

How to make

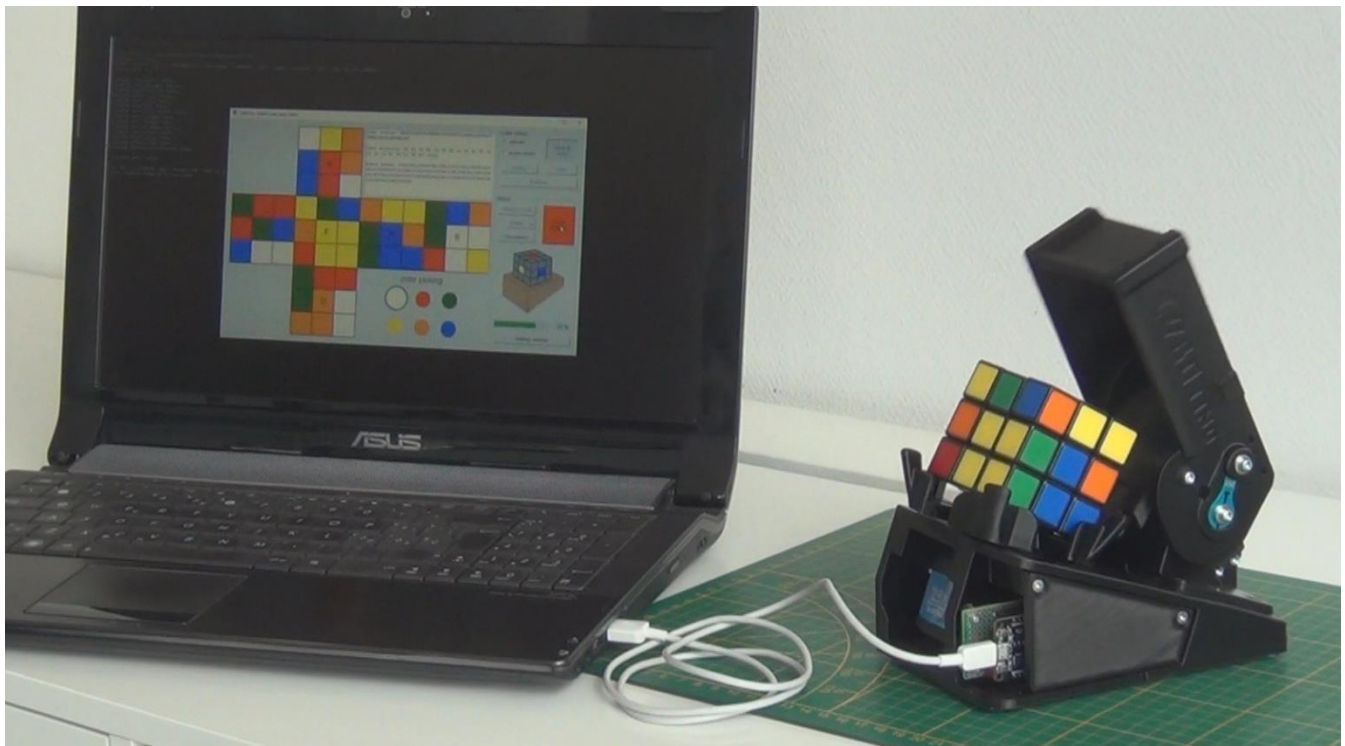
## CUBOTino: A small, simple, inexpensive Rubik's cube robot solver

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

Andrea Favero, Groningen (NL)

06/04/2022

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



This robot is meant to be very simple and inexpensive.

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

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

# Summary

1. Introduction:	3
2. Project scope:	3
3. Robot name:	3
4. Conclusions:	4
5. Next steps:	4
6. Commitment:	4
7. Safety:	5
8. Models:	5
9. High level info (Base version):	6
10. Construction:	7
11. 3D printed parts:	8
12. Bought parts:	11
13. ESP32 dev. board (30 pin) pinout:	12
14. Connections board:	13
15. Setting up the ESP-32 microcontroller:	15
16. Files needed at PC:	18
17. GUI	19
19. Data logged	24
20. Assembly steps:	25
21. How to operate the robot:	26
22. Tuning:	27
23. Troubleshooting	28
24. Robot solver algorithm:	30
25. Colour's detection strategy (when using the webcam):	31
26. Credits:	32
27. Assembly details:	33
28. Collection of robot's pictures:	47

## 1. Introduction:

To explain why I've started this project I've to shortly mention my previous Rubik's cube solver robot....

That one is based on a Raspberry pi 4B (2Gb ram) with a Picamera, it reads the cube status via the camera system, and it solves it: A full autonomous robot.

The complete process takes less than one minute, some references:

- How to make it: <https://www.instructables.com/Rubik-Cube-Solver-Robot-With-Raspberry-Pi-and-Pica/>
- Youtube: <https://youtu.be/oYRXe4NyJgs>

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

## 2. Project scope:

Based on the introduction you probably can guess the project scope 😊

This second Rubik's cube solver robot project wants to be affordable, to attract more people and especially students into robotics and programming.

The overall idea is to build a scalable robot, based on a minimalist base version.

Project targets for this robot:

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

## 3. Robot name:

I've started this project with the idea to write and share the instructions, and along the way I thought a robot name would make the project more complete.

By combining CUBE, ROBOT and -INO, the Italian suffix for diminutives (to remark the small robot size), the chosen name is CUBOTino

By considering the Top cover, combined with the Lifter, has a "C" profile shape, then CUBOTino become:

The logo for CUBOTINO is rendered in a bold, orange, blocky font. The letter 'C' is stylized to resemble a mechanical profile or a cube's corner. The letters 'U', 'B', 'O', 'T', 'I', 'N', and 'O' are also blocky, with the 'O's having a small square cutout in the center. The 'I' is a simple vertical bar. The overall style is clean and industrial.

## 4. Conclusions:

Only the Base version is finished so far, therefore the conclusions are based on this version:

Pro:

1. The robot and the GUI are working fine
2. Costs are relatively low (about 30 to 40 euro of material, on early 2022)
3. Robot dimensions are very small
4. Rather easy to set the robot parameters via the GUI

Cons:

1. Solving time is from 60 to 90 seconds.
2. Micropython documentation is somehow limited.
3. 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.

## 5. Next steps:

Work out the other two robot versions

## 6. Commitment:

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

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

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

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

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

## 7. Safety:

Energize the robot only via USB ports having a class 2 insulation from the power supply net (laptops and PC normally have this safety feature).

Despite the robot mechanical force is limited, it has to be operated only under adults' supervision.

If you build and use a robot, based on this information, you are accepting it is on your own risk.

## 8. Models:

This project considers the robot to be scalable.

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

Model	Type	Main directions	Status
<b>Base</b>	PC dependent, for cube status and cube solution	<ul style="list-style-type: none"><li>• Cube status entered on the GUI, via mouse or PC webcam</li><li>• Cube solution (Kociemba) generated at PC</li></ul>	Finalized
<b>Medium</b>	PC dependent, for cube solution	<ul style="list-style-type: none"><li>• Cube status detection at the robot</li><li>• Cube solution (Kociemba solver) generated at PC</li></ul>	Started, see notes below
<b>High</b>	Autonomous	<ul style="list-style-type: none"><li>• Cube status detection at the robot, via a vision system</li><li>• Cube solution (Kociemba solver) generated at the robot</li></ul>	Not started yet

### Notes:

The cube status detection method I'm trying 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 is not successful yet; I'm even doubting if it really fits the overall project scope, as this method involves too many skills.

Anyhow I've some other ideas for this Medium model....

The **High** version is still on my head, I hope to get it started soon

## 9. High level info (Base version):

1. To minimize the robot costs of this Base version, and to enhance DIY, the below aspects were considered:
  - a. Use the PC computation power (Cube detection status, and cube solution).
  - b. Use the PC power, to energize the robot via USB (only components meant for 5Volts or lower); The usb port also used for communication.
  - c. Use two 180° servos for all the movements.
  - d. Easy to find components.
  - e. 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](https://en.wikipedia.org/wiki/Rubik%27s_Cube#Move_notation)
6. Cube’s orientation considers the Western colour scheme (<https://ruwix.com/the-rubiks-cube/japanese-western-color-schemes/>):

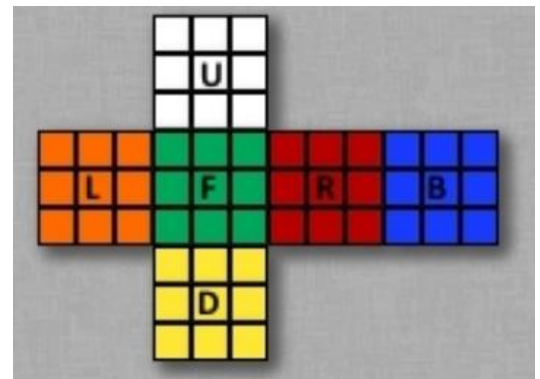
The Western color sheme (also known as BOY: blue-orange-yellow) is the most used colour arrangement used not only on Rubik’s Cubes but on the majority of cube-shaped twisty puzzles these days.

Cubers who use this colour scheme usually start solving the Rubik’s Cube with the white face and finish with the yellow; This colour scheme is also called **Minus Yellow** because if you add or extract yellow from any side you get its opposite.

white + yellow = yellow

red + yellow = orange

blue + yellow = green



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

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

## 10. Construction:

The robot mechanical targets are:

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

Construction principles:

1. The inclined cube-holder is inspired to Hans construction [Tilted Twister 2.0 - LEGO Mindstorms Rubik's Cube solver - YouTube](#);  
This is a clever concept, as it allows to flip the cube around one of the horizontal axes by forcing a relatively small angle change (about 30 degrees, over the 20 degrees of the starting cube holder angle); Once the cube center of gravity is moved beyond the foothold, the cube falls on the following face thanks to the gravity force.  
Overall, it allows to flip the cube via a relatively small and inexpensive movement.
2. The Top-cover, combined with the cube Lifter, is the logical simplification step from my previous robot:
  - The Top-cover provides a constrainer for cube layer rotation, further than suspending sensors for the cube status detection, while keeping a compact robot construction.
  - The cube Lifter flips the cube around one of the horizontal axes.
  - Top-cover with integrated lifter is directly actuated by a servo, therefore controlled via angle.Overall, it allows to combine multiple functions in a relatively small space, parts quantity, and costs.
3. Cube-holder is mounted directly to a servo, therefore controlled via an angle.
4. All parts are made by 3D printing:
  - This makes possible to pursue the needed geometries, also complex shapes.
  - The biggest parts can still be printed on a relatively small plate (min plate 200x200 mm).
  - Some of the parts are split, mainly for easier, and better, 3D printing; Others are split for assembly reasons.
  - All the overhangs have been designed to enable 3D printing without support.

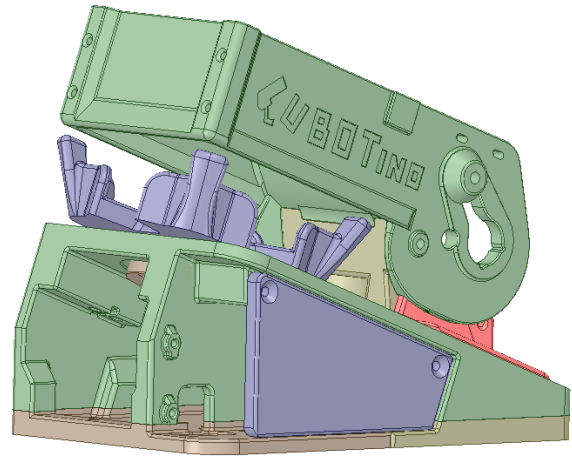
There are many different examples of Rubik's cube solver robot, based on two servos; I think most of them are variations from the one made by Matt's on 2014: <https://hackaday.com/2014/06/28/rubiks-cube-solver-made-out-of-popsicle-sticks-and-an-arduino/>

In these executions, the solution used to flip the cube on one of the horizontal axes, requires the main pivot (servo) to be placed rather far from the cube; This obviously increases a lot the overall dimensions.

