

How to make

CUBOTino autonomous: A small, 3D printed, Rubik's cube solver robot

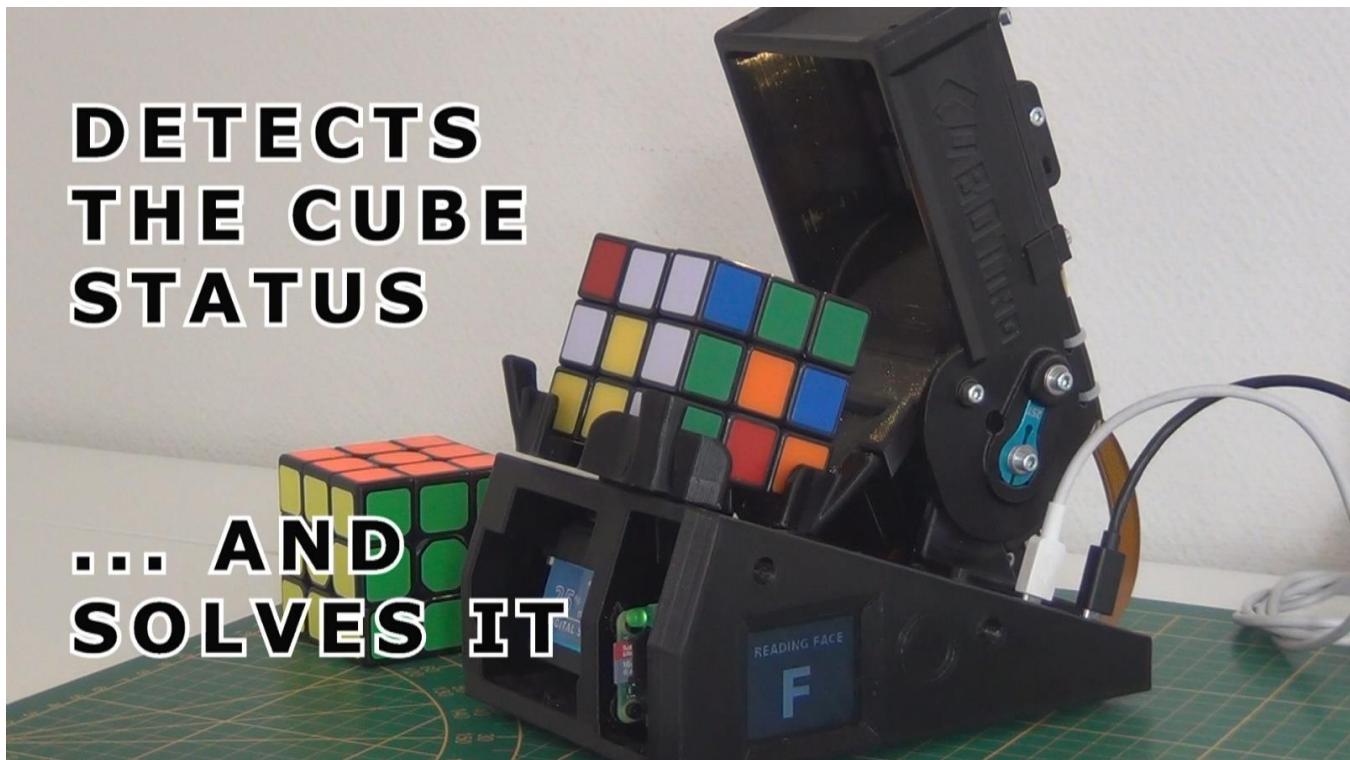
<https://www.instructables.com/CUBOTino-Autonomous-Small-3D-Printed-Rubiks-Cube-R/>

<https://github.com/AndreaFavero71/cubotino>

Andrea Favero, Groningen (NL)

01/04/2024 Rev. 7.2 (always check if a newer version is available)

Robot demonstration at YouTube: <https://youtu.be/dEOLhvVSBCg>



This is an autonomous Rubik's cube solver robot, the "Top version" of CUBOTino series.

(Cubotino Base version can be seen at: <https://youtu.be/ZVbVmCKwYnQ>)

It uses the same mechanical solutions of the Base version: It is rather simple, very small (the base is about 160 x 110mm) and it does not require any special gripping for the cube.

As average it takes 90 seconds for scanning and solving a scrambled cube; this is for sure not a fast robot, yet it is rather simple, small, and fully autonomous.

This is one of the smallest autonomous robots, for normal size and unchanged Rubik's cube; From February 2023 there is a smaller robot version, CUBOTino micro, for Rubik's cube sizing 30mm:

<https://youtu.be/EbOHhvg2tJE>.

1. Instructions: Order and organization

This document is organized in about 40 chapters, divided into 5 main sections:

- Sections 1 to 4 to make the robot.
- Section 5 providing useful (or interesting) info.

Sections:

1. Supplies:

2. Connections_board & Raspberry Pi setup:

- a. Make the Connections_board.
- b. Setup the Raspberry Pi.
- c. Test the servos rotation range.
- d. Set the servos to mid position.

3. 3D print and assembly:

- a. Print the parts.
- b. Assemble the robot.
- c. In case of a Raspberry Pi 2, 3 or 4.

4. Tuning and robot operation:

- a. Robot tuning.
- b. Troubleshooting.
- c. How to operate the robot.
- d. Automatic start and Rpi shut-off by the robot.

5. Info (my preferred part 😊, yet not strictly needed to build the robot):

- a. Project background.
 - b. High level information.
 - c. Robot solver algorithm.
 - d. Computer vision.
 - e. Colour detection strategy.
-and much more.

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; Raspberry Pi power supply and phone charger normally have this safety feature: Check it for your own safety.
Despite the robot mechanical force is limited, it must be operated only under adult supervision.
If you build and use a robot, based on this information, you are accepting it is on your own risk.

3. Manage expectations

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

- Servos.
- servo arm positioning to the servo.
- cube dimensions.
- print quality.
- Assembly.

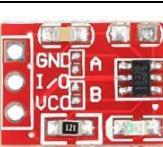
But hey, don't worry Other makers have successfully tuned their own Cubotino, and you will too 😊

... and now, let's start!

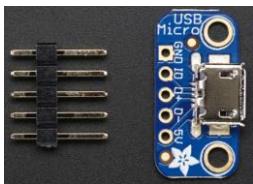
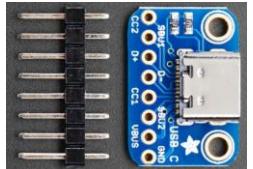
Summary

1.	Instructions: Order and organization	2
2.	Safety	3
3.	Manage expectations	3
4.	Supplies.....	5
5.	Raspberry Pi Zero 2 GPIO pinout	11
6.	Connections board.....	12
7.	Setting up Raspberry Pi.....	22
8.	Files copied to Raspberry Pi.....	41
9.	Servos test and set to mid position	43
10.	3D printed parts.....	44
11.	Assembly steps	48
12.	Assembly details	49
13.	Raspberry Pi 2, 3 or 4.....	67
14.	Tuning	73
15.	Tuning via GUI.....	85
16.	Tuning via CLI.....	93
17.	Parameters and settings.....	97
18.	Troubleshooting.....	102
19.	How to operate the robot	115
20.	Automatic robot start.....	120
21.	Rpi shut down via Touch button.....	120
22.	Introduction.....	121
23.	Project scope	121
24.	Robot name	122
25.	Models	122
26.	High level info (Top version).....	124
27.	Construction	125
28.	Computer vision part.....	126
29.	Robot solver algorithm	136
30.	Python main scripts, high level info.....	138
31.	Virtual solvers	143
32.	Set Thonny IDE interpreter.....	144
33.	A convenient cube	150
34.	Collection of robot's pictures	152
35.	Conclusions.....	160
36.	Next steps	160
37.	Commitment.....	161
38.	Credits.....	161
39.	Myself	162
40.	ChatGPT	163
41.	Revisions	164

4. Supplies

Q.ty	Part	link to the shop I used	Cost (€)	Notes
2	Servos I used in first place: TD-8325MG (180deg 25Kg metal, Pulse width 1to2ms) and metal arm "25T" Check the (better) alternative servos on next pages before ordering	https://www.aliexpress.com/item/32298149426.html?gateWayAdapt=glo2ita&spm=a2g0o.9042311.0.0.5d1e4c4d14Qjaz	25 (2 servos + 2 arms)	180 Degree Servo 2PCS + 25T Arm 2PCS (Control by Remote Control)  max 31mm protrusion under the flange
1	Raspberry Pi Zero2W (H needed, yet the header can be bought at side) Raspberry Pi ZeroW works too, see performances notes	https://www.sossolutions.nl/	19 (2W version, January '24)	
1	microSDHC 16GB	https://www.dataio.nl/sandisk-ultra-micro-sdhc-16gb-uhs-i-a1-met-adapter/	8	
1	PiCamera V1.3 (PiCamera V2 can be used, by changing the <code>s_mode</code> parameter)	https://www.amazon.nl/gp/product/B01M6UCEM5/ref=ppx_yo_dt_b_asin_title_o05_s00?ie=UTF8&th=1	7.5	
1	30cm cable Raspberry Pi Zero/Camera	https://www.amazon.nl/gp/product/B079H33VCM/ref=ppx_yo_dt_b_asin_title_o05_s00?ie=UTF8&th=1	5	
1	SPI TFT 128x160 Pixels Display (1.77 inch) with ST7735 driver. Check alternative on next pages	https://www.amazon.nl/gp/product/B078JBBPXK/ref=ppx_yo_dt_b_asin_title_o03_s00?ie=UTF8&th=1	8	
1	Led module 3W	https://www.aliexpress.com/item/1005001984730729.html?spm=a2g0o.cart.0.0.58be3c00SgxRBT&mp=1	2.6 (4 pcs)	
1	2.5-5.5V TTP223 capacitive touch switch button self-locking module for Arduino	https://www.amazon.nl/en/gp/product/B07BVN4CNH/ref=ppx_yo_dt_b_asin_title_o01?ie=UTF8&psc=1	4 (5 pcs)	
1	AMS1117-3.3 DC 4.75V-12V to 3.3V Voltage Regulator	https://www.amazon.nl/gp/product/B07MY2NMQ6/ref=ppx_yo_dt_b_asin_title_o00_s00?ie=UTF8&psc=1	6 (5 pcs)	

Section1: Supplies

Q.ty	Part	link to the shop I used	Cost (€)	Notes
2	USB MICRO-B BREAKOUT BOARD Not suitable for Raspberry Pi 3 or 4	https://www.kiwi-electronics.nl/nl/usb-micro-b-breakout-board-3908	3.8	
1	Alternative solution: USB Type-C BREAKOUT BOARD	https://www.adafruit.com/product/4090	3.5	
~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 connect all the parts
1	40pin (2x20) GPIO male header	In case you could not get the WH version of Raspberry Pi
1	40pin (2x20) GPIO female header (Plastic body height ca 8 to 8.5mm)	To connect the Connections board to Raspberry Pi Zero 2
1x8	Female Header (plastic height ca 8.5mm)	To connect the display to the Connections board
4x3	Male Headers 90deg	To connect the servos, touch pads, Led module
2x3	Female Headers	To connect the touch pads, Led module
3	Capacitor 16V 220uF	To limit voltage-drop when servos are activated
1	Heat sink for Raspberry Pi	

Screws:

Quantity	Dimension	Head type	Info
1	M4x20	Cylindrical	Pivot for Top_cover
~20	M3x12	Cylindrical	
~30	M3x12	Conical	
4	M2.5x10	Cylindrical	Rpi to Structure (can be used L=6mm or L=x8mm)
8	M2.5x4	Cylindrical	Micro-usb break-out boards to PCB cover
1	M2.5x20 + 4 nuts	Cylindrical	Used on the Connections_board as spacer for the display

Power supply:

- If micro-USB: 2x 2A phone charger with micro-USB cable
- If USB type C: 1x 3A

Other:

Off course some other common materials are needed (wires, solder and solder device, tire wraps, self-adhesive rubber feet, hot glue, 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 in the "A convenient cube" chapter, later in these instructions.

Alternative servos

Option 1:

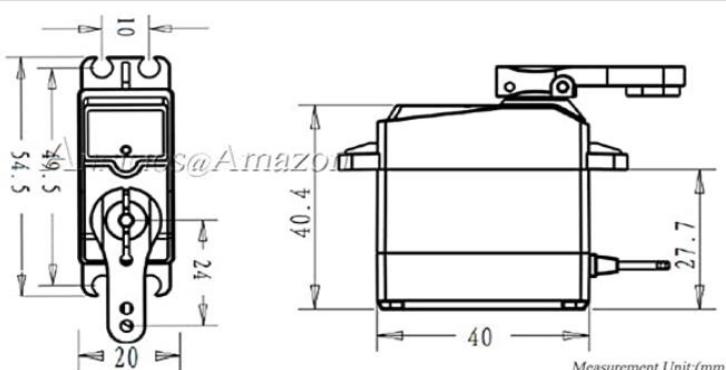
It is possible to use servos having 270deg rotation combined with an extended Pulse Width (from 500 μ s to 2500 μ s): When I started this project, I didn't know about the extended pulse width... a missed opportunity I would call it now! Rotation and Pulse Width ranges makes this servo a much better choice for this robot:

- Servo rotation range will always be sufficient, preventing from soldering resistors into the servo to enlarge the range.
- The resolution is higher, making easier the servo angle setting process.

This model isn't cheap yet is seems widely available.

As reference this servo has these characteristics:

Q.ty	Part	link to the shop I used	Cost (euro)	Notes
2	Servo DS3225MG (270deg 25Kg metal, Pulse width 500 μs to 2500μs) and metal arm "25T"	https://www.amazon.com/ANNIMOS-Digital-Waterproof-Crawler-Control/dp/B07GK1G5FV/	38 (2 servos + 2 arms)	



Operating Voltage	5V	6.8V
Idle current(at stopped)	4mA	5mA
Operating speed (at no load)	0.15 sec/60°	0.13sec/60°
Stall torque (at locked)	21 kg-cm	24.5 kg-cm
Stall current (at locked)	1.9A	2.3A
Control System	PWM(Pulse width modification)	
Pulse width range	500~2500 μ sec	
Neutral position	1500 μ sec	
Rotating direction	Counterclockwise (when 500~2500 μ sec)	

For the tests I've ordered the 180° version from Amazon Netherlands:

https://www.amazon.nl/gp/product/B01MU78A29/ref=ppx_yo_dt_b_asin_title_o02_s01?ie=UTF8&psc=1

On the received servo, the real rotation range is about 192°, therefore still ok for the robot.

I've measured a resolution <1° (vs 1.8 of the servos I've initially used, and suggested), making easier the angles settings.

When moving from a 180° to a 270° version, the angle resolution should still be around 1.4°: Again, better than the servo I initially used and suggested.

Option 2 (a cheaper servo model):

- The MG996R servo is enclosure in a plastic body, while having the reduction gears made by metal (MG might mean Metal Gear) as well as a metal bush at the outlet gear.
- Torque is ca 10Kg*cm, therefore much lower than the metal version I've initially used.
- It looks like this model is rather common for 180° rotation range, and rather rare for the 270°.
- The several online shops I've verified don't specify the PWM range.

I ordered them to Amazon Netherland https://www.amazon.nl/AZDelivery-Versnellingsbak-Helikoptervliegtuigen-compatibel-Inclusief/dp/B07H88DB8R/ref=asc_df_B07H88DB8R/?tag=nlshogostdde-21&linkCode=df0&hvadid=594732236372&hvpos=&hvnetw=g&hvrand=8695186734418167946&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=9103947&hvtargid=pla-653808722062&th=1

The servos I got have a Pulse Width ranging from 500µs to 2500 µs.

- Because of the lower rated torque, and less metal parts, these servos are cheaper, and apparently widely available; It is typically sold with a set of plastic arms.

Notes:

I've assembled two of these servos on my second robot.

22/10/2022: One of these servos (motor) broke after a few months of working; I don't know if it was because of excess of load or because production quality, yet I hadn't any rupture with more powerful servos so far.

Bottom servo:

- The servo connected to the Cube_holder has sufficient torque to also solve an old and rather hard to turn cube.
- The servo tuning has been rather easy to, because of the angle resolution and because the servo accepted pulse width min and max well behind the 500 and 2500 µs.
- Vibrations make the servo to swap back and forward of one resolution angle; This doesn't affect the cube status reading neither the cube solving process.

Top servo:

- The Top_cover stopping position is slightly less repetitive than those achieved with the more powerful servos. Anyhow, if the image cropping leaves some margin (upper and lower), it isn't of an issue for the robot to recognize the facelets and to perform all the other actions. The unfolded cube status pictures don't suffer from this position repeatability issue, as each face is cropped based on the detected facelets.
- Despite I've done about 100 successful cycles with this servo, by considering the rated torque and the torque dissipated on friction between the Top_cover and the Hinge, the usage of this servo might be challenged for the Top_cover.

As reference:

Q.ty	Part	link to the shop I used	Cost (euro)	Notes
1	Servo MG996R (180deg 10Kg metal, metal gears)	I assume these should be like those I've ordered in amazon.nl. https://www.amazon.com/Servo-Torque-Metal-Robot-Helicopter/dp/B08T7M8S42/ref=sr_1_22?crid=36VL0CM478RT0&keywords=mg996R&qid=1660395026&suffix=mg996r%2Caps%2C186&sr=8-22	8 (1 servo)	

Raspberry Pi ZeroW (instead of Zero2W)

By considering the severe chip shortage situation of 2022, Raspberry Pi Zero board is a valid alternative for this project:

Pro:

1. Availability, see notes below.
2. Hardware and Software compatibility.
3. Size.
4. Price.

Cons:

1. Performances:
 - a. Boot with script loading takes about 120secs, roughly double the time of Zero2W.
 - b. Cube status detection takes about 10 to 15 seconds more than Zero2W.
 - c. Solving time takes about 10% more than Zero2W.

OS 10 (Buster): As average the cube detection and solving is reached in 110 seconds vs 90 seconds of a Zero2W

OS 11 (Bullseyes): The ZeroW performance are further impacted. It works yet doubling the time of a Zero2W.

Purchasing a Raspberry pi Zero2W or ZeroW (Info updated o January 2024)

The Raspberry Zero2 and Zero have suffered for severe shortage in 2022; Starting from December 2022, Raspberry Pi ZeroW is back available on selected shops.

Starting from the summer 2023, also the Zero2 is coming back to market too.

From the Raspberry Pi official site (<https://www.raspberrypi.com/products/raspberry-pi-zero-w/> or <https://www.raspberrypi.com/products/raspberry-pi-zero-2-w/>) select *Buy now*, enter your Country and check one by one the proposed shops for availability.

Alternative display

In March 2024, Paul has reported difficulties to find the original screen:

SPI TFT 128x160 Pixels Display (1.77 inch) with ST7735 driver	https://www.amazon.nl/gp/product/B078JBBPXK/ref=ppx_yo_dt_b_asin_title_o03_s00?ie=UTF8&th=1
---	---

He has developed the complete solution (code, models, information) for below alternative displays:

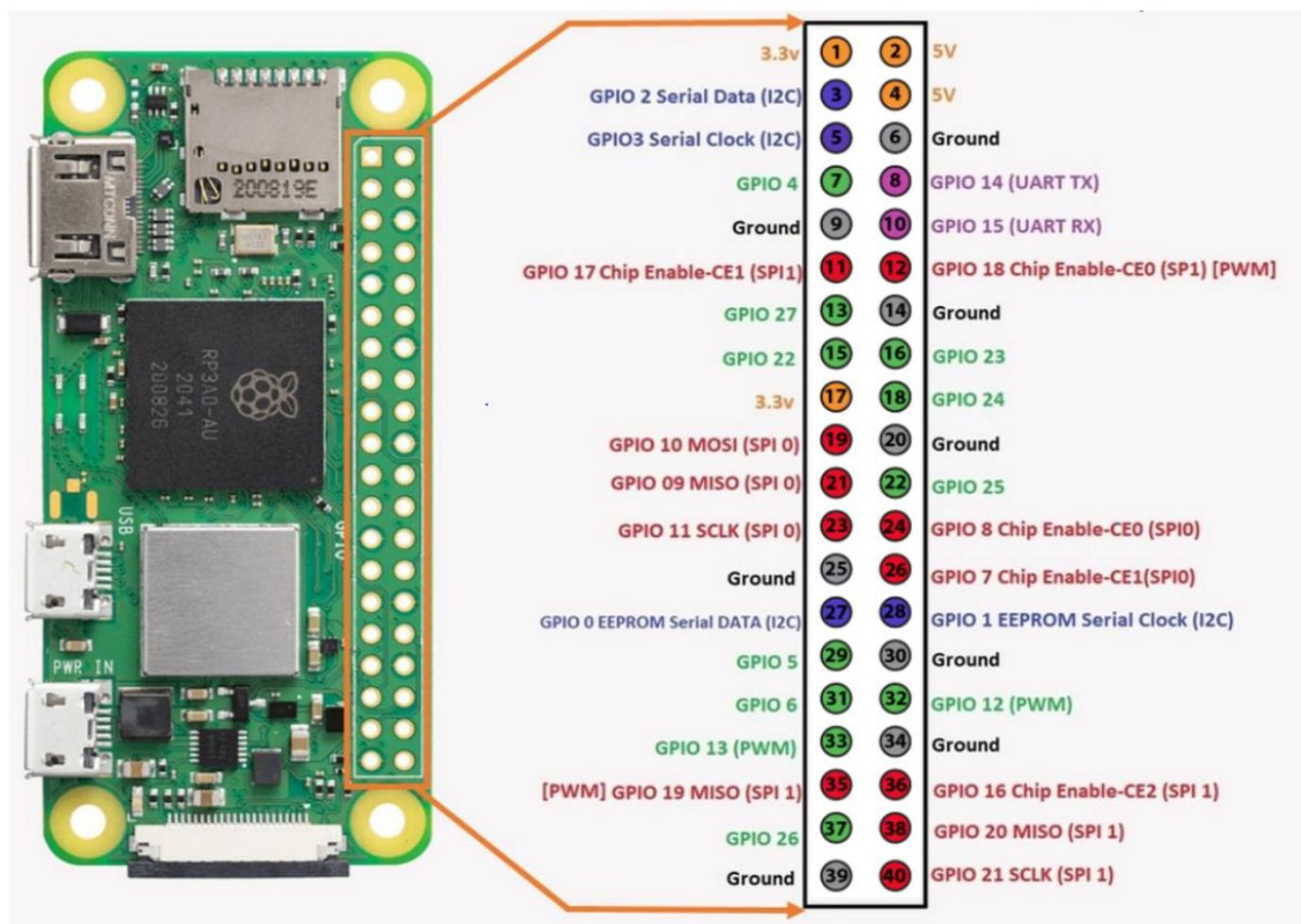
Option	Resolution	Info	Link
1	128x160	128x160, General 1.8inch LCD display Module, ISP, st7735 driver	https://www.waveshare.com/1.8inch-lcd-module.htm
2	240x320	2inch LCD Display Module, IPS Screen, 240x320 Resolution, SPI Interface, st7789 driver	https://www.waveshare.com/2inch-lcd-module.htm

Notes for these alternative displays:

- Assemble directly to the PCB_cover, via specific models. Files at :
 - https://github.com/AndreaFavero71/cubotino/blob/main/personalizations/waveshare_displays/PCB_covers_waveshare_models.zip
- Display connection to the Connections_board:
 - via cables (pay attention to the different pinout order).
 - Paul has used the Connections_board designed by Gerald Maurer, having a good match for the display connection:
https://www.pcbway.com/project/shareproject/W88385ASJ23_CUBOTino2_ea8e6776.html
- When the 240x320 display is used:
 - It must be specified at the Cubotino_settings.txt, parameter ("disp_type": "st7789")
 - Cubotino code automatically resizes the displayed content.

For more info: https://github.com/AndreaFavero71/cubotino/tree/main/personalizations/waveshare_displays

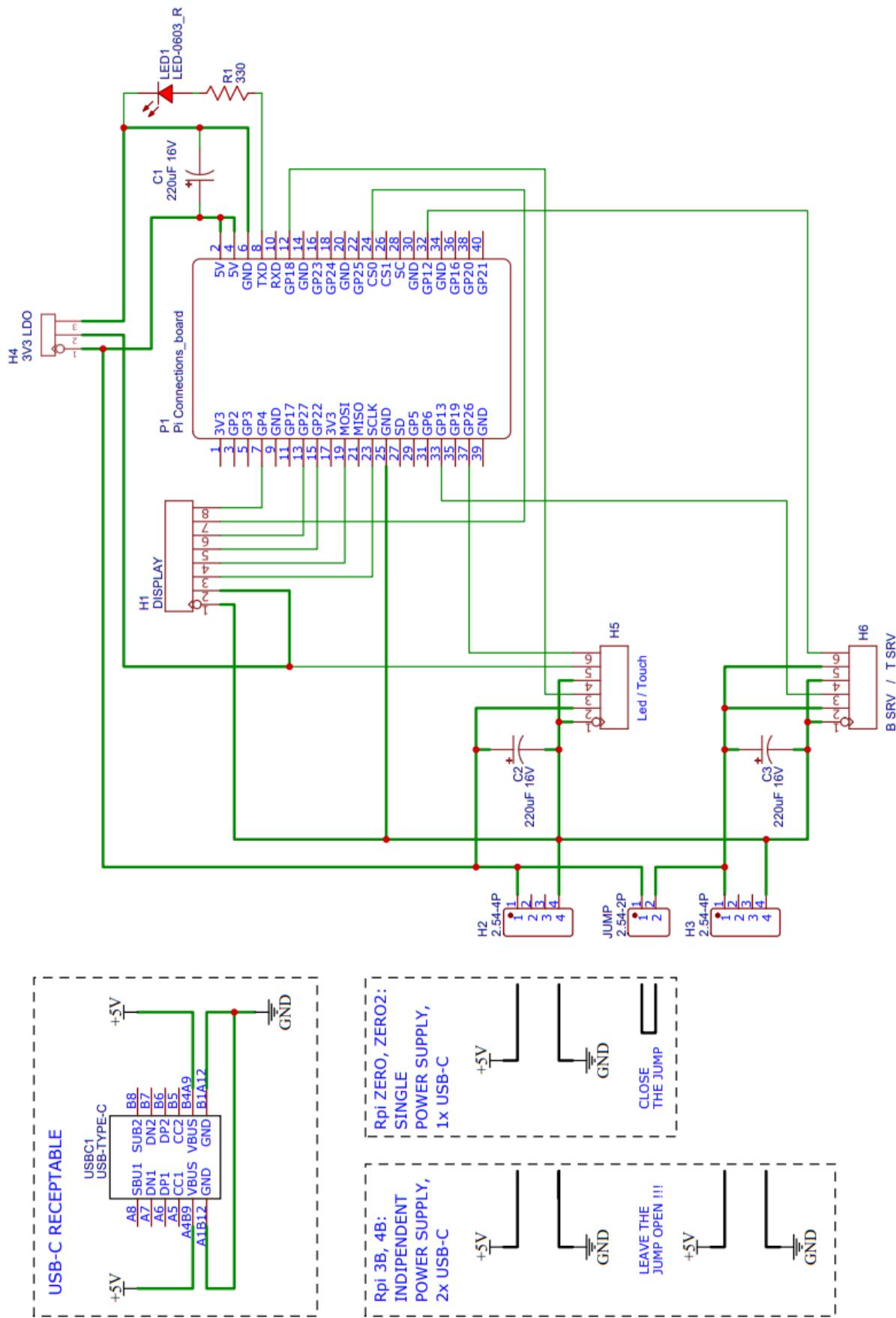
5. Raspberry Pi Zero 2 GPIO pinout



Used pins:

Pin	Label/GPIO	Purpose
2, 4	5V	Energize the SBC board
25, 6	GND	Energize the SBC board. (Note: GND continuity between pins 25 and 6 is made by the Raspberry Pi board)
8	GPIO 14 (UART TX)	Led of Rpi ON status
7	GPIO 4	8 Display backlight
13	GPIO 27	6 Display RS (Register select)
15	GPIO 22	5 Display reset
24	GPIO 8 (CHIP ENABLE SPI0)	7 Display CS (Chip select)
19	GPIO 10 MOSI (SPI0)	4 Display SDA (Serial Data Pin)
23	GPIO 11 SCLK (SPI0)	3 Display SCK (Serial Clock)
12	GPIO 18 [PWM]	PWM Led at Top_cover
32	GPIO 12 (PWM)	PWM servo Top_cover
33	GPIO 13 (PWM)	PWM servo Cube_holder
37	GPIO 26	Touch button

6. Connections board



Section2: Connections_board & Raspberry Pi setup

There are some options for the Connections board:

- A) Chad (Vanblokland) has drawn and made available the Gerber files to order the pcb to a pcb manufacturer;
On the received board you'll 'just' solder the few components needed.
- B) You can make it by your own via a (perfboard) prototyping board.
- C) You can energize it via 2 x 2A microUSB power supply, or via a single 3A USB type-C power supply.

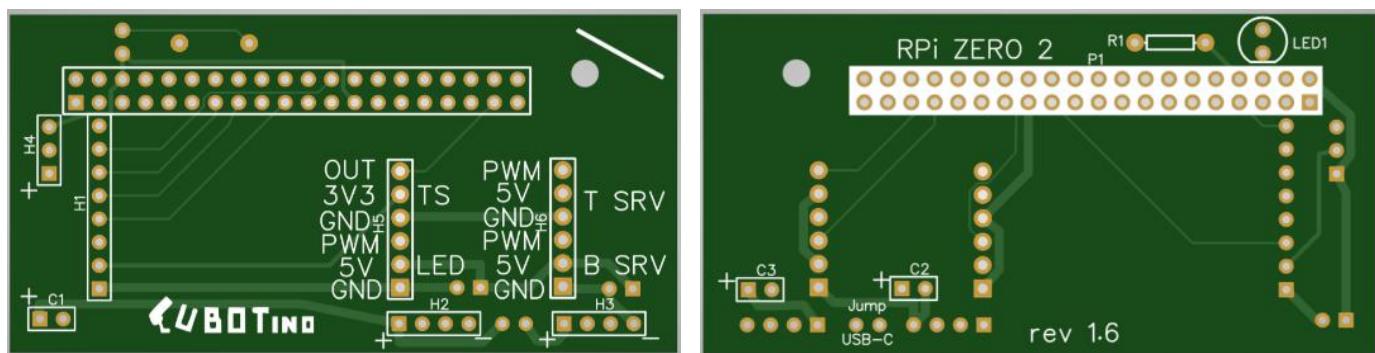
A) Notes for the printed circuit board:

1. This cover the V1.1 and V1.6 revision, that have been positively tested; V1.6 replaces the V1.1
2. Gerber files are available at

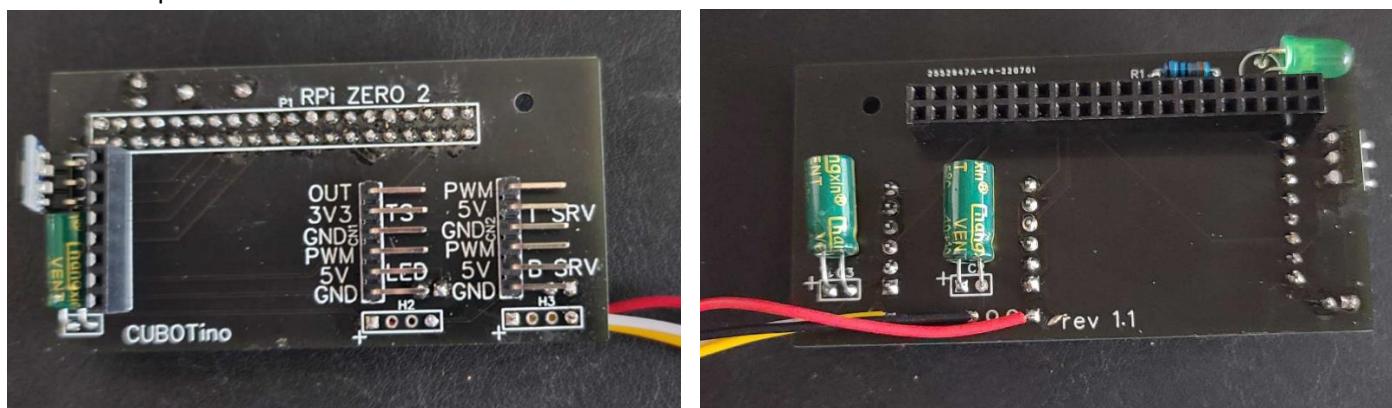
https://github.com/AndreaFavero71/cubotino/tree/main/connections_board/gerber

To download the Gerber files folder from GitHub:

1. Via this link a helpful online tool opens: <https://download-directory.github.io/>.
2. Paste the address of the Gerber folder.
3. Click Download, and a zip file of the folder will be downloaded into your download folder.
3. For the pcb order check: <https://docs.eeasyeda.com/en/PCB/Order-PCB>
4. The boards I got from Chad, have been ordered at: <http://jlpcb.com/>



Just as example:

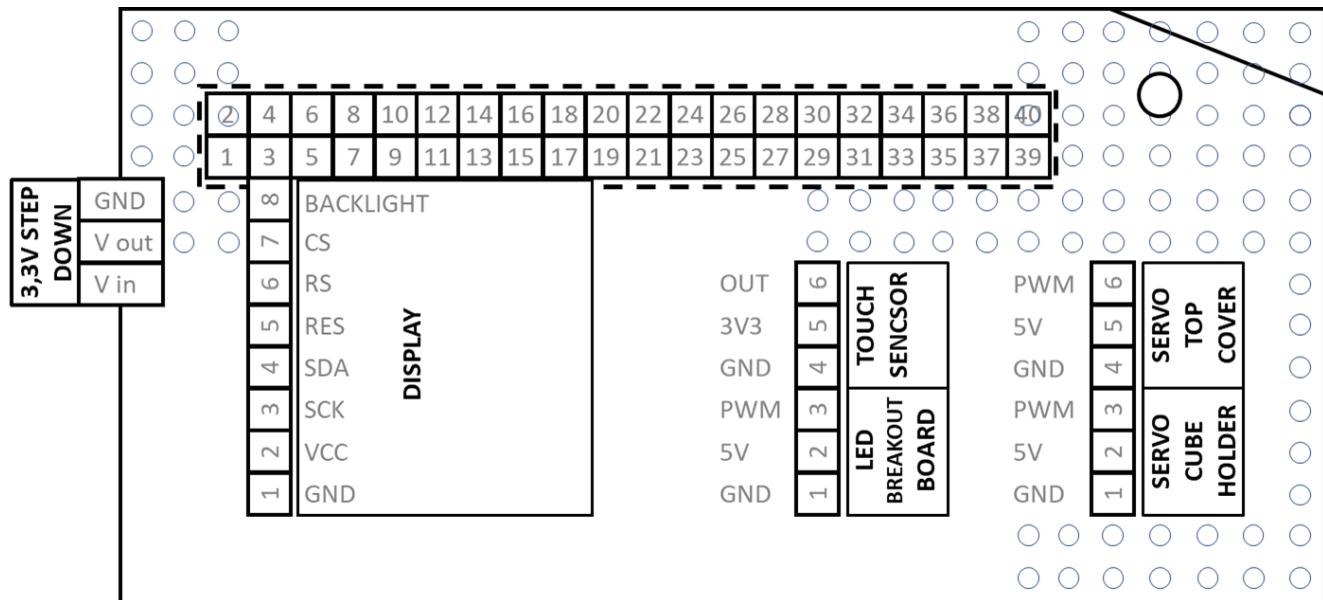
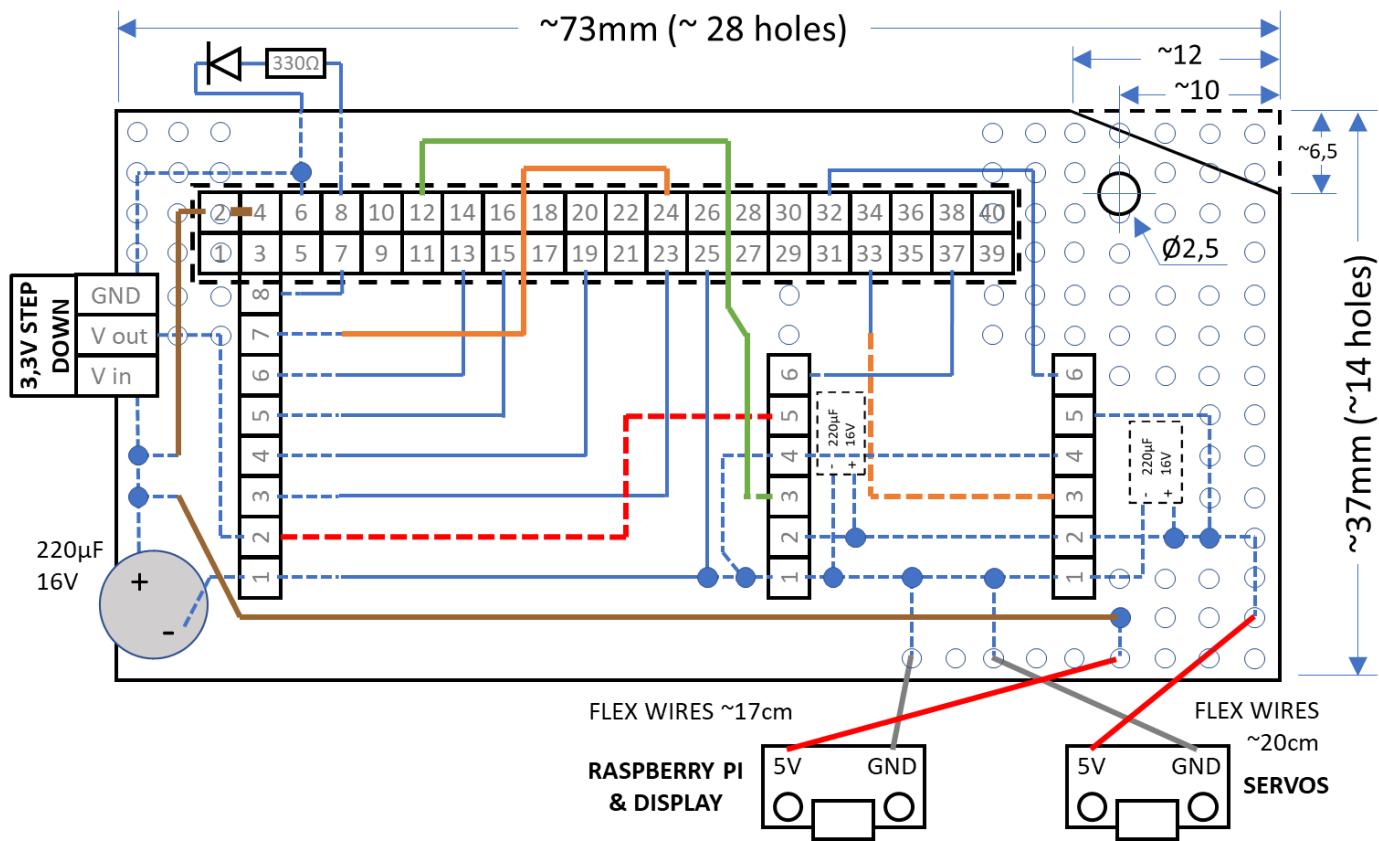


B) Notes for the prototyped board:

1. The prototype board should be of a double face type.
2. Start by positioning the 2x20 header, at 3rd hole from the corner.
3. Count the holes for the other parts positioning.
4. Dashed lines for parts/wires on the opposite side.
5. Blue lines are wires without insulation.
6. Red, Brown, Orange and Green lines are insulated wires.
7. Use insulated wires when crossing other lines, also when these are on the board opposite side.
8. For the power lines, toward the servos and the raspberry Pi, preferably use 20AWG solid wire; Thick power lines are especially needed when using a Raspberry Pi model 3 or 4.
9. Filled dots are connections.
10. 15mm is the max height for the 3V3 voltage regulator board, and the capacitor close to the display connector.
11. Ø2,5mm hole is for a M2,5mm bolt (and 3 nuts) to support the display. The highest nut should be at ~10.5mm from the board surface.
12. Add the last 2 capacitors, close to the servo connectors, once the board has been tested.

