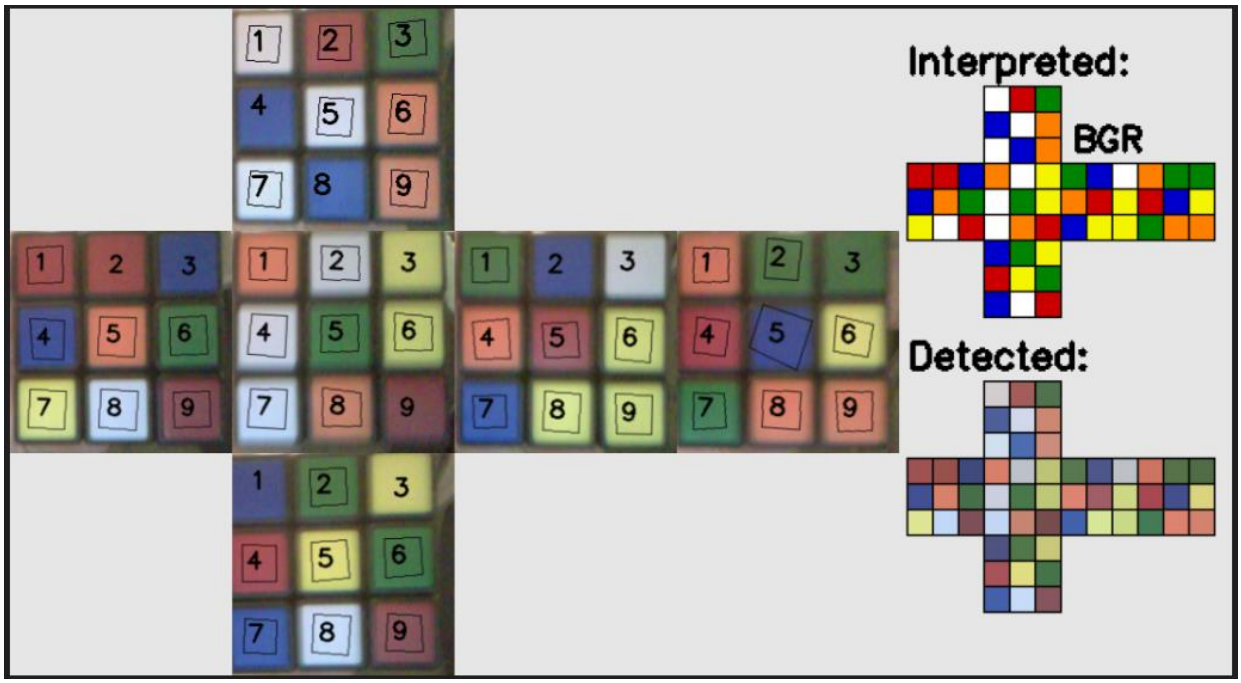
- Dynamic colour reference:
  Facelets association to the cube face (to the cube faces center colours) is made after ordering the facelets by colour distance, and by starting with those having the least distance (the less uncertain choice).
  Once a facelet is associated to the reference, the reference is updated by averaging it with the just associated facelet; In this way the reference keeps updating with the other facelets from the same colour; In a way, the reference becomes more and more representative of that cube face colour.
  This approach is useful to mitigate the camera vignetting effect, or a center face having a logo or a little defect.

- Non-western colour scheme cubes:
  Because of the colour detection strategy used, the cube status can be detected also on non-western colour scheme cubes:



28. Images and data log

Each time a cube status is determined via the webcam a cube_collage of the six faces is generated and saved.
The image name is "cube_collage_yyyymmdd_hhmmss.png and it's saved in the PC_files/CubeStatusPictures folder.
The image is saved also when the interpreted cube is not coherent (**Error**)
When correctly detected, the Interpreted sketch will be indicated **BGR** or **HSV** according to the colour detection method delivering a coherent cube status (see colours detection strategy.
Facelets without a contour are those that have been estimated on their positions, based on those previously detected.