## 11. 3D printed parts:

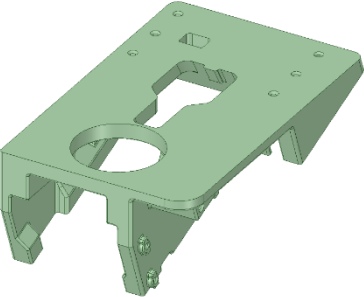
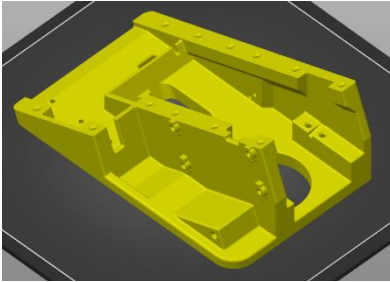
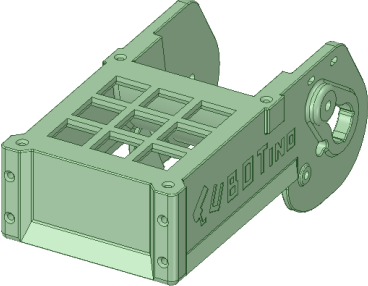
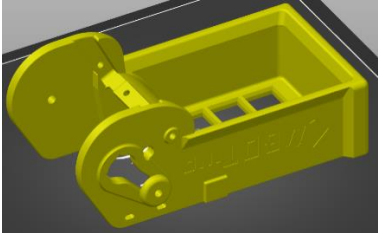
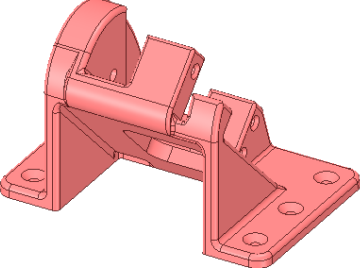
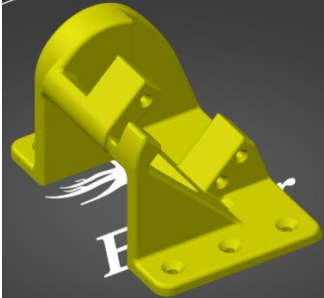
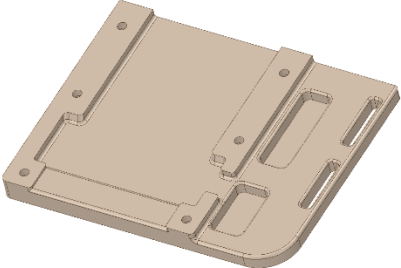
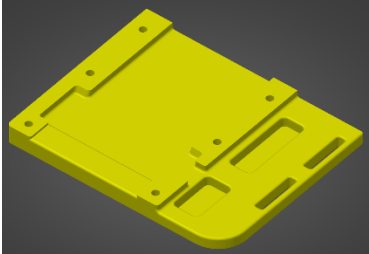
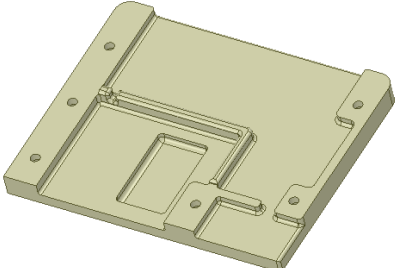
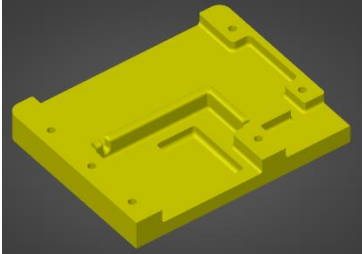
See notes below for filament quantity, and printing time ...

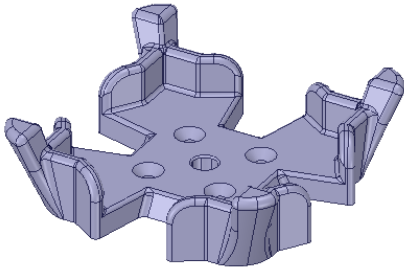
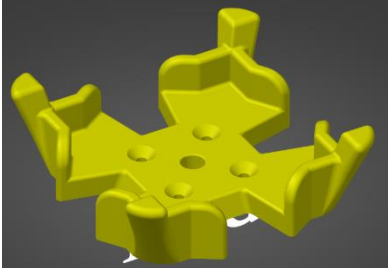
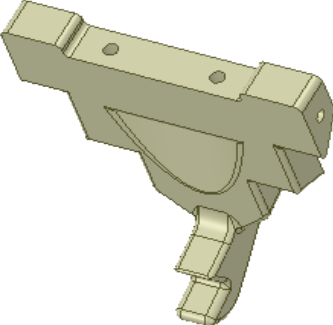

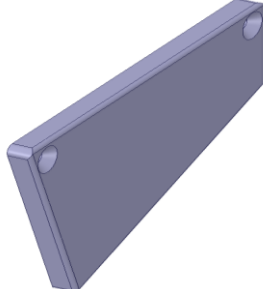
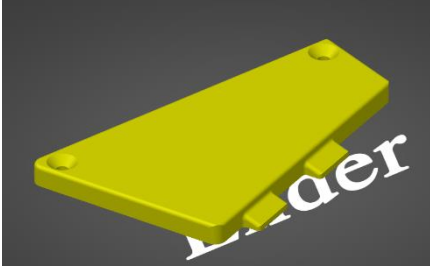
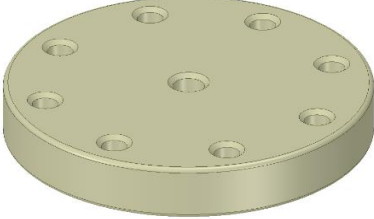
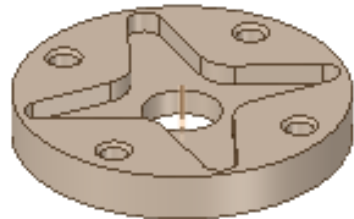
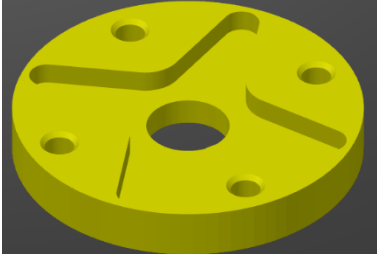
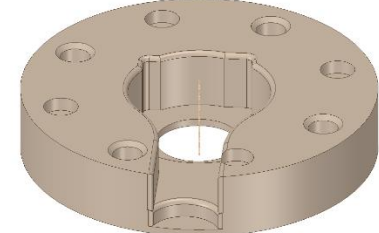
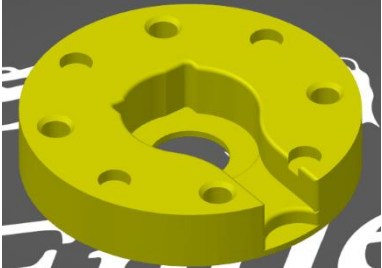
Ref	Part	Filament		Printing time
		meters	grams	
1	Structure	49	145	14h33m
2	Top_cover	39,5	117	12h26m
3	Hinge	17,2	51	5h37m
4	Baseplate front	12,2	36	3h33m
5	Baseplate back	12,6	37,3	3h36m
6	Cube_holder	11,6	34,4	3h36m
7	Cube_lifter	5,5	16,2	1h48m
8	PCB_cover	5,1	15	1h23m
9	Servo_axis_sup (or its alternative)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





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  $\varnothing 1.75\text{mm}$ .
3. Filament weight is based on PETG density ( $1.23\text{g/mm}^3$ ), and printing settings I've use on my Ender 3 printer, for accurate result:
  - 0.2mm layers
  - Low speed (between 25 to 40mm/s for the external parts and 1<sup>st</sup> layer)
  - 4 layers on vertical walls
  - 5 layers on horizontal walls
  - 30% filling
  - 8mm brim
4. The filament quantity, and printing time, in the table should be considered as an upper limit. After printing, the parts total weight is closer to 400grams than 466grams estimated by the slicer SW.
5. All parts have been designed to be printed printing without supporting the overhangs.
6. Some parts have been split, for easier and better 3D printing.
7. The suggested part orientation for the 3D print is showed on below Table.
8. The stl files are available on below websites:
  - <https://www.instructables.com/CUBOTino-a-Small-Simple-3D-Printed-Inexpensive-Rub/>
  - <https://www.thingiverse.com/thing:5342368>

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

Cube_holder		
Cube_lifter		
PCB_cover		
Servo_axis_sup (or its alternative)		Symmetrical
Servo_axis_inf		
Alternative Servo_axis_inf		

## 12. Bought parts:

Q.ty	Part	link to the shop I used	Cost (euro)	Notes
1	ESP32 30 pins development board	<a href="https://www.aliexpress.com/item/32864722159.html?gatewayAdapt=glo2ita&amp;gatewayAdapt=glo2ita&amp;spm=a2g0o.9042311.0.0.27424c4dBteSlp">https://www.aliexpress.com/item/32864722159.html?gatewayAdapt=glo2ita&amp;gatewayAdapt=glo2ita&amp;spm=a2g0o.9042311.0.0.27424c4dBteSlp</a>	4.5	
2	Servo TD-8325MG (180deg 25Kg metal) and metal arm "25T"	<a href="https://www.aliexpress.com/item/32298149426.html?gatewayAdapt=glo2ita&amp;spm=a2g0o.9042311.0.0.5d1e4c4d14Qjaz">https://www.aliexpress.com/item/32298149426.html?gatewayAdapt=glo2ita&amp;spm=a2g0o.9042311.0.0.5d1e4c4d14Qjaz</a>	25 (2 servos + 2 arms)	180 Degree Servo 2PCS + 25T Arm 2PCS (Control by Remote Control) 
<500g	Filament 1.75mm		~10	Suggested PETG, yet other material will do the job

### 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:

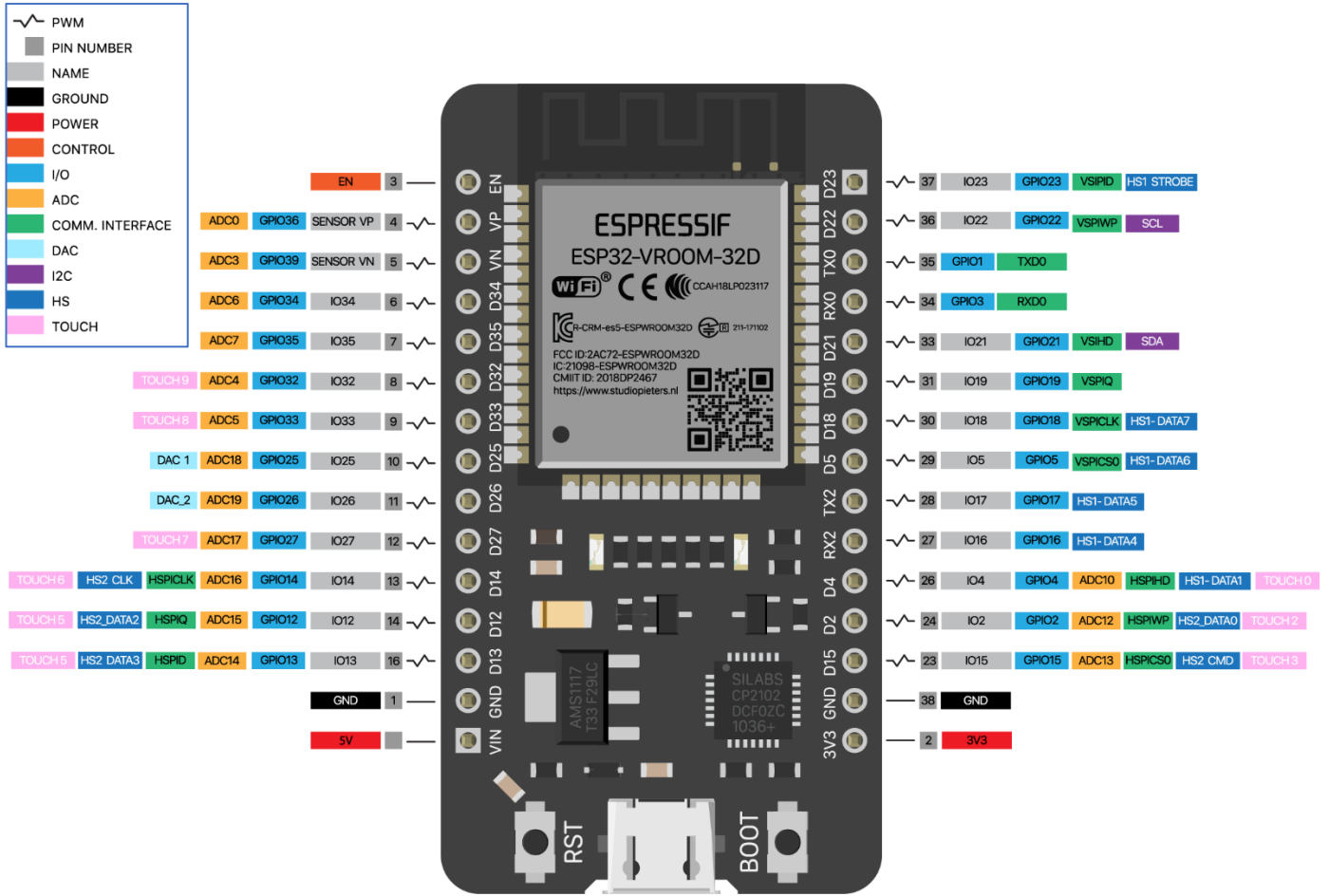
Quantity	Dimension	Head type
1	M4x20	Cylindrical
~ 20	M3x12	Cylindrical
~30	M3x12	Conical
4	M2.5x10	Cylindrical

### 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

Off course some other common materials are needed (USB-Microusb cable having with data lines, wires, solder and solder device, tire wraps, self-adhesive rubber feet, etc).