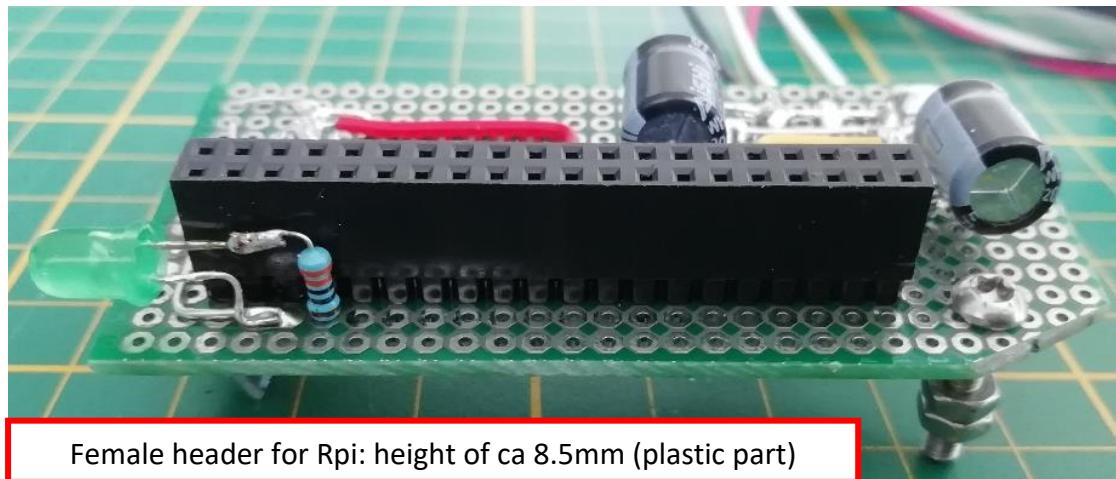
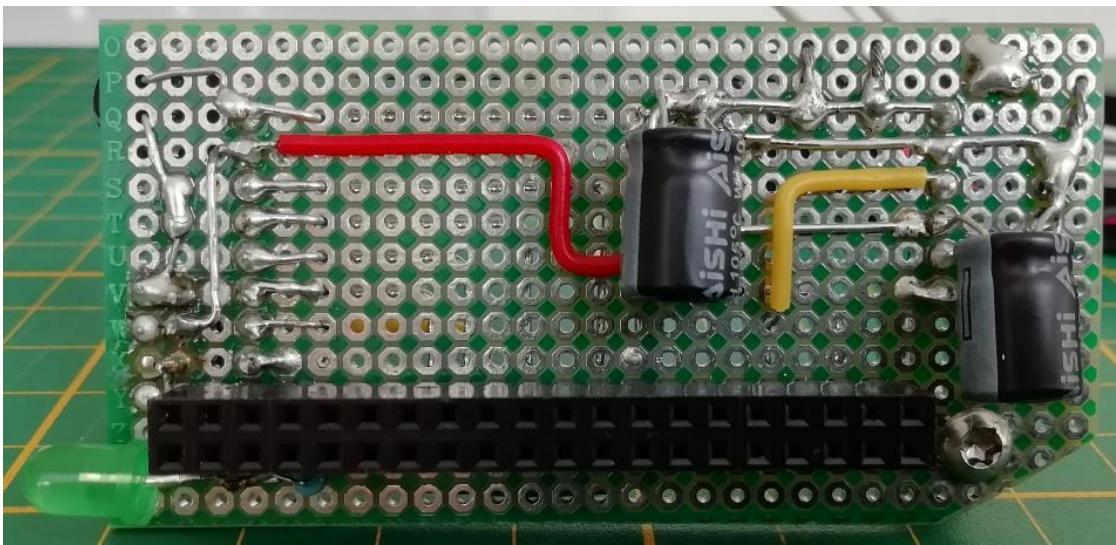
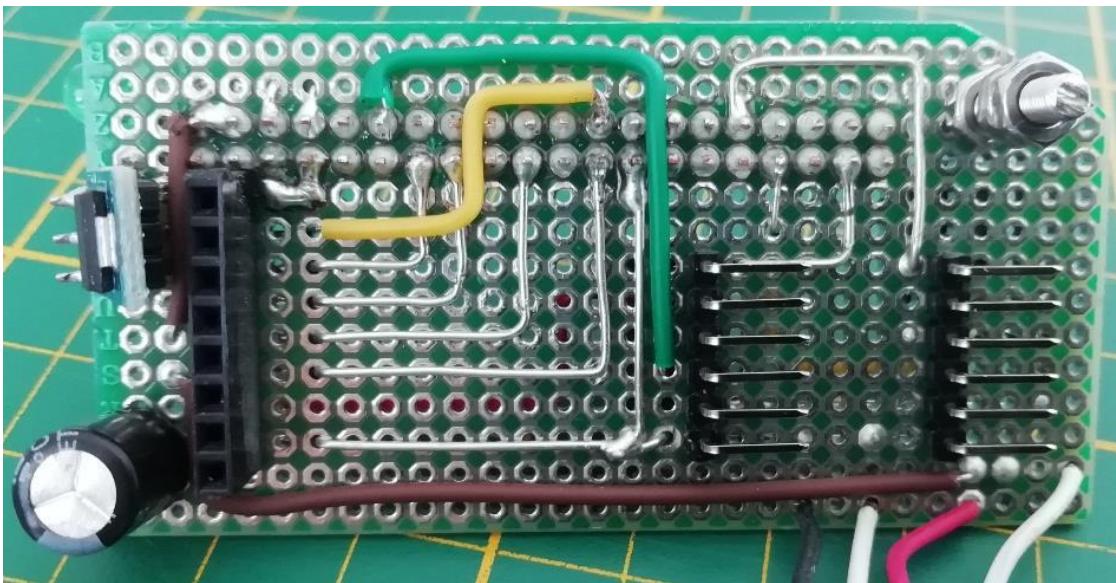
Section2: Connections_board & Raspberry Pi setup

Front view (the 2x20 header is on the opposite side):



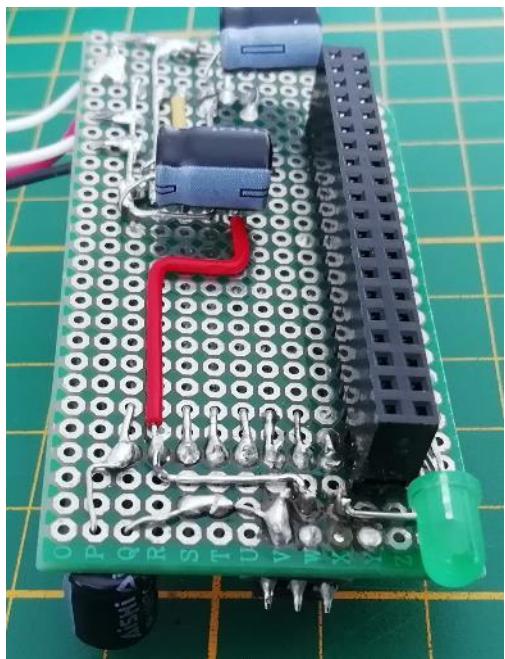
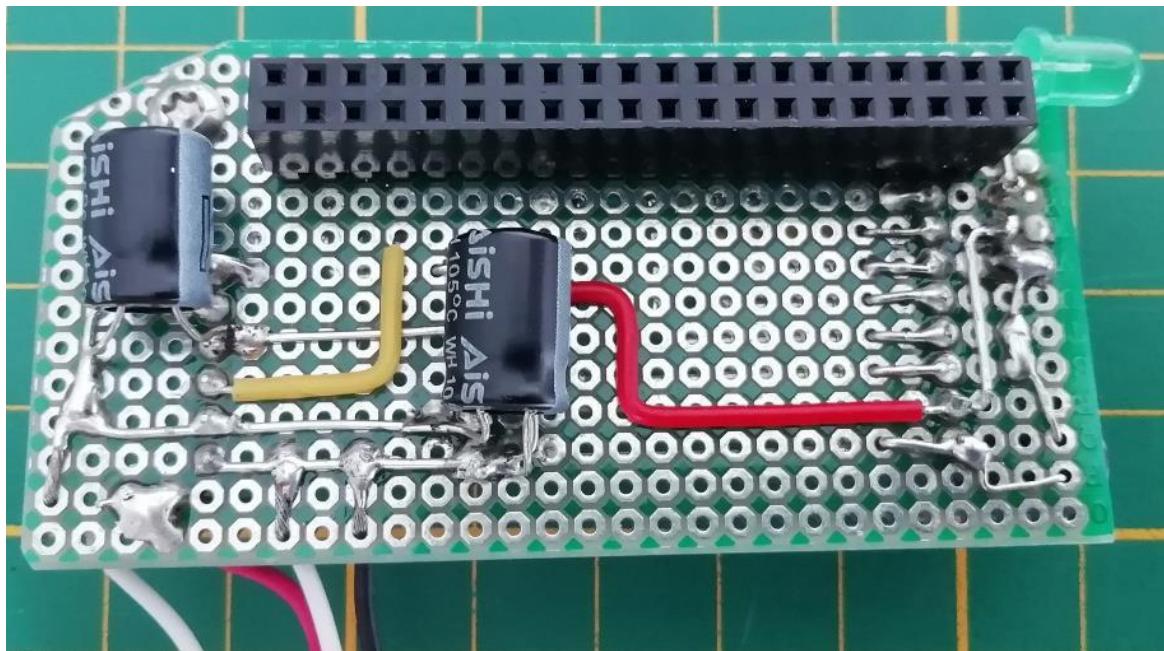
Section2: Connections_board & Raspberry Pi setup

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



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

Section2: Connections_board & Raspberry Pi setup



microUSB breakout board:
Wire soldering orientation



Wiring	Wire length (cm)	MIN Wire size (see next page)	
		Rpi Zero, 2 and 3	Rpi 4
USB breakout board	~18	20AWG or 0.5mm ²	18AWG
Touch sensor breakout board	~18		
3W Led breakout board	~35		

Power supply for the Raspberry Pi boards model 3 and 4.

According to official Raspberry Pi documentation, the model 4b requires a power supply of 5Vdc 3A (or 2.5A of good quality) at the GPIO pins (<https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/>)

For the Raspberry Pi 3 Model B+, it is required a power supply of 5Vdc 2.5A (<https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus/>).

Official Raspberry Pi documentations does not provide information about the wire size.

Perhaps good to notice a few aspects:

1. Official Raspberry Pi power supply, with the USB Type-C connector, delivers 5.1V and it uses 1.5meter of cable 18AWG (<https://datasheets.raspberrypi.com/power-supply/usb-c-power-supply-product-brief.pdf>);
This is the power supply intended for the model 4b
2. Similarly, the Official Raspberry Pi power supply, with the micro USB connector, delivers 5.1V and it uses 1.5meter of cable 18AWG (<https://datasheets.raspberrypi.com/power-supply/micro-usb-power-supply-product-brief.pdf>).

“Power Supply Warnings”

(<https://www.raspberrypi.com/documentation/computers/raspberry-pi.html#power-supply>)

On all models of Raspberry Pi since the Raspberry Pi B+ (2014) except the Zero range, there is low-voltage detection circuitry that will detect if the supply voltage drops below 4.63V (+/- 5%). This will result in a warning icon being displayed on all attached displays and an entry being added to the kernel log.

If you are seeing warnings, you should improve the power supply and/or cable, as low power can cause problems with corruption of SD cards, or erratic behaviour of the Raspberry Pi itself, for example, unexplained crashes.

Voltages can drop for a variety of reasons, for example if the power supply itself is inadequate, the power supply cable is made of too thin wires, or you have plugged in high demand USB devices.

Some of the Makers have reported issues with the power supply (Low Voltage Warning) when using the Rpi 4b.

Kevin has solved by using 18AWG wires for the Rpi power lines, further than using two independent USB connectors.

In case you're going to use Rpi model 4b, you're encouraged to use thick wires.

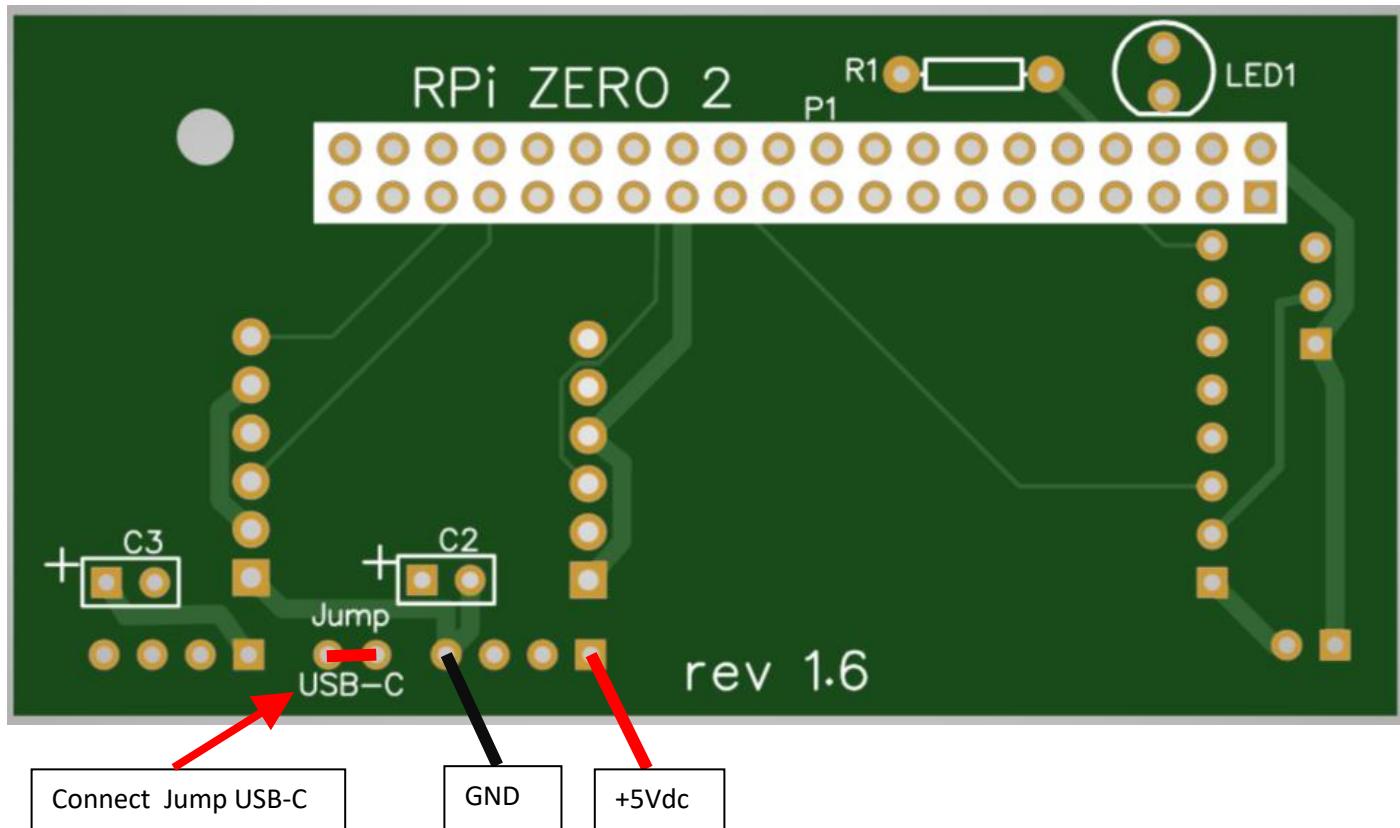
In case of Low Voltage Throttle check the tips at troubleshooting.

Connecting a single power supply input (board V1.6):

USB type-C connector is rated 3A, therefore possible to use a single power supply connections for the robot.

The V1.6 PCB Connections_board drawn by Chad is prepared for this situation:

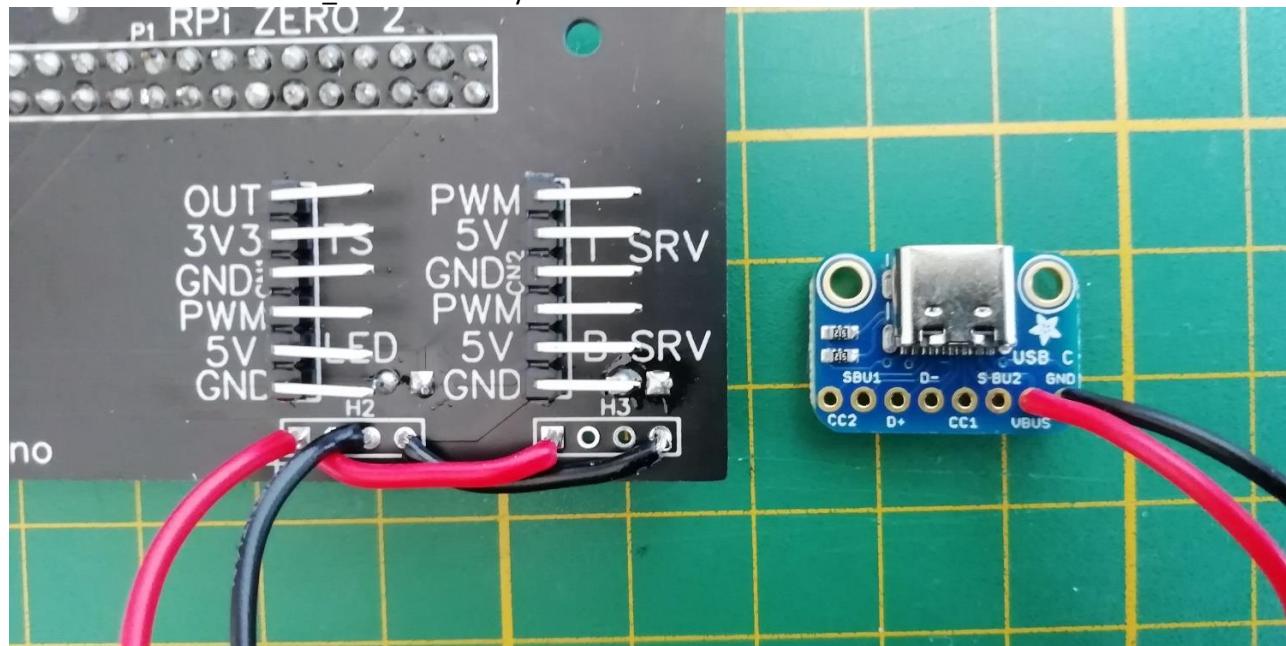
1. Connect the USB-C breakout board to either H2 or H3
2. Connect the "Jump USB-C" pads.



Connecting a single power supply input (board V1.1):

USB type-C connector is rated 3A, therefore possible to use a single power supply connections for the robot.

The V1.1 PCB Connections_board drawn by Chad can be used:

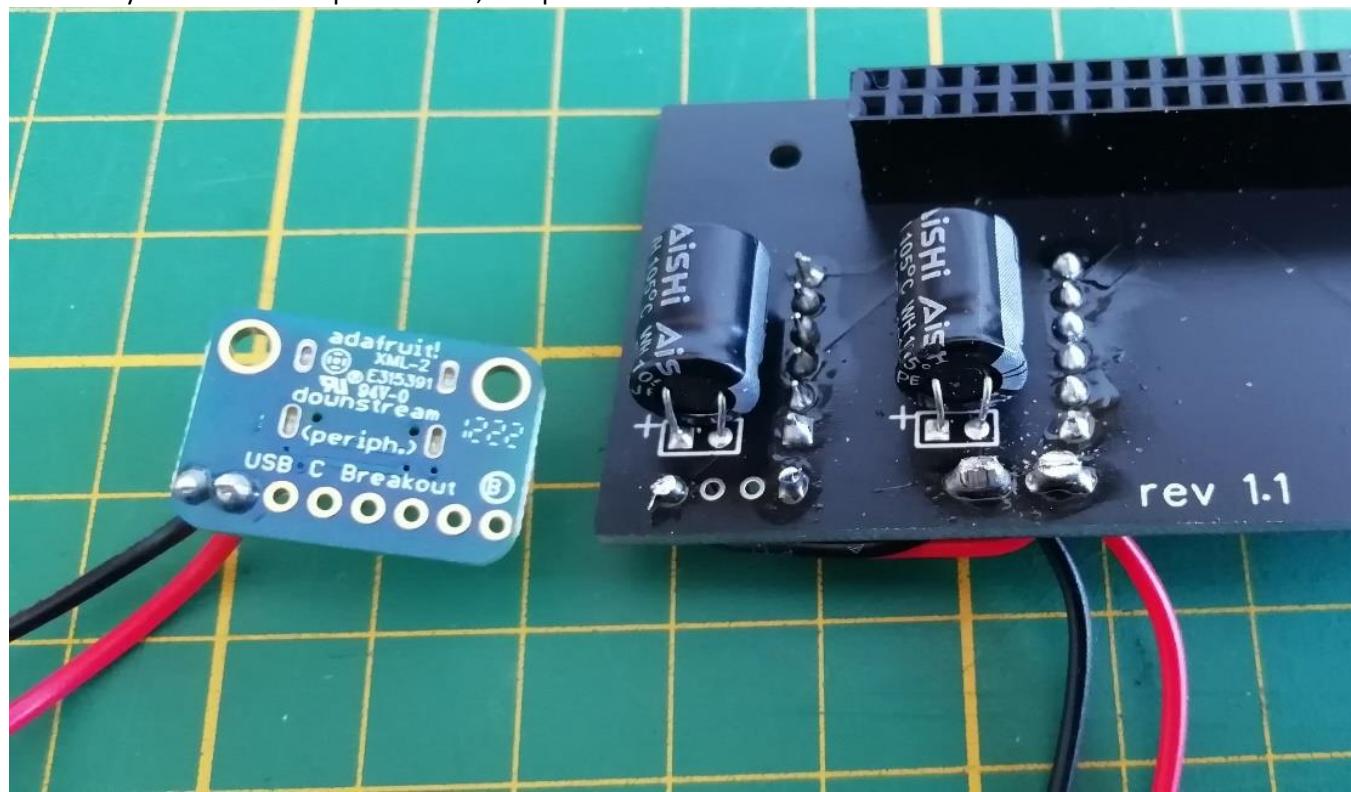


Notes:

At H2 and H3 the soldering pads #2 and #3 are not connected.

Make use of these to mechanically constrain the wires at H2 pins 2 and 3.

Electrically connect the H2 pins 1 and 2, and pins 3 and 4:



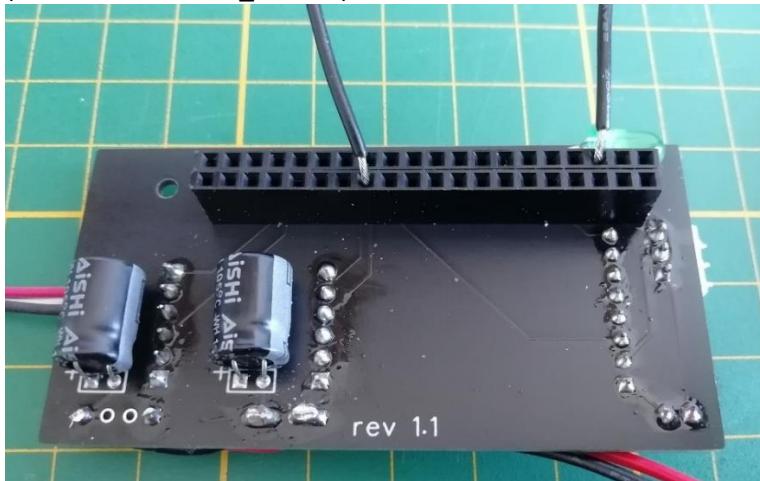
Testing the Connections_board before connecting it to the raspberry Pi:

Before connecting the Connections_board to the Raspberry Pi there are a couple of tests that can be done:

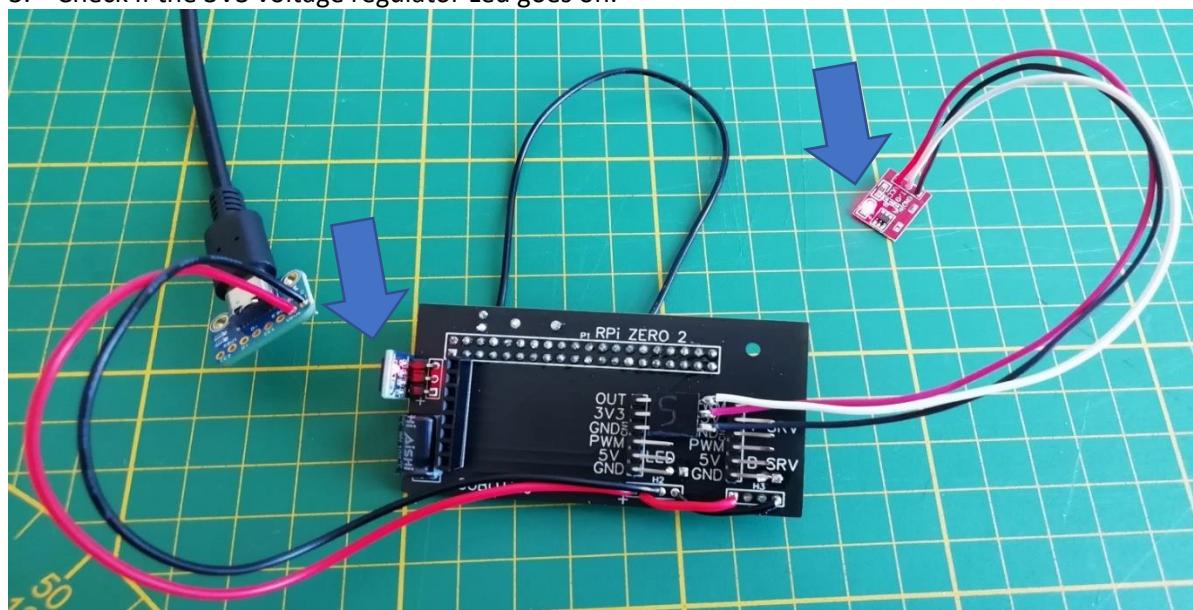
1. Connect the Touch_sensor breakout board.
2. Energize the board, via the USB type-C break out board or via the micro-USB breakout board connected to H2.
3. Touch the Touch_sensor pad and check if the breakout board Led goes on.

Note the sensor sets its level 'off' right after being energized: It is suggested to prevent the sensor from touching the table or your hands when you energize the board, so you can later touch it or drop it to the table to see if it detects the variation.

- 1. (Not necessary with V1.6 board version)** Place a jumper wire between the pins 6 and 25 of the 'Pi' connector (at the Connections_board !).



2. Energize the board, via the USB type-C breakout board or via the micro-USB breakout board connected to H2.
3. Check if the 3V3 voltage regulator Led goes on.



7. Setting up Raspberry Pi

Notes:

1. Starting from 30th June 2022 only the simplified installation (via GitHub repository) is described in this document.
2. Starting from 12th October 2023:
 - o The manual installation method is not anymore maintained.
 - o Cubotino code is updated for “Bullseye” OS while keeping compatibility with “Buster” OS.

This simplified installation method isn't just easier and faster, but it makes very easy to update your robot in case newer updates will be made available at GitHub.

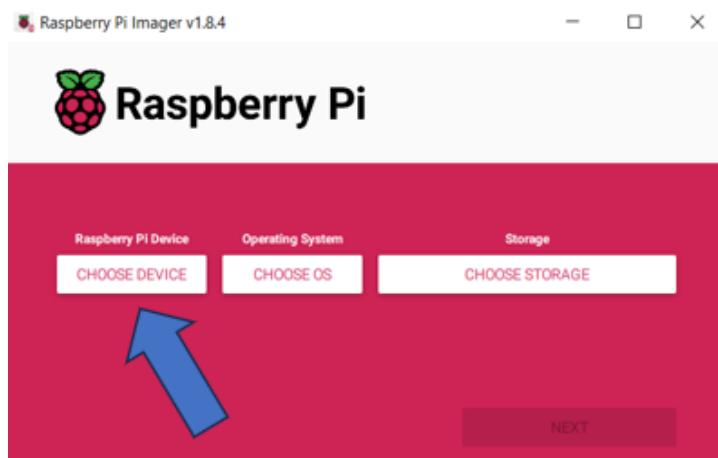
In case of:

- Raspberry Pi Zero2W, or more performant boards, the OS 11 (Bullseye) is suggested.
- Raspberry Pi ZeroW, or similar boards, the OS 10 (Buster) is suggested.

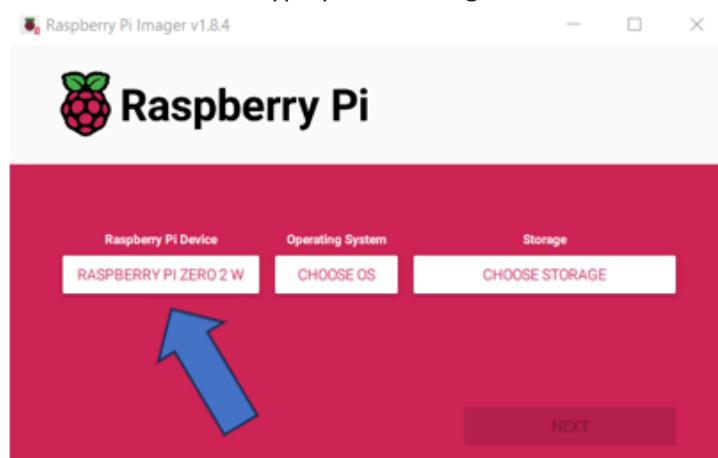
Step 1: Download the Raspberry Pi Imager, from <https://www.raspberrypi.com/software/>

Step2: Flash the OS to the microSD:

- a. Select **CHOOSE DEVICE**

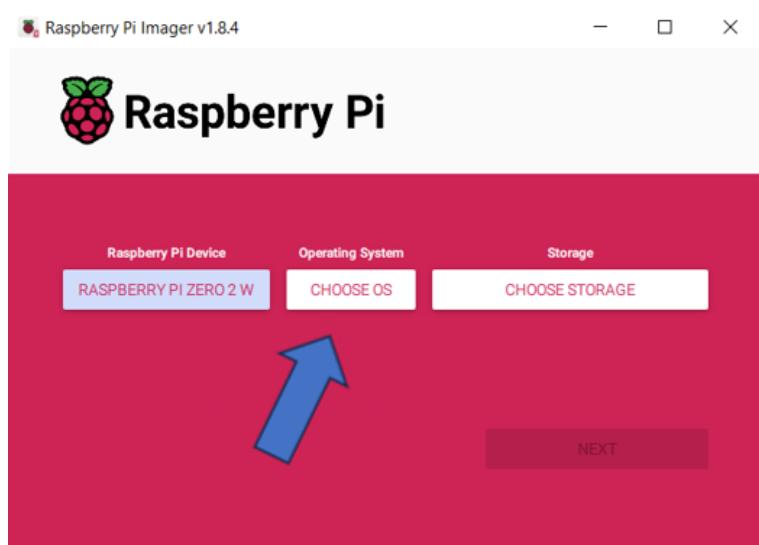


and select the board type you are using:

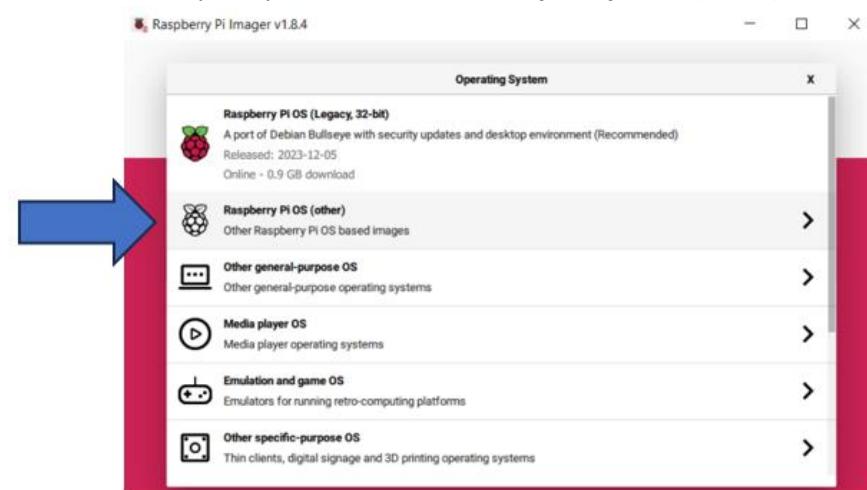


Section2: Connections_board & Raspberry Pi setup

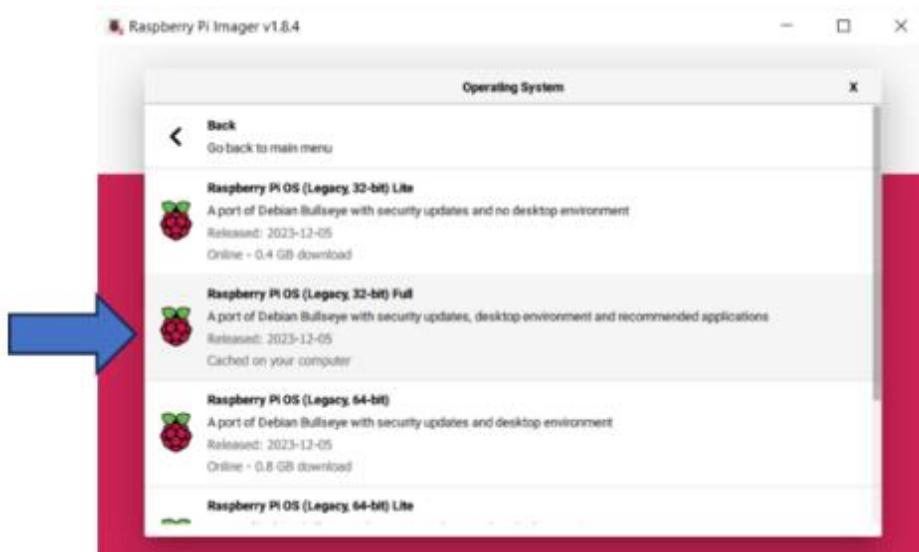
- b. Select **CHOOSE OS**:



- c. Selection of Raspberry Pi OS: under the “**Raspberry Pi OS (other)**”.