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

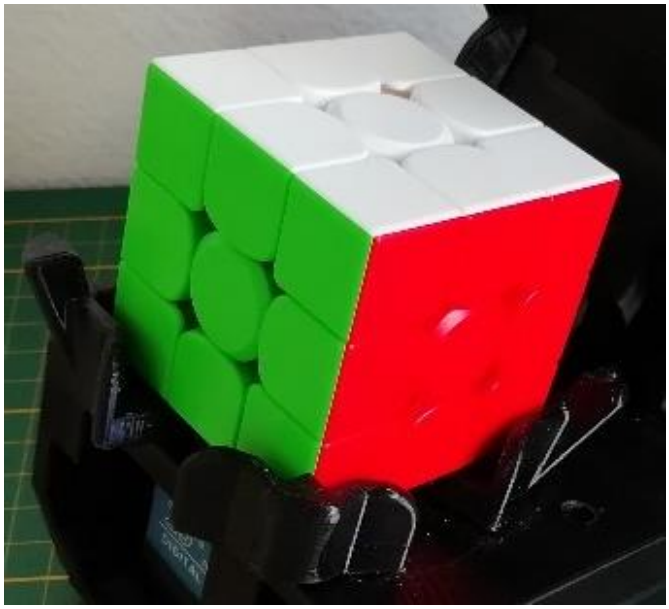
Column name	**Info**
Date	Date and time (yyyymmdd_hhmmss), i.e. 20220308_213439
CubeStatusEnteringMethod	"screen sketch" or webcam
CubeStatus	i.e. RLFDUUDBBLFRURRBDDRDDFFLURULDRFDFLUFDLBRLRFLBUBFUBBLBU
CubeSolution	i.e. D2 L2 F2 R3 F2 L1 D2 F2 R3 U2 F1 L1 F3 R1 B2 F3 D3 L2 F2 (19f)
RobotoMoves	i.e. R1S3R1S3S3F1R1S3R1F1R1S3R1S3F3S1R3S3F1R1S3R1S3F3R1S3 F1R1S3R1F3R1S3R1S3S1F3R3S3F1R1S3R1F3R1S3F3R1S3S1F1R3F1R1 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)

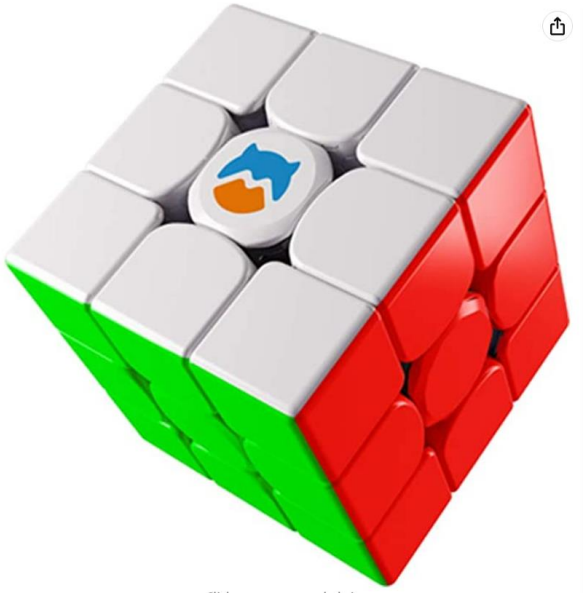
Notes:

1. The folder *data_log_folder* is made from the folder where Cubotino_GUI.py is running (PC_files).
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.

29. A convenient cube





Thanks to a note from Art, I got to know the existence of “frameless” cubes
Thanks to a note from Martin, I got to know about the existence of “magnetic” cubes

This two information were shared close to each other, therefore I decided to buy one cube having both those characteristics.
I ordered a “Monster Go Magnetic 3x3 Cube” in Amazon for about 16€ (https://www.amazon.com/Monster-Go-Magnetic-Learning-Beginners/dp/B087RMGJ12/)

Well, my children and myself are very impressed by this cube, to the point I immediately ordered a second one.
This cube is very smooth, requires little force to turn, it easily cuts corners

The reason I’m especially pleased by this cube is because of the magnets, that keep the layers very well aligned: This makes a perfect cube for the robot.
In terms of dimensions, this cube is halfway the two I had home, on which I based the robot parts.
Of course, the logo must be scratched away.