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



Used pins:

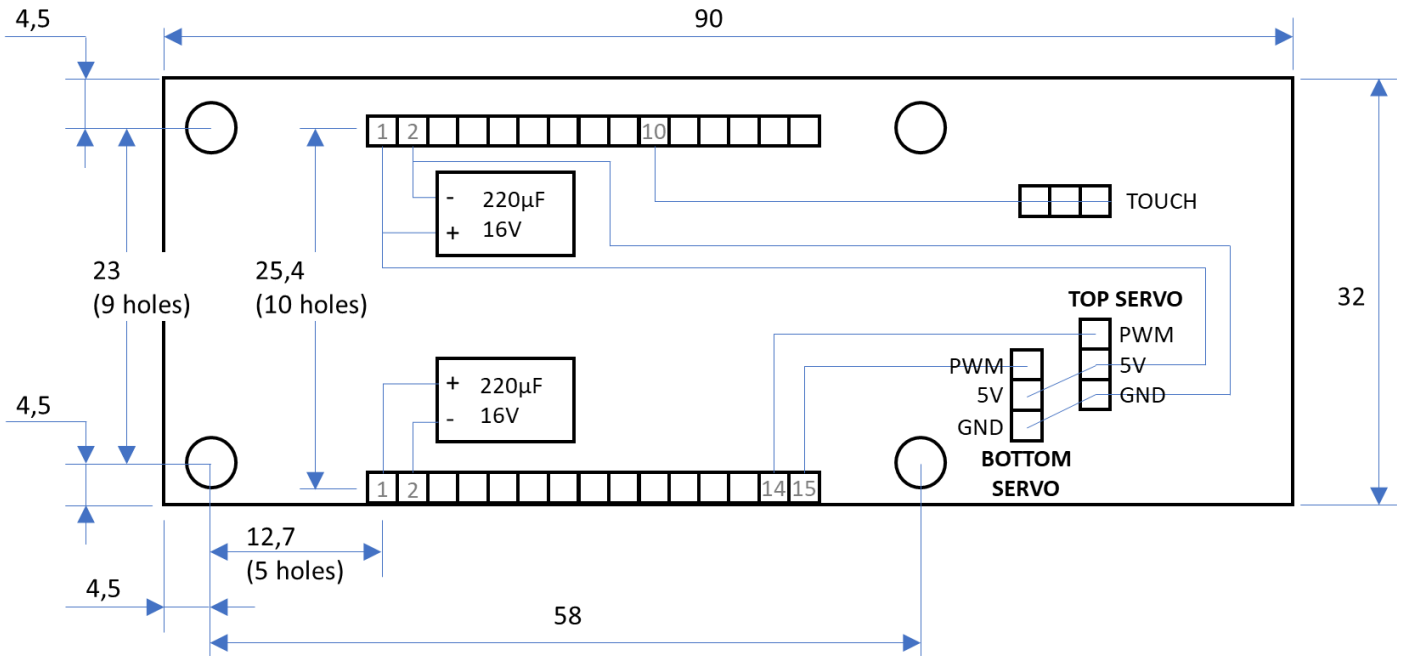
Pin	Purpose
D22	Cube Holder servo
D23	Top cover servo
D32	Touch plate for robot stop

Notes:

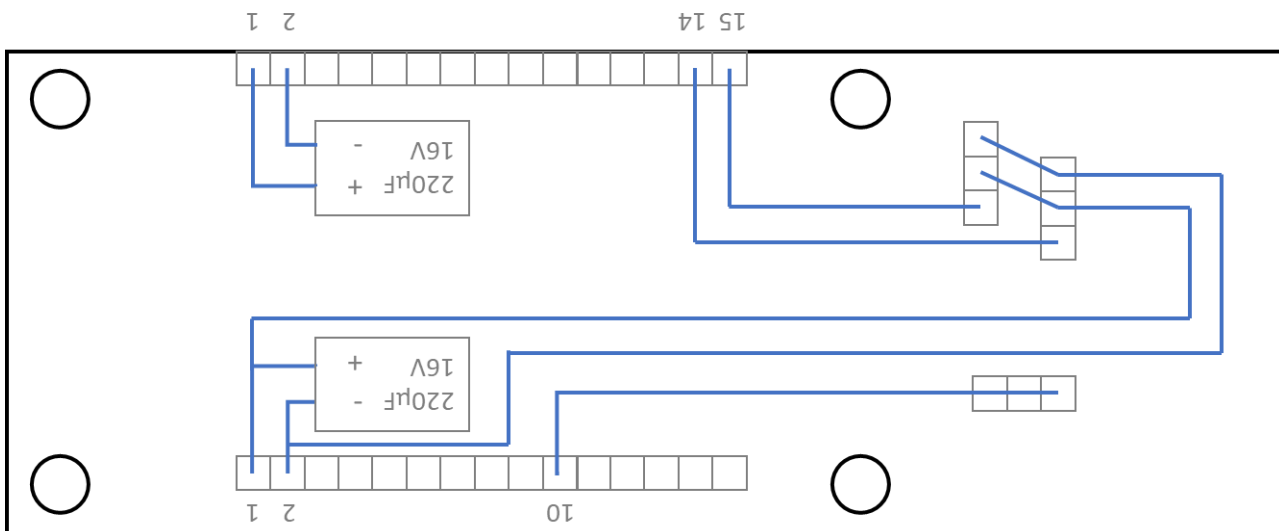
Added two capacitors (16V 220uF) at Vin and 3V3 vs GND

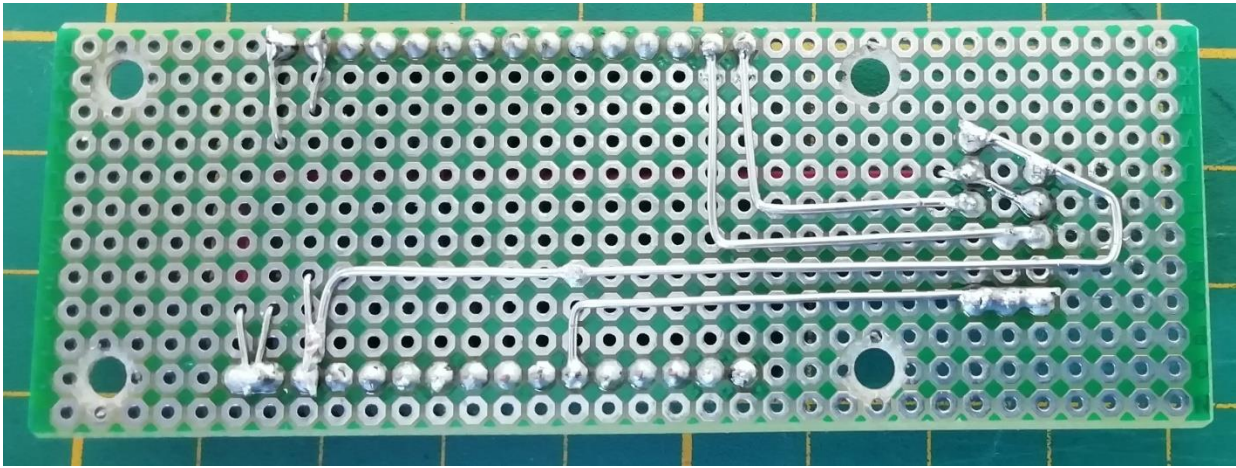
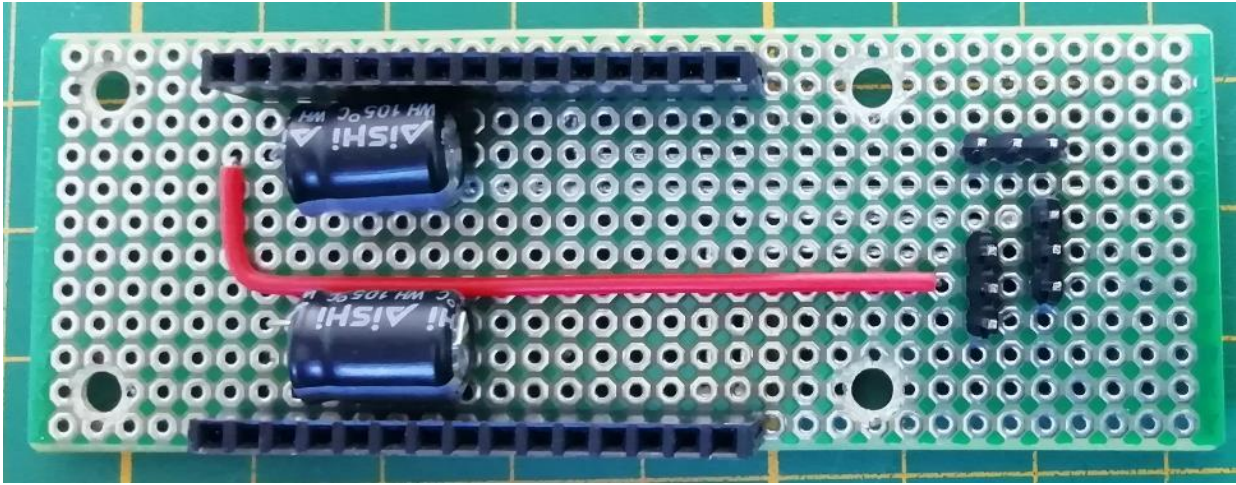
## 14. Connections board:

Top view:



Bottom view:



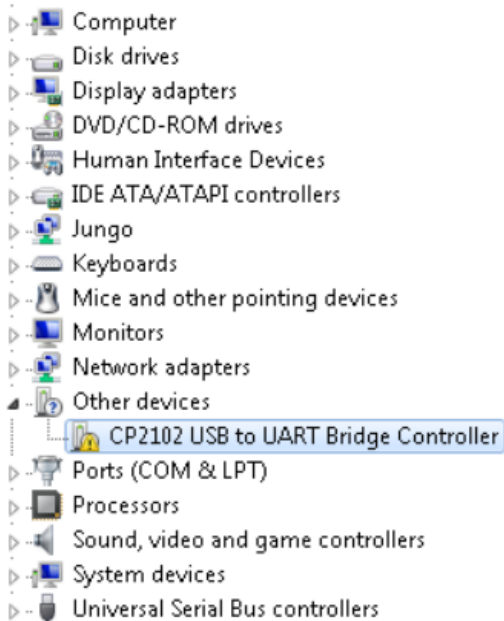


## 15. Setting up the ESP-32 microcontroller:

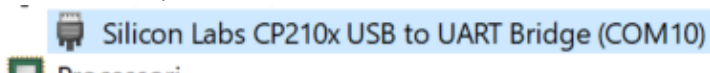
Step1: Before starting, do check if you've the driver for USB to UART communication installed:

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

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



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



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

Step2: Download Micropython firmware for the ESP32;

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



Step3: Download Micropython firmware [v1.17 \(2021-09-02\)](#)

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

### Firmware

Releases



[v1.18 \(2022-01-17\) .bin \[elf\] \[.map\] \[Release notes\]](#) (latest)  
[v1.17 \(2021-09-02\) .bin \[elf\] \[.map\] \[Release notes\]](#)  
[v1.16 \(2021-06-23\) .bin \[elf\] \[.map\] \[Release notes\]](#)  
[v1.15 \(2021-04-18\) .bin \[elf\] \[.map\] \[Release notes\]](#)  
[v1.14 \(2021-02-02\) .bin \[elf\] \[.map\] \[Release notes\]](#)  
[v1.13 \(2020-09-02\) .bin \[elf\] \[.map\] \[Release notes\]](#)  
[v1.12 \(2019-12-20\) .bin \[elf\] \[.map\] \[Release notes\]](#)

Nightly builds

[v1.18-221-g94a9b5066 \(2022-03-20\) .bin \[elf\] \[.map\]](#)  
[v1.18-218-g63f0e700f \(2022-03-17\) .bin \[elf\] \[.map\]](#)  
[v1.18-215-g962ad8622 \(2022-03-16\) .bin \[elf\] \[.map\]](#)  
[v1.18-212-geec07332b \(2022-03-15\) .bin \[elf\] \[.map\]](#)