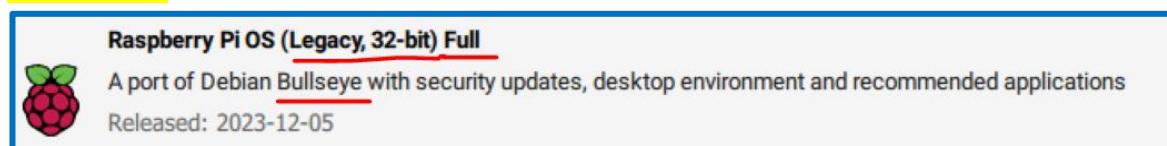


- d. Scroll down, and select **Raspberry Pi OS (legacy, 32-bit) Full**



Section2: Connections_board & Raspberry Pi setup

Double check it mentions "A port of Debian **Bullseye**" and "Desktop environment".

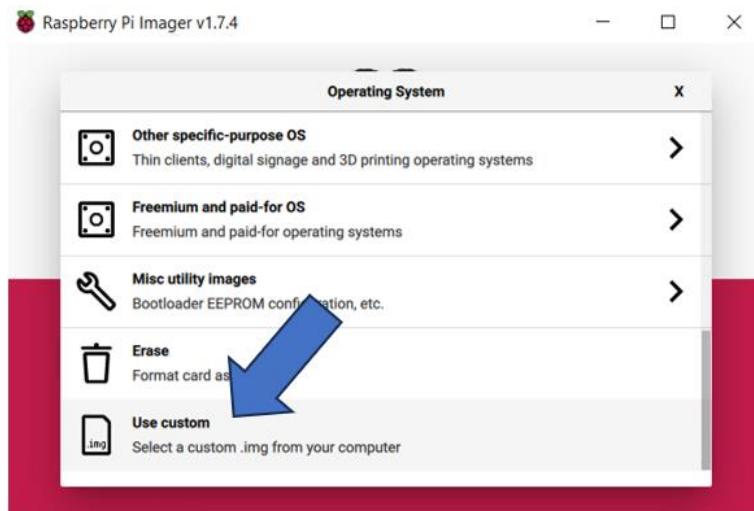


Step2d alternative.

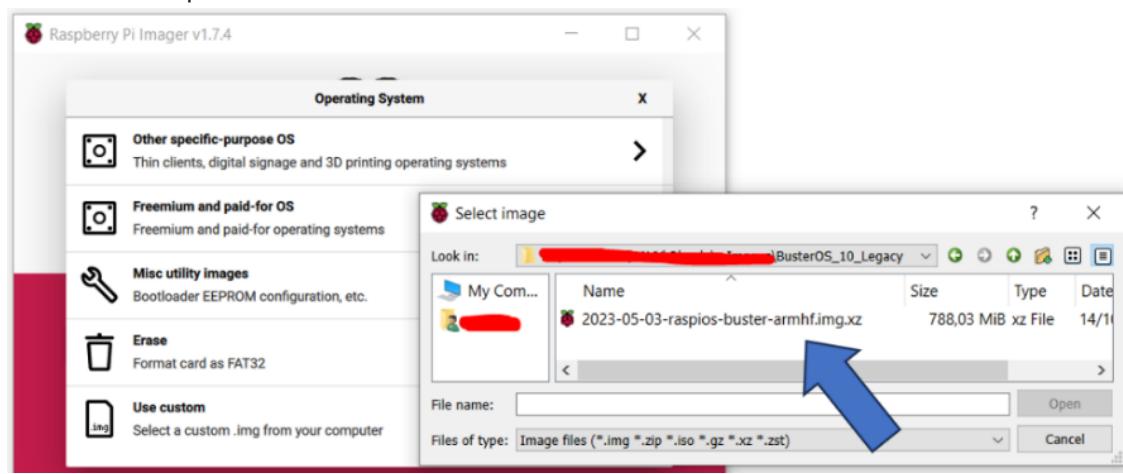
In alternative to the previous step, or as needed approach once Bullseye won't be listed into the Imager, or when the Buster OS is a preferred choice. In this case the OS image needs to be first downloaded in your PC; The official image for the Bullseye OS is:

https://downloads.raspberrypi.org/raspios_oldstable_armhf/images/raspios_oldstable_armhf-2023-10-10/2023-05-03-raspios-bullseye-armhf.img.xz

Scroll completely down and select **Use custom**:



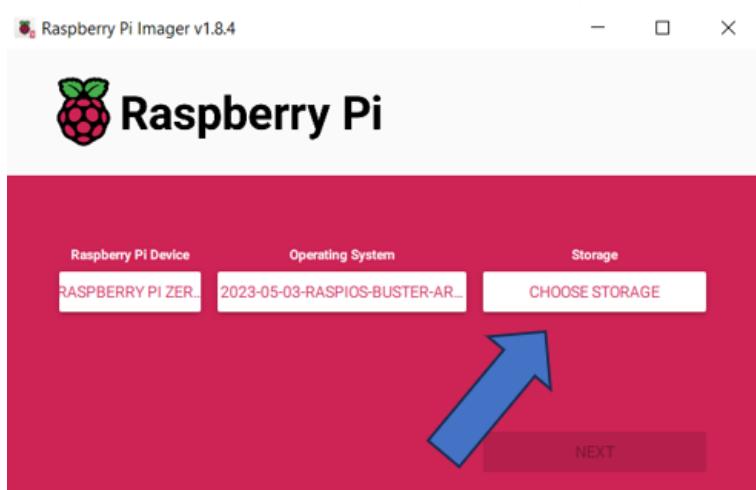
Navigate on your PC and select the downloaded Image (2023-05-03-raspios-buster-armhf.img.xz), confirm with Open.



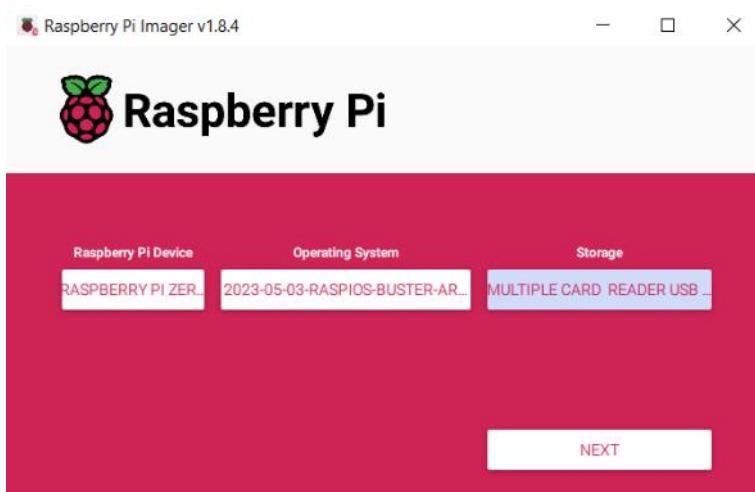
From here onward, the remaining steps are the same.

Section2: Connections_board & Raspberry Pi setup

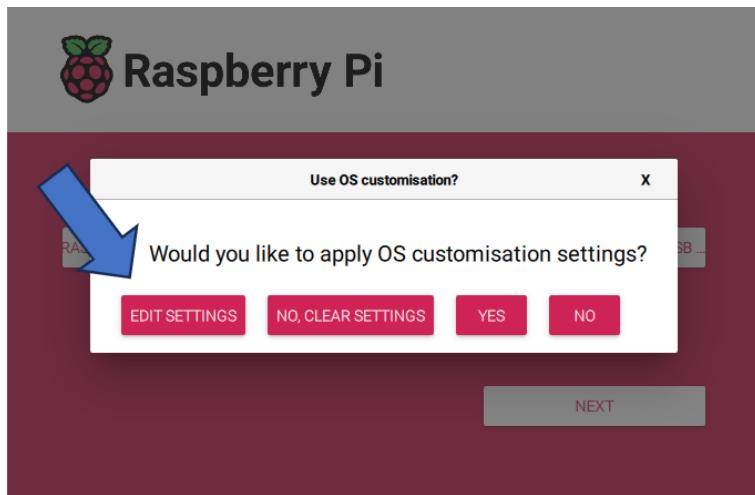
- e. Select the **CHOOSE STORAGE** disk (microSD card):



- f. Select **NEXT**:

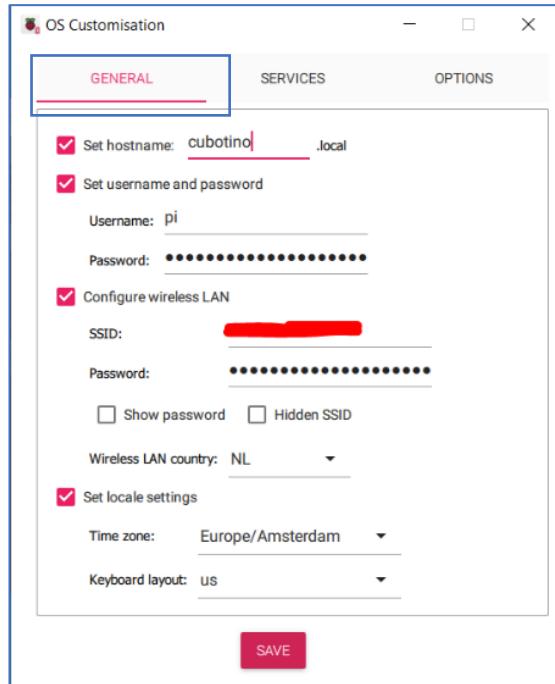


- g. Select **EDIT SETTINGS**:



Section2: Connections_board & Raspberry Pi setup

On settings, GENERAL tab, enter:



- a. *cubotino*.local
- b. enter *pi* as username.
- c. enter a password (my choice has been *raspberry*, as no sensitive data, but you are encouraged to use a more secure password).
- d. enter your SSID.
- e. enter your network password.
- f. enter your Country code.
- g. set your local settings.

On settings **SERVICES** and **OPTIONS** tabs, leaves the default values.

Double check **Enable SSH** is selected

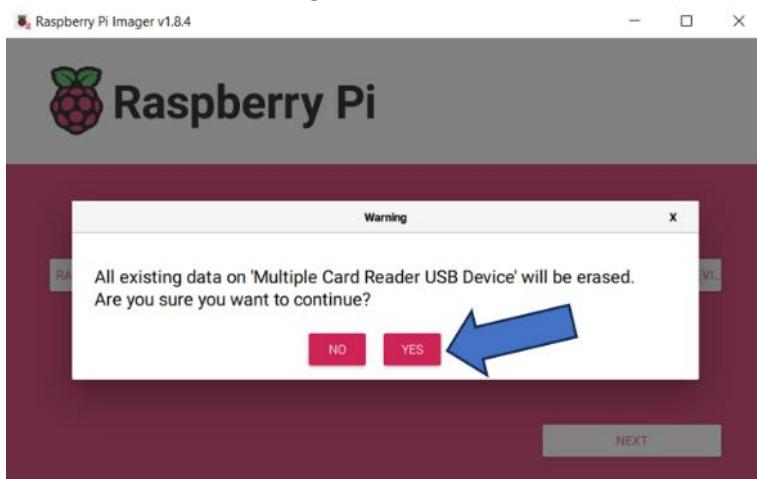
Press **SAVE** to store these settings (these settings will also be available the next time).

Section2: Connections_board & Raspberry Pi setup

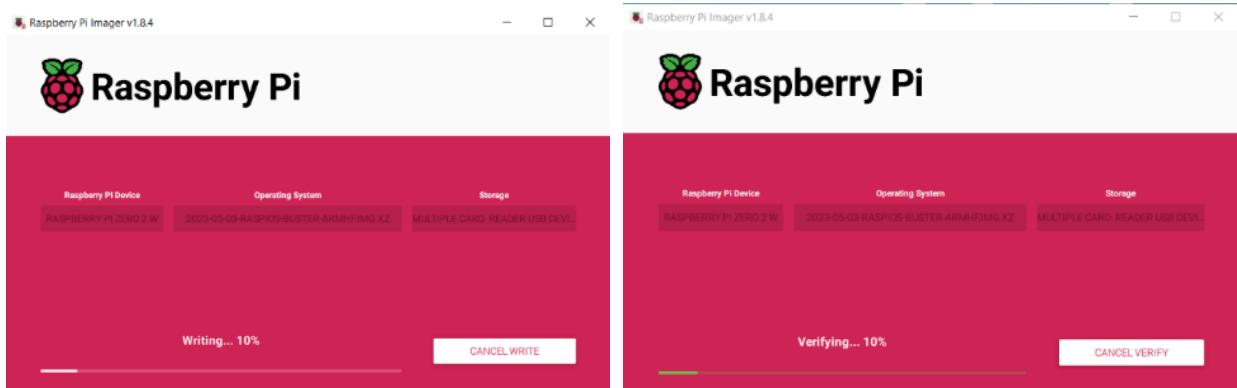
- h. Select **YES** to use the settings:



- i. Select **YES** to start flashing the OS to the microSD:



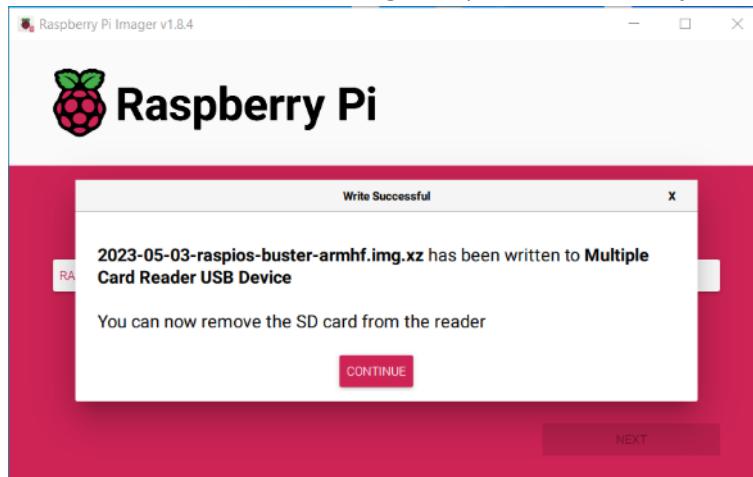
Raspberry Pi Imager starts writing the OS in the microSD, and it's followed by a verification:



Writing and verifying takes around 10 minutes.

Section2: Connections_board & Raspberry Pi setup

- j. Select **CONTINUE** to acknowledge the process end and eject the microSD:



Note: In case you'd like to have multiple WiFi possibilities:

- 1) Eject the card
- 2) Re-insert the card
- 3) jump to Step15.
- 4) Once done come back to Step3.

Step 3: Insert the SD card into the Raspberry Pi, and power it.

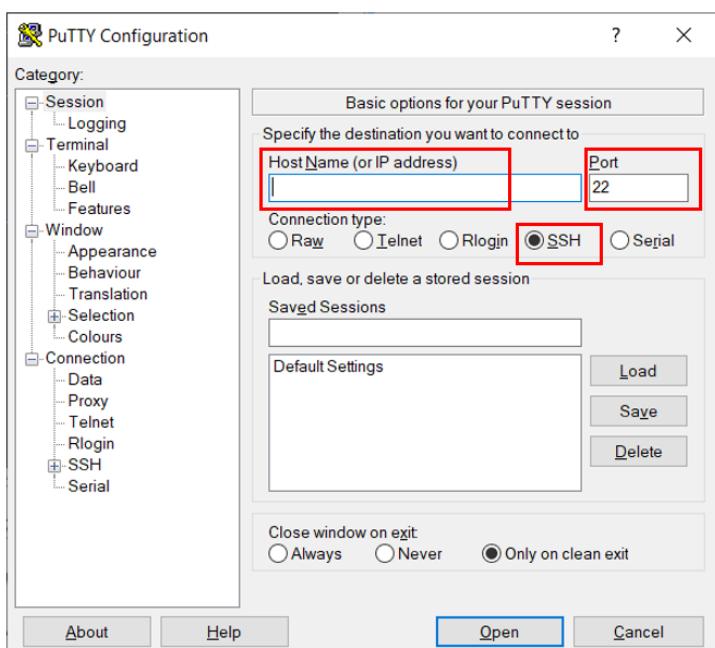
First boot takes longer, up to two minutes; Wait until the led stop blinking.

Step 4: from a command prompt try to connect to the raspberry Pi via ***ssh pi@cubotino.local***; insert the password and jump to Step 6

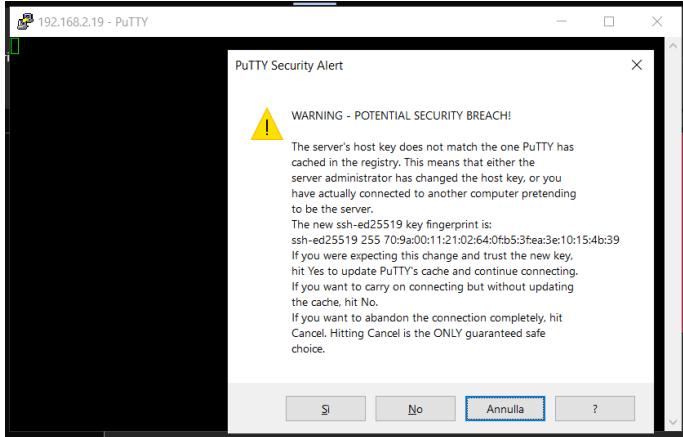
Step 4b: In case you cannot connect via ***ssh pi@cubotino.local***, search the Raspberry Pi IP address.

Different tools can be used to detect which IP address has been assigned to the Raspberry Pi; For instance, Advanced IP Scanner (<https://www.advanced-ip-scanner.com/>)

Step 5: Connect to the Raspberry Pi via SSH, i.e. by using Putty: Run Putty, with the IP address of the Raspberry Pi on the Host Name, remain settings as per Putty default. Accept the warning....



Section2: Connections_board & Raspberry Pi setup



Login as: *pi*

You'll be prompted to enter a password, "pi@xxx.xxx.x.xx's password:" enter *your_password* (*raspberry* in my case).

Note: You can copy the commands from this doc and press Shift + Enter to paste it in the CLI.

Step 6: Clone the repository; From the root (pi@cubotino:~ \$) type (or copy -paste)

git clone https://github.com/AndreaFavero71/cubotino.git

In one or a few minutes (a bit longer for Zero W), also depending on the internet connection, the files will be cloned into raspberry Pi:

```
pi@cubotino:~ $ git clone https://github.com/AndreaFavero71/cubotino.git
Cloning into 'cubotino'...
remote: Enumerating objects: 832, done.
remote: Counting objects: 100% (298/298), done.
remote: Compressing objects: 100% (211/211), done.
remote: Total 832 (delta 173), reused 146 (delta 78), pack-reused 534
Receiving objects: 100% (832/832), 150.57 MiB | 862.00 KiB/s, done.
Resolving deltas: 100% (458/458), done.
Checking out files: 100% (96/96), done.
pi@cubotino:~ $
```

Notes: Commands can be copied, and pasted in the shell with shift + insert 😊

Step 7: Start the installation:

- Enter cubotino/src folder from the root type: `cd cubotino/src`
- Start the bash file that takes care of the installation: `sudo ./install/setup.sh` (attention to the dot).
- In about 10 minutes, also depending on the internet connections speed, a Raspberry Pi Zero 2 will complete the setup.
- Once requested confirm the reboot with a `Y` and press enter.
- When the Raspberry Pi boots, the additional LED powers ON.
- See Appendix 1 for terminal printout reference.

The simplified installation takes care of enabling the necessary Raspberry Pi interfaces: Camera, SPI, Serial com.

The simplified installation takes up to 20 minutes for Rpi ZeroW, much less for a Zero 2W.

At this point the installation is pretty much completed.

The below folder structure will result at Raspberry Pi:

`/home/pi/cubotino/` is the main robot folder; into `/home/pi/cubotino` :

<code>/connections_board</code>	contains the gerber files.
<code>/doc</code>	contains the How_to_buid... .pdf file.
<code>/extra</code>	contains the link to the Instructables page of this robot.
<code>/images</code>	contains the Cubotino logo image for the display.
<code>/stl</code>	contains all the robot stl files.
<code>/stp</code>	contains all the robot stp files.
<code>/scr/</code>	contains all the robot specific files.
<code>/src/twophase</code>	contains the lookup tables for Kociemba solver.
<code>/src/install</code>	contains the setup.sh bash file: Do not use this file!

Note: This folder structure differs from the original one used until 30th June 2022.

After the simplified installation ends, with a raspberry Pi reboot:

At the next connections with the Raspberry Pi, it will be more convenient using VNC Viewer, because of the graphical support.

For VNC Viewer installation: <https://www.realvnc.com/en/connect/download/viewer/>

Step 8: Updating the installation:

The simplified installation isn't just easier and faster to be made, but it makes easier to keep your robot updated, in case newer updates will be made available at GitHub:

1. Enter cubotino folder : `cd ~/cubotino`
2. Type `git status` to check if there are updates.
3. In case there are updates available, type `git pull` to receive them.
4. Starting from 27th January 2024, caused by a mistake of mine, the online repo is messed up.
To get the updates, it's now necessary the below commands:
 - i. `git reset --hard origin/main`
 - ii. `git pull`

Notes:

1. Starting from 10th March 2024, the local repository update should (finally) prevent your local settings to be overwritten:
 - a. The *settings_default.txt are the tracked files by git, and never modified by the Cubotino.
 - b. The *settings.txt are your local setting files, modified by the Cubotino, yet untracked files by git.
2. For your information, the robot makes a backup copy of your settings every time the Cubotino is started; This gives you one more chance to recover your previous settings in case something goes wrong (it shouldn't but you never know). If the Cubotino hasn't been started after the update, in that case remember to rename the backup files:
`Cubotino_T_settings_backup.txt → Cubotino_T_settings.txt`
`Cubotino_T_servo_settings_backup.txt → Cubotino_T_servo_settings.txt`
3. Eventual new parameters that are/will be added to the setting files, will be automatically appended to your settings the first time Cubotino is started after the update.
4. In case 'git pull' returns "error: The following untracked working tree files would be overwritten by merge.", it means you've applied some changes to your local files (files not meant to be changed, like settings files). This problem can be solved by overwriting your local files:
 - a. `git fetch --all`
 - b. `git reset --hard origin/main`

Please bear in mind the updates are based on my robot; There might be other changes you've made: Those must be handled by you another time.

I can think of the display orientation if you've a Raspberry Pi 3 or 4, and you've oriented the display differently.

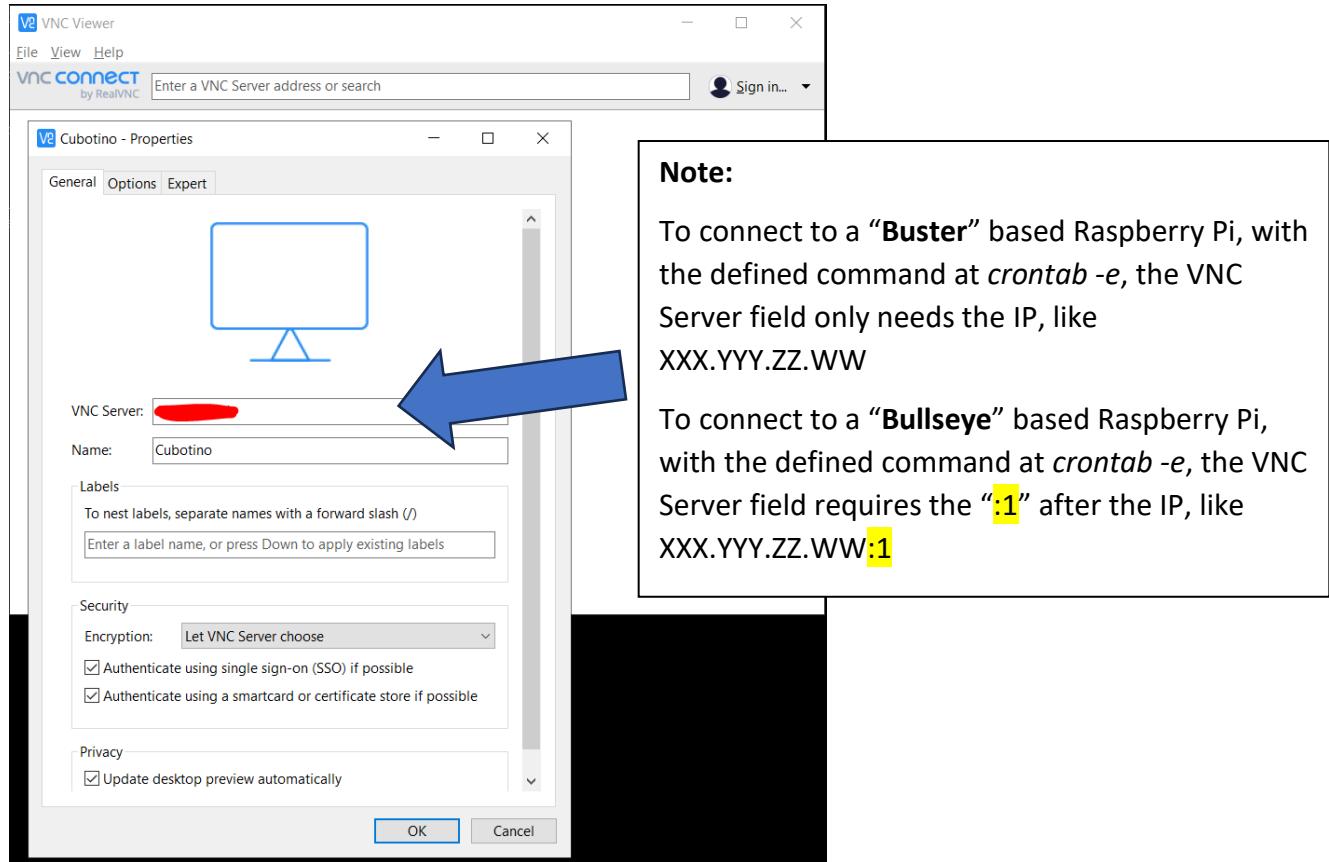
You might also have personalized some prints to the display: Recall to save notes/snippets.

Section2: Connections_board & Raspberry Pi setup

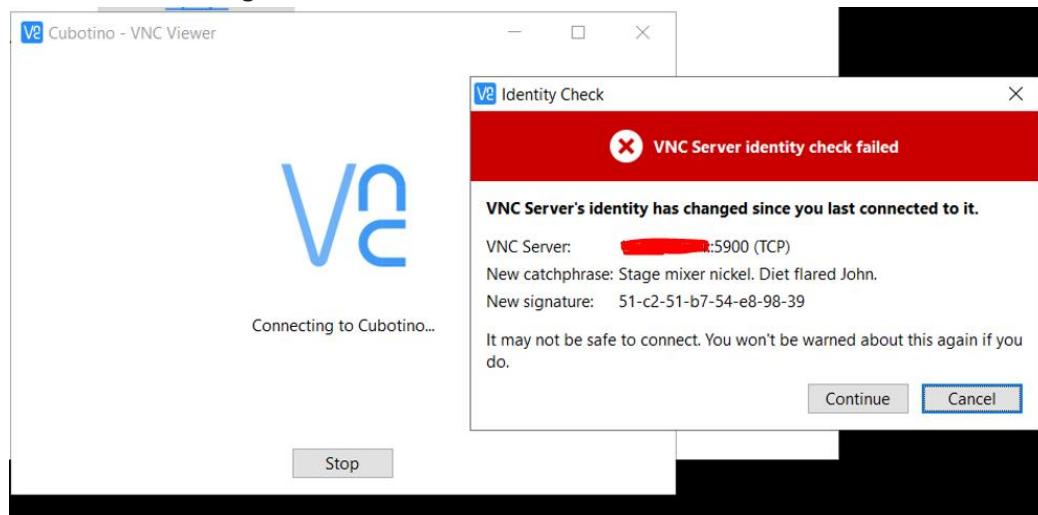
Step 9: Make a new connection at VNC Viewer

Make a new connection: File, New connections, insert the IP address at VNC Server, and a Name.

Note you need to add “:1” after the IP in “Bullseye” (see the note below)



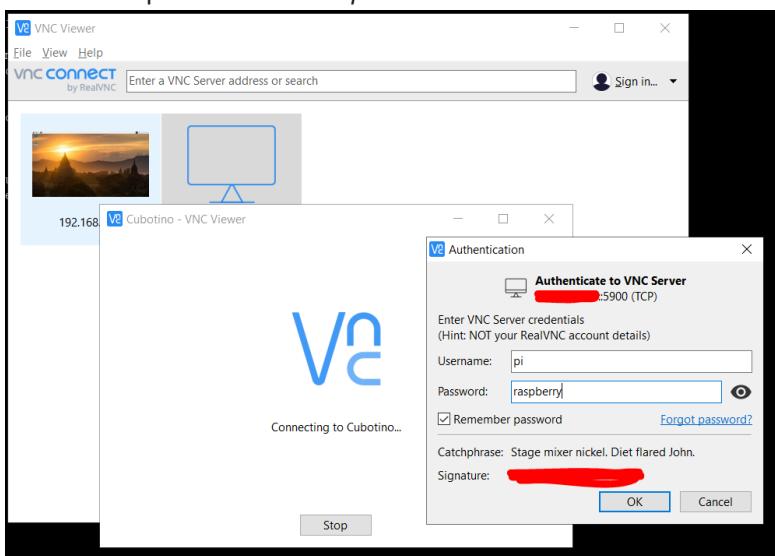
Confirm the warning:



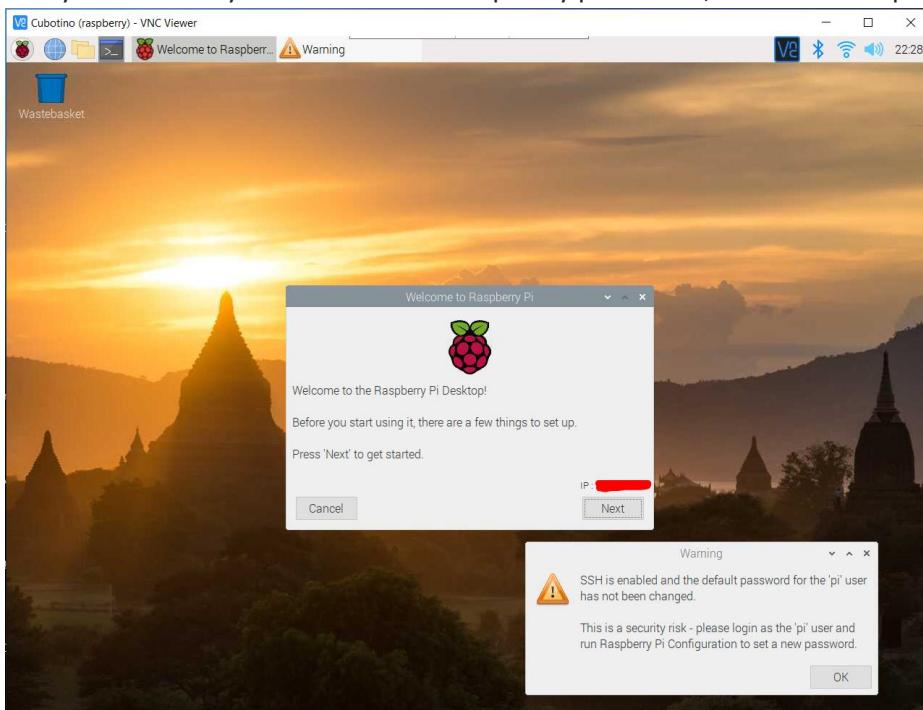
Section2: Connections_board & Raspberry Pi setup

Insert username, *pi* in my case, and the password, *raspberry* in my case.

Check the option *Remember password*.



And you're virtually connected to the raspberry pi monitor; "Buster" example:



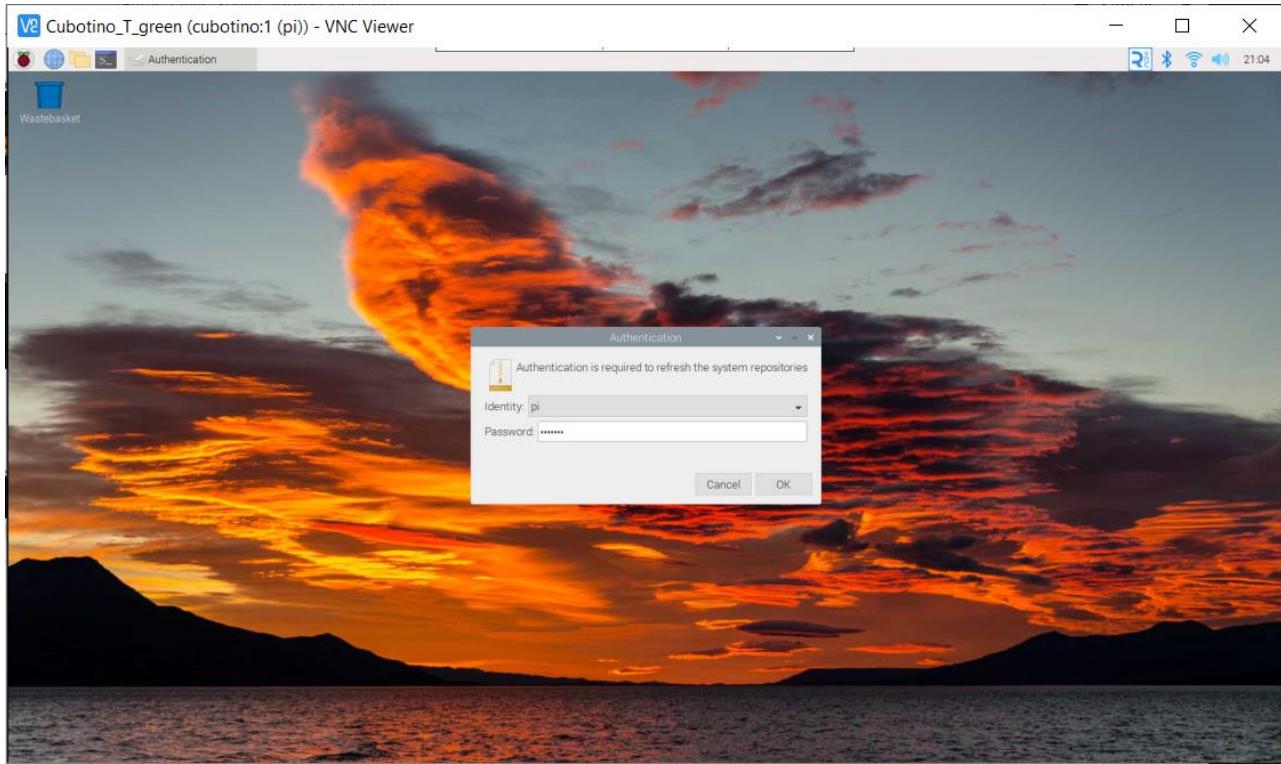
Check, and accept (⌚), the warning.