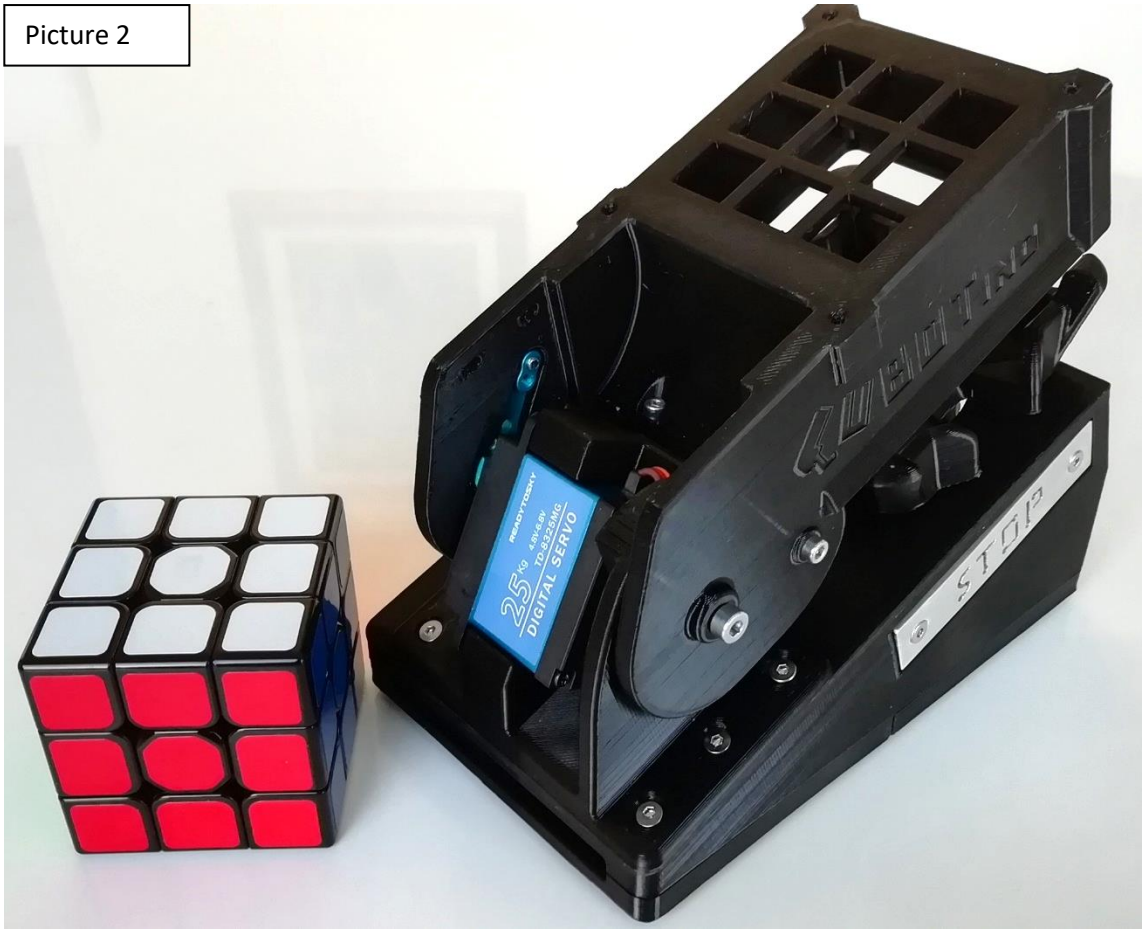
Detecting the cube status with the camera is slightly less fast, because of the black frame absence.

Note: I’m not affiliated with Amazon, nor any other Company, I’m just sharing my positive experience with this cube 😊

30. Collection of robot's pictures

Picture 1

Picture 2



Picture 3

31. 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 (Lifter integrated with Upper_cover) 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 two power inputs.
    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

32. Next steps

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

34. 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 leaving 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

35. 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.
- to Andreas, the guy with Swiss accent, for the good suggestions to make this project easier to others.
- to Yannick, who has "forced" me to learn about GitHub repo, making easier the project management for me and for the Makers.
- to John, who discovered the /twophase/files incompatibility across different architectures; John did not accept my simplistic solution proposal, he dig the problem up to the source and made possible to get best possible solution, directly in the solver.
- all the people who have provided feedbacks, both positive and negative 😊