A binary file will be saved on your downloads folder

Step4: Flash the firmware to the ESP32

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

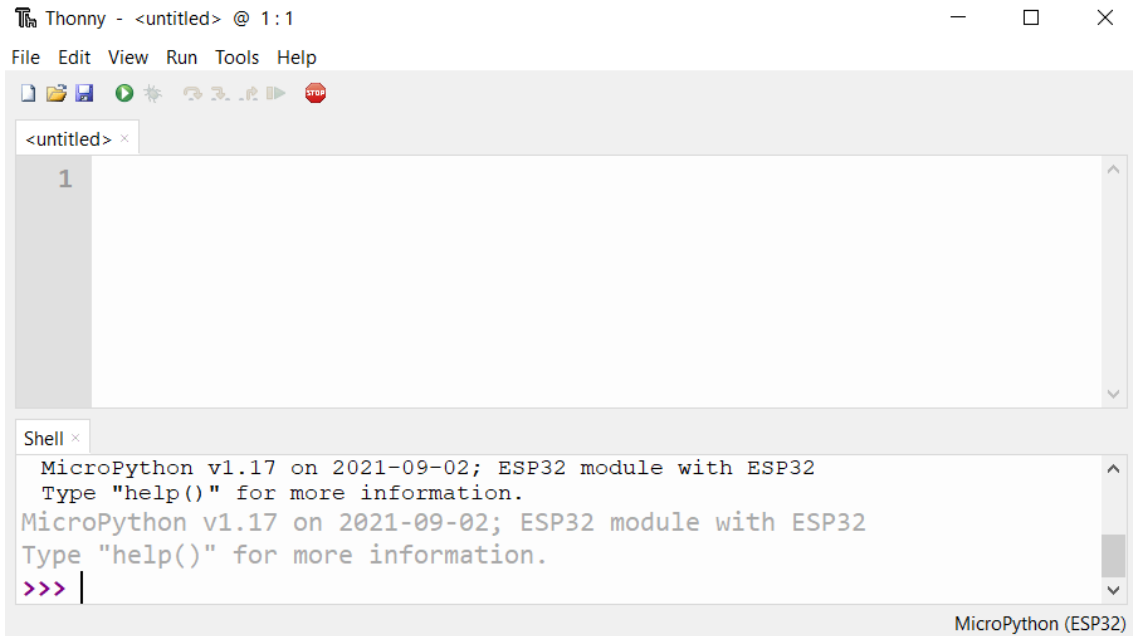
Good guide for installing micropython on ESP32: <https://microcontrollerslab.com/getting-started-thonny-micropython-ide-esp32-esp8266/>

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

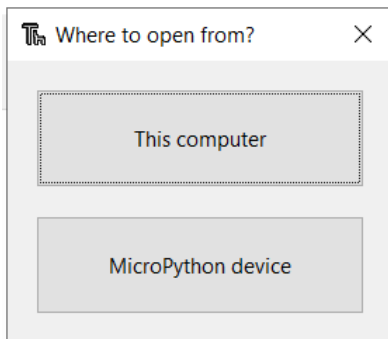
Type `esp.flash_size()` to check the flash memory, that should be around: 4194304

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

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



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



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

- main.py
- Cubotino\_moves.py
- Cubotino\_servos.py
- Cubotino\_settings.txt (select *all files (\*.\*)* option to be able to see the .txt file)

Save the above files, by using the *Save as* command, and by selecting the *MicroPython device*. Note that files names should not be changed, as these are the reference for module import.

## 16. Files needed at PC

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

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

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

Library	Main scope	Notes
pyserial	To manage serial communication	pyserial==3.5
cv2	For the vision part associated to the webcam usage	cv2.__version__ '4.5.3'
solver	Kociemba solver for the (almost optimum) cube solution	<a href="https://github.com/hkociemba/RubiksCube-TwophaseSolver">https://github.com/hkociemba/RubiksCube-TwophaseSolver</a> Note: The first time the Kociemba solver is called, it builds many tables. This takes quite some time on a PC

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

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

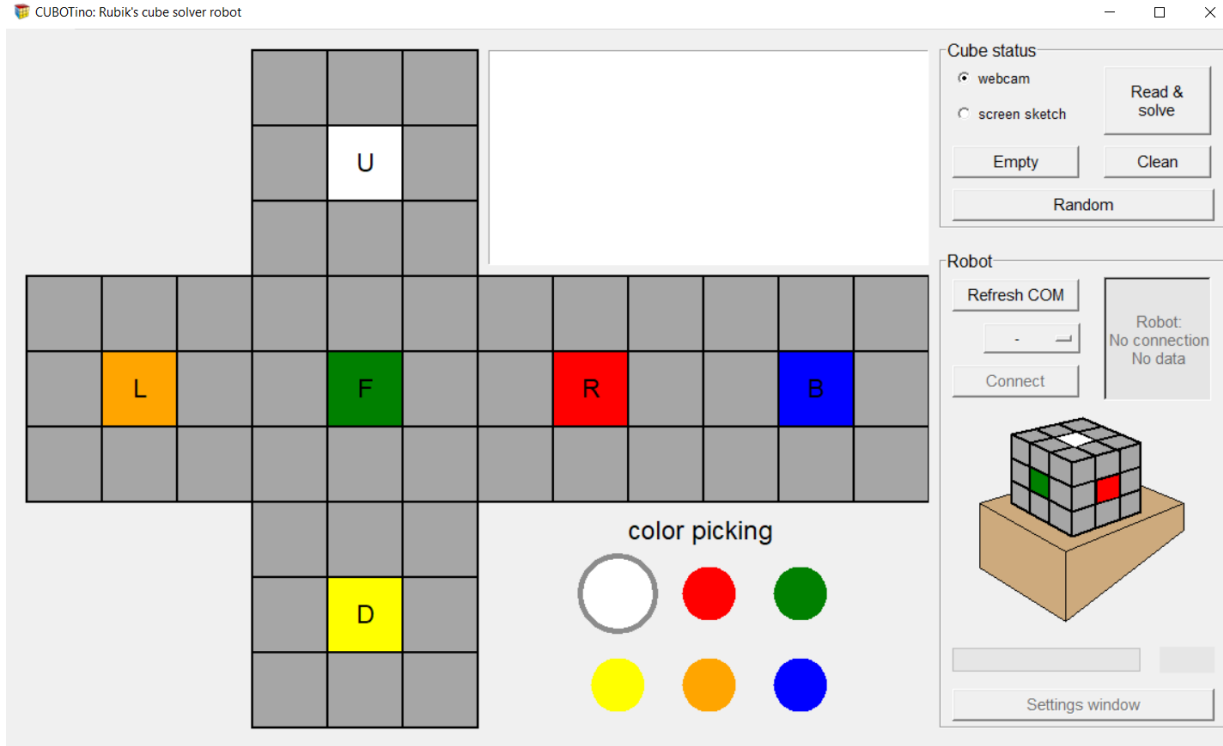
The project has been developed / tested with:

- Windows (W10) with Python version: 3.8.12 [MSC v.1916 64 bit (AMD64)] and cv2 ver: 4.5.3
- Windows (W10) with Python version: 3.9.7 | packaged by conda-forge | [MSC v.1916 64 bit (AMD64)] and cv2 ver: 4.5.3
- Thonny 3.3.13

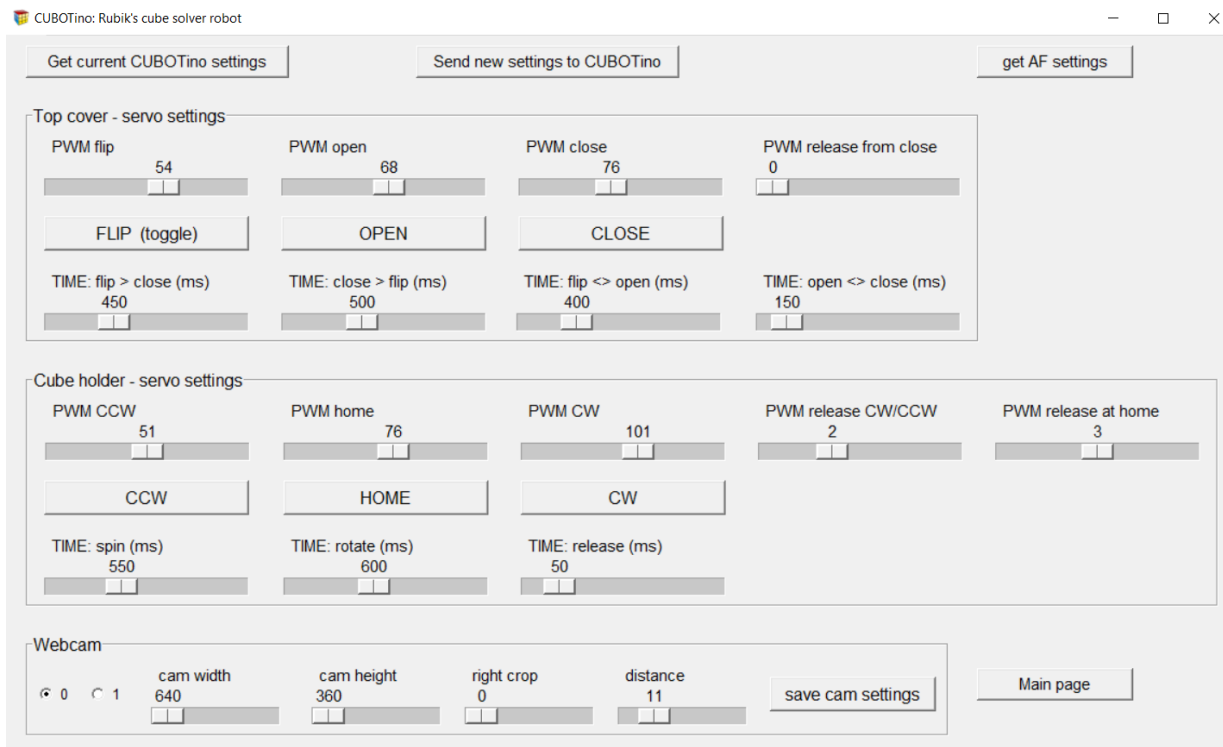
# 17. GUI

The GUI has two windows, "Main page" and "Settings windows"

At GUI launch (Cubotino\_GUI.py), the Main window appears, as per below:



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



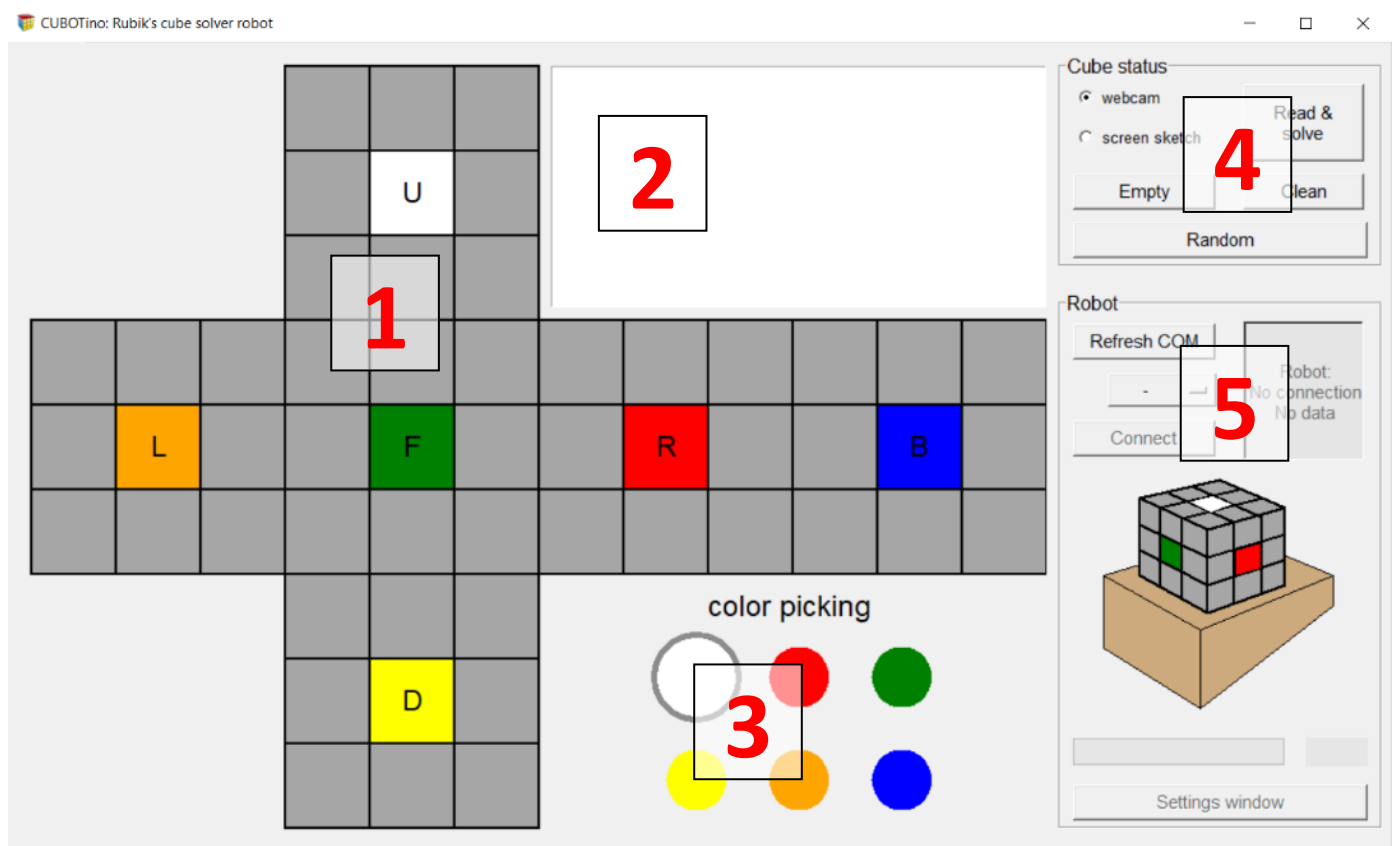
Each window is somehow divided in areas:

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

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

Notes:

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

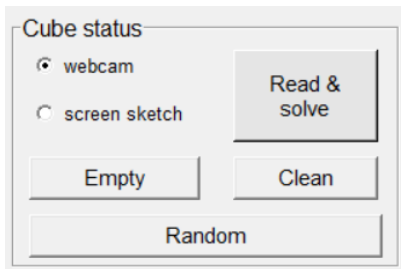


Area 1 and 3:

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

The Area2 is just informative

Area 4, Cube status interface:

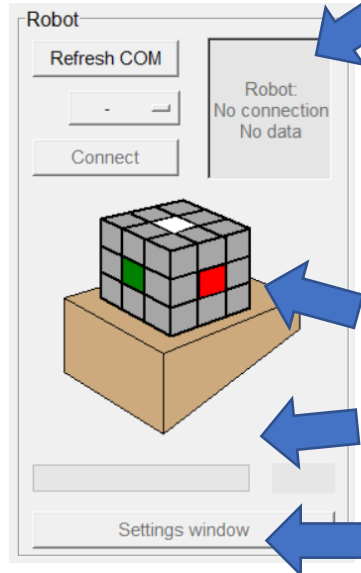


It's possible to select the cube status source, webcam or screen sketch. The *Read & solve* button sends to the Kociemba solver the cube status depicted on the sketch; in case the cube status is coherent, on the Text window will be printed:

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

The *Random* button generates random cube status on the screen sketch; By *Read & solve* them we have countless amount of cube solution to test/tune the robot movements

Area 5, Robot interface:



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

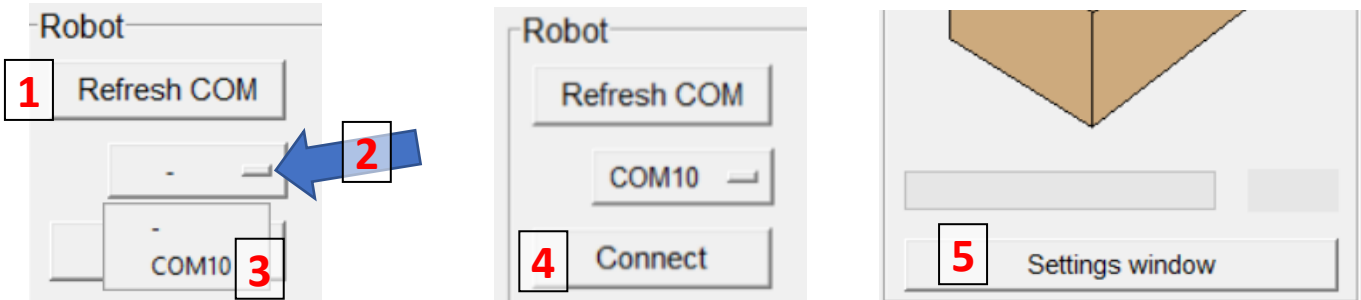
- No connection, No data (No data means there isn't a robot solution to send)
- Connected, No data
- Send data to robot
- STOP ROBOT

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

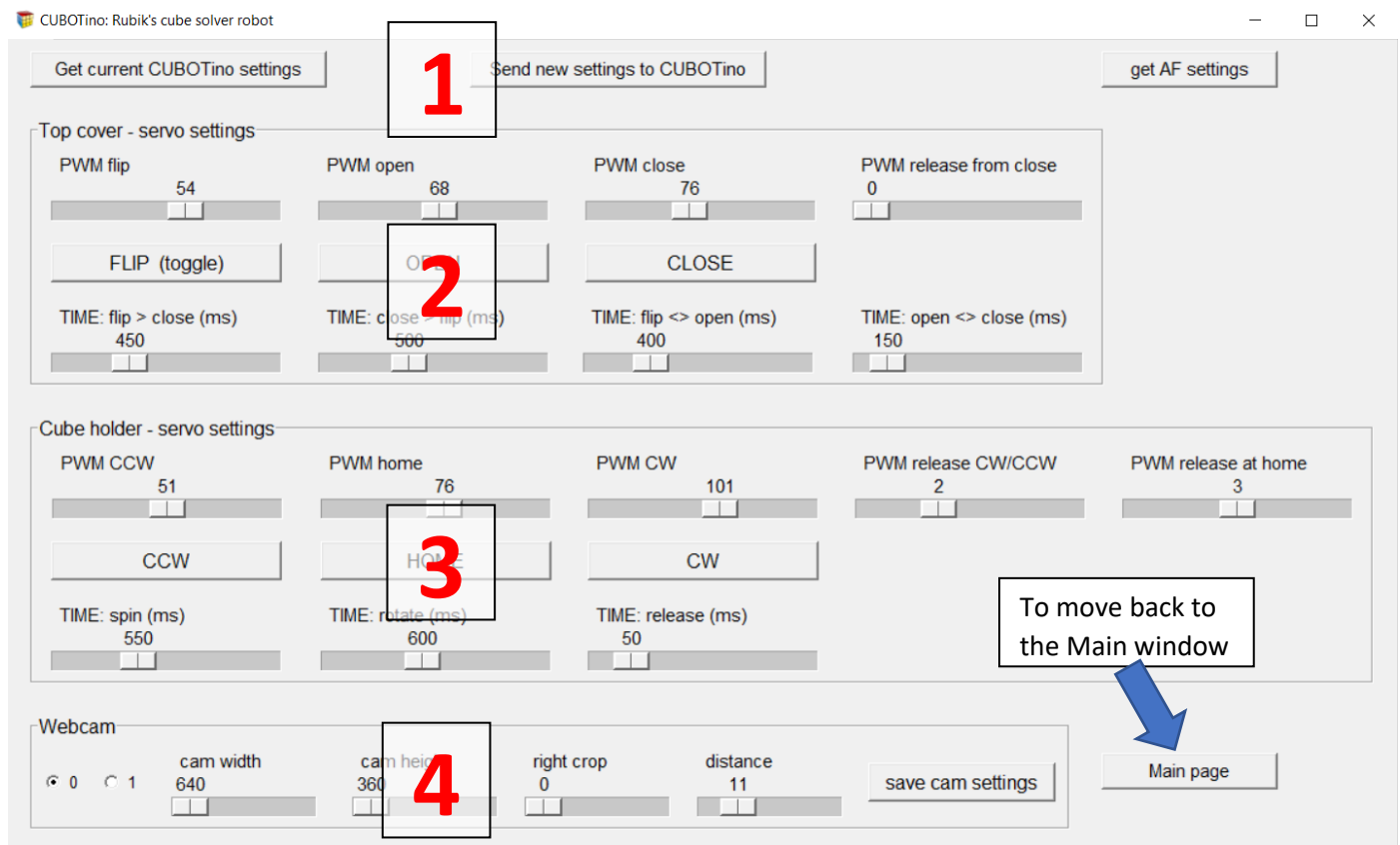
The *Settings window* button activates the GUI dedicated window for the robot and webcam settings

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

Area	Name	Functions
1	Receive and sending 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	Top cover servo	- Settings for the Top cover (and lifter) - Buttons allow to test the specific positions, and some of the associated timers
3	Cube holder servo	- Settings for the Cube holder - Buttons allow to test the specific positions, and some of the associated timers
4	Webcam	- Settings for the webcam

Notes:

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



Area1, Receive and sending robot setting:

*Get current CUBOTino settings* button, to receive the servo settings stored in the ESP32 microcontroller.

*Send new settings to CUBOTino* button, to send the sliders servo settings to the ESP32 microcontroller.

*Get AF settings* button, to receive the servo settings of my CUBOTino, just as reference (hard coded settings).

Area2, Top cover – servo settings:

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

Notes:

1. For the PWM smaller values means the Top cover rotating 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

Area3, Cube holder servo-settings:

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

Area4, Webcam settings:

Name	Type	Function
0 1	Radio-button	Set the webcam number 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:

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

## 19. Data logged

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

Column name	Info
Date	Date and time, i.e. 20220308_213439
CubeStatusEnteringMethod	screen sketch or webcam
CubeStatus	i.e. RLFDUUDDBBLFRURRBDDRDDFFLURULDRFDLUFDLBRLRFLBUBFUBBLBU
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. R1S3R1S3S3F1R1S3R1F1R1S3R1S3F3S1R3S3F1R1S3R1S3F3R1S3F1R1S3R1F3R1S3R1S3S1F3R3S3F1R1S3R1F3R1S3F3R1S3S1F1R3F1R1S3S3F3R1S3R1F2S1R3S1F3R3S3F3R1S3R1F3R1S3R1S3
TotCubotinoMoves	i.e. 97
EndingReason	Solved or stopped
RobotTime(s)	i.e. 77.2 (this is reported also when the robot gets stopped)

Notes:

1. The folder *data\_log\_folder* is made from the folder where *Cubotino\_GUI.py* is running.
2. The logged data is saved in the *Cubotino\_log.txt* file.
3. Text file uses tab as separator.

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

## 20. Assembly steps:

Before assembling the robot:

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

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

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

### Tools necessary:

Allen keys 3mm, 2.5mm and 2mm



## 21. How to operate the robot:

1. 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).
2. 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
3. GUI → Refresh the COM ports on the GUI.
4. GUI → Select the used COM from the drop-down menu.
5. GUI → Connect the robot; If the communication works correctly, then the blue LED of ESP32 lights up.
6. 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.
7. 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.
8. 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 the 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.
9. Robot → The robot can be unplugged at any moment (no shut down procedure needed).
10. GUI → The GUI can be closed at any moment; The cube status on screen sketch is never saved.

## 22. Tuning:

### 1. Reference angles for servos:

The servos I bought, have 180-degrees of rotation, that is more than sufficient for the (lifter and Upper-cover) angles of this robot; I don't suggest buying 270-degrees servo as this will further affect the angle resolution.

The point is that the connection between the servo arms, and the servo's outlet gear, have many possible positions (believe there are 25 teeth).

This means the reference angles set on Cubotino\_settings.txt file, are working fine on my robot, are not necessarily the best choice on other systems.

These parameters must be tuned on your system, and the Settings windows GUI is the convenient

### 2. Setting servos positions:

Servos are controlled on angle, via a PWM signal

The PWM resolution with Micropython V1.17 is rather poor at 50Hz, the frequency used by servos; The possible range is in unit from ca 50 to 100, meaning a 3.6° resolution for a 180° servo.

Despite the low resolution, this robot is quite forgiven, but the positions (angle=PWM) must be properly chosen.

With the servos I bought (link on the material list), and Micropython V1.17, the servos move for PWM values from 51 to 101 included (these are the values for the cube Holder CCW and CW, on my CUBOTino); With this range the servos make slightly more than 180°, that is simply perfect for this application.

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

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

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

### 3. Timers for servos:

The servos don't provide feedback when they have completed the requested angular rotation; For this reason it's necessary to set appropriate waiting time to allows the servo to complete the action.

It will be convenient to use larger delays at the beginning, and progressively reduce them.