Complete the settings:

- Set Country
- Change password, in my case again *raspberry*
- **SKIP** the Setup screen (see below)
- **SKIP** Select Wi-Fi Network
- **SKIP** Update Software

Section2: Connections_board & Raspberry Pi setup

“Bullseye” example:



Not yet clear to me when convenient to enter the requested password and when skipping it is just fine.

Step 10: Change the monitor size to use servos_settings_GUI.py ,or in case you don't feel comfortable with the initial proposed resolution of 1280x720.

From the root or from the venv: ***sudo crontab -e***

The first time you'll be asked to choose an editor, use 1 for nano.

```
pi@cubotino:~ $ sudo crontab -e

Select an editor. To change later, run 'select-editor'.
 1. /bin/nano      <---- easiest
 2. /usr/bin/vim.tiny
 3. /bin/ed

Choose 1-3 [1]:
```

Section2: Connections_board & Raspberry Pi setup

The crontab content in “Buster” installation will look like:

```
MAILTO=""
@reboot su - pi -c "/usr/bin/vncserver :0 -geometry 1280x720"
#@reboot su - pi -c "/usr/bin/vncserver :0 -geometry 1920x1080"
#@reboot /bin/sleep 5; bash -l /home/pi/cubotino/src/Cubotino_T_bash.sh > /home/pi/cubotino/src/Cubotino_T_bash.log
```

The crontab content in “Bullseye” installation will look like:

```
MAILTO=""
@reboot su - pi -c "/usr/bin/vncserver-virtual :1 -randr=1280x720"
#@reboot su - pi -c "/usr/bin/vncserver-virtual :1 -randr=1920x1080"
#@reboot /bin/sleep 5; bash -l /home/pi/cubotino/src/Cubotino_T_bash.sh > /home/pi/cubotino/src/Cubotino_T_bash.log
```

The crontab in Bullseye and Buster differs for the :1 instead of :0 at the vncserver command.

To tune the servos via the GUI if will be necessary to use a larger screen.

You can comment the row with the 1280x720 pixel setting and uncomment the row with the 1920x1080 pixel setting.

To save the change: Ctrl + X, then Y, then Enter

Reboot the system (*sudo reboot*) to get the changes effective.

Step 11: Make a backup image of the microSD.

This is the perfect moment to secure the stime spent to get here.....

There are many tutorials available for this task, as reference: <https://howchoo.com/g/nmexndnlmdb/how-to-back-up-a-raspberry-pi-on-windows>

Once the robot will be tuned in your system (see Tuning chapter), then a final backup image will capture the tuning part too.

Step 12: Set things to get the robot starting automatically at the raspberry Pi boot.

See 'Automatic robot start chapter.'

Step 13: Set things to get the robot starting automatically at the raspberry Pi boot.

See 'Automatic robot start chapter.'

Step14: Set Thonny to work with the virtual environment.

Thonny will be handy to eventually tune the parameters hard coded in the scripts; This shouldn't be necessary, as most of the parameters are into the text Json files.

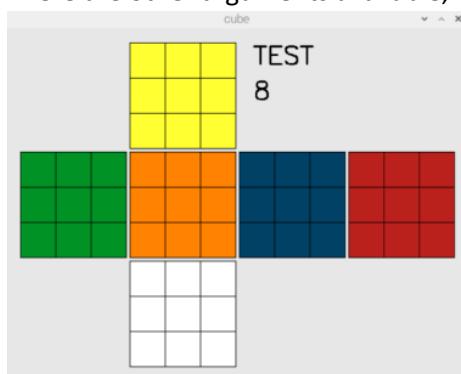
See 'Virtual solver'

Starting from 1st April 2024, CUBOTino has a virtual solver: **Cubotino_T_test_random.py**

The virtual solver:

1. Uses the Kociemba solver to generate a random cube status.
2. The random cube status it sent to the solver to get the solution.
3. The solution is sent to the robot solver to get the necessary moves
4. The robot moves are individually, virtually, applied.
5. After each movement the cube status is updated.
6. After the virtual manipulation of the cube, the final cube status is achieved: The final cube status is verified if conforming to a solved cube.
7. It saves the relevant data in a text file (tab separated data).
8. Considers the servos' timers to estimate the solving time.
9. Returns the expected servos time, servos movements, and how often the optimizations (1 and 2) are used.

The quantity of tests can be passed with '-- runs' argument (i.e., '-- runs 1000'), otherwise it stops after 100 tests. There are other arguments available, like --plot to visualize the solving on the screen:



Below how the outcome looks like, after testing 1'000 random cube status:

```
#####
Tested 1000 random cube's status with 0 failures
The test took 810.34 seconds
Average time to analyze each cube_status: 810.3 ms

Average quantity of robot movements: 96.2
Average estimated solving time (servos): 79.5
std on estimated solving time (servos): 5.287
Min estimated solving time (servos): 59.6
Max estimated solving time (servos): 95.1

Optimization type 1 being used: 595
Optimization type 2 being used: 25

Saved the cube depth and estimated servo time, of the 1000 runs
File saved at: /home/pi/cubotino/src/TestRandom_1000cubes_20240401_201240.txt
```

Section2: Connections_board & Raspberry Pi setup

The virtual solver is a tool for CUBOTino development, to verify the effectiveness of different moves strategies 😊. A variant of it has been used to compare the servos' time from Paul's code vs the original code (180 deg. Rotation instead of 2x90deg), by using the same random generated cube status, thus a fair comparison (Paul's code is ca 13% faster than the original, with standard deviation of 5.8%).

Section2: Connections_board & Raspberry Pi setup

Set Thonny IDE interpreter' chapter.

Step15: multiple Wi-Fi settings:

Adding a second (or more) Wi-Fi connections, for instance to add your phone Wi-Fi_33 hotspot, allows you to show the robot on different locations and still sharing the image processing part on a screen.

For instance, by adding the phone Wi-Fi_33 hotspot details, you can use the Real VNC app on the phone to show the image processing part.

On September 2022 I've presented this project to the Eindhoven Maker Faire, and I used the phone hotspot to show on a large screen what the robot's camera sees and related image processing.

Steps:

1. in the Boot partition of the microSD, create a text file named "wpa_supplicant.conf", and add below content:

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
```

```
update_config=1
```

```
country=NL (use your Country code)
```

```
network={
```

```
ssid="your_SSID_name" (use your SSID name; In my case this is the home Wi-Fi)
```

```
psk="your_PASSWORD" (use your PASSWORD; In my case this is the home Wi-Fi password )
```

```
priority=10
```

```
}
```

```
network={
```

```
ssid="your_SSID_name" (use your SSID name; In my case this is the Wi-Fi hotspot of my phone)
```

```
psk="your_PASSWORD" (use your PASSWORD; In my case this is the Wi-Fi hotspot password of my phone)
```