36. Revisions

Rev	Date	Notes
0	06/04/2022	First release
1	09/04/2022	Added the scrambling function; New files at PC: • Cubotino_GUI.py • Cubotino_moves.py Improved servos (re)initialization, after cube solving and robot stopping; New files at ESP32: • main.py • Cubotino_servos.py • Cubotino_moves.py
1.1	09/05/2022	Added some notes on this document: • Info to type the file extension when saving them in ESP32 • 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: • Cubotino_webcam.py file to work without “scipy” library; Not anymore needed to install that library. Updated (accordingly): • Files needed at PC on this doc • Troubleshooting
1.4	22/05/2022	Modified: • Cubotino_webcam.py An error was preventing the HSV color detection to work when “debug” was set True. Updated: • Doc revision
1.5	23/05/2022	Modified: • Cubotino_webcam.py A quote error was preventing script from working at all. Updated: • Doc revision
2.0	04/06/2022	Modified: • Hinge.stl, Lifter.slt Increased the gap between Top_Cover and Hinge, at the side of servo output gear Added: • Personaliz_plate Updated: • Cubotino models table, conclusions and next steps (Top version now available). • Doc revision

3.0	11/06/2022	Modified: • Supplies list, by adding a microUSB breakout board • Connections board, by splitting the power supply for servos from the ESP32 • Info related to the two power supply, and related How to use the robot • Cubotino_GUI.py and Cubotino_webcam.py to tentatively import Kociemba solver from multiple installation locations, and to allow window dimension adjustment • Structure.stl for (better) compatibility with the Top_version Added: • Kociemba installation chapter Updated: • Troubleshooting, with more details for the Top_cover < -- > Hinge gaps and friction • Doc revision
3.1	30/06/2022	Modified Lifter.stl to increase the clearance toward the Hinge
4	09/10/2022	Very large update: 1. Uploaded the project to GitHub, to make easy the complete installation process 2. Add the servo_to_mid.py script at ESP32, to test servos and set them to mid position ~~3. Modified Cubotino_GUI.py to deal with 2 different servo Pulse Width ranges~~ 4. Modified Cubotino_webcam.py to better deal with non-MS Windows OS 5. Instructions: • Re-ordered the content, with initial part fully focused on the making process • Improved the tuning session • Added: • alternative servo type to the supplies list • GitHub repo related info • how to flash MicroPython • info about libraries to install • images for the relevant robot positions • info/pictures on how to increase the servo rotation range • convenient cube type info
4.1	11/10/2022	Modified Cubotino_GUI.py: • to deal with 2 different servo Pulse Width ranges • to save a backup copy of the settings • to upload the backup personal settings after a (git) update • removed the button that was loading my personal settings. Modified Cubotino_servos.py: • to deal with 2 different servo Pulse Width ranges • corrected a small bug Instructions: • Added the edge detection chapter GitHub, uploaded: • updated Cubotino_GUI.py • updated Cubotino_servos.py • updated “How_to_=make ..pdf” instruction

4.2	20/10/2022	Modified Cubotino_GUI.py: • (V4.1.1) Solved a bug when only one of the two backup settings files is available. • Added a checkbutton (beta version) to activate the (last) facelets positions • Added a checkbutton to printout info related to webcam cube status detection • Added some more prints to the terminal. Modified Cubotino_webcam.py: • Solved a bug when the variable debug was set true (at __main__), and the script was executed directly (not from the GUI) • Added text on cube_collage for the colour detection method • Cube_collage images are now saved (on a PC_files sub-folder) • Cube_collage is quickly shown also when cube status is coherent. Instructions, added info on: • repo cloning via git • how to update the local repository made via git • webcam usage • cube_collage image saved at every detection via the webcam GitHub, uploaded: • updated Cubotino_GUI.py • updated Cubotino_webcams.py • updated “How_to_=make ..pdf” instruction Modified Hinge.stl to accept taller servos (up to 43mm)
4.3	12/11/2022	Modified Cubotino_GUI.py: Solved a bug generating an error when the file was updated from a project made with version 4.1 or earlier
4.4	21/11/2022	Added info about “EOFError: read() didn't return enough bytes”
4.5	17/01/2023	Modified Cubotino_GUI.py: Added arguments to be passed via the CLI. Modified Cubotino_webcam.py: Added a delay to start detecting the cube facelets from the cube face change. Updated instructions: Arguments are explained at “How to use the robot”.

Note: Instructions and files will be updated whenever needed