## 23. Troubleshooting

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

1. Servos don't move smoothly
  1. Don't use jumper wires, or use quality jumper wires
  2. Don't use bread boards, make the connection board instead.
  3. Add the capacitors, to prevent voltage drops when servos are activated.
  4. Use a USB3, that delivers higher power.
  5. Use a 20 to 25Kg/cm servo.
2. Connection from the GUI to the robot fails:
  1. Ensure the ESP32 microcontroller isn't still connected to Thonny or an IDE.
3. Connection from the IDE to the robot fails:
  1. Ensure the ESP32 microcontroller isn't still connected to the GUI.
4. Bottom cube layer doesn't align nicely:
  1. 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)
  2. Verify if the cube Holder makes an extra rotation, before stopping home; If this doesn't happen, adapt the PWM release home value.
5. The Top\_cover isn't intended to keep pushing the cube when it's in the close position; In case the cube layers don't align nicely, by playing with the cube\_Holder settings, it's possible to use the Top\_cover to level the cube. By lower the Top\_cover close position to have a little interference with the cube, will improve the cube layer alignment in particular after flipping the cube. In this case it will be convenient to set one or few units on *PWM release from close setting*. Via this setting is possible to release the tension between the Top\_cover and the cube, after pressing it, to allow the cube\_Holder to rotate with less effort.
6. Webcam app reads twice the same cube face: After one face has been detected, there are 3 seconds time in which the facelets are ignored, to give the time to move the cube to another face. A solution is to move the cube apart as soon as a face has been detected, and presented it again to the webcam once rotated to the next face.
7. Webcam app is slow: Speed is largely affected by the number of pixels under analysis. On the *Settings window* at GUI, it's possible to reduce the frame size (width, height, crop) to gain speed.
8. Webcam app ends with incoherent cube status statement: There are few reasons:
  1. 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.

2. 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.

9. Cube's facelet and light reflection (webcam cube status detection):

Detection of edges, as well as colours, can be largely affected by light reflection made by the facelets.

I have two cubes available, one with in-moulded coloured facelets, and the other with glossy stickers.

On the cube with plastic facelet, I made the surface matt by using a fine grit sandpaper (grit 1000); This makes the cube status detection much more insensitive 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)



## 24. Robot solver algorithm:

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

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

The Kociemba solver provides a string with the rotations to be applied on the 6 faces, like `U2 F1 R3` etc (I will refer to these three moves as example on the below explanation).

The precondition for the cube solution is that the cube orientation doesn't change, meaning the U (upper) side remains up oriented and the F (front) side remains front oriented during the solving process; This pre-condition is clearly not fulfilled by the robot.

The robot solver follows instead the below approach:

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

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

1. The vision system is used to detect the cube's facelets edges (contours), more or less as explained at <https://medium.com/swlh/how-i-made-a-rubiks-cube-color-extractor-in-c-551cceb80f0>
2. Average BGR is calculated for the areas defined by the contours, for the 54 facelets, and stored in a dictionary; Average HSV (of a small area at each contour's centers) is also calculated and stored on a second dictionary.

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

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

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

3. The average HSV, detected on the 6 centers, is used to determine which colour is located on the 6 centers:
  - a. White facelet is the one having the largest V-S delta (difference between Value, or Brightness, and Saturation), while the yellow one is located at opposite face.
  - b. Remaining 4 centers are evaluated according to their Hue, and the Hue at opposite face.
  - c. Orange has very low Hue, and red should be very high (almost 180); Depending on light condition, the red's Hue could be very low (few units) and lower than the orange, in that case both Orange and red are very low, yet orange is always higher than red.
  - d. Out of the two remaining centers, blue is the one with highest Hue, and consequently the green is also known.
4. Based on previous step, the 6 cube colours (at least their centers) have a known average HSV and therefore an average BGR colour; This also informs on the cube orientation (colours) as placed on the cube-holder.
5. Facelets colour interpretation is made, by using two methods, with 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.
  - b. In the second method the Hue value of each coloured (non-white) facelet are compared to the Hue of the 5 reference centers; White facelets are retrieved according to 3 parameters (Hue, Saturation, Value), in comparison to the white center HSV.

First method is in general better than the second one, yet the second one "wins" when there is lot of light; The second method is only used (called) when the first one fails.

As result both methods are used, to get reliable cube status detection under different light situations.

## 26. Credits:

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

## 27. Assembly details:

**Step4** (Mount the bottom servo to the structure):

4x M3x12mm cylindrical head

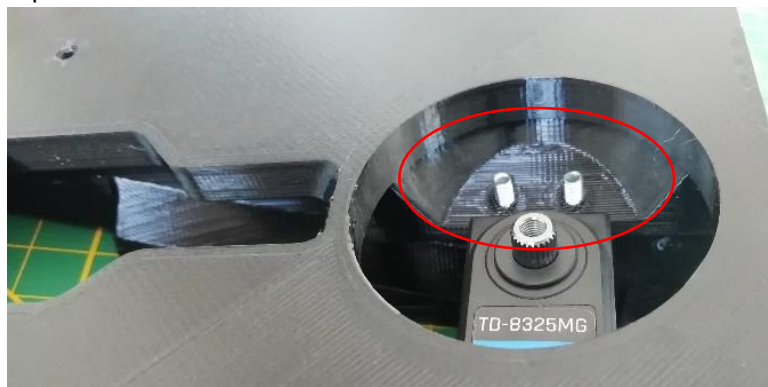
Couple of washers

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



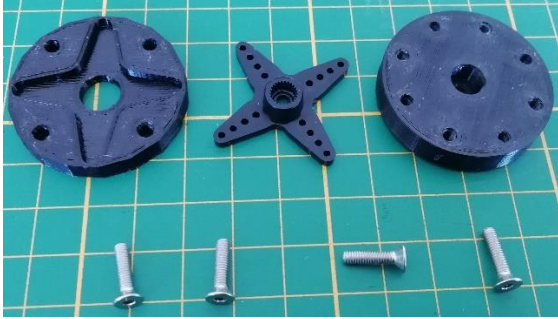
Couple of notes:

- 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

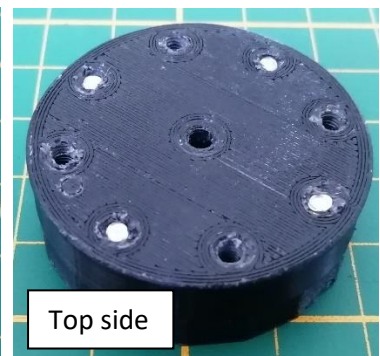
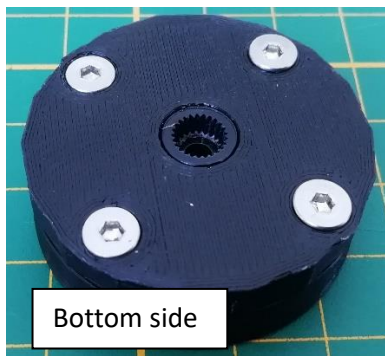


Check if the supplied X arm fits with the indentation of Servo\_axis\_inf part:



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



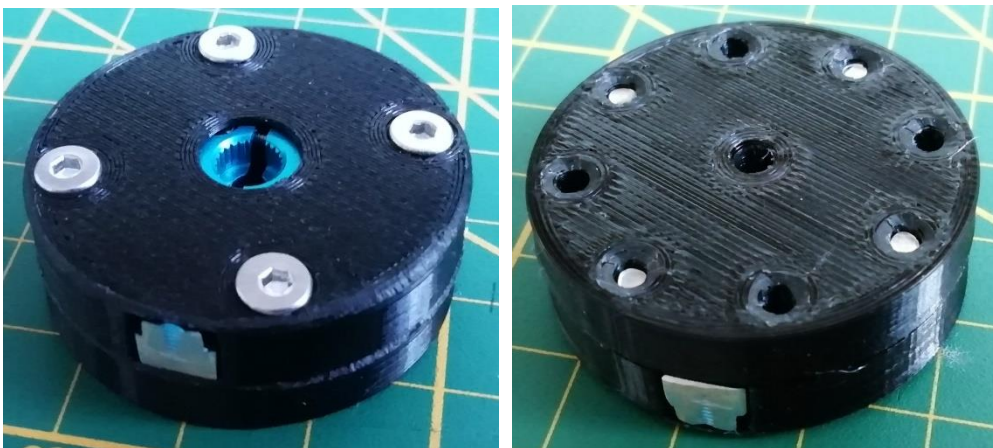
**Step5a** Alternative servo axis assembly:



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):

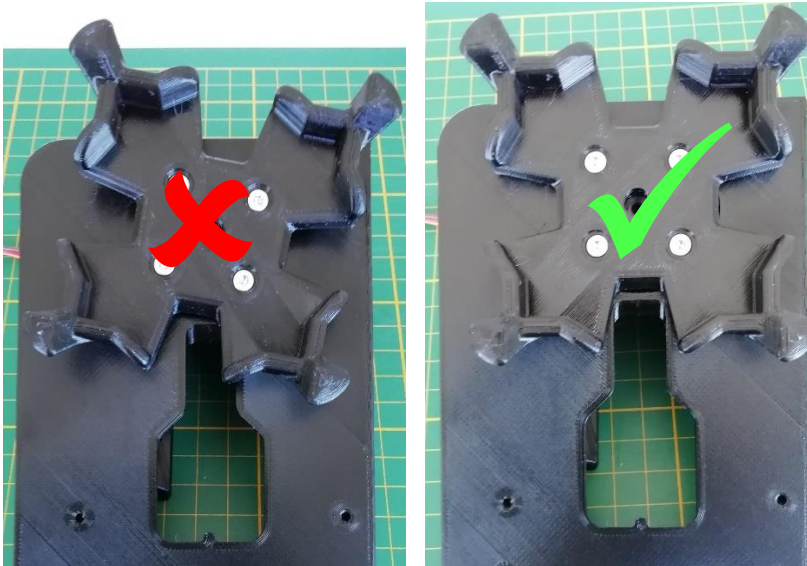
4x M3x12mm conical head



**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

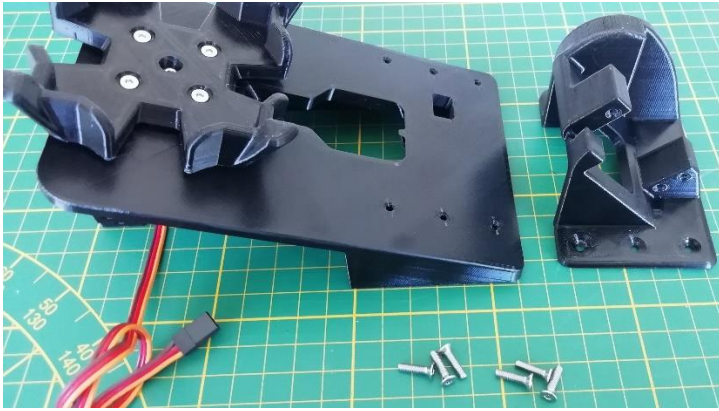


1x M3x12mm cylindrical head

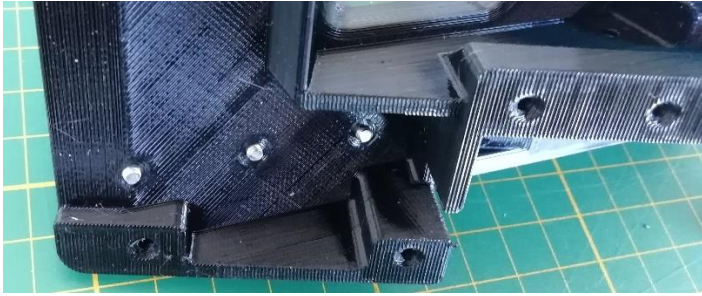


**Step8** (Assemble the Hinge to the Structure):

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).