```
priority=20
```

```
}
```

Note: The priority command is needed when both the WiFis are available on the same time; The higher the value, the higher the priority.

2. in the Boot partition of the microSD, create an empty text file named "ssh" without extension. To create the file, you can use the command "create a new text file" and afterward you change the name and remove the extension.

Step 16: Rpi memory management settings:

(I'm not a real Raspberry Pi expert, below description might not be as precise as it should be).

In case of a Raspberry Pi with 512Mb of RAM (Rpi ZeroW, Zero2W, etc), it is convenient to increase the **swap memory** size, and consequently the **swappiness** and **cache_pressure** parameters.

Starting from 12 October 2023, the SWAP memory is automatically increased during the installation phase.

Swap memory:

My overall understanding is the Kernel optimizes the RAM usage, by loading data and processes expected to be used; When there is more RAM demand, to prevent running out of memory, the Kernel moves some of the (less requested) processes out from the RAM and write them to the swap memory (microSD).

The out of memory problems typically arise in case of accidents, for instance when playing with the settings in case of a syntax error: This might result in an unresponsive microprocessor.

The suggested swap size ranges from 0.5 to 2 times the RAM size, meaning from 256 to 1024Mb.

A too large swap size might reduce the speed, as the microSD I/O access time is much lower than RAM one.

Default swap_size = 100 → suggested value 512Mb

Steps to change the swap memory, from the terminal:

1. stop the current swap process: `sudo dphys-swapfile swapoff`
2. edit the setting file: `sudo nano /etc/dphys-swapfile`
3. modify the swap size: change from `CONF_SWAPSIZE=100` to `CONF_SWAPSIZE=512`
4. save and close the file: `Ctrl+X`, then `Y`, then `Enter`.
5. initialize the swap: `sudo dphys-swapfile setup`
6. start the swap memory service: `sudo dphys-swapfile swapon`

Swappiness and cache_pressure

To minimize the swap_memory from writing too often to the microSD (performance reduction, and potentially reducing the microSD lifespan), it is possible to reduce the **swappiness** parameter: This parameter gives a balance between the memory_swapping and chaching.

Cache_pressure parameter influences the Kernel tendency to reclaim memory from the cache.

In Rpi boards with limited RAM (51Mb), and very slow swap_memory writing (microSD) it is suggested to make more use of the cache memory than swap_memory, to maintain certain responsivity from the system.

Default swappiness = 60 → suggested value 20 (lower values reduces chances for microSD access)

Default cache_pressure = 100 → suggested value 200 (higher value increases the RAM cache)

Steps to modify swappiness and cache_pressure, from the terminal:

1. edit the sysctl file: `sudo nano /etc/sysctl.conf`
2. add at the end: `vm.swappiness=20`
3. add at the end: `vm.vfs_cache_pressure=200`
4. save and close the file: `Ctrl+X`, then `Y`, then `Enter`
5. reboot the system: `sudo reboot`

Section2: Connections_board & Raspberry Pi setup

Step17: Wi-Fi stability:

When the VNC connection drops for long time inactivity, it can be set again without problems.

Differently, when the connection suddenly drops, and it isn't possible to re-establish a connection, then it's necessary to search for the potential cause:

1. Check if the power supply at Rpi is ok.
2. Check if the Rpi green light isn't flashing; This indicates the Rpi processor being busy/freezing (see out of memory info above) and not capable to handle the VNC connection.
3. Check the network at your PC is up and running.

If the problem isn't related to the above listed causes, then you might want to try different Wi-Fi settings:

1. remove the Wi-Fi power management: `sudo iwconfig wlan0 power off`
(to verify the Wi-Fi status: `iwconfig wlan0`)

If the problem persists

2. edit the file /etc/ssh/sshd_config: `sudo nano /etc/ssh/sshd_config`
3. add at the end: `IPQoS cs0 cs0`

Info at https://manpages.debian.org/stretch/openssh-server/sshd_config.5.en.html and
https://en.wikipedia.org/wiki/Differentiated_services

8. Files copied to Raspberry Pi

Note: The simplified installation takes care to copy these files into the Raspberry Pi

Below listed robot specific files, are copied into `/home/pi/cubotino/src` folder:

File	Purpose	Notes
Cubotino_T.py	Main robot script	
Cubotino_T_moves.py	Translates the cube solution (Singmaster notation) in robot moves	From 1 st April 2024 180deg instead of 2 x 90deg.
Cubotino_T_servos.py	Manages the servos	
Cubotino_T_set_picamera_gain.py	Manages the Camera gains settings	Used with OS 10 (Buster)
Cubotino_T_Logo_265x212_BW.jpg	Cubotino logo, plot on display	
Cubotino_T_display.py	Display management	From 6 th August 2022
Cubotino_T_servos_GUI.py	GUI to help the servos and camera settings.	From 18 th April 2023
get_macs_AF.py	Checks the Rpi mac address, to match different settings on different robots.	From 18 th April 2023
Cubotino_T_camera_OS10.py	Camera management, OS 10 (Buster)	From 12 th October 2023
Cubotino_T_camera_OS11.py	Camera management, OS 11 (Bullseye)	From 12 th October 2023
Cubotino_T_pigpiod.py	Stop / start pigpiod at robot start	From 12 th October 2023
Cubotino_T_settings_manager.py	Class interacting with the setting files	From 10 th March 2024
Cubotino_T_settings_default.txt	Json file with settings for Cubotino_T.py script	Default values, to start the tuning
Cubotino_T_settings_AF.txt		Optimized values for my robot
Cubotino_T_servo_settings_default.txt	Json file with settings for Cubotino_T_servos.py script	Default values, to start the tuning
Cubotino_T_servo_settings_AF.txt		Optimized values for my robot
Cubotino_T_bash.sh	Bash file to start-up the robot script automatically after Rpi boots	
Cubotino_T_test_random.py	Test the robot solver by generating random cube to be solved by a virtual cube handler	From 1 st April 2024

Section2: Connections_board & Raspberry Pi setup

Kociemba solver library is necessary. Note: The simplified installation takes care to install this package, further than to copy the lookup tables on a well-defined folder location

Library	Scope	Notes
Kociemba solver (twophase)	Kociemba solver for the (almost optimum) cube solution	This is made by 20 python files and 19 Lookup tables files. https://github.com/hkociemba/RubiksCube-TwophaseSolver

Specific python libraries (automatically installed by the simplified installation):

Library	Forced version	Scope
numpy	1.21.4	Image array manipulation and many other functions
picamera[array]		Outputs camera images in array format
st7735	0.0.4.post1	Driver for the display, used in this project
st7789	0.0.4	Driver for alternative display types.
RubikTwoPhase	1.1.1	Kociemba TwoPhase solver
getmac	0.8.3	Get MAC address (useful when more Cubotino bots with different settings)

9. Servos test and set to mid position

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

- Check if both servos have 180° rotation.
- Check that at least one of them have about 190° rotation, to be used for the cube holder.
- Set both the servos on their mid angle position, prior to the robot assembly.

Check the servo rotation angle, and to set them to their mid position:

1. Connect a connections arm to the non-assembled servos.
2. Connect the servos to the Connections_board.

Argument set, and related value, can be passed to *Cubotino_T_servos.py*, to play with the servos angle and specially to **set them to the mid position before the assembling**.

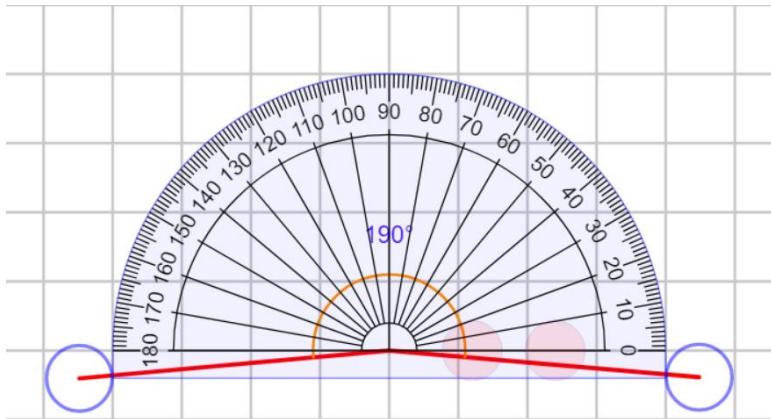
3. Enter home/pi/cubotino/src folder: *cd ~home/pi/cubotino/src*
4. Activate the venv: *source .virtualenvs/bin/activate*
5. Set the servos to the mid position: *python Cubotino_T_servos.py --set 0* (Attention to the double '-' without space in between, and the space before the zero).
6. To check the rotation angle of the servos, you can type a different value (float value between -1 and 1) after the double '-'; Once the script has been started with the “--set” argument, followed by a value, further values can be entered without closing the script.

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!

10. 3D printed parts

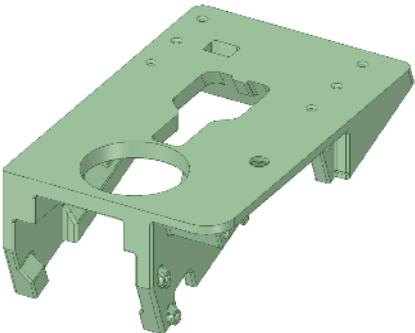
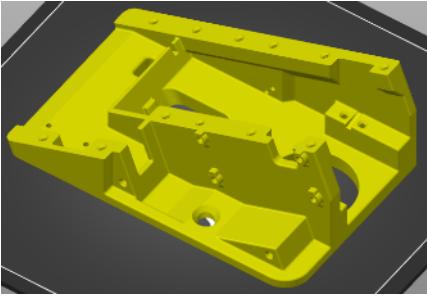
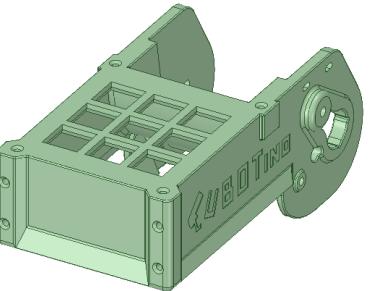
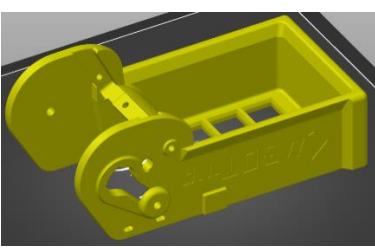
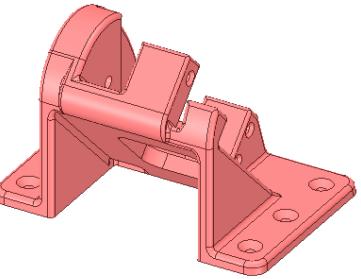
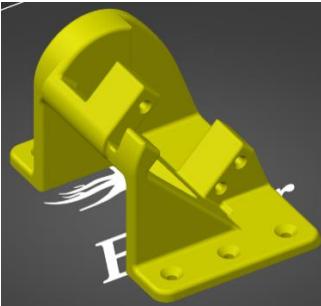
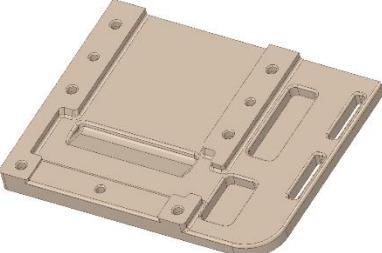
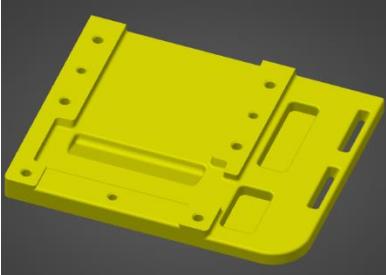
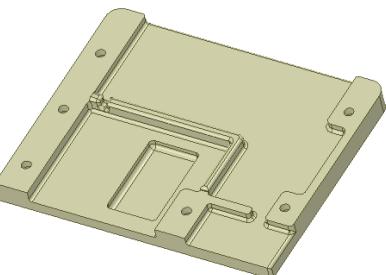
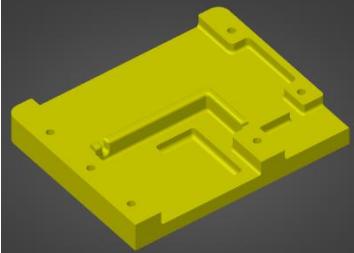
See notes below for filament quantity, and printing time ...

Ref	Part	Filament		Printing time	Version specific parts
		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	Servo_axis_sup (or its alternative)	2,4	7,2	0h55m	
9	Servo_axis_inf (or its alternative)	2,4	7	0h52m	
10	PCB_cover_display (or its alternative)	14	41.4	4h13m (4h27m)	Top specific
11	PiCamera holder	2.2	6.5	0h50m	Top specific
12	PiCamera holder frame	6.1	18	2h17m	Top specific
13	Personaliz_plate	1.5	5	0h30	Top specific
TOTAL		175m	692g	51h30m	

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. 0.2mm layers
 2. Low speed (between 25 to 40mm/s for the external parts and 1st layer)
 3. 4 layers on vertical walls
 4. 5 layers on horizontal walls
 5. 30% filling
 6. 8mm brim
4. The filament quantity, and printing time, in the table should be considered as an upper limit. After printing the parts for the base version, total weight was closer to 400grams than 466grams estimated by the slicer SW.
5. Possible to upgrade the Base version, by printing 3 (or 4) more parts, requiring ~ 8h.
6. All parts have been designed to be printed without supporting the overhangs.
7. Some parts have been split, for easier and better 3D printing.
8. The suggested part orientation for the 3D print is showed on below Table.
9. The **stl** files are available at
 - <https://www.instructables.com/CUBOTino-Autonomous-Small-3D-Printed-Rubiks-Cube-R/>
 - <https://www.thingiverse.com/thing:5407226>
 - <https://github.com/AndreaFavero71/cubotino/tree/main/stl>
10. The **stp** files are only available at <https://github.com/AndreaFavero71/cubotino/tree/main/stp>

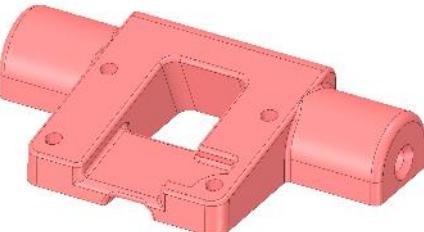
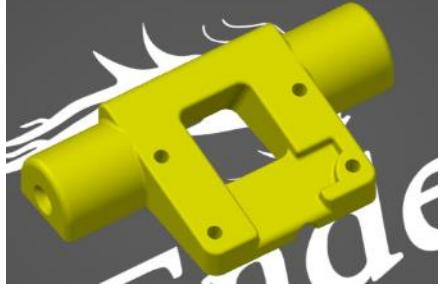
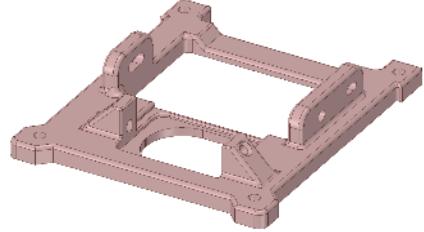
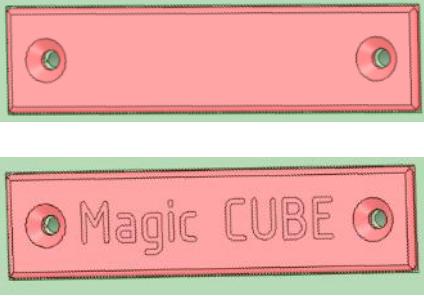
Section3: 3D print and assembly

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

Section3: 3D print and assembly

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

Section3: 3D print and assembly

PiCamera holder		
PiCamera frame		
Personaliz_plate Or Personaliz_plate01		

11. Assembly steps

Before assembling the robot:

- Make the connections board.
- Setup the Raspberry Pi.
- **Position the two servos output gear to their middle position (see Servos test and set to mid position chapter)**

Assembly order is initially as per the Cubotino_base_version:

4. Screw the bottom servo to the structure.
5. Prepare the sandwich Servo_axis_inf / servo lever / Servo_axis_inf.
6. Assemble the Cube_holder to the Servo_axis assembly.
7. Assemble the Cube_holder assembly to the bottom servo.
8. Assemble the Hinge to the Structure.
9. Assemble the "T25" servo arm to the upper servo, only after knowing the servo is on its middle position.
10. Insert the Top_servo assembly into the Top_cover slot.
11. Assemble the Top_cover assembly to the Hinge.
12. Complete the top servo assembly to the Hinge.
13. Assemble the Lifter to the Top_cover.

Starting from here, the steps are different from the Cubotino_base_version.

14. Assemble the PiCamera holder frame to the Top_cover.
15. Assemble the LED breakout board to the Top_cover.
16. Position the cables and assemble the Baseplate_rear.
17. Connect the PiCamera flex cable to Raspberry Pi Zero2 board and fix it to the Structure.
18. Fix the Touch_sensor to the PCB cover.
19. Assemble the USB breakout board to the PCB_cover.
20. Connect the servos, LED breakout board and Touch sensor.
21. dress the cable and connect the display.
22. Assemble the PCB_cover.
23. Assemble the Baseplate_front to the Structure.
24. Stick the PiCamera to its board, via the self-adhesive tape underneath the camera.
25. Assemble the PiCamera module to the PiCamera_holder.
26. Assemble the PiCamera holder to the PiCamera holder frame.
27. Connect the flex cable to the PiCamera module.
28. Personalize the formal Stop plate, transformed to a personalization plate in this robot version.

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



12. Assembly details

Step4 (Mount the bottom servo to the structure):

4x M3x12mm cylindrical head

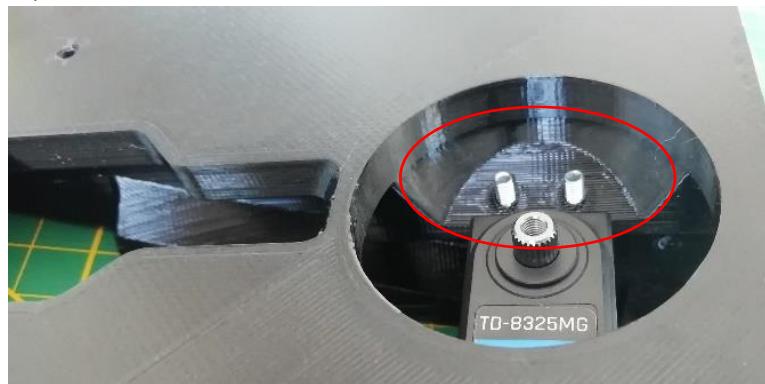
Couple of washers

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.



Section3: 3D print and assembly

Step5 (sandwich Servo_axis_inf / servo arm / Servo_axis_inf):

4x M3x12mm conical head

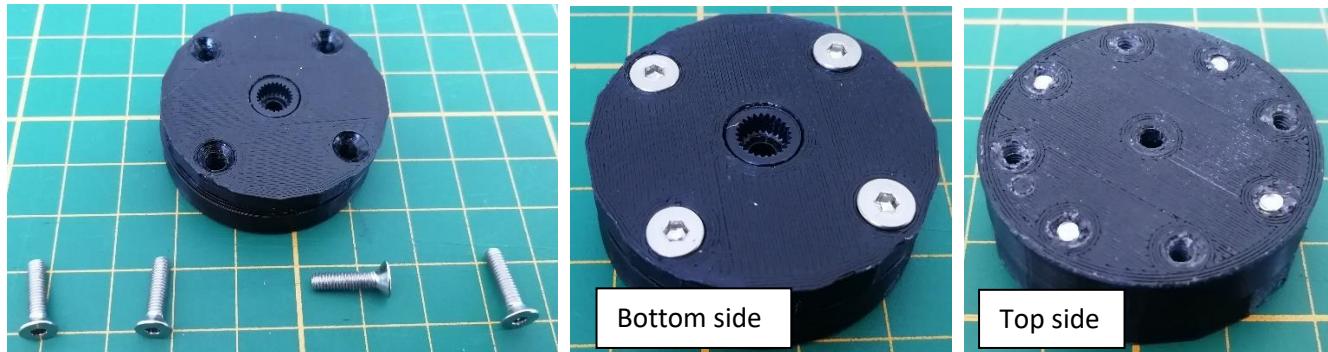


Check if the supplied X arm fits with the indentation of Servo_axis_inf part:



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 must be reduced in length).

Make the sandwich



Section3: 3D print and assembly

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



Section3: 3D print and assembly

Step6 (Assemble the Cube_holder to Servo_axis assembly):

4x M3x12mm conical head



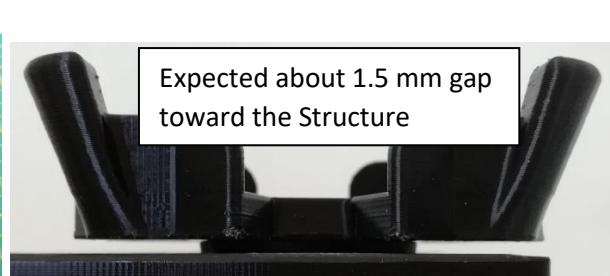
Step7 (Assemble the Cube_holder assembly to the bottom servo):

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.



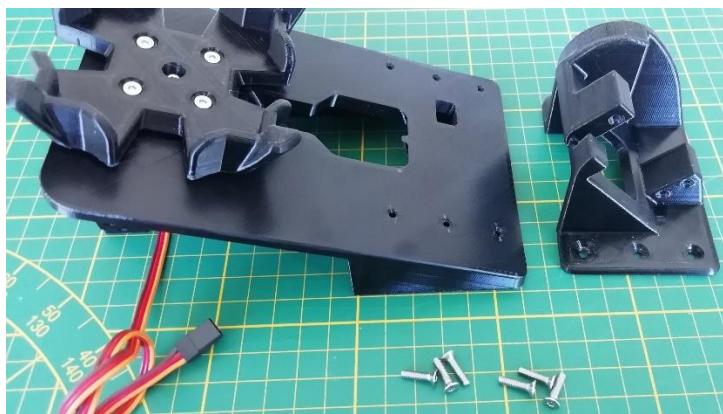
1x M3x12mm cylindrical head



Section3: 3D print and assembly

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 tiny screws at the arm.



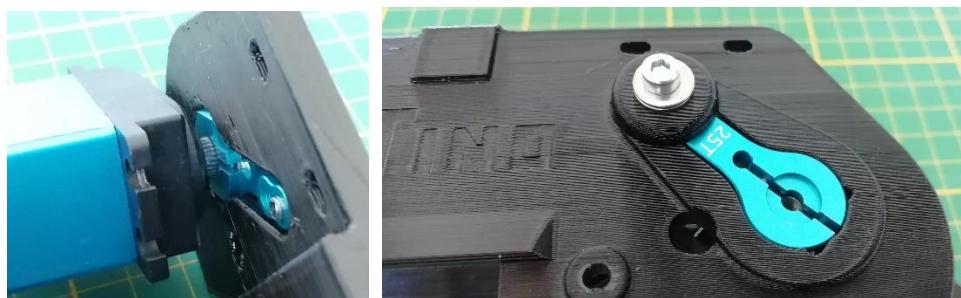
Section3: 3D print and assembly

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 constrain the screw (it should have $\varnothing 4.1$ to $\varnothing 4.3$ mm)



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.



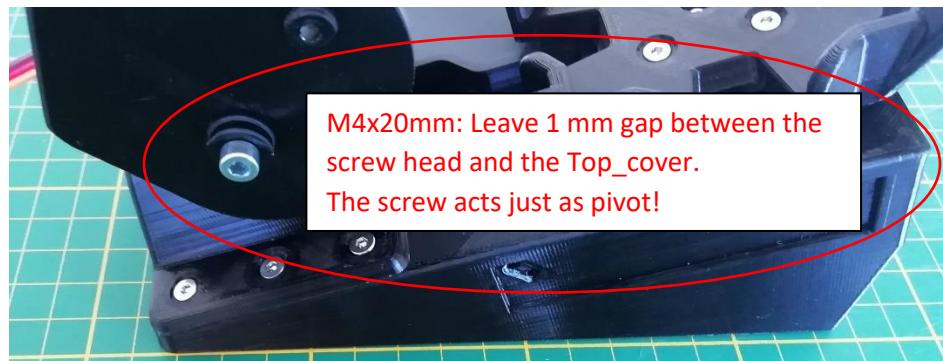
Section3: 3D print and assembly

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 constrain the screw (it should have $\varnothing 4.1$ to $\varnothing 4.3$ mm)

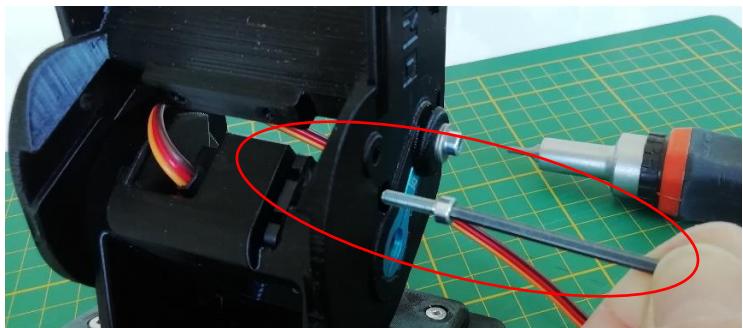


Do not fully tighten the two M3x12mm, until also the third screw is positioned.

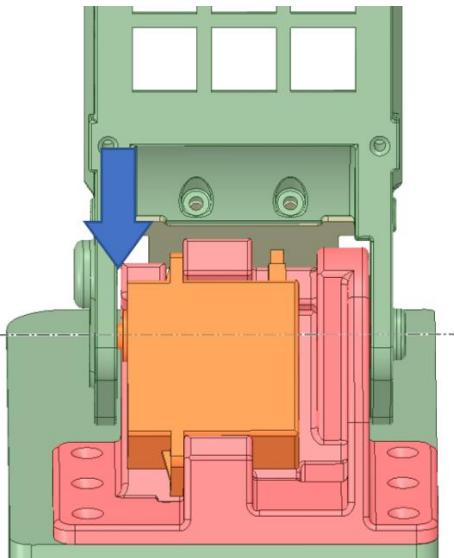
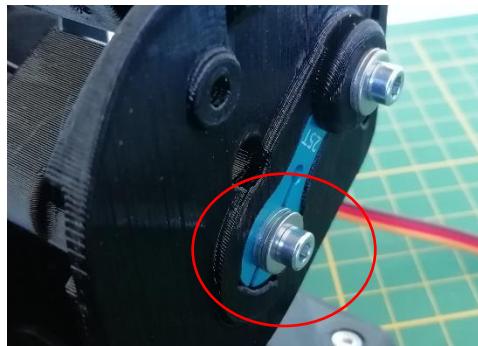


Step12 (Complete the top servo assembly to the Hinge):

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



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



Verify the 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.

Section3: 3D print and assembly

Step13 (Assemble the Lifter to the Top_cover):

4x M3x12mm cylindrical head

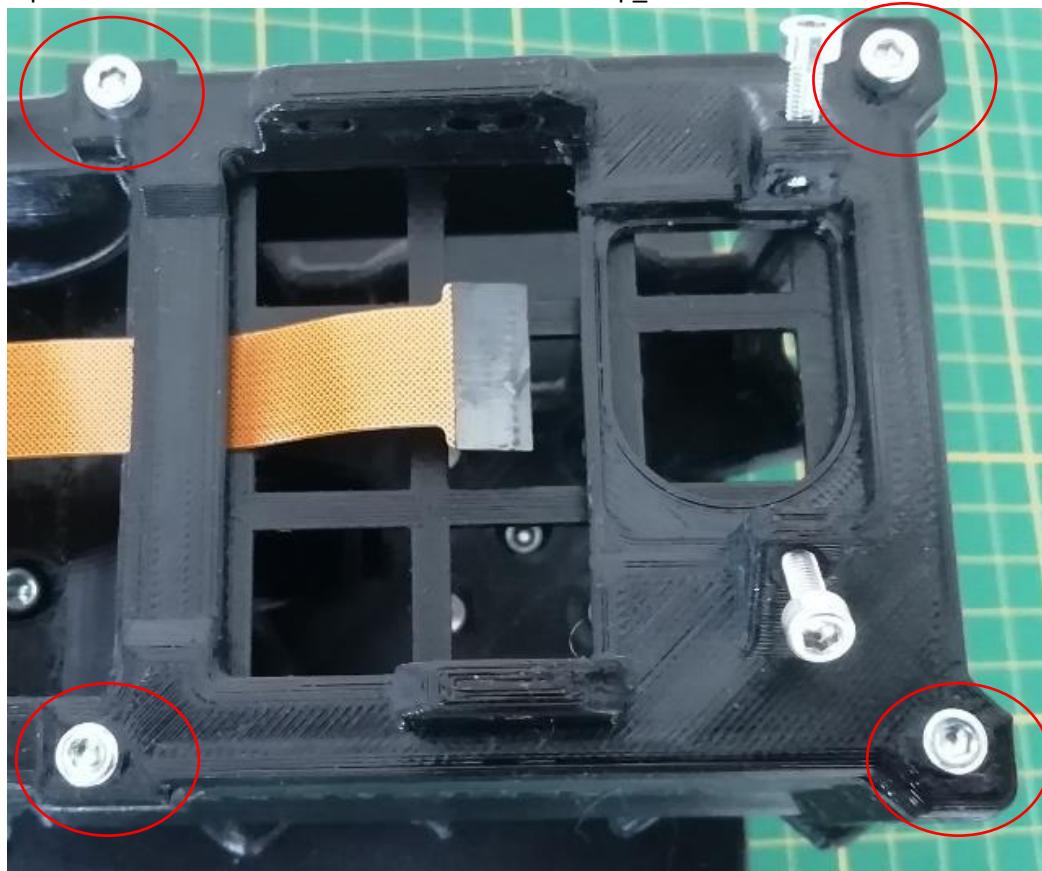
Slide the Lifter into the Top_cover slots:



Step14 (Assemble the PiCamera holder frame to the Top_cover):

4x M3x12mm cylindrical head

Squeeze the PiCamera flex cable in between the Top_cover and the PiCamera holder frame:

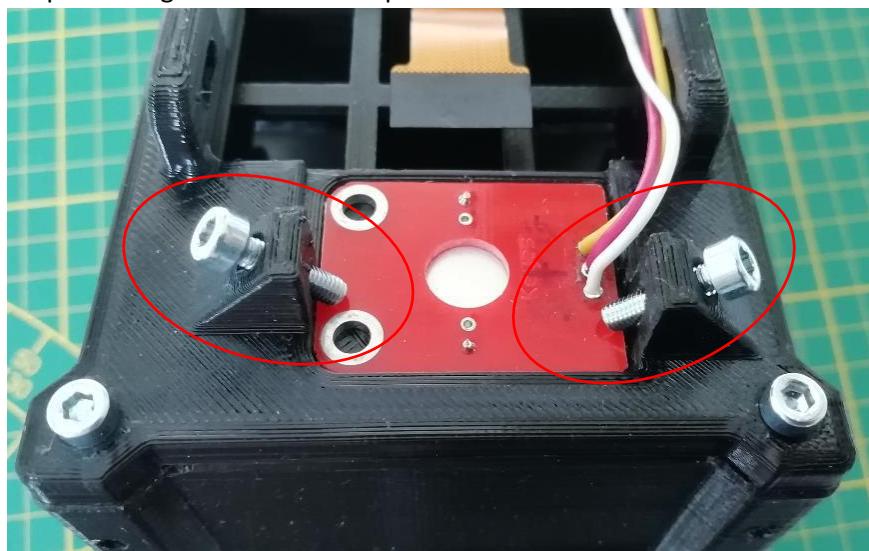


Step15 (Assemble the LED breakout board):

Add 2x M3x12mm cylindrical head.

Because of limited space, it will be convenient to solder the wires instead of the connector at the board.

Stop screwing once the screw tip touches the board:



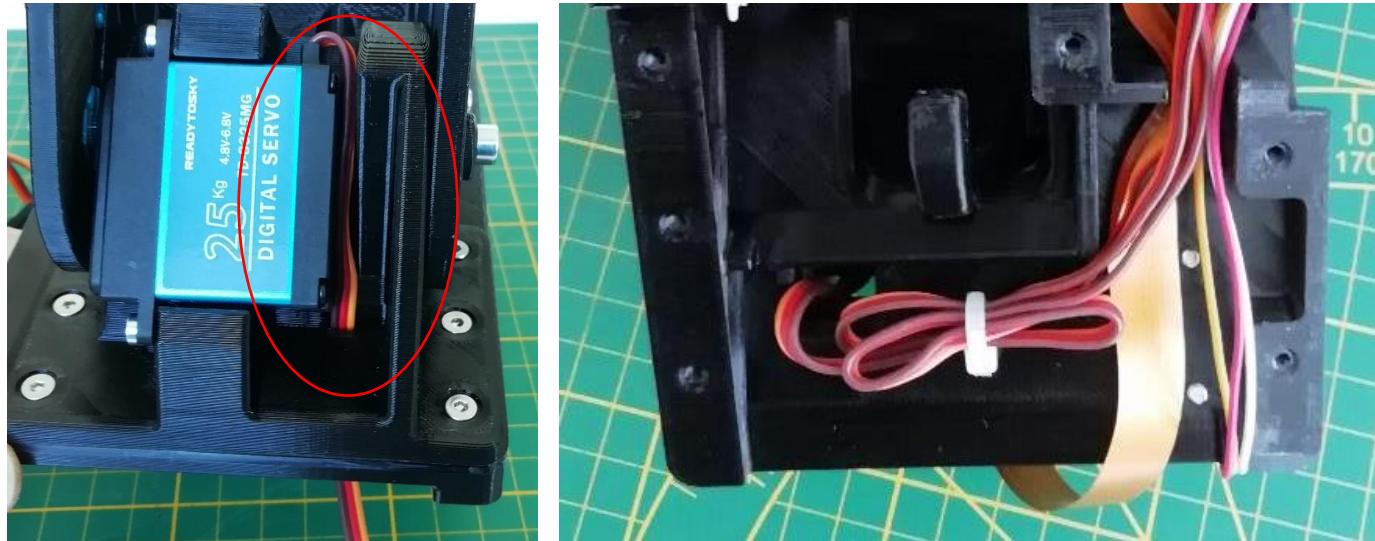
Section3: 3D print and assembly

Step17 (Position the cables, and assemble the Baseplate_rear):

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

Note: It's convenient to 'store' the excess cable length as per below picture, as there is very little space left at the board location

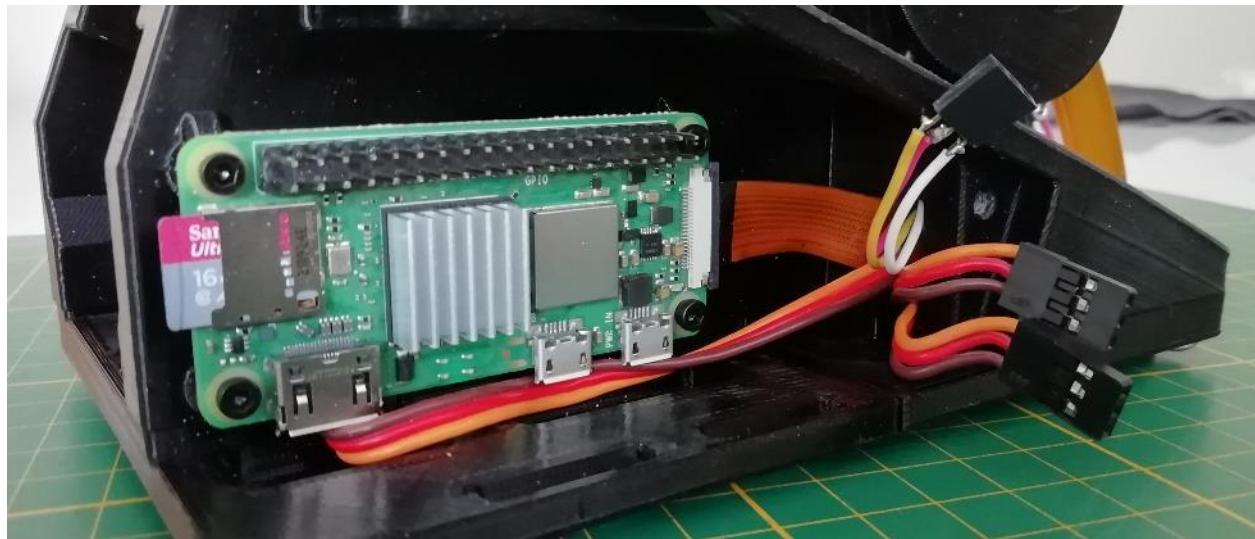
Dress the cables and close the Baseplate_rear.



Section3: 3D print and assembly

Step18 (Connect the PiCamera flex cable to Raspberry Pi Zero2 board, and fix it to the Structure)

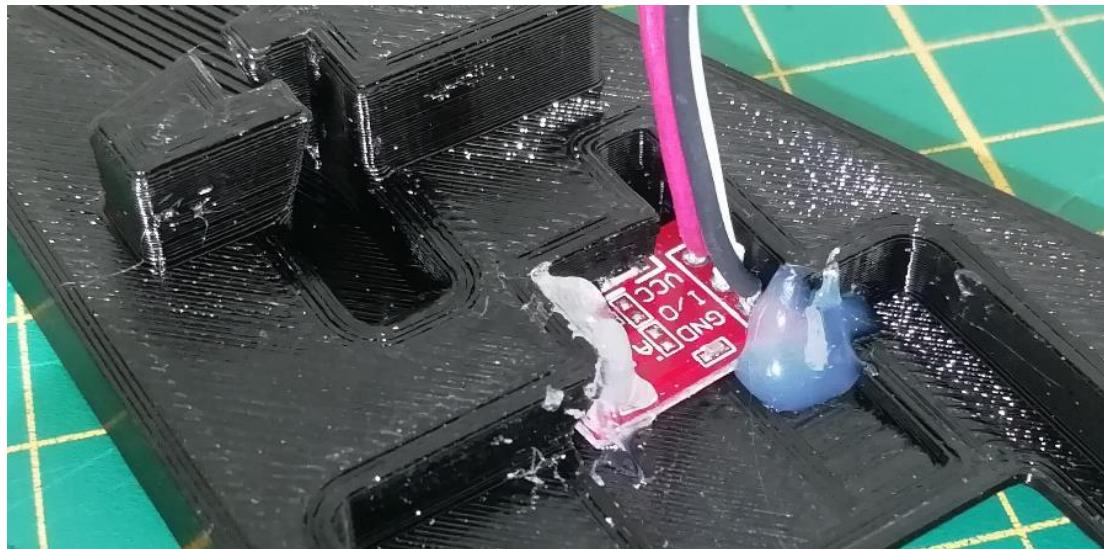
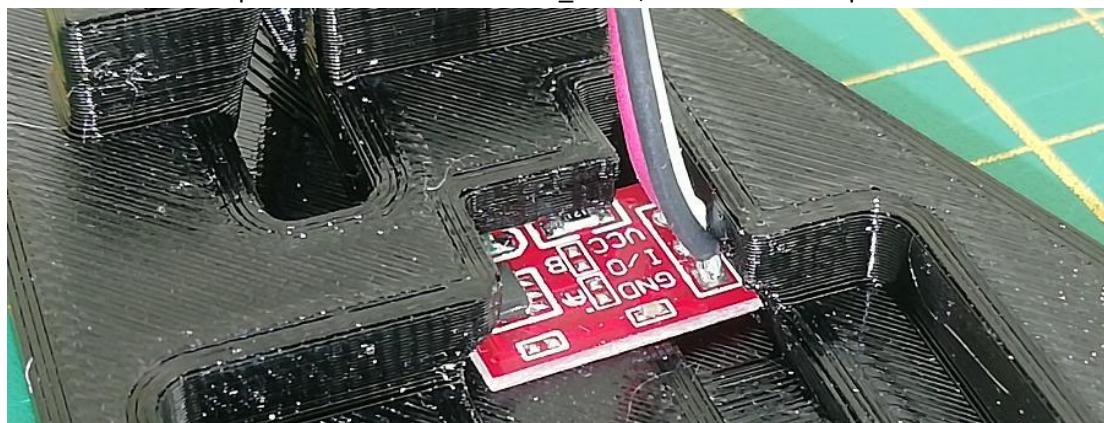
4x M2.5x4mm (or to M2.5x max 12mm) cylindrical head



Step19 (Fix the Touch_sensor to the PCB cover):

Insert the board as per picture and add a couple of hot glue droplets.

If the board centre part doesn't touch the PCB_cover, reduce the wire protrusion underneath the board.

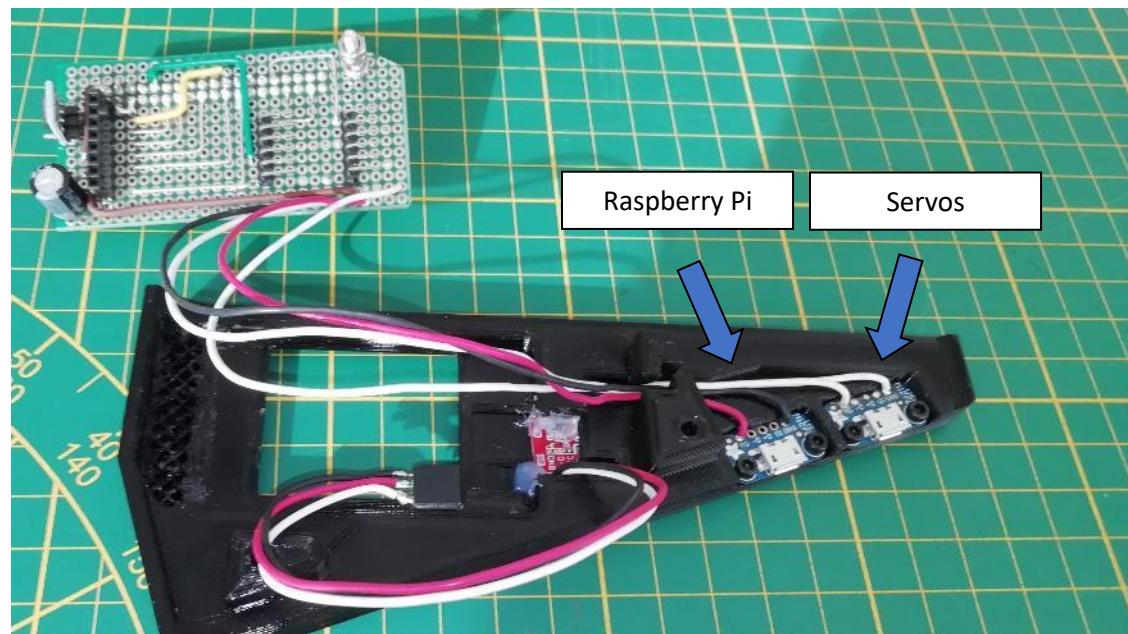


Section3: 3D print and assembly

Step20 (Fix the microUSB breakout board to the PCB_cover):

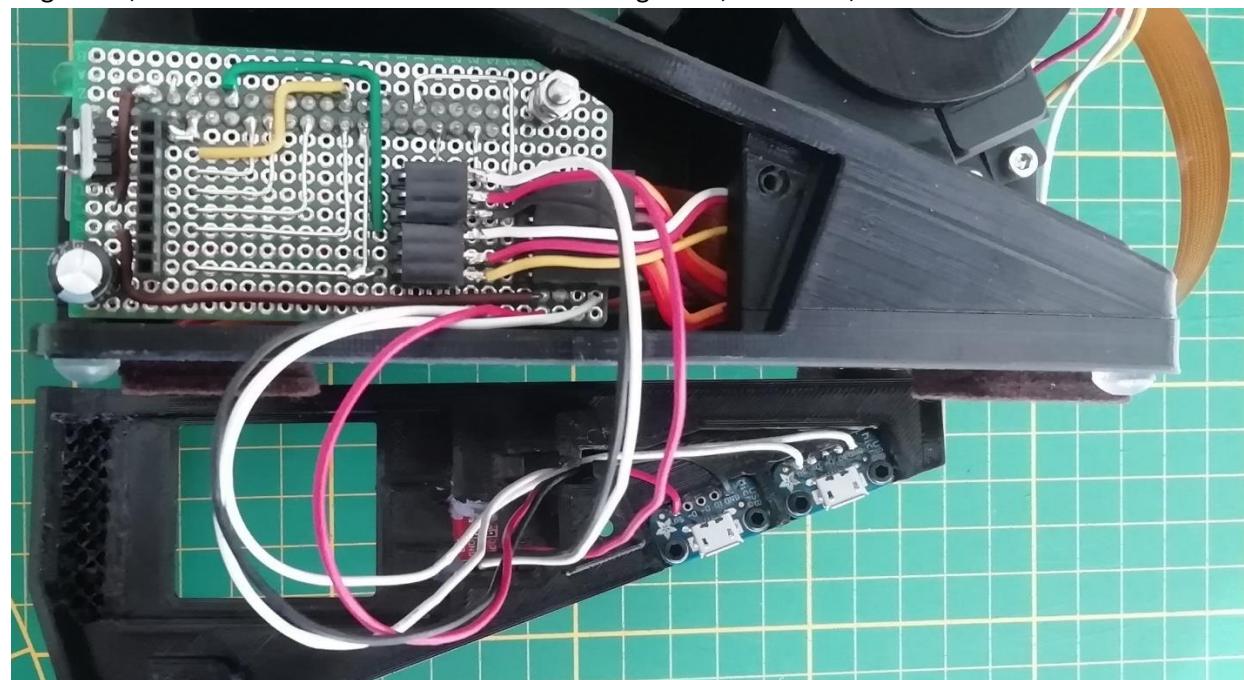
2x M2.5x4mm cylindrical head.

The order of the two boards is not critic, below just a proposal:



Step21 (Connect the servos, LED breakout board and Touch sensor):

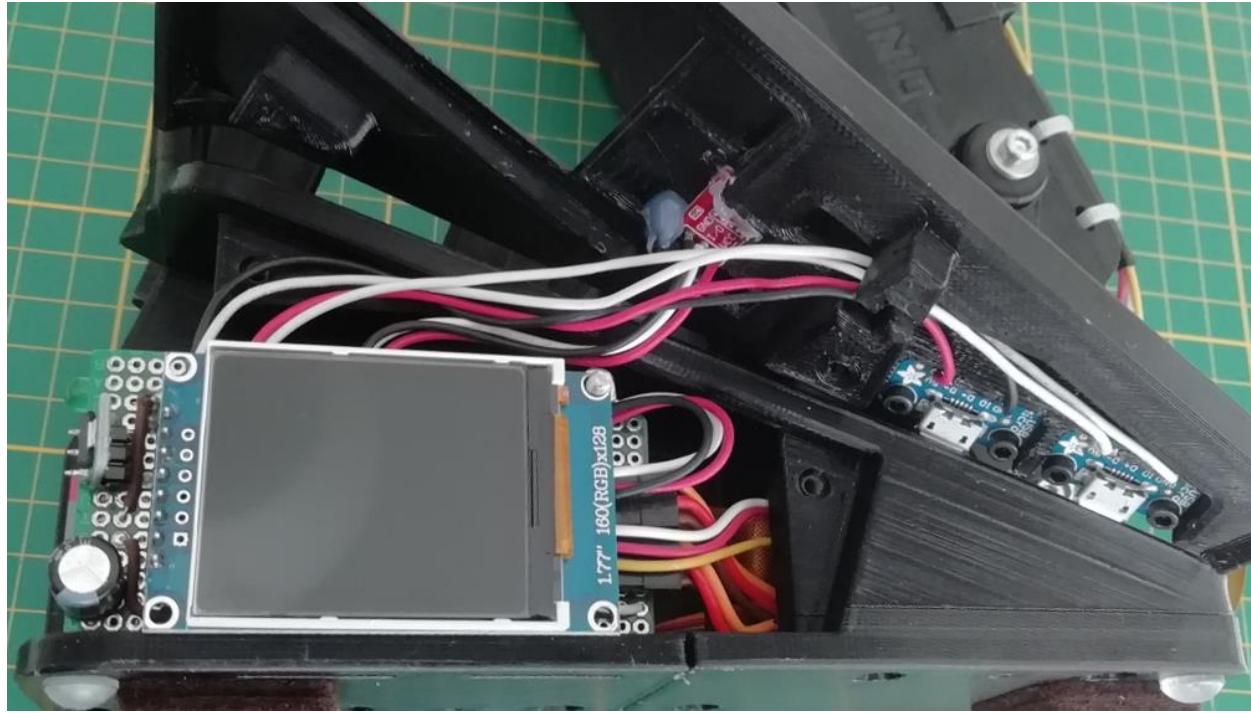
In general, the brown wire of servo connector is the ground, therefore, to be oriented toward the bottom



Section3: 3D print and assembly

Step22 (dress the cable and connect the display):

Cables at the microUSB breakout boards must be well positioned into the groove:



Step23 (Assemble the PCB_cover):

2x M3x12mm conical head

Attention to don't squeeze cables against the Structure:

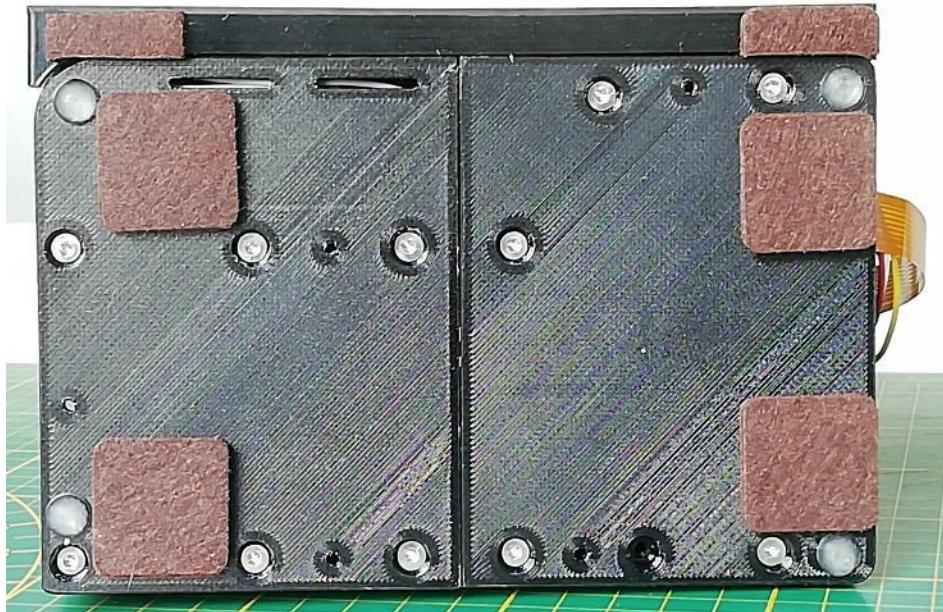


Section3: 3D print and assembly

Step24 (Assemble the Baseplate_front to the Structure):

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

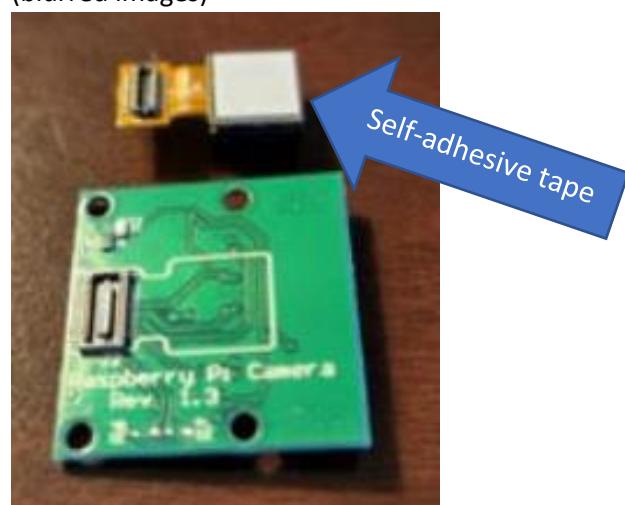
Stick self-adhesive rubber feet (or felt pads) to the baseplate; Those at the back should be placed as close as possible to the corners:



Step25 (Stick the PiCamera to the board):

Underneath the PiCamera there is a little piece of self-adhesive tape.

Make use of that tape to secure the camera to the board, to prevent the camera oscillating on its flexible cable (blurred images)



Section3: 3D print and assembly

Step26 (Assemble the PiCamera module to the PiCamera holder):

Necessary some little pieces of filament Ø1.75mm.

Force 4 pieces into the holder, slide the board over, deform the filament with hot blade.



Alternatively, 4x M2x3mm screws can be used or 4x M2.5x4mm screws can be used.

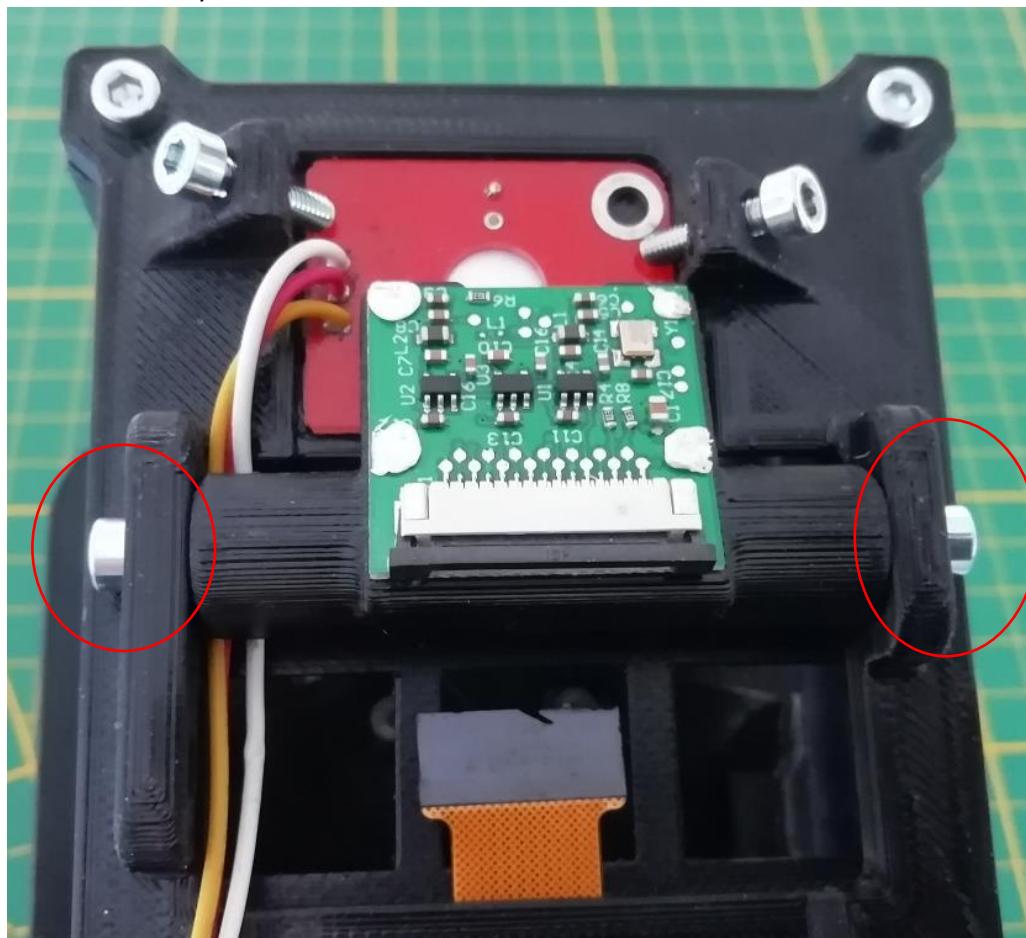
In this case, first self-thread the slots via the screw without the camera.



Section3: 3D print and assembly

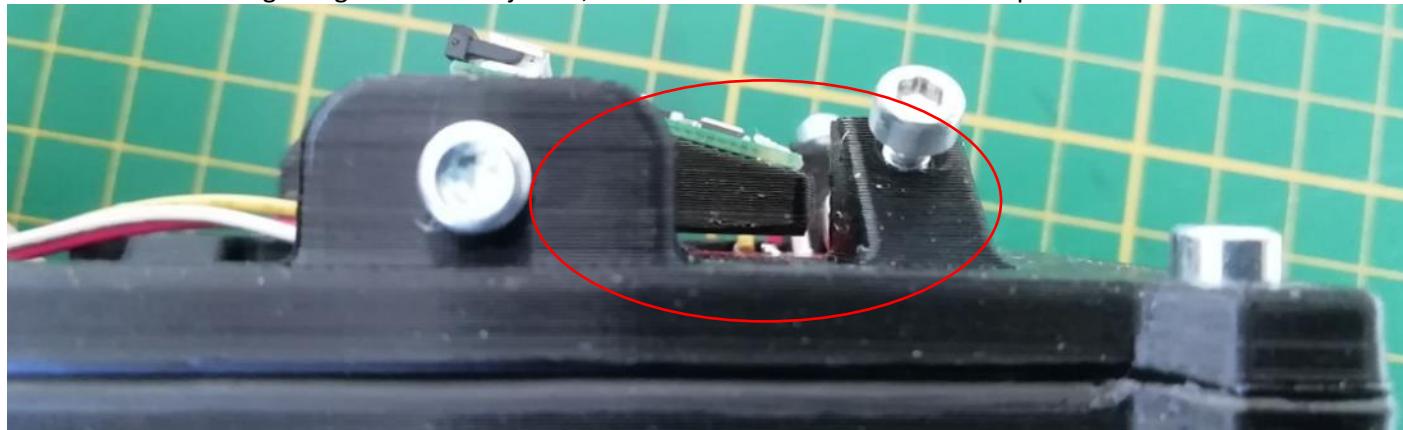
Step27 (Assemble the PiCamera holder to the PiCamera holder frame):

2x M3x12mm cylindrical head

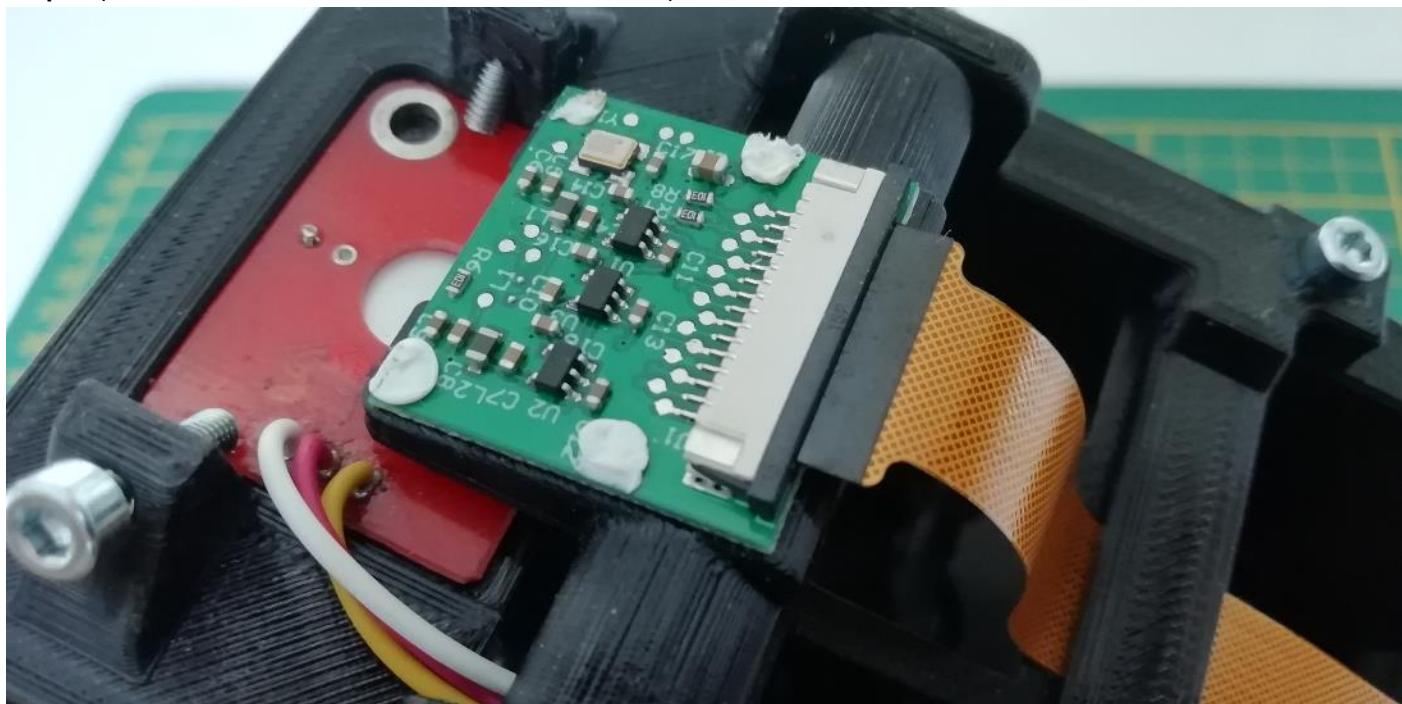


Keep the PiCamera holder parallel lo the Top_cover, while tightening the screws:

Be considered this angle might be later adjusted, to orient the camera to the cube top face:



Step28 (Connect the flex cable to the PiCamera module):



Step29 (Personalize the formal Stop plate):

2x M3x12mm conical head

The Structure has a recess, meant to hold a metal plate working as a Stop touch sensor on the Base version.

On this version, the stop function is available on the PCB_cover, making that recess quite useless and unesthetic.

I made available two alternative stl files of a plate to 3D printed as cover such a recess; One plate is neutral, the second one has 'Magic CUBE' words engraved.

Of course, you might consider using the neutral one as baseline to personalize your own Cubotino 😊.

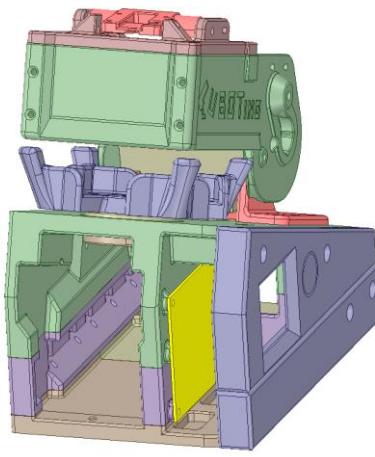
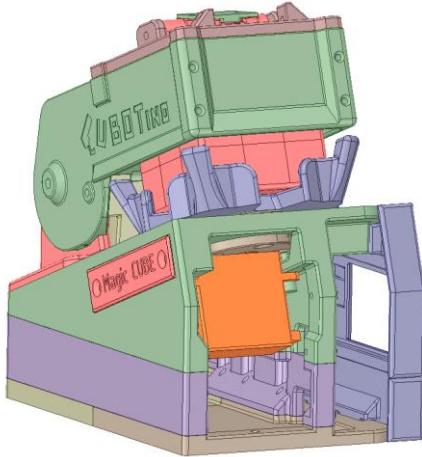
13. Raspberry Pi 2, 3 or 4

Cubotino_Autonomous version has been designed in late winter 2021, by considering the Raspberry Pi Zero 2 an easy to source and relatively cheap (ca 15\$) component; Unfortunately, the market situation has changed: Raspberry Pi Zero2 boards are scarcely available and are gold priced.

If you have a Rpi 2b / 3b / 4b, a possible solution is to increase the robot height by 26mm to embed that larger SBC. Some notes:

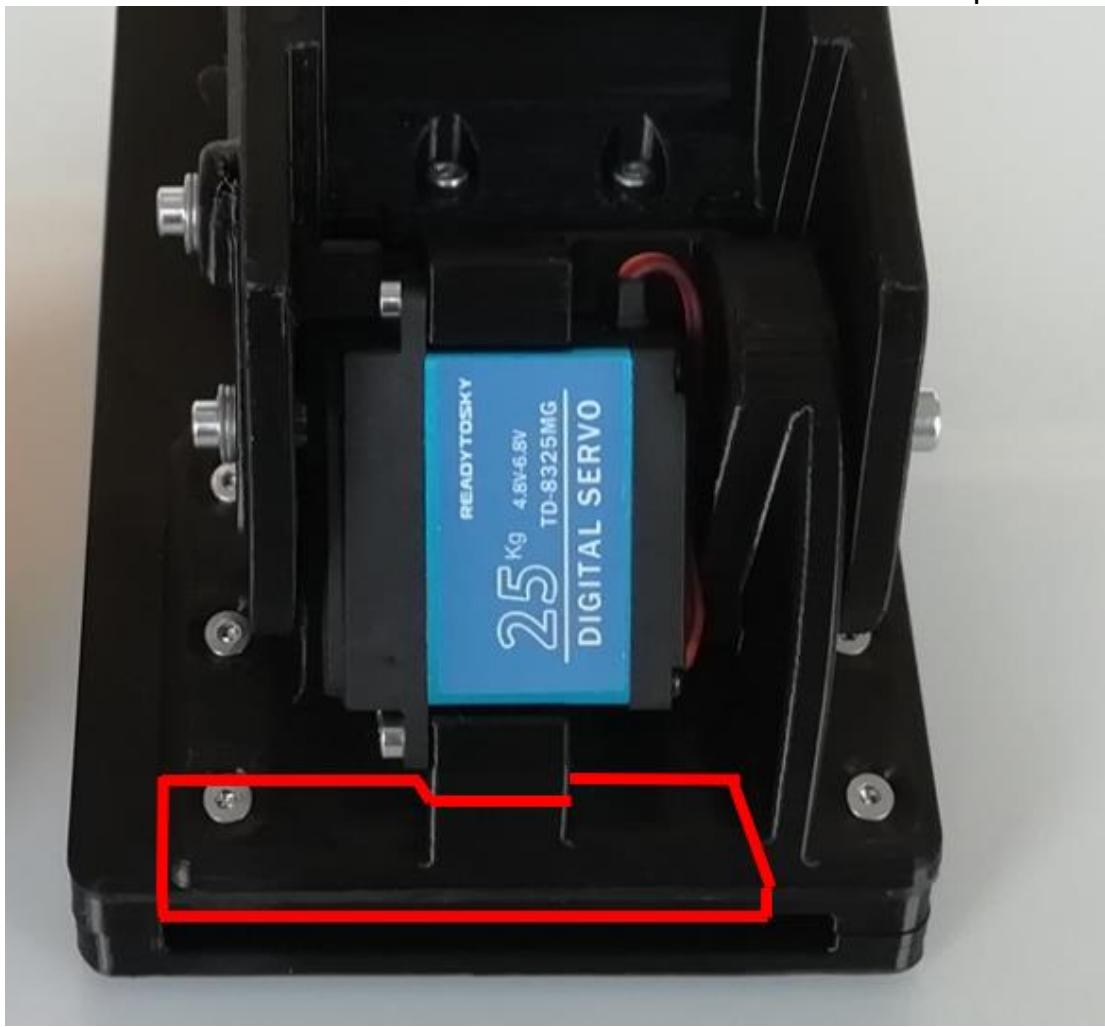
- The Connections_board need to be slightly smaller, to prevent interference with some of the raspberry Pi 3 or 4 connectors.
- Thicker wires are needed for these SBCs, especially for Raspberry Pi 4b; See Connections_board chapter.
- A 3A power supply is needed (preferably the Raspberry Pi original one for model 4b)
- The height increment can be gained via 3 extension parts to be 3D printed and screwed to the other parts; This choice allows to shrink the robot in future, in case the smaller boards will be reasonably priced again.
- This approach has been already suggested with Rev. 2.2 on 18/06/2022, yet at that moment I hadn't any "large" Raspberry Pi board available, and I did not realize the interferences with the boards; On that period, I had very little time to work things out, and decided to remove the proposal in less than a week.

Below a few images of how the robot will look like (And a picture from Maker **nrdlrd**):



Notes:

1. Raspberry Pi 2 / 3 and 4 have a Low Voltage detection circuitry; In case you experience erratic behaviour or Low Voltage Throttle, check the notes I wrote at the Connection-board and Troubleshooting chapters.
2. Raspberry Pi 2 / 3 and 4 have "standard" CSI camera, meaning the flex cable differs from the one at supply list for Raspberry Pi Zero (and Zero2).
3. Raspberry Pi 3 and 4 have the CSI camera port in a less convenient position than Raspberry Pi Zero: This makes the 30cm flex cable just enough, meaning it might be just not enough....
 - The obvious option is to buy a longer cable, meaning the commercial 450mm or 500mm.
 - In case you already have a 30cm cable that is just enough, and you'd like to reduce cable tension, you might consider modifying a couple of parts: Structure and Hinge



Removing that material is very laborious if you file it or cut it off, differently you might consider revising the files and re-print.

The Hinge currently has 6 fixing screws, considering some screws might be less effective; The removal of one screw is clearly not a problem.

As said, the suggested solution is to buy a longer cable (400 or 450mm), yet this might be a creative solution in case.

Section3: 3D print and assembly

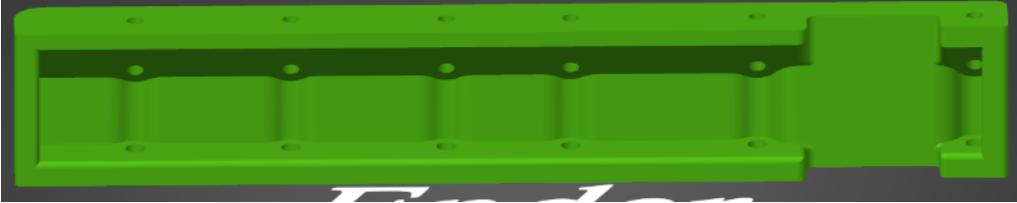
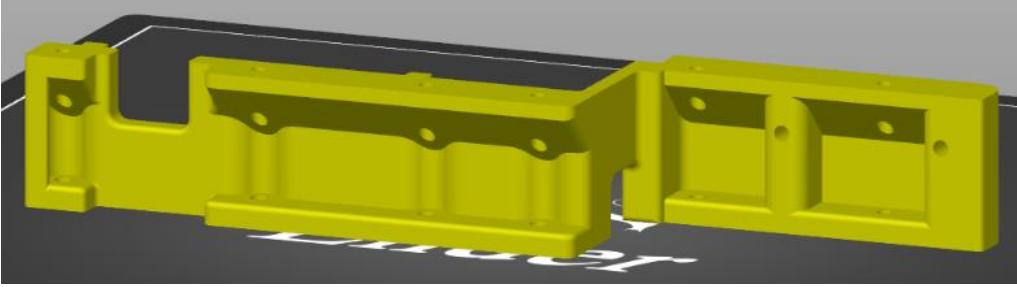
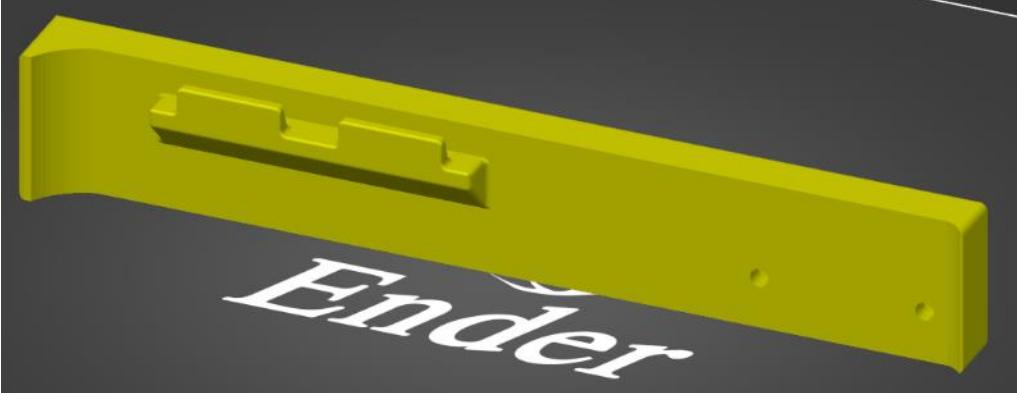
Stl files for the Extensions are (back) available since 22/10/2022 at Instructables and GitHub:

Filament quantity, and printing time ...

Ref	Part	Filament		Printing time	Version specific parts
		Meters	Grams		
1	Extension_left	10.7	31.5	3h40m	Top_version Rpi 2b or 3b or 4b
2	Extension_middle	10.8	32	3h50m	Top_version Rpi 2b or 3b or 4b
3	Extension_right	11.7	34.6	3h20	Top_version Rpi 2b or 3b or 4b
	TOTAL	33m	98g	10h50m	

Notes:

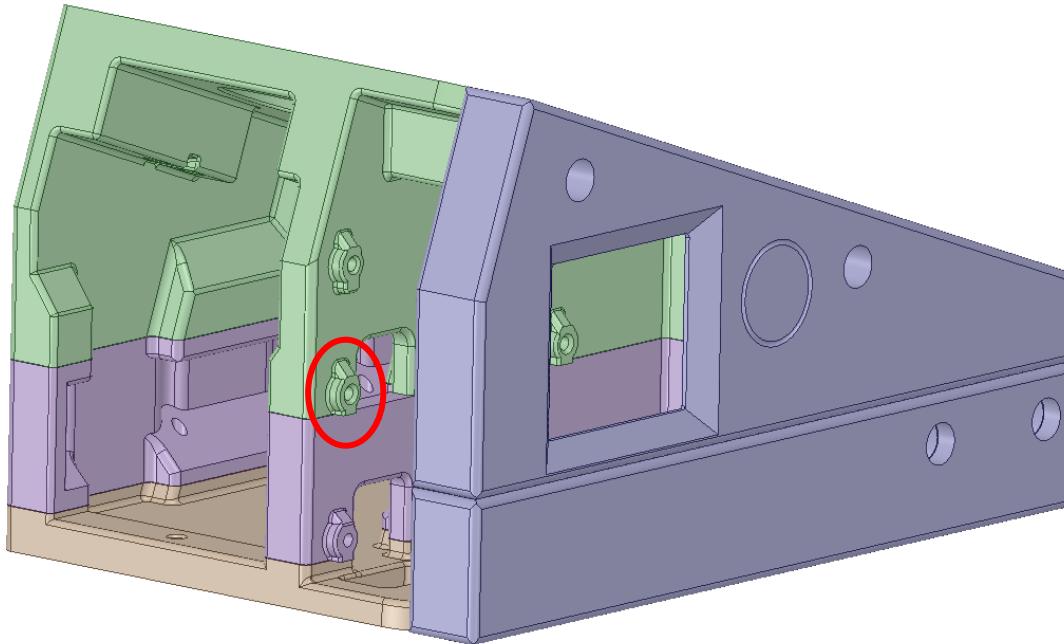
1. Still valid the notes on chapter 3D printed parts (i.e. no support needed).
2. Above Filament consumption and printing time are based on a quality printing setting; If these parts are really meant to be temporary in your case, a much lower quality could be considered.
3. The suggested part orientation for the 3D print is showed on below Table.

Part name	3D print orientation
Extension_left	
Extension_middle	
Extension_right	

Before starting the assembly, grind off a couple of millimetres from the protrusion highlighted below.

That space is needed for the microSD reader of Raspberry Pi 2b or 3b or 4b.

This won't be a problem to use a Raspberry Pi Zero 2 in future, as it can be assembled with three screws instead of 4, or by adding a spacer (plastic washer) to compensate for the removed material.



Assembly:

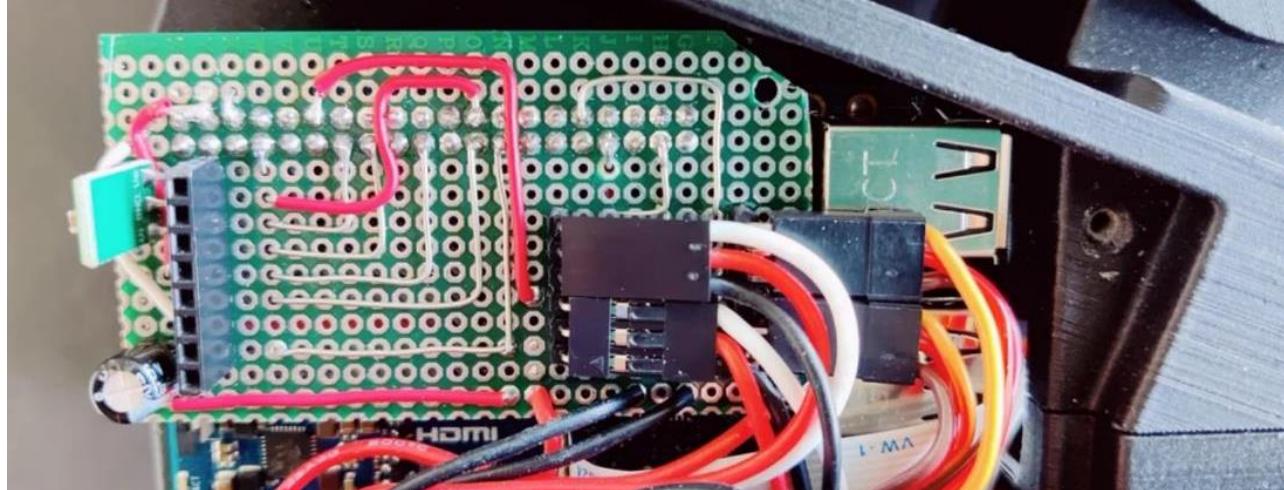
- The three additional parts are connected by screws.
- Use M3x12mm cylindrical, to connect the Extension_left and the Extension_middle toward the Structure.
Note: In case you don't have Allen keys with "spherical" head, to reach the screw under an angle, it might be necessary to use screws with conical head; M3 screws with conical head uses smaller Allen key, that can be used through the holes straight in front.
- Use M3x12mm conical head, to connect the Extension_right toward the Extension_middle.
- Raspberry Pi 2b or 3b or 4b must be assembled by keeping the GPIO connector on top, like for the Raspberry Pi Zero 2.
- Do not tight much the screws of the board, as there might be small electronic parts of Rpi 3b or 3b or Rpi 4b in contact with the second unused screw seat for Rpi Zero 2.
- The bottom_servo cable must be dressed through the recess between the Extension_middle and the Base_front.
- The Extension_left and the Extension_middle parts should be assembled after Step4 (of Assembly details chapter), therefore after assembling the bottom servo.
- The Extension_right must be assembled before assembling the Base_front and before assembling the PCB_cover_display.

Adapting the Connections_board, to fit Raspberry Pi 2 (or 3 or 4) component layouts.

If you make the Connections_board, out of a proto board, the important changes are:

- limit the protrusion of the proto board, to the right side, to max 4 / 4.5 holes from the last used pin of the Raspberry Pi GPIO connector.
- Solder the C3 capacitor directly to pins 1 and 2 of the (H6) header for the bottom servo.

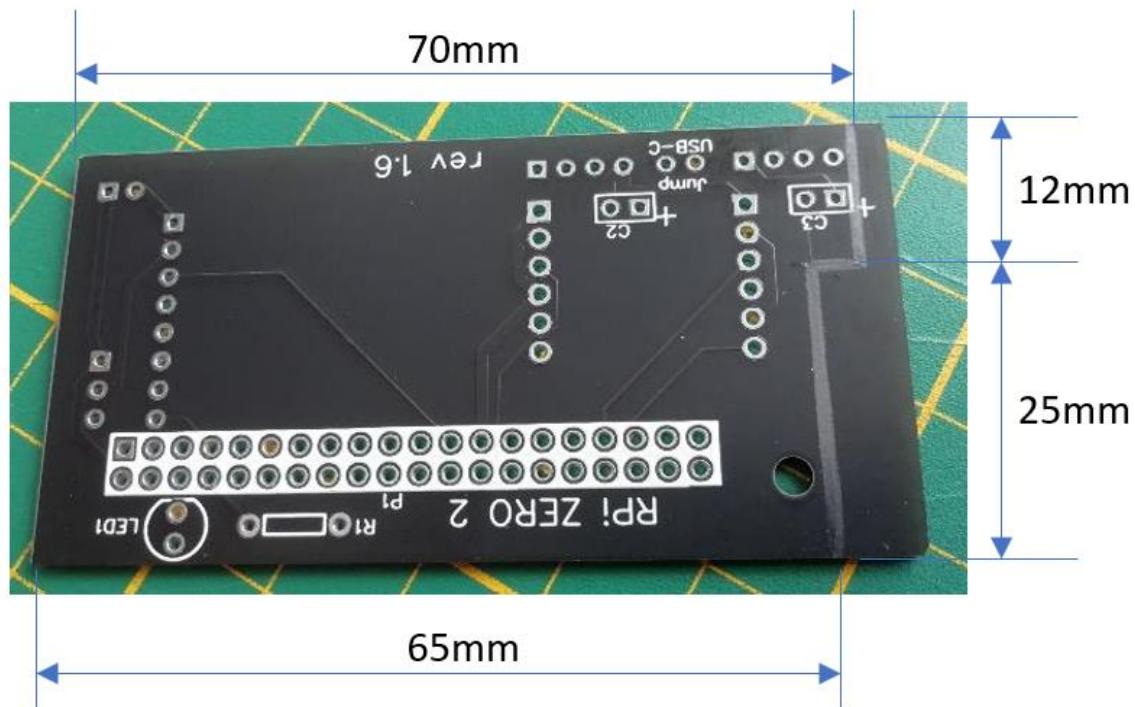
Below an image from Maker **nrldrd**, who had used a Raspberry Pi 3:



The V1.6 board can also be used, in combination with Raspberry Pi 3 or 4, by cutting out part of the right side.

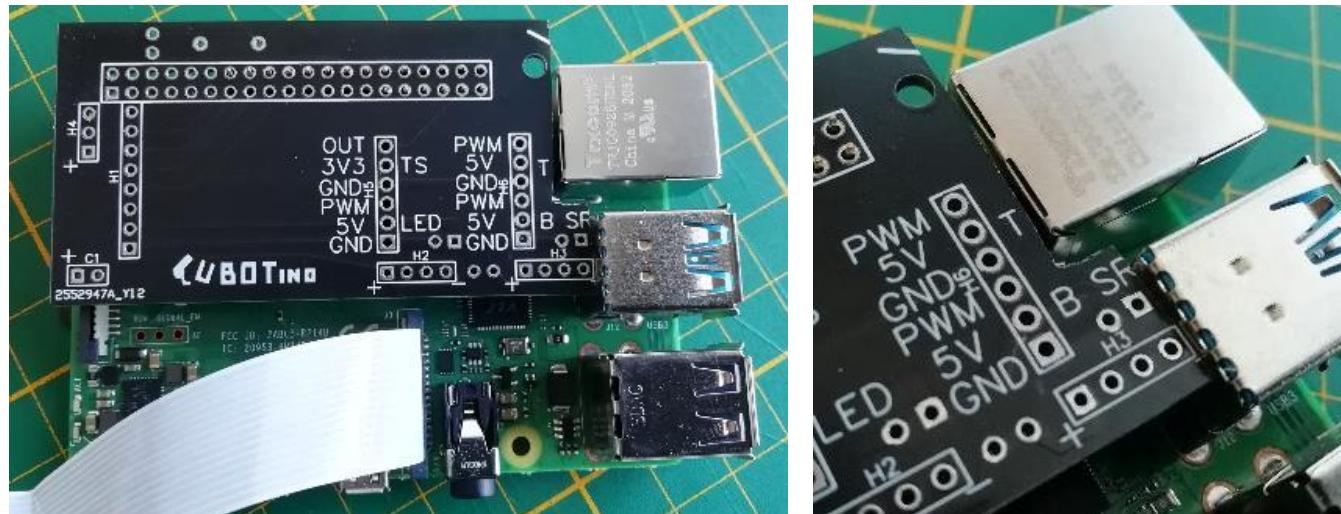
In case of **Rpi 2b or 3b** the board can have a straight cut, to reduce the board length from 74mm to ~70mm.

In case of **Raspberry Pi 4b** the board requires a slightly more complex cut:



Section3: 3D print and assembly

V1.6 Connections_board cut to make it fit with Raspberry Pi 4b:



Notes

- Solder the C3 capacitor directly to pins 1 and 2 of the (H6) header for the bottom servo.
- With reference to the PiCamera flexible cable, I'm not fully sure the 30cm version to be sufficient.
- The M3x16mm screw, meant to keep the display parallel to the Connections_board and the PCB_cover, need to be fixed to the display hole instead of the Connections_board hole.
- Verify the absence of short circuits between the Connections_board and the Raspberry Pi 2b or 3b or 4b connectors, in particular for the C3 positive solder pad.
- Thicker wires are needed for these SBCs, especially for Raspberry Pi 4b; See Connections_board chapter.

14. Tuning

As anticipated, be prepared the robot won't magically work right after assembling it: **Tuning is needed!**

This has to do with differences between each robot, in particular:

- Servos.
- arm positioning to the servo.
- cube dimensions.
- print quality.
- Assembly.

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

In case things are getting too difficult, check out the Instructables chats (there are chances other people have asked the same questions); Differently, consider dropping specific questions to that chat.

1. General:

There are parameters that are expected to be differently tuned on each robot.

These parameters are grouped into two (Json) text files: See Parameters and settings chapters.

Some of those parameters are quite likely to require tuning, because each robot will slightly differ from others:

- a) Servo angles, and servo timers
- b) Frame Cropping, as Top_cover angle dependent and PiCamera assembly angle dependent

Other parameters in the Json files, aren't so likely to be tuned, but it might be something you'd like play with 😊.

2. Setting servos angles:

The servos at supplies list have 180° of rotation, that is more than sufficient for the lifter and Top_cover angle of this robot, and it should be right sufficient to the Cube_holder; I don't suggest buying 270° servo as this will affect the angle resolution.

Apart from tolerances between different servos, one variation source is the connections between the servo arms, and the servo's outlet gear, having many possible positions (I believe there are 25 teeth).

This means the reference angles set on Cubotino_T_servo_settings.txt working fine on my robot, are not necessarily the best choice on other systems: **These parameters must be tuned on each system!**

Servos are controlled on angle, via a PWM signal (https://en.wikipedia.org/wiki/Servo_control)

The servos at supplies list, accept a Pulse Width signal from 1ms to 2ms, wherein 1.5ms is the mid angle.

It is anyhow possible to use servos with different pulse width range, in that case adjust the min_pulse_width and max_pulse_width parameters for the related servo, see Parameters and setting for more info.

The Cubotino_T_servos.py uses gpiozero library to manage the servo PWM.

This library uses a target servo position/angle with a parameter ranging from -1 to 1 (0 is the mid angle, value is a float), based on the Pulse Width range (i.e. from 1 to 2 ms, or from 500 to 2500us).

See the specific "Tune servos via GUI" chapter.

Detailed info on servo management:

1. On Parameters and settings chapter are listed the involved variables and the default values.
2. The gpiozero library is used to control the servos, with a (float) parameter ranging from -1 to 1.
3. The float value entered on the ‘–set’ argument (see Servos test and set to mid position chapter), represents a normalized rotation; The applied rotation is based on the Pulse Width range. Be considered your servos might have a different Pulse Width range...
4. Value -1 is the max CCW servo rotation; The library sends the minimum Pulse Width to the servo.
5. Value 1 is the max CW servo rotation; The library sends the maximum Pulse Width to the servo.
6. **CW and CCW notations used in this document are from the servo point of view; When you stand in front of the servo it will be the other way around.**
7. Changing from a smaller value to a larger one it results to a CW rotation of the servo outlet.
8. On servos with a Pulse Width ranging from 1 to 2ms, every 0.02 step in the “–set” argument determines a rotation of 1.8 degrees: -0.02 rotates the servo outlet of 1.8deg CCW, while 0.02 rotates the servo outlet of 1.8deg CW.
9. It is strongly suggested checking the servo rotation range before the assembly, therefore prior to have mechanical constrains on the servo rotation.
10. In case your servos don’t make 180 degrees rotation, when the ‘–set’ parameter is changed from -1 to 1 (or the other way around), it might be the case those servos have a Pulse Width ranging from 0.5ms to 2.5ms (default range considered is from 1 to 2ms).

In this case it is necessary to change the minimum and maximum pulse width parameter at the Cubotino_T_servo_settings.txt file (t_min_pulse_with, t_max_pulse_with, b_min_pulse_with, b_max_pulse_with) with values that best fit your servos.

Save the text file and re-launch the Cubotino_T_servo.py (pulse width parameters are uploaded when this script is started)

Re-check the servo angle range: If 180 degrees are now covered set the servos to their mid position.

11. In case the servo for the Cube_holder makes less than 180° rotation, despite you’ve tried to enlarge the Pulse Width range, it is necessary to increase the rotation range to slightly more than 180° 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).

2k2 Ω resistors were used by Jonesee (reference <https://www.thingiverse.com/make:1034575>).

Andy (search for G7UHN at <https://www.instructables.com/CUBOTino-a-Small-Simple-3D-Printed-Inexpensive-Rub/#discuss>) has reported ‘my servos had only about 90° range of motion out of the box (despite being advertised as 180° servos) so I had to modify them, adding 3k resistors to both ends of the internal (5k) potentiometer’.

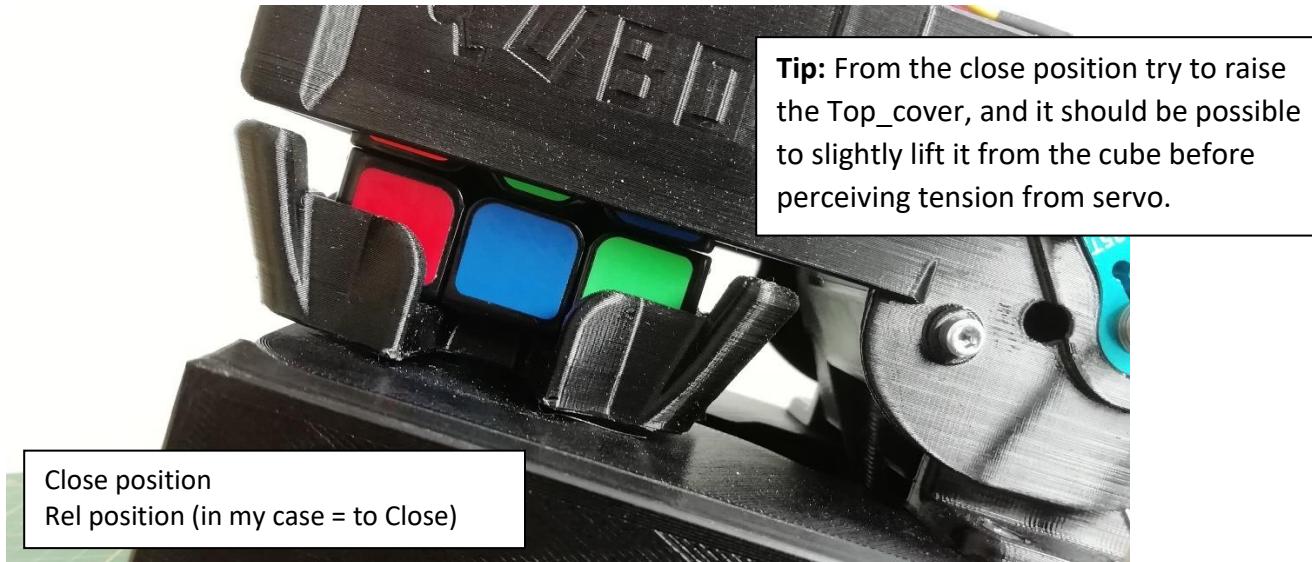
Section4: Tuning and robot operation

Top_cover (t_servo) angles:

Working angles for the servos, are set in a (Json) text file: Cubotino_T_servo_settings.txt.

There are 5 defined angles (most of time mentioned as positions):

- a. Close: position to constrain the top and mid cube layers.
- b. Rel: position to release tension, from cube, at Close position.
- c. Open: position without interferences with the cube and Cube_holder.
- d. Read: position for camera reading, with the Lifter almost touching the cube (and unfortunately constraining the Cube_holder).
- e. Flip: position for the Lifter to flip the cube (about 2 cube layers height).

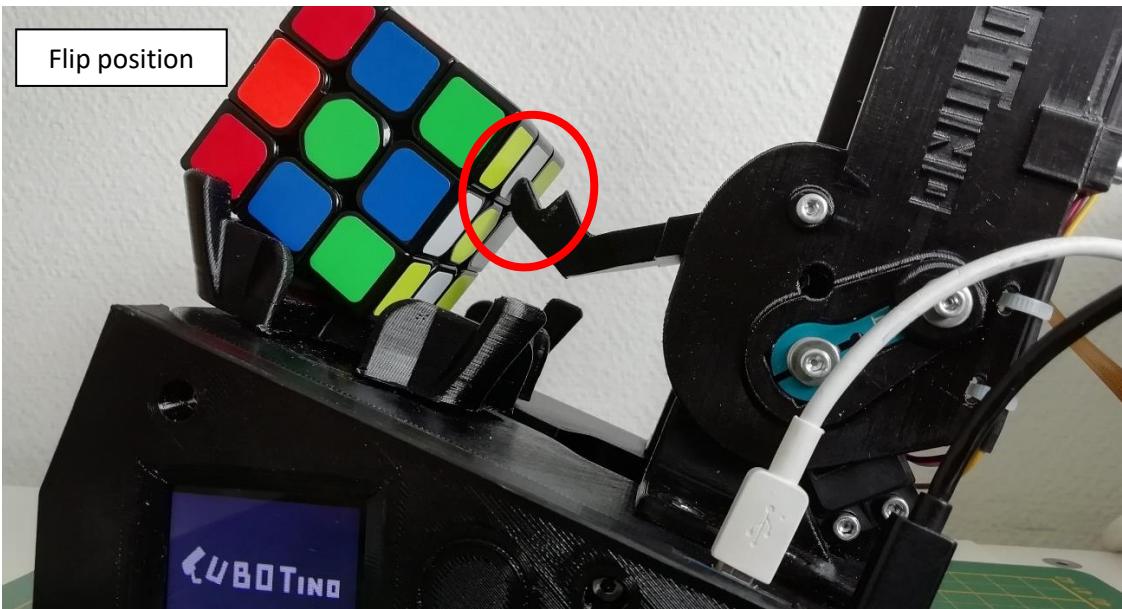


Section4: Tuning and robot operation

Read position



Flip position

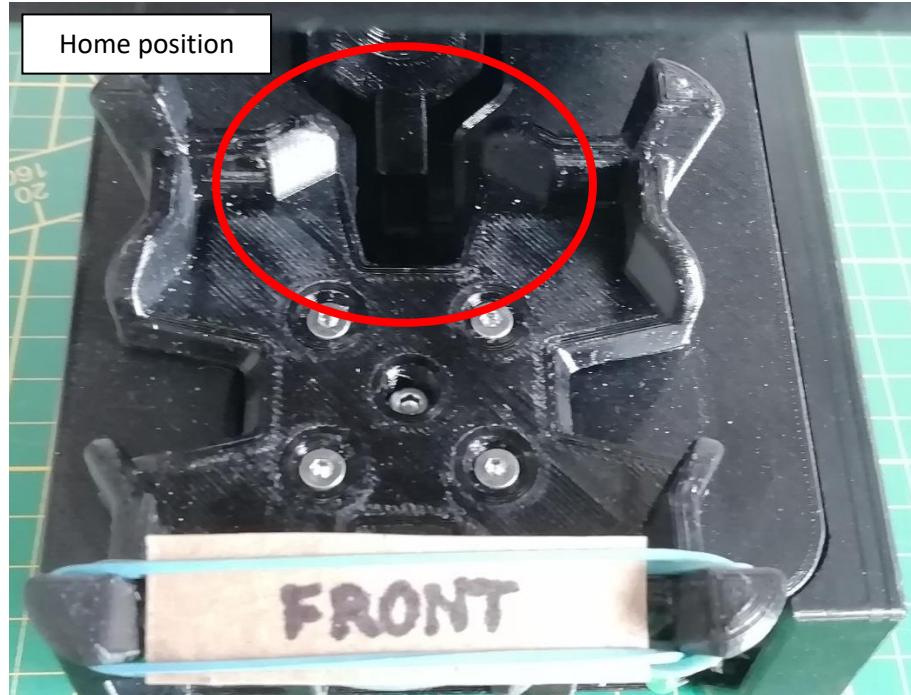


Cube_holder (b_servo) angles:

There are 7 defined angles (most of time mentioned as positions):

Position	Description
Home	mid position between CCW and CW
CCW	position to spin or rotate the Cube_holder ca 90° CCW from Home (Direction according to the motor point of view)
CW	position to spin or rotate the Cube_holder ca 90° CW from Home (Direction according to the motor point of view)
Release_CW	release cube tension at CW (called <i>b_extra sides</i> before)
Release_CCW	release cube tension at CCW (called <i>b_extra sides</i> before)
Extra_home_CCW	release cube tension at home, when rotating from CCW (called <i>b_extra home</i> before)
Extra_home_CW	release cube tension at home, when rotating from CW (called <i>b_extra home</i> before)

The Home position must be well centred.



To centre the Home position, it might be necessary to adjust the below parameters:

- *b_min_pulse_width*
- *b_max_pulse_width*
- *b_home*

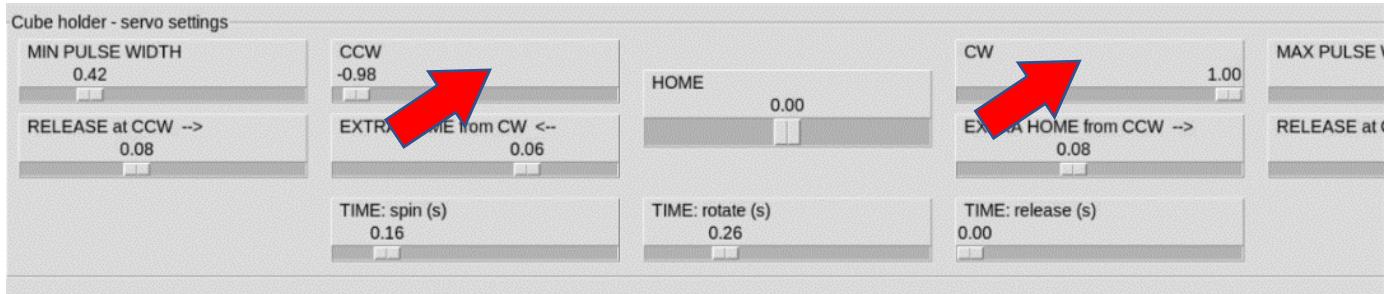
Section4: Tuning and robot operation

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

This is needed 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.



When using the GUI to tune the servos, the CCW and CW positions (without the tension release) are visible by pressing the slider background, at the Cube holder – servo settings:



Section4: Tuning and robot operation

After a Cube_holder rotation toward CCW or CW , there is a backward rotation defined by the value Release_CCW and release_CW:

- This releases tension between the cube and the Cube_holder, and between the cube and Top_cover.
- It aligns the Cube_holder for cube flipping at CCW and CW.

Note: When the Cube_holder spins toward CCW or CW (spins means the Top_cover doesn't constraints the cube), the rotation stops before making the extra rotation.

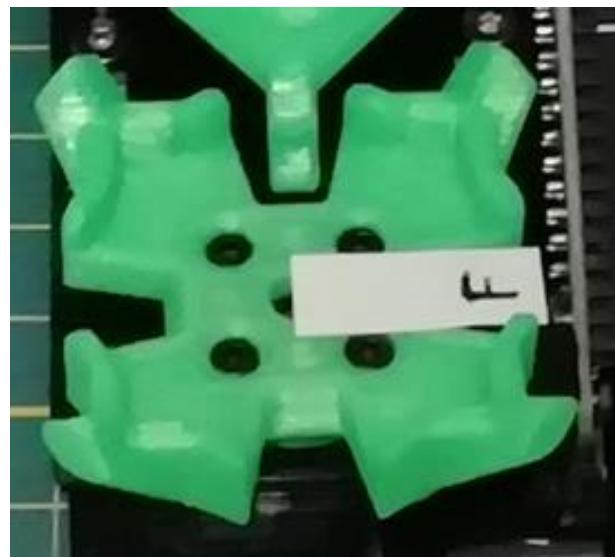
When using the GUI to tune the servos, the CCW and CW positions with the tension release are visible by pressing the CCW:



CCW position, after Release_CCW:



CW position, after Release_CW:

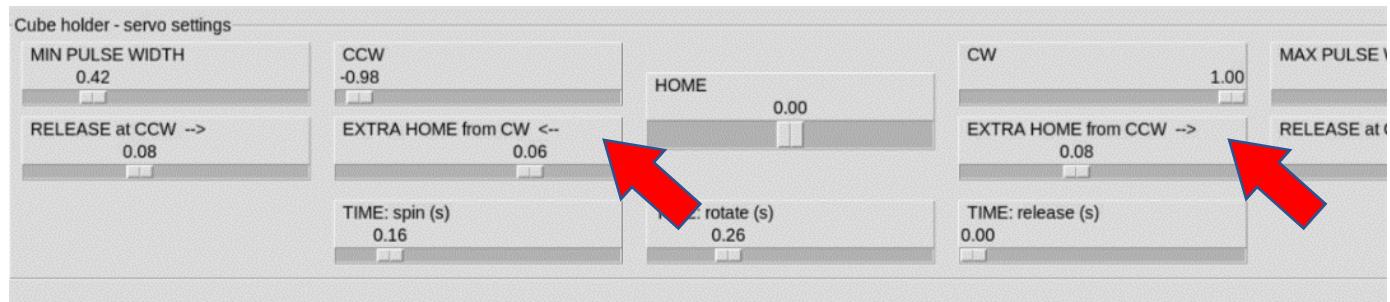


Section4: Tuning and robot operation

When the Cube_holder rotates toward Home (rotation means the Top_cover constraints the cube), there is first an extra rotation defined by the value Extra Home from CCW and Extra Home from CW; After the extra rotation, the Cube_holder rotates backward to Home position:

- This releases tension between the cube and the Cube_holder, and cube toward Top_cover.
- This aligns the Cube_holder for cube flipping at Home.

When using the GUI to tune the servos, the Extra Home CCW and Extra Home CW positions can be tested by pressing on the sliders background:



Note: On below pictures the Lifter has been raised only to help visualizing the Cube_holder angle

Extra Home CW position:



Extra Home CCW position:



Section4: Tuning and robot operation

3. Timers for servos:

Servos don't provide feedback when they have completed the requested angular rotation; It's necessary to set appropriate waiting time in the script, to allow sufficient time for the servo to complete the action.

It will be convenient to use larger delays at the beginning, and progressively reduce them once the servo angles are adjusted to your system: The default values I've set for the default should be more than sufficient to allow the servos intended rotations.

Once your robot runs smoothly, and in case you' like to reduce the solving time, then you might start reducing those timers. As reference you can check on "Parameters and settings" for the values I'm using on my robot.

Timers for the servos, are set in a (Json) text file: Cubotino_T_servo_settings.txt.

4. Frame cropping:

Cropping parameters are set in a (Json) text file: Cubotino_T_settings.txt

PiCamera position, and its Field of View (FoV), are likely to read both top and back cube faces.

Cubotino_T.py file is delivered with no cropping effect (variables set to zero): this to help the camera assembly angle to be set to get the cube top face centred: First picture below as example.

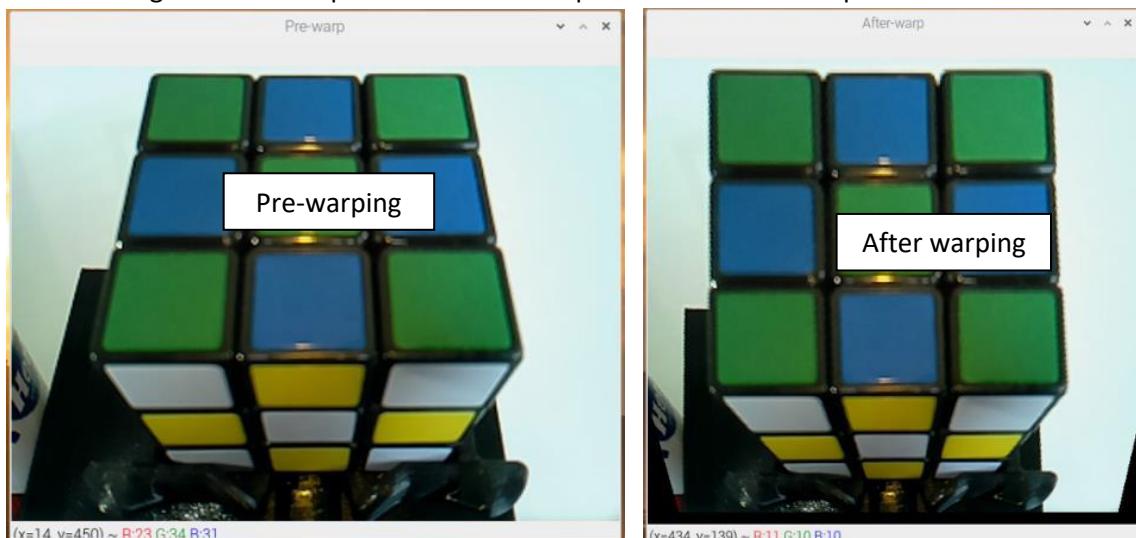
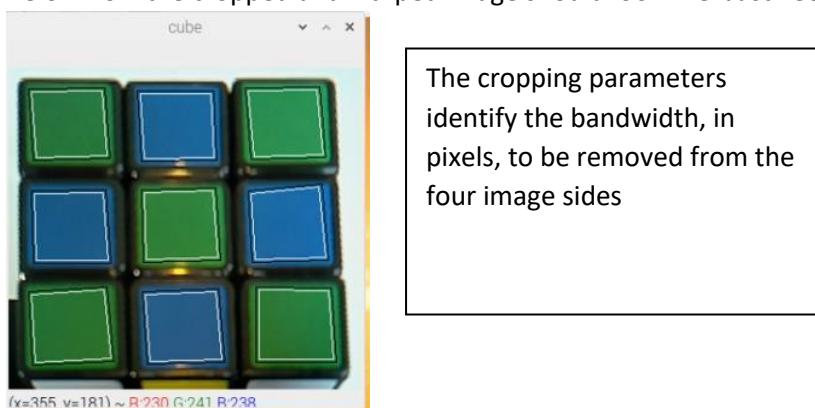


Image cropping must be tuned, to reduce the image to the region of interest (ROI).

Be noted the image cropping is made before warping it; Pictures on this page have been made by inverting these processes, to better show the potential problem.

Image warping does not prevent the risk to have some of facelets at back face, to be detected as part of the top face. Another reason to set proper cropping parameters is to reduce the image size and gaining process speed.

Below how the cropped and warped image should look like: Just keep a little part visible of the back face:



5. Frameless cube:

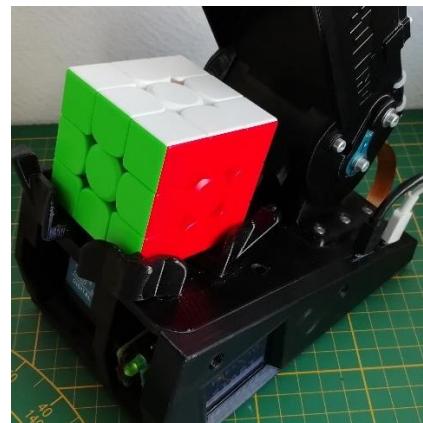
classic (with stickers)



classic (in-molded)



frameless



The cube facelets detection algorithm has been initially developed for the classic cubes, those having the black frame around the facelets.

Starting from 6th August 2022, it is also possible to use the frameless type of cubes.

At Cubotino_T_settings.txt there is a “frameless_cube” parameter, with below options:

- ‘false’ for the classic cube type (default value)
- ‘true’ for the frameless type
- ‘auto’ for both types,

The ‘auto’ mode is computationally more demanding, therefore it will take slightly longer (about 1 to two seconds more for the six cube faces).

In the case the Cubotino_T_settings.txt doesn’t have the key “frameless_cube”, because you are re-using an old setting file, then the new Cubotino_T.py file version will simply consider the classic cube type.

7. Display initialization and test:

For installation up to 6th August 2022, the display was initialized twice; This wasn't pleasant to be seen, and potentially leading to hardware race by the code.

Starting from 6th August 2022, thanks to Yannick solution, the display is initialized once, with better display management.

The new solution requires one more file: Cubotino_T_display.py to the project.

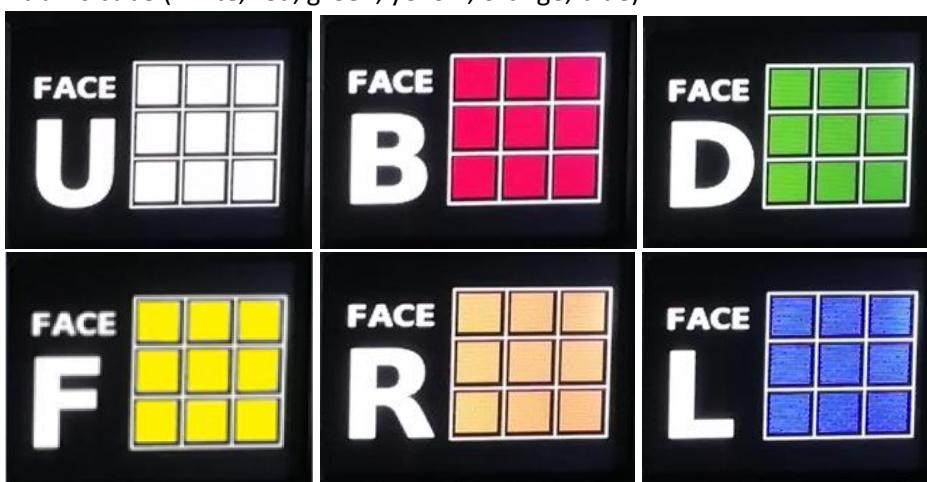
The new file can be used to test the display after its connection:

- a. enter the cube folder (`cd ~/cubotino/src`)
- b. activate the venv (`source .virtualenvs/bin/activate`); Attention to the dot in front of virtualenvs.
- c. run the script `python Cubotino_T_display`; Attention to the space in between ‘‘

For 20 seconds, on the display should alternate the Cubotino logo to “DISPLAY TEST” text placed into three rectangles:



Afterward, the display test will take additional 20 seconds to display the URF colours sequence of a western Rubik's cube (white, red, green, yellow, orange, blue).



Note: In case the display is unreadable, check for “display” at the troubleshooting: There is a fix that has worked for many of us.

8. PiCamera focus.

The V1.3 PiCamera has fix focus.

Because of the relatively short distance between the PiCamera and the cube face, it might be convenient to adjust the focus.

To verify the camera focus:

1. Connect via VNC
2. Enter home/pi/Cubotino_Ticro/src folder: `cd ~home/pi/cubotino/src`
3. Activate the venv: `source .virtualenvs/bin/activate`
4. Run the camera test: `python Cubotino_T.py --picamera_test`

The camera will be activated, and its image sent via the VNC:

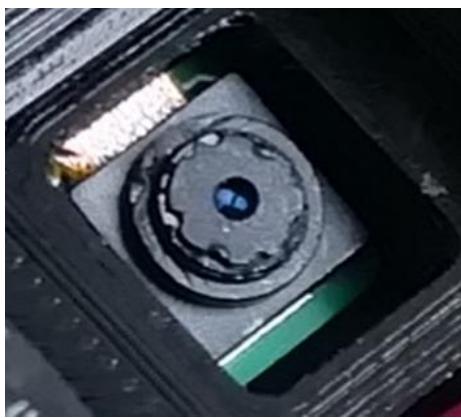
After 20 seconds the test ends, and the command can be repeated whether needed.

Check if the focus is roughly ok at 9cm distance, if not proceed with the following instructions until the focus is satisfactory.

Reference tutorial I followed <https://projects.raspberrypi.org/en/projects/infrared-bird-box/6>

I did change the focus on three PiCamera so far, and not always I find it easy, yet it seems I've now got the key learnings:

- Remove **all** the glue, all around, **before** trying to rotate the lens.
- Use a cutter with a new blade, to have a very sharp and thin tip.
- To reduce the focus distance, the lens must be rotated CCW (unscrew it).
- To prevent detaching the lens from the camera, proceed in small steps (i.e. 1/3 of revolution per time).
- Do not rotate the lens for more than one full revolution.
- To check the camera focus result, prior the robot assembly, consider the cube face will be at 80~100mm distance from the camera lens (mid-point at ~90mm).
- After correcting the focus, try to evaluate if the lens cover has enough friction, to avoid adding glue; I did not glue the lens, as there still was quite some friction, preventing the lens from getting loose.



15. Tuning via GUI

This method is an alternative to use the CLI, and it allows a more intuitive tuning process.

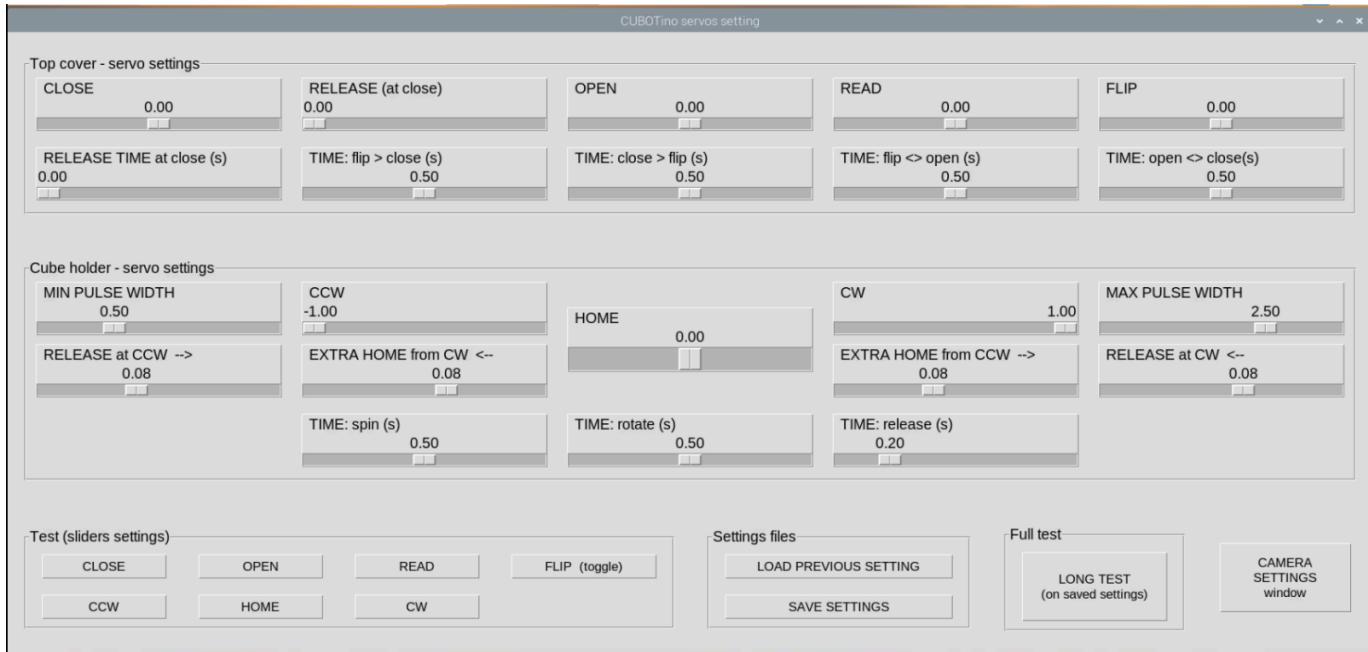
Steps before tuning the robot.

1. set the Servos to the mid angle (see Servos test and set to mid position chapter).
2. assemble the robot.
3. Increase the VNC resolution (see “Setting up Raspberry Pi, Step 10).
4. enter the cube folder (`cd cubotino/src`) and activate the venv (`source .virtualenvs/bin/activate`);
Attention to the dot in front of virtualenvs.
5. in case you have updated from a release older than 5.0:
 - a. it is necessary to run one time `python Cubotino_T_servos.py --tune`

```
pi@cubotino: ~ $ cd cubotino/src
pi@cubotino:~/cubotino/src $ source .virtualenvs/bin/activate
(.virtualenvs) pi@cubotino:~/cubotino/src $ python Cubotino_T_servos.py --tune
```
 - b. The script will add at the end a one-time statement:

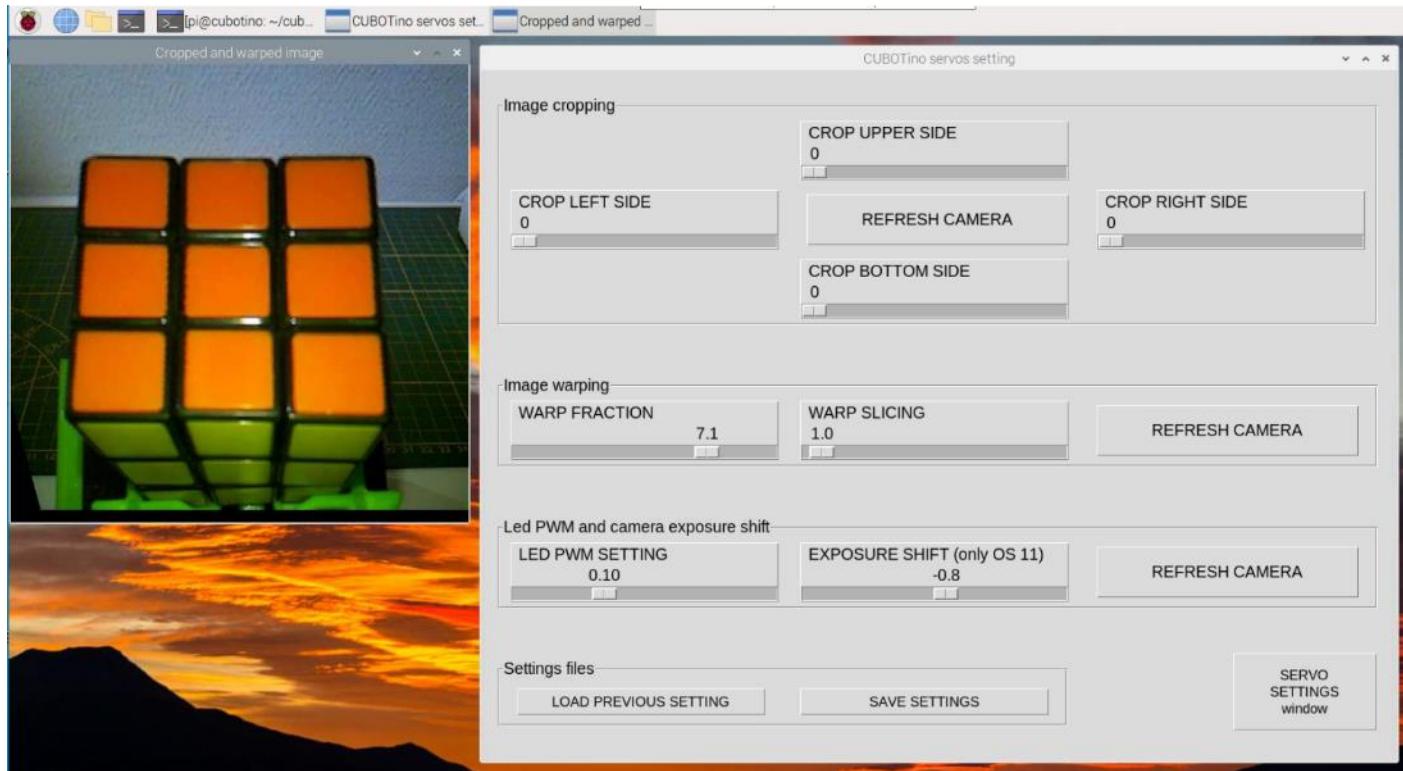