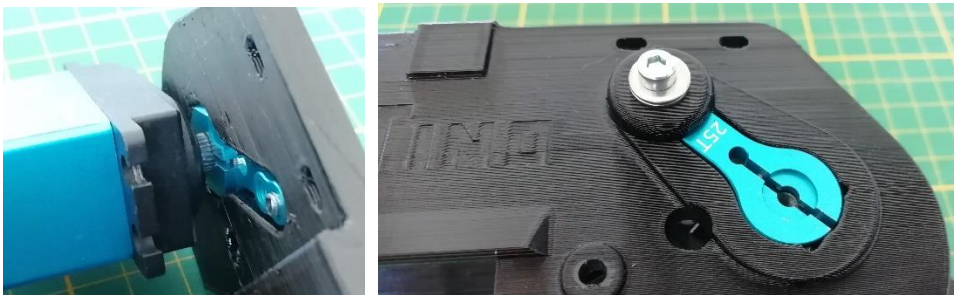
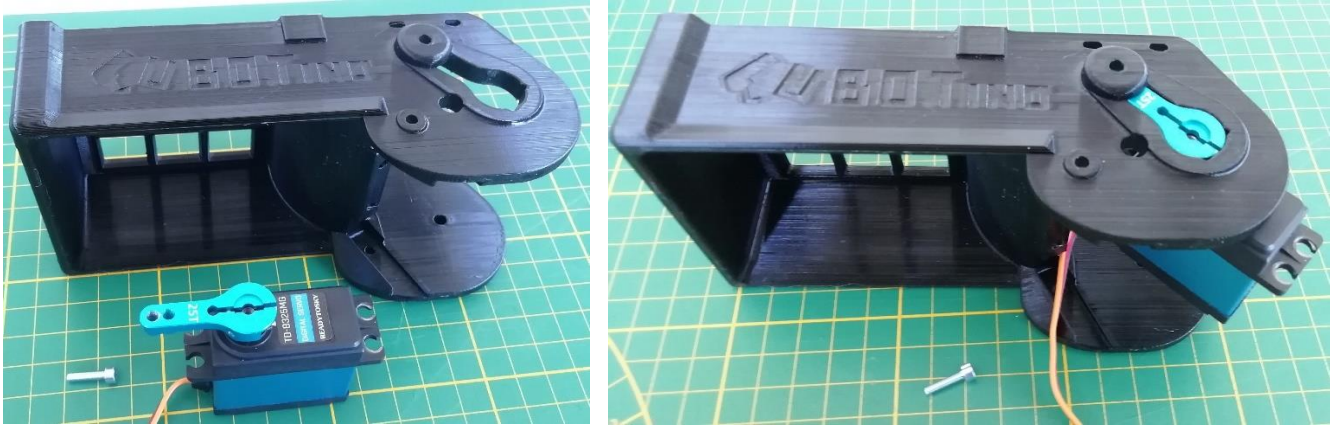
Close the two arm tiny screws



**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



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



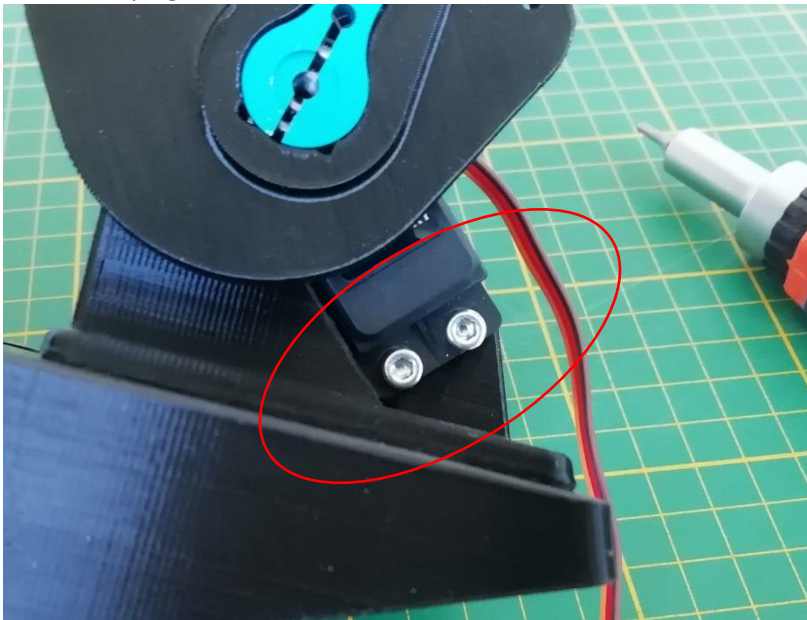
**Step11** (Assemble the Top\_cover assembly to the Hinge):

1x M4x20mm cylindrical head

3x M3x12mm cylindrical head

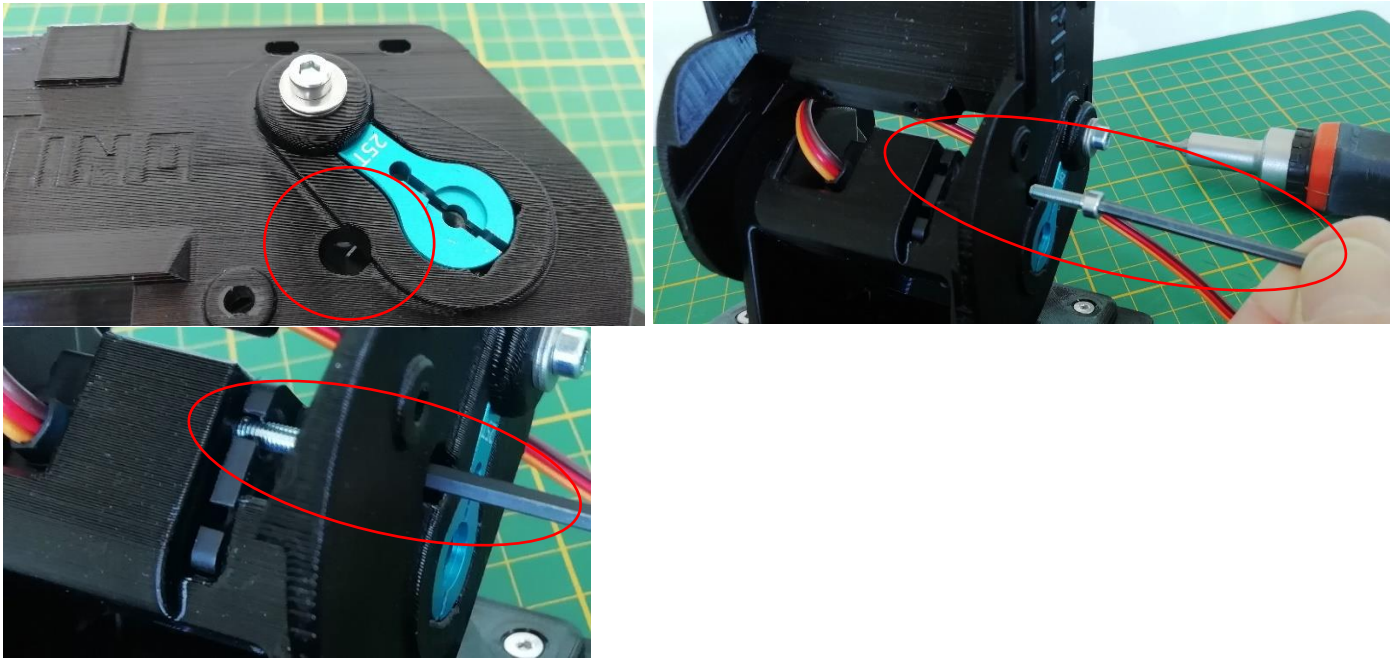


Do not fully tighten the two M3x12mm, until also the third screw is positioned

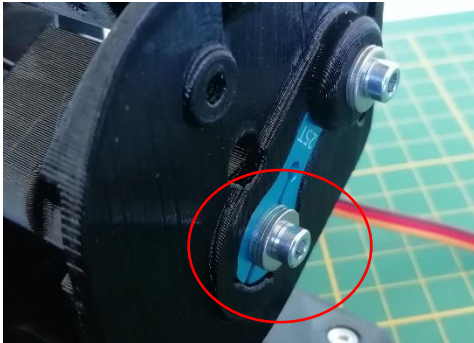


**Step12** (Complete the top servo assembly to the Hinge):

Rotate the Top\_cover to have the hole facing the third screw accessible



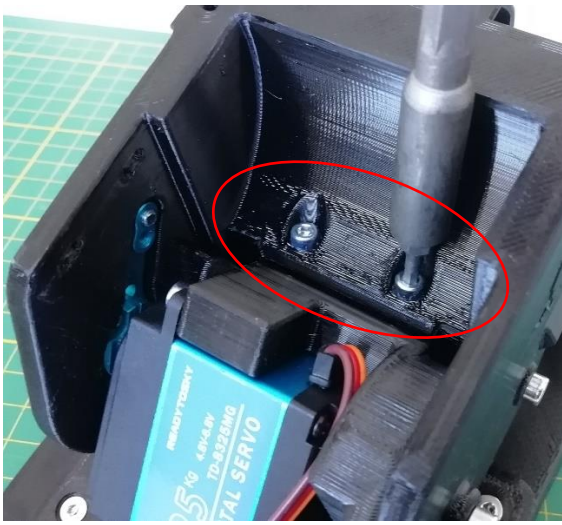
1x M3x12mm cylindrical head (add washers if the screws doesn't push on the "T25"Arm)



**Step13** (Assemble the Lifter to the Top\_cover):

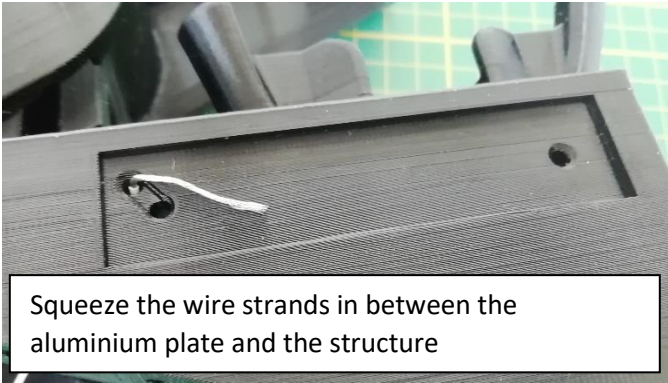
4x M3x12mm cylindrical head

Slide the Lifter into the Top\_cover slots

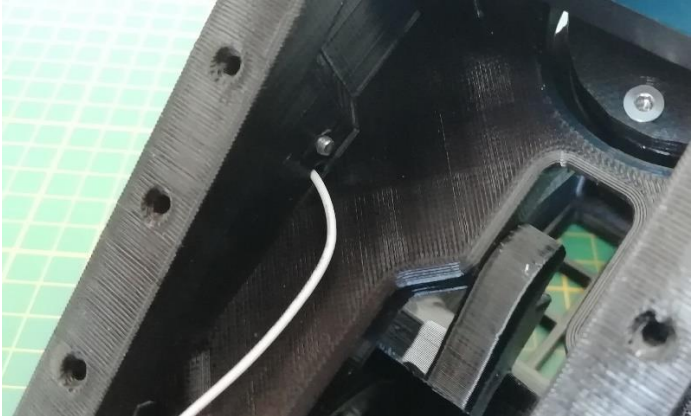


**Step14** (Assemble the "STOP" touch plate, and related cable):

2x M3x12mm conical head

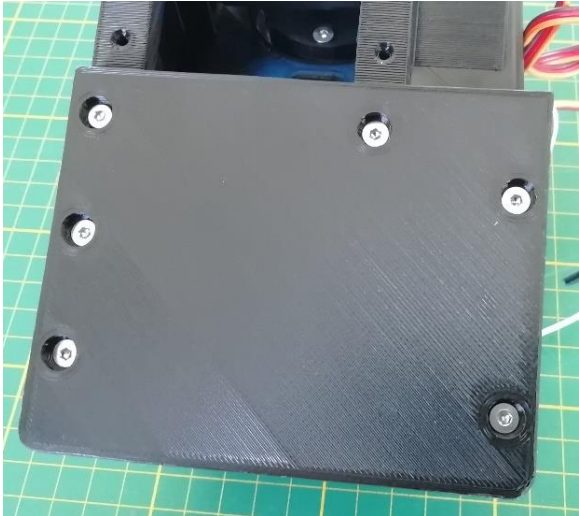
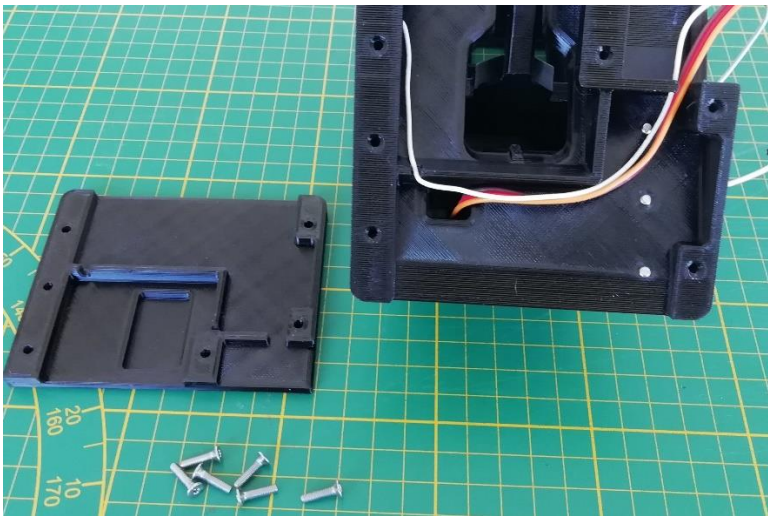
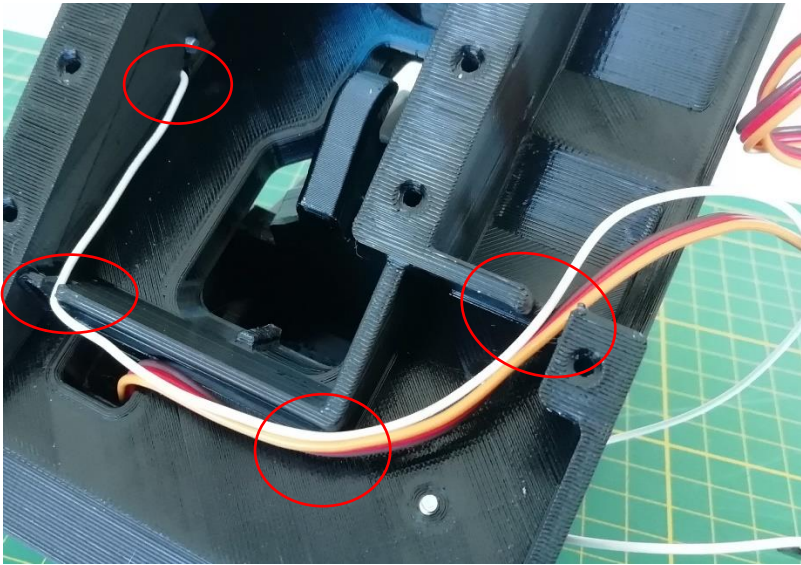
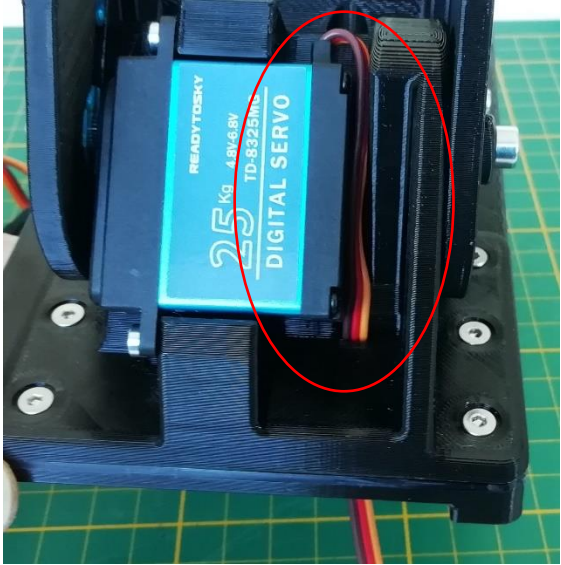