```
One time action: Adding new parameters to the Cubotino_T_settings.txt
Action necessary for compatibility with the latest downloaded Cubotino_T.py
```
 - c. The one-time action will remove the parameter “b_extra_home”; The related value will be associated to the new parameters “b_extra_home_CCW” and “b_extra_home_CW”.
 - d. The one-time action will remove the parameter “b_extra_sides”; The related value will be associated to the new parameters “b_rel_CCW” and “b_rel_CW”.
 - e. Exit the script by typing ‘q’ followed by Enter.
6. run the script `python servos_setting_GUI.py` to get the GUI up and running.
7. the GUI sliders are on the default settings, or on your last saved settings.

a first window will appear for the section related to the servos:



Section4: Tuning and robot operation

The Camera settings window can be reached via the bottom-right button:



The servos GUI section is divided in 5 areas:

Area	Description	Note
Top cover – servo settings	Sliders to set the relevant positions and timers for the upper servo (Top_cover / Lifter)	
Cube holder – servo settings	Sliders to set the relevant positions and timers for the bottom servo (Cube holder)	Area blocked at the start, and when the Top_cover is not in Close or Open positions
Test (slider settings)	Buttons to check the settings currently visible at the sliders.	The Cube_holder buttons are locked at the start, and when the Top_cover is not in Close or Open positions
Settings files	Allows to save to file the settings currently at the sliders. It allows to upload the last saved settings, and sliders get updated (save again if you'd like to keep these)	Previously saved settings are renamed by adding date and time. The last 10 files are maintained (older are deleted)
Full test	It runs a predefined sequence of movements, based on the last saved settings	Also used to check the overall time when trying to speed up the servos

Section4: Tuning and robot operation

Servos tuning high level order:

- Do not care about the timers, those are the very last thing to tweak.
- Start with the Top_servo positions, by using the sliders and the buttons.
- Maximize the Bottom_servo rotation range.
- Set the Bottom_servo positions, by using the sliders and the buttons.
- Verify a few times with the LONG TEST
- Optimize the timers 😊

Suggested setting sequence for the Top_cover:

1. A correct "Servo_axis" selection should now show a Cube_holder rather aligned with the robot main axes, at least letting the Lifter passing freely across the Cube_holder; If this is not the case, check again the Servo_axis "selection" chapter and choose a more suitable geometry.
2. Set the Top_cover: CLOSE, OPEN, READ and FLIP sliders positions (leave RELEASE to zero)
3. Keep the timers on the default values, until all the positions (also those for the bottom servo) are set and tested to working fine.

Notes:

- When you change a slider setting, the servo gets updated only when the slider is released.
 - When clicking to the slider background, the servo moves to the value of the slider.
 - Every time the servo changes position, the last command and the value are plotted to the display.
4. Press the SAVE SETTINGS button.
 5. Check the CLI for the file names (you might keep it visible right below the GUI). In your case the files will not have the _AF suffix.
 6. Close and reopen the GUI: Check if the GUI loads your last settings.



Section4: Tuning and robot operation

Notes:

- a. If the Top_cover is not in close or open positions, the bottom_servo related widgets are blocked.
- b. The bottom_servo related buttons, at Test sliders settings, mimic what the robot does:
 - o When the **Top_cover is in close position**, the positions “CCW”, “Home” and “CW” are reached by first making the extra-rotation followed a rotation back; This is needed to:
 - recover the play between the cube and the part.
 - align the cube layers.
 - release the tension.
 - centre the Cube_holder to allow Flipping at “CCW”, “Home” and “CW” positions.
 - o When the **Top_cover is in open position**, the positions “CCW”, “Home” and “CW” are without making any extra-rotation; The only target is:
 - centre the Cube_holder to allow Flipping at “CCW”, “Home” and “CW” positions.

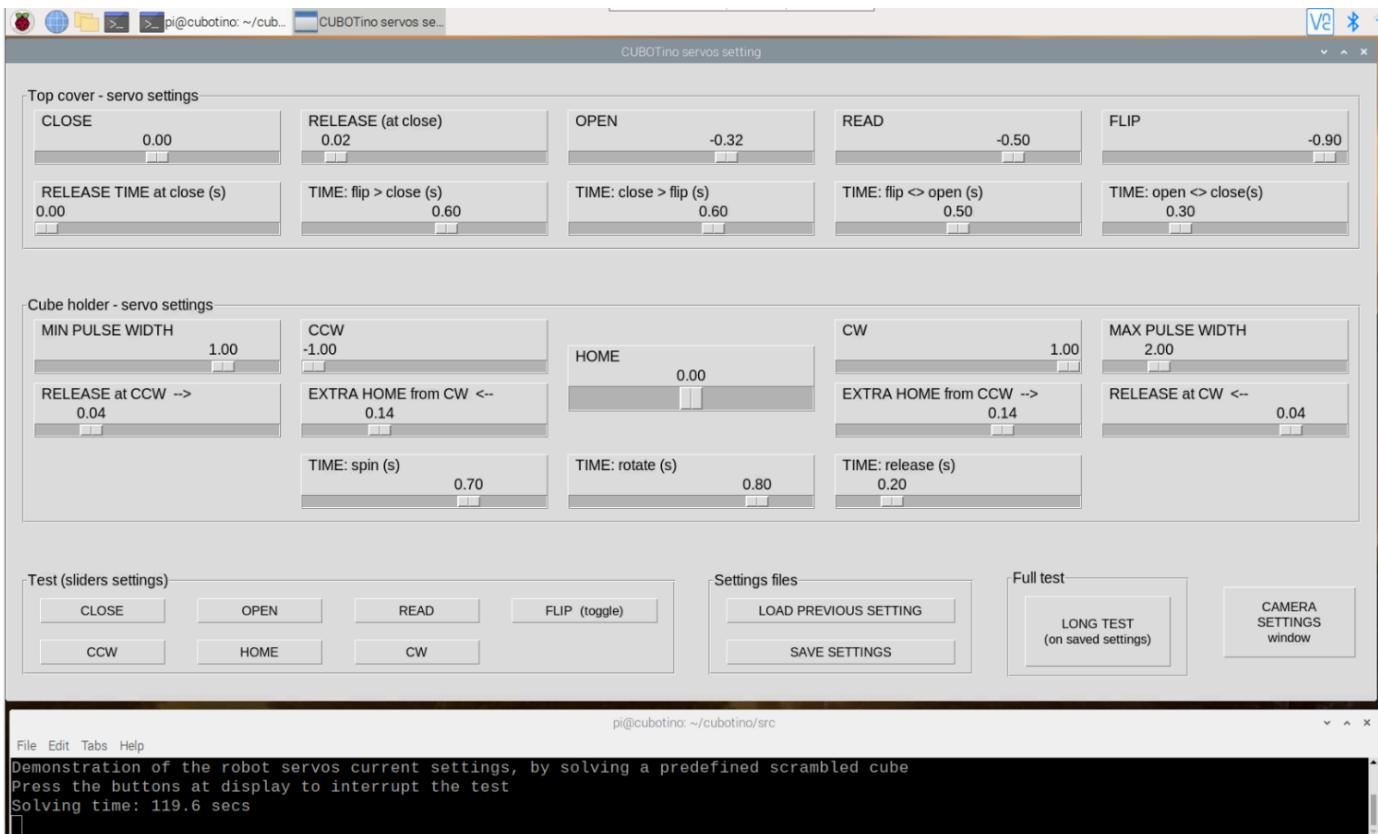
Section4: Tuning and robot operation

Suggested sequence for the Cube_holder settings:

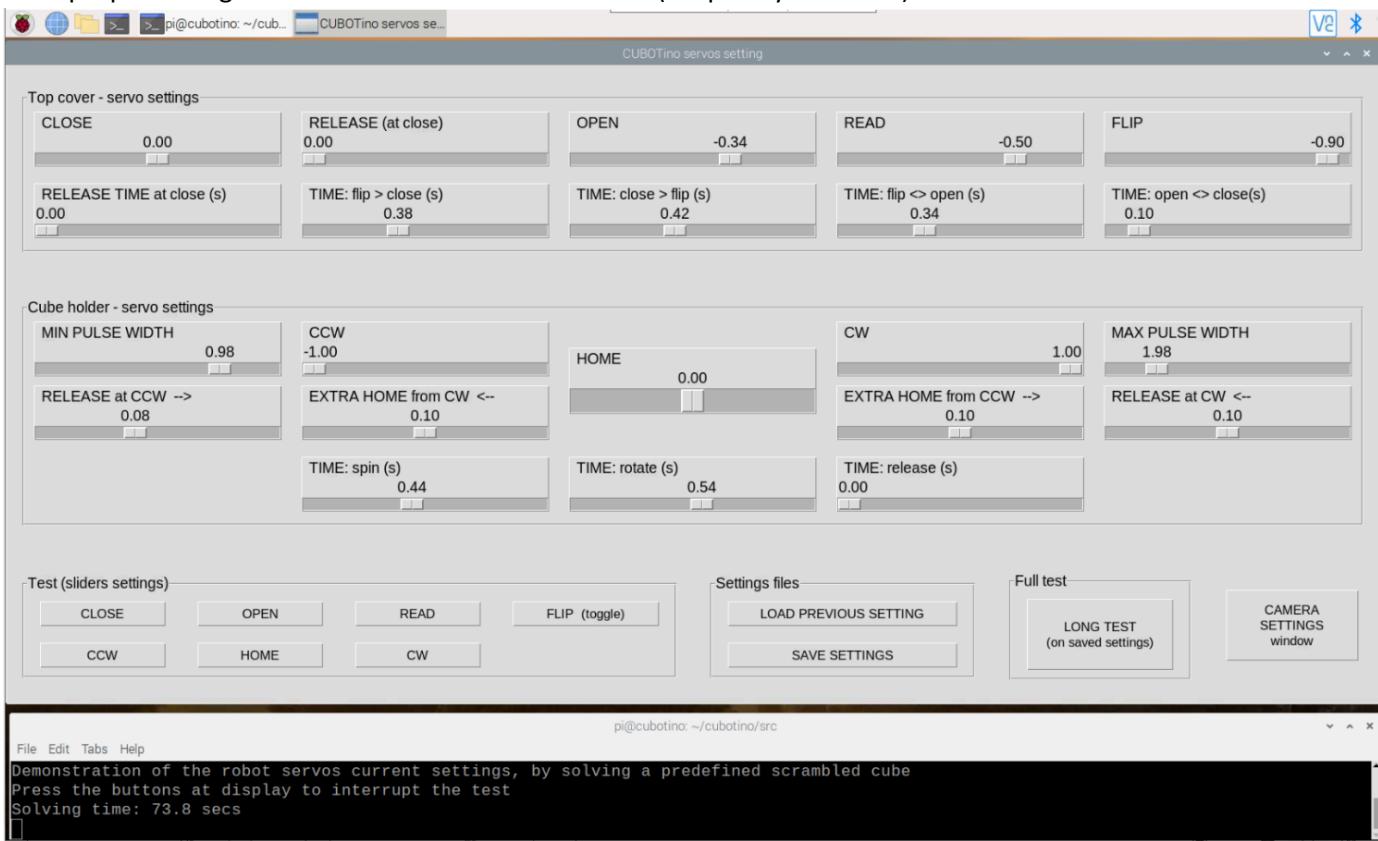
1. Press OPEN button to position the Top_cover to the open position: This activates the Cube holder – servo settings widgets.
2. Search the max possible “Min Pulse Width”:
 - a. Verify the CCW slider to be at -1.00.
 - b. Press CCW button.
 - c. Decrease the Min Pulse Width slider, with decrements of 0.02.
 - d. After every increment press Home button and again CCW button.
 - e. Repeat steps b and c until the smallest Min Pulse Width is still accepted (= Cube_holder rotates to CCW).
3. Search the max possible “Max Pulse Width”:
 - a. Verify the CW slider to be at +1.00.
 - b. Press CW button.
 - c. Increase the Max Pulse Width slider, with increments of 0.02.
 - d. After every increment press Home button and again CW button.
 - e. Repeat steps b and c until the largest Max Pulse Width is still accepted (= Cube_holder rotates to CW).
4. Once the rotation range has been maximized, set the Home position to be well centred.
5. Press SAVE SETTINGS to memorize these first settings.
6. Place the cube on the Cube_holder and press CLOSE button to constrain the cube.
7. Press CCW button and check the cube layer alignment:
 - a. If over rotation, increase the CCW slider.
 - b. If under rotation the servo has too little rotation range; park the problem for now and move on.
8. Press OPEN button, remove the cube and press Read to raise the Lifter:
 - a. Check if the Cube_holder is well aligned, to permit the Lifter passing through freely.
 - b. Increase or decrease the Release at CCW to improve this aspect.
9. Repeat the same approach (steps 6 to 7) for the CW position.
10. A very similar approach must be used for the Home and Extra home settings.
11. Press SAVE SETTINGS to memorize the settings.
12. If all the positions are reasonably ok, you can run a LONG TEST:
 - a. The LONG TEST is based on the last saved settings.
 - b. During this test the GUI is not anymore active.
 - c. Use the buttons at the robot (display) to eventually interrupt the test.
 - d. Once the LONG TEST is finished, the GUI is back active.

Section4: Tuning and robot operation

LONG TEST: With the timers set as per default, the solving test time is ca 120 seconds (Raspberry Pi ZeroW):



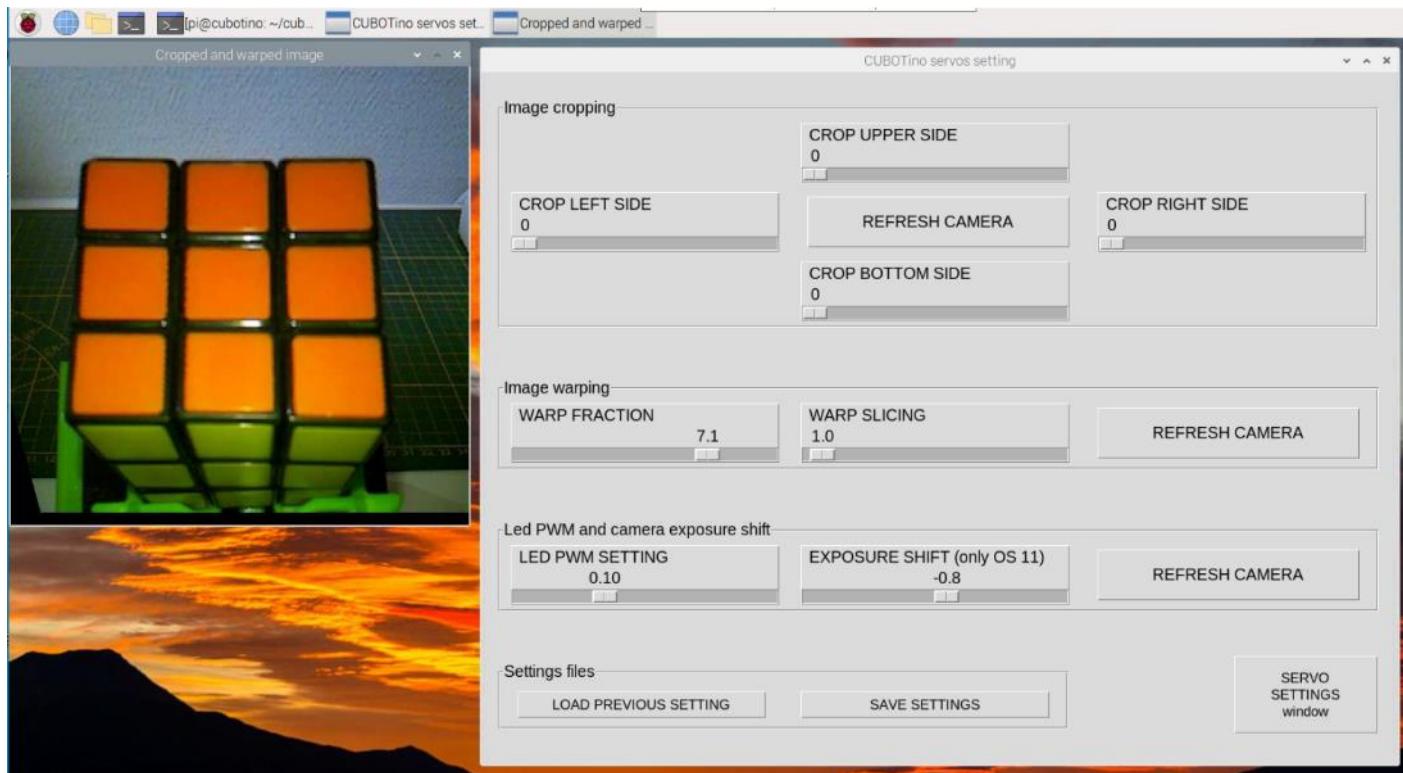
With proper tuning it can be lowered to ca 74 seconds (Raspberry Pi ZeroW).



Section4: Tuning and robot operation

The Camera GUI section is divided in 4 areas:

Area	Description	Note
Image cropping	Sliders to set how many pixels to remove from the four image sides	Areas blocked when the camera is refreshing the image
Image warping	Sliders to alter the image warping, aiming to get top cube side appearing like a square	
Led PWM and camera exposure shift	Slider to set the Top_cover_led intensity. Slider to set the exposure shift at the camera. On OS 11 it seems useful to slightly lower the Exposure time (set the exposure shift to -0.5 ~ -1.0).	The exposure is only applied when OS11
Settings files	Allows to save to file the settings currently at the sliders. It allows to upload the last saved settings, and sliders get updated (save again if you'd like to keep these)	Previously saved settings are renamed by adding date and time. The last 10 files are maintained (older are deleted)



Camera tuning order:

- Cropping
- warp fraction.
- warp slicing.
- Led PWM and Exposure shift.

Press the SAVE SETTINGS button to have the settings active at the next usage.

Section4: Tuning and robot operation

Notes:

1. First adjust the servos positions.
2. Top_cover is forced to the (camera) read position, when accessing the Camera settings window.
3. Leaves some margin around the image.
4. Do not leave complete facelets to be visible on the bottom image side (back cube face).
5. The image is refreshed at every applied change; Anyhow the refresh buttons are available.
6. If the sliders do not produce visible changes, check if any feedback at the CLI: Not all the values combinations produce acceptable results.
7. Be noted that the cropping influences the warping; It might be necessary to repeat the process a couple of times.
8. Hit the REFRESH CAMERA button to have more image references.
9. Be noted the camera is set to AUTO mode, therefore it will adjust all the parameters (Exposure time, Auto White Balance, Gains, etc). This means the Led PWM and the Exposure shift might require some fine tuning based on how the cube scanning performs.

16. Tuning via CLI

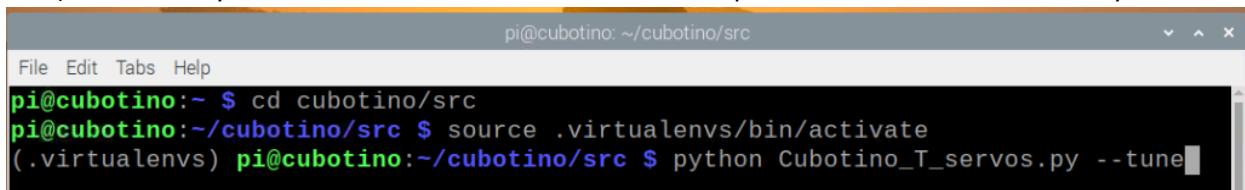
This method is an alternative to use the GUI.

Notes:

1. The CLI is a good tool to set the servo to their mid position and to verify their max rotation range (b_sweep or t_sweep).
2. The CLI methos is not very user friendly to tune the servos positions.
3. The GUI makes the tuning process easier, and in my view the only drawback is the need to use a graphical VNC consuming more resources from the Raspberry Pi.

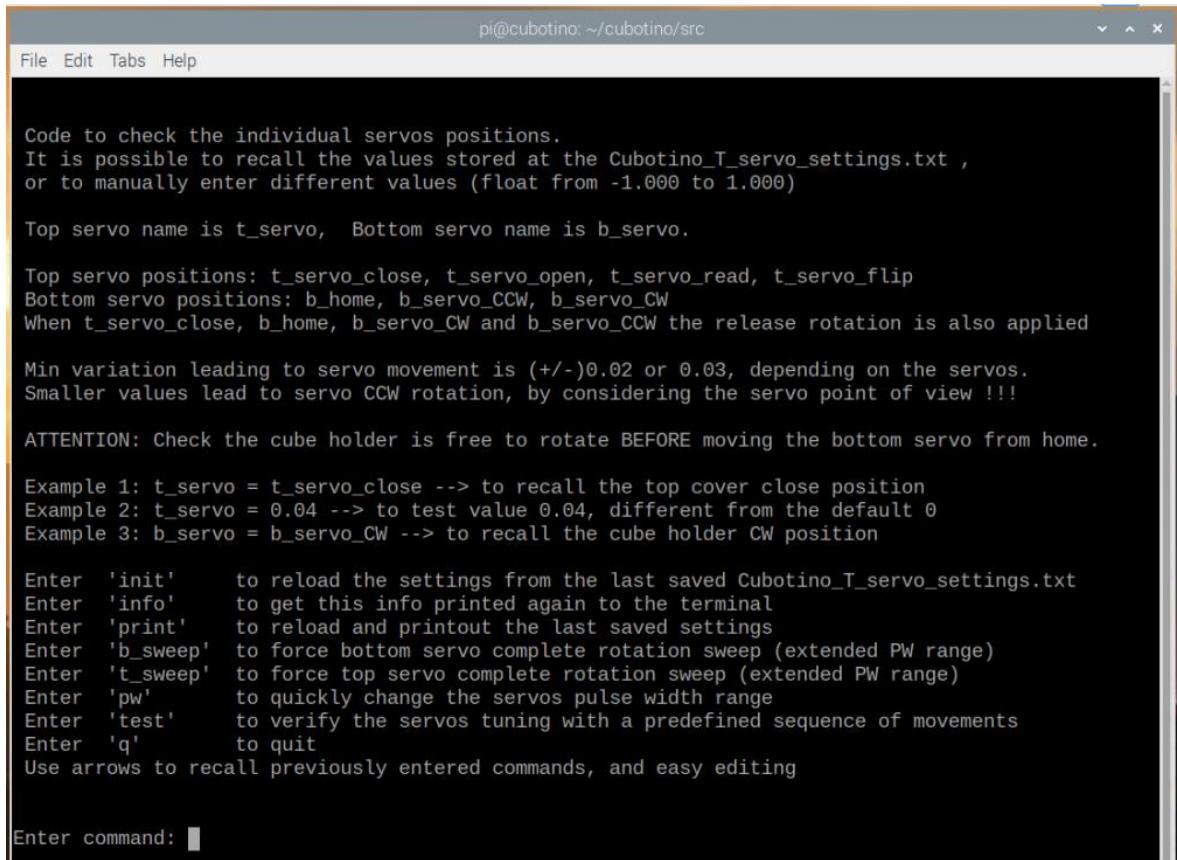
Steps

1. set the Servos to the mid angle (see Servos test and set to mid position chapter).
2. assemble the robot.
3. enter the cube folder (`cd cubotino/src`) and activate the venv (`source .virtualenvs/bin/activate`);
Attention to the dot in front of virtualenvs.
4. run the script `python Cubotino_T_servos.py --tune`; Attention to the space in between '-'
(Note: not anymore needed to use "true" after "tune": Up to revision 4.0 it was necessary to use --tune **true**)