Squeeze the wire strands in between the aluminium plate and the structure



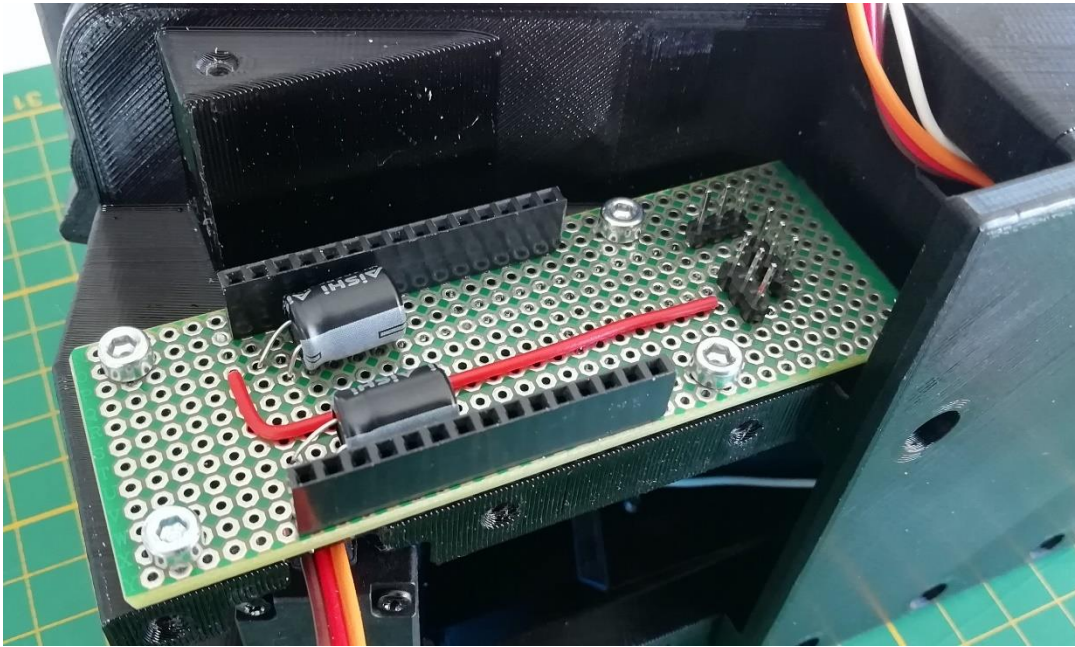
**Step15** (Position the cables, and assemble the Baseplate\_rear):  
6x M3x12mm conical head (4 screws at corners are sufficient)

Dress the cables and close the Baseplate\_rear



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

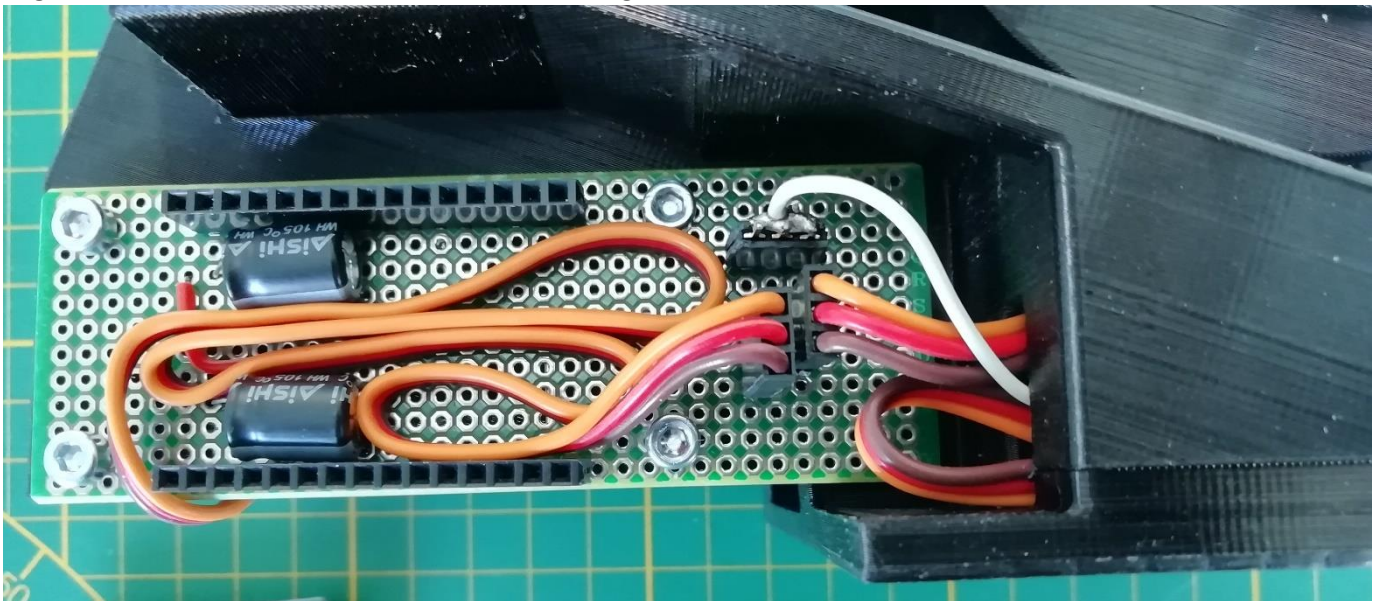
4x M2.5x10mm cylindrical head



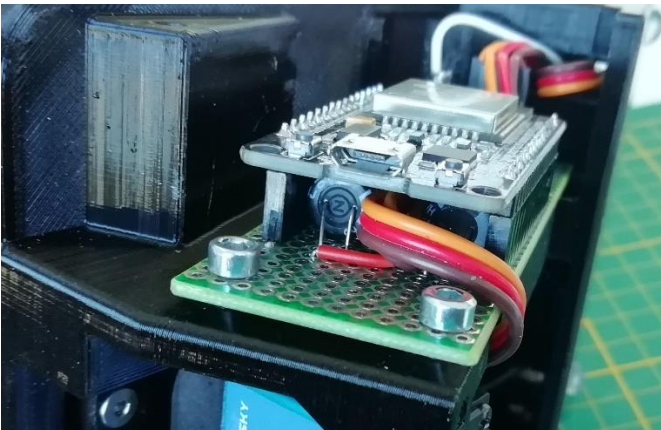
**Step17** (Connects servo and touch pad)

Dress the wires and connect

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

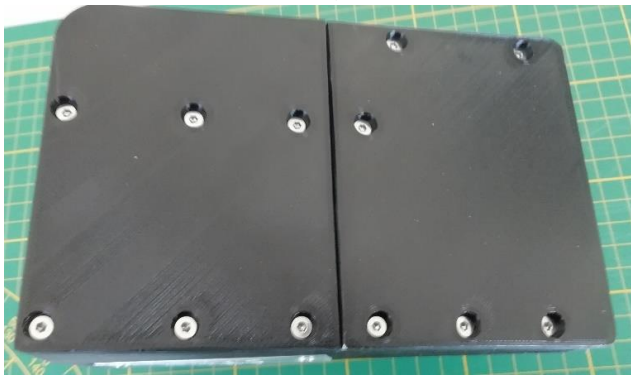


**Step18** (Insert the ESP32 dev board):



**Step19** (Assemble the Baseplate\_front to the Structure):

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



**Step20** (Stick self-adhesive rubber feet to the baseplates):

4x Self adhesive rubber feet

Rubber feet at the back should be placed as close as possible to the corners



**Step21** (Assemble the PCB cover):

2x M3x12mm conical head

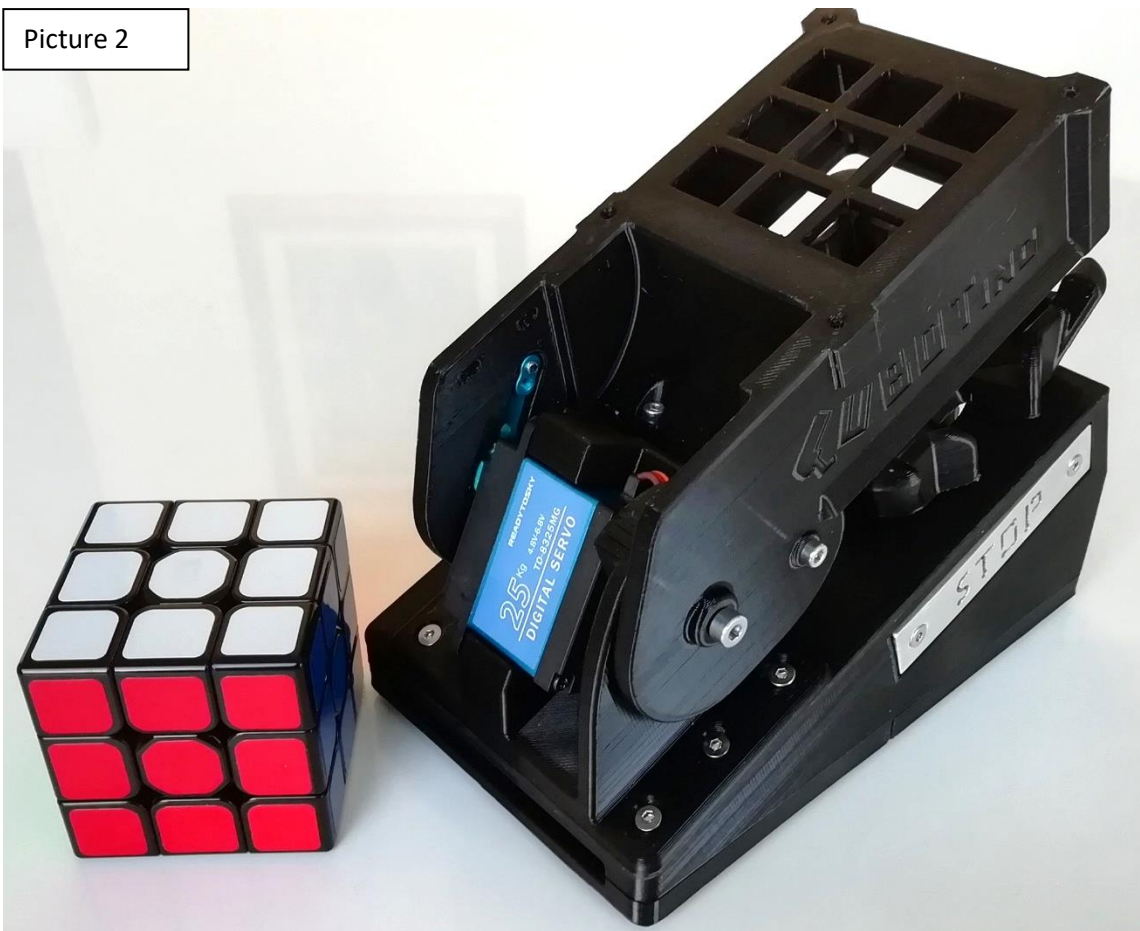


28. Collection of robot's pictures:

Picture 1



Picture 2



Picture 3