```
pi@cubotino:~/cubotino/src
File Edit Tabs Help
pi@cubotino:~ $ cd cubotino/src
pi@cubotino:~/cubotino/src $ source .virtualenvs/bin/activate
(.virtualenvs) pi@cubotino:~/cubotino/src $ python Cubotino_T_servos.py --tune
```

5. some info will be printed on the Terminal, to guide this process.



```
pi@cubotino:~/cubotino/src
File Edit Tabs Help
Code to check the individual servos positions.
It is possible to recall the values stored at the Cubotino_T_servo_settings.txt ,
or to manually enter different values (float from -1.000 to 1.000)

Top servo name is t_servo, Bottom servo name is b_servo.

Top servo positions: t_servo_close, t_servo_open, t_servo_read, t_servo_flip
Bottom servo positions: b_home, b_servo_CCW, b_servo_CW
When t_servo_close, b_home, b_servo_CW and b_servo_CCW the release rotation is also applied

Min variation leading to servo movement is (+-)0.02 or 0.03, depending on the servos.
Smaller values lead to servo CCW rotation, by considering the servo point of view !!

ATTENTION: Check the cube holder is free to rotate BEFORE moving the bottom servo from home.

Example 1: t_servo = t_servo_close --> to recall the top cover close position
Example 2: t_servo = 0.04 --> to test value 0.04, different from the default 0
Example 3: b_servo = b_servo_CW --> to recall the cube holder CW position

Enter 'init'      to reload the settings from the last saved Cubotino_T_servo_settings.txt
Enter 'info'       to get this info printed again to the terminal
Enter 'print'      to reload and printout the last saved settings
Enter 'b_sweep'    to force bottom servo complete rotation sweep (extended PW range)
Enter 't_sweep'    to force top servo complete rotation sweep (extended PW range)
Enter 'pw'          to quickly change the servos pulse width range
Enter 'test'        to verify the servos tuning with a predefined sequence of movements
Enter 'q'           to quit
Use arrows to recall previously entered commands, and easy editing

Enter command: 
```

Section4: Tuning and robot operation

6. it is possible to recall the settings stored at Cubotino_T_servo_settings.txt as well as to enter different target values (value should be a float ranging from -1.000 to 1.000).
7. After the 'Enter command:' type the below commands to test the servos positions stored at Cubotino_T_settings.txt:
 - t_servo = t_servo_close (a backward rotation is applied if t_servo rel_delta is not zero)
 - t_servo = t_servo_open
 - t_servo = t_servo_read
 - t_servo = t_servo_flip
 - b_servo = b_home (a backward rotation is applied if b_extra_home is not zero)
 - b_servo = b_servo_CCW (a backward rotation is applied if b_extra_sides is not zero)
 - b_servo = b_servo_CW (a rotation backward is applied if b_extra_sides is not zero)
8. To adjust the Top_cover and/or the Cube_holder positions, you might enter the value instead of the saved parameters; Check the example below.

```
Enter command: t_servo = -0.45
testing top servo, argument value: -0.45
done
```

```
Enter command: 
```

9. Once the position(s) are satisfactory, edit the Cubotino_T_servo_settings.txt and save the file to apply the new settings; It is suggested to keep open the Cubotino_T_servo_settings.txt file at the side, to copy paste the command (use shift Ins to paste) and to update and save the new settings.

The screenshot shows two windows side-by-side. The left window is a terminal session titled '*Cubotino_T_servo_settings.txt - Mo'. It contains the JSON configuration file:

```
{
  "t_min_pulse_width": "1",
  "t_max_pulse_width": "2",
  "t_servo_close": "0",
  "t_servo_open": "-0.33",
  "t_servo_read": "-0.5",
  "t_servo_flip": "-0.9",
  "t_servo_rel_delta": "0.02",
  "t_flip_to_close_time": "0.6",
  "t_close_to_flip_time": "0.6",
  "t_flip_open_time": "0.5",
  "t_open_close_time": "0.3",
  "b_min_pulse_width": "1",
  "b_max_pulse_width": "2",
  "b_servo_CCW": "1",
  "b_servo_CW": "1",
  "b_home": "0",
  "b_extra_sides": "0.04",
  "b_extra_home": "0.13",
  "b_spin_time": "0.7",
  "b_rotate_time": "0.8",
  "b_rel_time": "0.2"
}
```

The right window is another terminal session titled 'pi@cubotino: ~/cubotino/src'. It displays a help message for servo control commands:

```
Code to check the individual servos positions.
It is possible to recall the values stored at the Cubotino_T_servo_settings.txt ,
or to manually enter different values (float from -1.000 to 1.000)

Top servo name is t_servo, Bottom servo name is b_servo.

Top servo positions: t_servo_close, t_servo_open, t_servo_read, t_servo_flip
Bottom servo positions: b_home, b_servo_CCW, b_servo_CW
When t_servo_close, b_home, b_servo_CW and b_servo_CCW the release rotation is also applied

Min variation leading to servo movement is (+/-)0.02 or 0.03, depending on the servos.
Smaller values lead to servo CCW rotation, by considering the servo point of view !!

ATTENTION: Check the cube holder is free to rotate BEFORE moving the bottom servo from home.

Example 1: t_servo = t_servo_close --> to recall the top cover close position
Example 2: t_servo = 0.04 --> to test value 0.04, different from the default 0
Example 3: b_servo = b_servo_CW --> to recall the cube holder CW position

Enter 'init'      to reload the settings from the last saved Cubotino_T_servo_settings.txt
Enter 'info'       to get this info printed again to the terminal
Enter 'print'      to reload and printout the last saved settings
Enter 'b_sweep'    to force bottom servo complete rotation sweep (extended PW range)
Enter 't_sweep'    to force top servo complete rotation sweep (extended PW range)
Enter 'pw'          to quickly change the servos pulse width range
Enter 'test'        to verify the servos tuning with a predefined sequence of movements
Enter 'q'           to quit
Use arrows to recall previously entered commands, and easy editing

Enter command: 
```

Section4: Tuning and robot operation

To verify if everything goes well:

1. If you type **info**, instead of a command after the ‘Enter command:’, guidance is printed again on the screen.
2. If you type **init**, instead of a command after the ‘Enter command:’, the new settings from Cubotino_T_servo_settings.txt are re-applied to the robot.
3. If you type **print**, instead of a command after the ‘Enter command:’ the latest values saved at Cubotino_T_servo_settings.txt are printed to the screen.
4. If you type **b_sweep**, or **t_sweep**, the bottom or the top servo will make a couple of full rotations to show their maximum rotation range; These commands temporarily change the pulse width rotation to a very extended range, to ensure the maximum rotation, and after that the pulse width range is set to the previous setting:

```
Enter command: b_sweep
Bottom servo: Temporary set the pulse width to a very extended range (0.3 to 2.7 ms)
Bottom servo: Sweeping the full rotation range
Bottom servo: Set the pulse width to the previous setting

Enter command: █
```

5. If you type **pw**, the servos pulse width range can be quickly changed:

There are three options to choose from, two standard ranges and a custom one:

```
Enter command: pw
type:
 0      for range 0.5 to 2.5ms
 1      for range 1.0 to 2.0ms
 cust   to set custom min and max pulse width values (in milliseconds)
your choice: █
```

By entering 0, both the servos will be set to pulse width range from 0.5 to 2.5ms.

By entering 1, both the servos will be set to pulse width range from 1.0 to 2.0ms (this is the default setting).

By entering cust, you'll be asked to enter first the min pulse width and after the max one.

```
Enter command: pw
type:
 0      for range 0.5 to 2.5ms
 1      for range 1.0 to 2.0ms
 cust   to set custom min and max pulse width values (in milliseconds)
your choice: cust

      enter the min pulse width value in milliseconds
      your choice: █

      enter the max pulse width value in milliseconds
      your choice: █
```

The pw command allows to quickly change the pulse width range of the two servos, to verify the effective rotation range and pulse width.

6. If you type **test**, instead of a command after the ‘Enter command:’, a large sequence of movements, mimicking the solving of a cube, are applied to the robot. This part is also embedded in the GUI part.

Section4: Tuning and robot operation

7. Run the script `python Cubotino_T_servos.py`, without any argument, to test the robot manoeuvring the cube like during a solving process (this is the same of using `test` command); Take a close look to check if the cube handling is ok.
 - a. If the cube layers don't align well, it is suggested to apply some stickers on the cube holder: When the cube holder is home place an 'F' on the cube holder front side, 'L' on the cube holder left side and 'R' on the cube holder right side. Take a movie while the robot manoeuvres a cube and watch it back to see in which position the misalignment is generated.
 - b. Re-adjust the setting for the position that leads to the cube layers misalignment.

This part is also embedded in the GUI part.

Example:

When the command `t_servo = t_servo_close` is entered, the variable `t_servo_close` is assigned to the `top_servo` position.

Based on the Parameters and settings table (next chapter), the default value for the `t_servo_close` is 0 (zero).

In case the `Top_cover` is too far from the cube (reference pictures a few pages above), then the servo position requires to increase the CW position (CW and CCW are from the servo point of view).

To increase the CW rotation is requested a larger value ; If the needed variation is small (i.e. 1.8deg), the increment can be of 0.02.

Considering the default value for `t_servo_close` is zero, you might want to try 0.02 (0 + 0.02) by typing '`t_servo = 0.02`'.

In case the `Top_cover` is too close to the cube then the servo position requires to decrease the CW position (CW and CCW are from the servo point of view).

To decrease the CW rotation is requested a smaller value ; If the needed variation is of about 3.6degrees, the decrement shall be of 0.04.

Considering the default value for `t_servo_close` is zero, you might want to try -0.04 (0 – 0.04) by typing '`t_servo = -0.04`'.

On the `Cubotino_T_servo_settings.txt` file, use your defined values to better cope with your robot characteristics.

17. Parameters and settings

Parameters that are more likely to differ on each system, are stored into two Json files:

Cubotino_T_settings_default.txt and *Cubotino_T_servo_settings_default.txt*

Note: These two files will be automatically duplicated on your local repository, and renamed as:

Cubotino_T_settings.txt and *Cubotino_T_servo_settings.txt*, to prevent overwriting from “git pull”.

To provide a reference, the below tables capture the settings used on my two robots: *Cubotino_T_settings_AF.txt* and *Cubotino_T_servo_settings_AF.txt*.

On the tables are listed the parameters, the proposed values to start the tuning, the values working on my Cubotino robot, and some information about the parameters and settings.

Highlighted the default settings that differ from mine.

Cubotino_T_settings_default.txt (becoming *Cubotino_T_settings.txt* in your local repository) and *Cubotino_T_settings_AF.txt* as reference)

Parameter (dict key)	Default value	AF value	Data type	Info
frameless_cube	false	auto	string	Set the facelets edge detection according to the cube. Options are: ‘false’, ‘true’ and ‘auto’. If the parameter key is missed (i.e. old setting files) a classic cube is considered, meaning frameless_cube = false. frameless_cube = true should be used when small logos on the cube.
disp_type	st7735	st7735	string	st7735 or st7789, depending on the purchased display (st7735 is the default).
disp_width	128	132	Int	Display width (in pixels); At troubleshooting for Display the explanation to make easy to adjust this parameter this parameter
disp_height	160	162	Int	Display height (in pixels); At troubleshooting for Display the explanation to make easy to adjust this parameter this parameter
disp_offsetL	0	-2	Int	Display offset on width Left (in pixels); At troubleshooting for Display the explanation to make easy to adjust this parameter this parameter
disp_offsetT	0	-2	int	Display offset on height Top (in pixels); At troubleshooting for Display the explanation to make easy to adjust this parameter this parameter
camera_width_res	640	640	Int	PiCamera resolution on width. Changes to the Camera resolution has large influence on many functions, and computation time
camera_height_res	480	480	int	PiCamera resolution on height. Changes to the Camera resolution has large influence on many functions, and computation time
s_mode	7	7	int	PiCamera sensor mode (from 22/12/2022, V3.15): 7 for PiCamera V1.3 (Full Field of View, 4:3, binning 4x4) 4 for PiCamera V2 (Full Field of View, 4:3, binning 2x2) For more info look at “6.2 Sensor Modes” at https://buildmedia.readthedocs.org/media/pdf/picamera/latest/picamera.pdf
kl	0.95	0.95	float	Coefficient for PiCamera stability acceptance. Lower values are more permissive (range is 0 to 1). The camera is considered stable when all the parameters from the camera in AUTO mode (AWB, gains, Shutter time, etc) will vary less than $\text{abs}(1-kl)$ from the average of the previous readings. 0.95 stops the AUTO mode when all the parameters have a max deviation of 5% from the average

Section4: Tuning and robot operation

Cubotino_T_settings_default.txt (becoming **Cubotino_T_settings.txt** in your local repository) and

Cubotino_T_settings_AF.txt as reference)

Parameter (dict key)	Default value	AF value	Data type	Info
x_l	0	60	Int	Image cropping at the left, before warping (in pixels). This is meant to removes a slice of the image at the left side, external to the cube image. around the bot, and it will increase speed.
x_r	0	80	Int	Image cropping at the right, before warping (in pixels). This is meant to removes a slice of the image at the right side, external to the cube image. around the bot, and it will increase speed.
y_u	0	0	Int	Image cropping at the top, before warping (in pixels). This is meant to removes a slice of the image at the upper side, external to the cube image. around the bot, and it will increase speed.
y_b	0	110	int	Image cropping at the bottom, before warping (in pixels). This is meant to removes a slice of the image at the bottom side, external to the cube image. around the bot, and it will increase speed.
warp_fraction	7	7	float	Image warping index. This parameter is used to alter the perspective from the cube face images. Smaller values increase the effect, meaning it applies a larger variation to the camera image.
warp_slicing	1.5	1.5	float	Image cropping index, that crops the right side of the image after the warping process. Values from 0.1 to 0.9 increase the cropping and values bigger than 1.1 reduce the cropping.
square_ratio	1	1	float	Facelets contour squareness check filter. This parameter is the max threshold used to filter out non-square like contours, calculated as the delta between the max and min contour sides divided by the mean. Possible values are > 0. 0 is the perfect square therefore never possible to meet! 1 is a rather permissive threshold (max delta sides = average sides)
rhombus_ratio	0.3	0.3	float	Facelets contour rhombus check filter. This is the lower threshold used to discharge contours with excessive rhombus shape, calculated as the ration between the min rhombus axis and the max one. Smaller values are more permissive (1 is perfect Rhombus). 0.3 is a rather permissive threshold (max axis = 3.3 * min axis)
delta_area_limit	0.7	0.7	float	Facelets area deviation check filter. This is the upper threshold, for each contour calculated as the ratio between the contour area and the median area based on at least 7 detected facelets. Larger values are more permissive (0 means no deviation).

Section4: Tuning and robot operation

Cubotino_T_settings_default.txt (becoming **Cubotino_T_settings.txt** in your local repository) and **Cubotino_T_settings_AF.txt** as reference)

Parameter (dict key)	Default value	AF value	Data type	Info
sv_max_moves	20	20	int	Max number of moves requested to the Kociemba solver. When the solver finds a solution matching this movement quantity, that solution is returned even before the timeout expiration (sv_max_time).
sv_max_time	2	2	float	Timeout, in seconds, for the Kociemba solver. The best solution found within the timeout is returned at timeout end, even if it doesn't match the desired max quantity of moves.
collage_w	1024	1024	int	Image width for the unfolded cube file. This parameter determines the image collage realization, and it makes possible to save all images with the same size.
marg_coef	0.1	0.06	float	Defines the margin around the cube faces images. This margin is used to cut the six cube faces with some margin around, for the unfolded cube collage. The margin is calculated by multiplying the detected cube diagonal in pixels to this coefficient. The larger the value the more pixels margin around the cube 0.1 means the margin is 10% of the cube diagonal (calculated on the detected facelets contours).
cam_led_bright	0.1	0.1	float	PWM for the 3W led at Top_cover. Range from 0 (no PWM) to 1 (PWM=100%). At the Cubotino_T_servos.py the max value really delivered to the LED is 30%, therefore values > 0.3 are useless.
detect_timeout	40	40	int	Timeout, in second, for the cube status detection. If the six cube faces aren't detected within this time, the cycle is terminated with a timeout message on the display.
show_time	7	7	int	Time, in seconds, the unfolded cube image is kept on screen. This only applies when a screen (i.e. VNC) is connected.
warn_time	1.5	1.5	float	Time from touching the touch button (after 0.5s filter), after which a warning appears on display. If the button is released after the warning, and within quit_time, the robot stops without quitting the script.
quit_time	4.5	4.5	float	Time from touching the touch button (after 0.5s filter), after which the script starts the quit procedure. This timer starts right after the warn_time elapses. If the button is kept pressed longer than this time, the script quits (and the Rpi SHUT OFF if automated SHUT OFF is set).
cover_self_close	false	true	string	Top_cover auto closing at shut down. Options are 'false' and 'true'. If the parameter key is missed (i.e. old setting files) the cover_self_close is set false.

Section4: Tuning and robot operation

Cubotino_T_settings_default.txt (becoming **Cubotino_T_settings.txt** in your local repository) and
Cubotino_T_settings_AF.txt as reference)

Parameter (dict key)	Default value	AF value	Data type	Info
vnc_delay	0.5	0	float	<p>(from 12/01/2023, V3.17): Timer in seconds, to delay the cube moving to the next face during the cube detection phase. This delay compensates for the Rpi-VNC connection delay, obtaining a more synchronized images on screen and cube movement (pleasant experience). In case the Cubotino_T_settings.txt belongs to an older version (absence of this parameter), 0.5secs will be used as default.</p>
built_by	""	""		<p>If a text is entered, it will be displayed in BLUE under the CUBOTino logo. In the example below the built_by parameter is "DAVID".</p>  <p>When built_by parameter is not an empty text, then "Andrea FAVERO's" is displayed in red above the logo. Run <i>Cubotino_T_display.py</i> to check the result.</p>
built_by_x	20	20		Adjusts the horizontal position of the built_by text (bigger values move to the right).
built_by_fs	16	16		Adjusts the font size of the built_by text.
expo_shift	-0.6	-0.6	float	<p>(from 21/01/2024, V6.3): Correct the camera exposure time (range is from -2.0 to +1.0). It is the equivalent of the clicks on the camera, when adjusting the exposure setting. A small reduction (negative values) seems beneficial to prevent overexposure in OS 11.</p>
fcs_delay	3	3	float	<p>(from 21/01/2024, V6.3): Timer in seconds, to switch to Fix Coordinates System (FCS). Before this time elapse, the cube facelets are searched via edge detection, afterward the facelets position are based on fix coordinates. The fix coordinates are automatically generated, on cycles when the fcs is not used. The coordinates are saved on a text file.</p>

Section4: Tuning and robot operation

Parameters related to the servos.

Notes:

1. ‘t_’ refers to Top_servo while ‘b_’ refers to Bottom_servo.
2. “Angles” are in gpiozero range for the Servo class (range from -1 to 1, with 0 as mid angle)
3. Time is in seconds.

Cubotino_T_servo_settings_default.txt (becoming **Cubotino_T_servo_settings.txt** in your local repository), and Cubotino_T_servo_settings_AF.txt as reference:

Parameter (dict key)	Default value	AF value	Data type	Info
t_min_pulse_width	1	1	float	Min pulse width, in ms of the used top servo. Most of the servos accepts a slightly extended value (<1)
t_max_pulse_width	2	2	float	Max pulse width, in ms, of the used top servo. Most of the servo accepts a slightly extended value (>2)
t_servo_close	0	0	float	“Angle” for Top_cover to constrain the top and mid cube layers
t_servo_open	-0.32	-0.32	float	“Angle” for Top_cover not constraining the cube and Cube_holder
t_servo_read	-0.5	-0.5	float	“Angle” for Top_cover for PiCamera reading. Lifter almost touching the bottom cube face
t_servo_flip	-0.9	-0.9	float	“Angle” for Top_cover to flip the cube (~2 cube layers)
t_servo_rel_delta	0.02	0.06	float	Delta “angle” for Top_cover to retract after closing
t_flip_to_close_time	0.6	0.38	float	Time for t_servo to move from Flip to Close position
t_close_to_flip_time	0.6	0.42	float	Time for t_servo to move from Flip to Close position
t_flip_open_time	0.5	0.34	float	Time for t_servo to move from Flip to Open position and vice versa
t_open_close_time	0.3	0.1	float	Time for t_servo to move from Close to Open position and vice versa
t_rel_time	0.2	0	float	Time for t_servo to retract after closing
b_min_pulse_width	1	0.98	float	Min pulse width, in ms, of the used bottom servo. Most of the servos accepts a slightly extended value (<1)
b_max_pulse_width	2	1.98	float	Max pulse width, in ms, of the used bottom servo. Most of the servo accepts a slightly extended value (>2)
b_servo_CCW	-1	-1	float	“Angle” for the Cube_holder at ~90° CCW from Home. CCW is from motor point of view
b_servo_CW	1	1	float	“Angle” for the Cube_holder at ~90° CW from Home. CW is from motor point of view
b_home	0	0	float	“Angle” for the Cube_holder in between CW and CCW positions
b_rel_CCW	0.04	0.06	float	(b_extra_sides until 18/04/2023) “Delta angle” for the Cube_holder to retract from CCW
B_rel_CW	0.04	0.06	float	(b_extra_sides until 18/04/2023) “Delta angle” for the Cube_holder to retract from CW
b_extra_home_CCW	0.12	0.1	float	(b_extra_home until 18/04/2023) “Delta angle” for the Cube_holder before retracting Home when rotation from CCW
b_extra_home_CW	0.12	0.1	float	(b_extra_home until 18/04/2023) “Delta angle” for the Cube_holder before retracting Home when rotation from CW
b_spin_time	0.7	0.44	float	Time for the Cube_holder to spin ~90° (cube not constrained)
b_rotate_time	0.8	0.54	float	Time for the Cube_holder to rotate ~90° (cube constrained)
b_rel_time	0.2	0	float	Time for the Cube_holder to rotate back, at CCW, CW and Home

Note: On Cubotino_T.py and Cubotino_T_servos.py, the string ‘#(AF ‘ is placed as comment start, where the above listed parameters are used. This because those variables weren’t initially collected in a Json file; Later I decided to comment those rows instead of cancelling them.

18. Troubleshooting

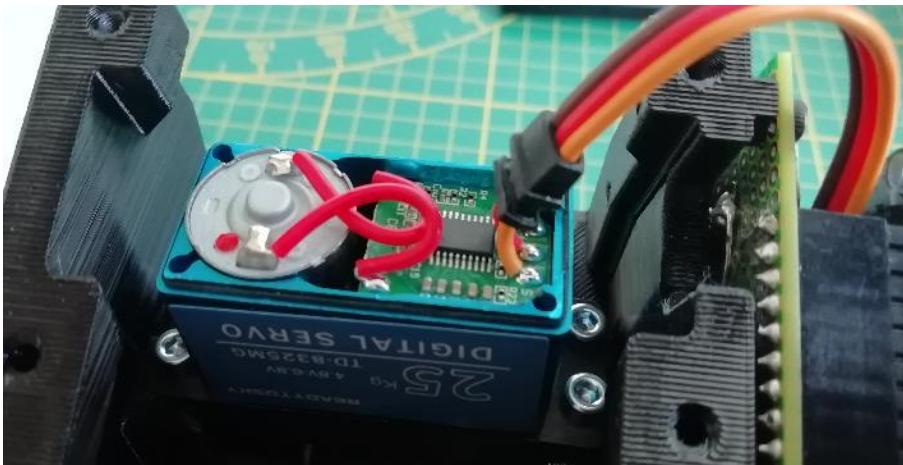
Some of the below aspects were encountered during the robot development, other were posted at Instructables, other were suggested by some of the Makers, and remaining are hypothetical:

1. Servos rotate about 180 deg, yet not more than that.
2. Servos rotation angle of about 90 deg, when you've ordered 180deg.
3. Servos rotation angle of about 270deg.
4. Servos not moving smoothly.
5. Cube layer (bottom, or central vertical) doesn't align nicely.
6. Top cover usage to flat the cube.
7. Cube status detection error.
8. Robot stuck on reading the same face.
9. Cube's facelets and light reflection (cube status detection).
10. Displayed text and images are un-readable.
11. Program doesn't work as intended.
12. PiCamera focus.
13. Cube misalignments after updating the scripts.
14. Low Voltage Throttle.
15. Updating the Cubotino software.
16. Raspberry Pi freezing (memory management).
17. Raspberry Pi Wi-Fi dropping.
18. Errors during the Raspberry Pi installation process.
19. Error "Total corner twist is wrong" after scanning the cube.
20. Display and/or Top_cover_led flickering right after the boot.
21. Touch sensor issues.

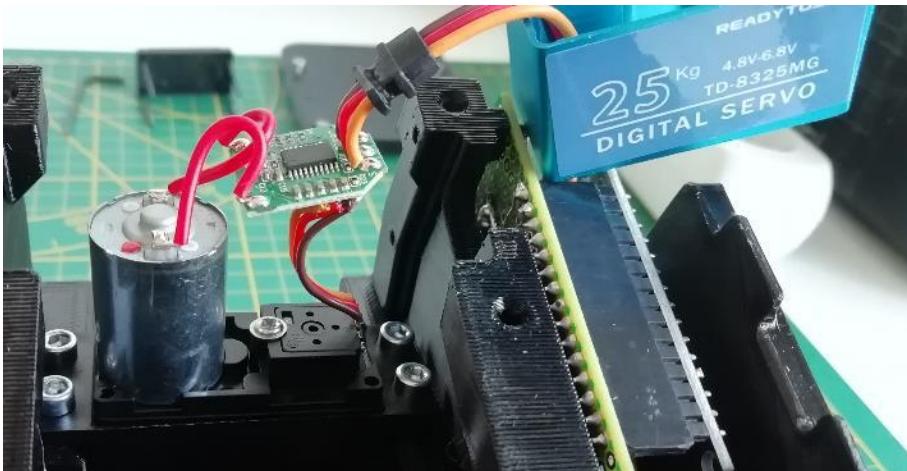
1. Increase the servo rotation range (if needed)

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)

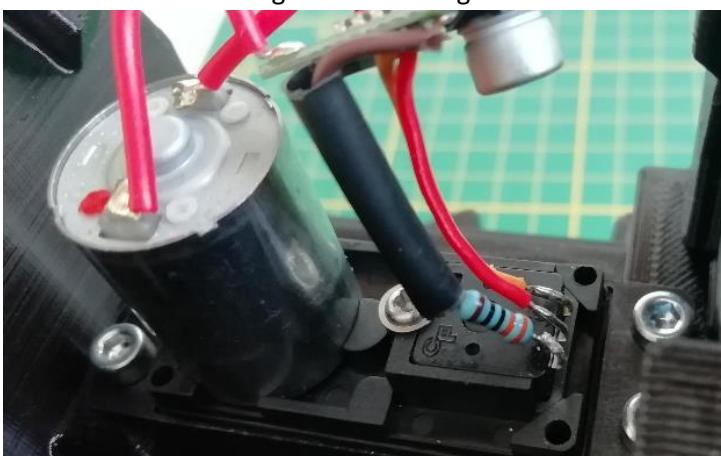


2. Slide out the pcb, and afterward slide the case out of the way.



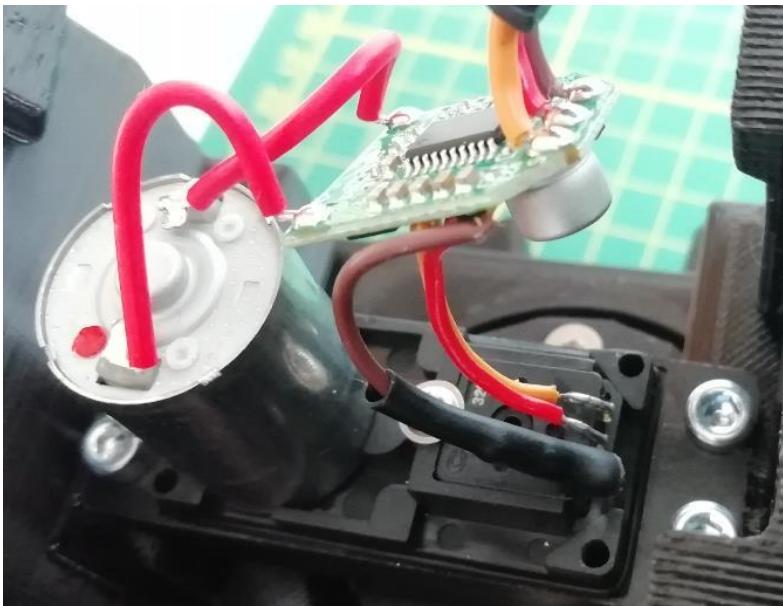
3. Solder one resistor to one external pin of the potentiometer (or one resistor per each of the two external potentiometer pins to keep the same PWM value for the mid position).

I've added 330ohm to gain about 5 degrees on a servo having pulse width from 1 to 2 ms.

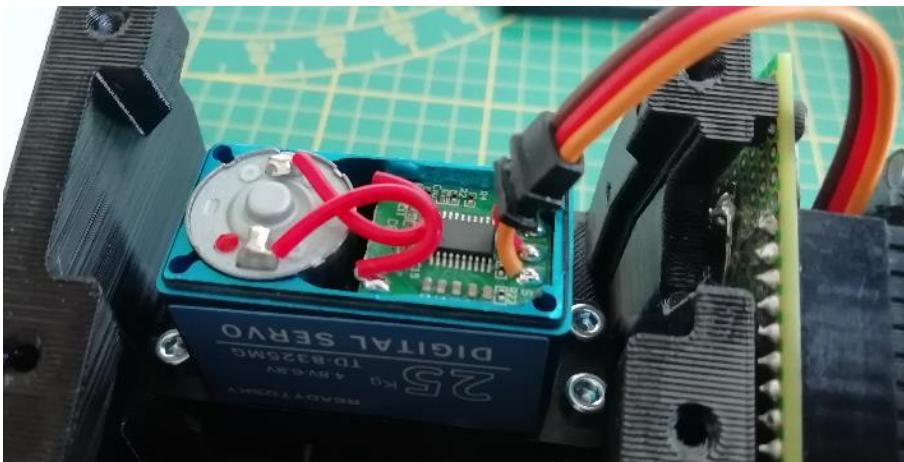


Section4: Tuning and robot operation

4. Protect the soldered parts with a sleeve:



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



6. Place the cover and close the 4 screws.

Section4: Tuning and robot operation

2. Servos rotate about 90deg while you've ordered 180deg servos.

This has probably to do with the Pulse Width of the received servos, ranging from 500 μ s to 2500 μ s instead of from 1 to 2 ms.

This project uses as default a Pulse Width range from 1m to 2ms, yet you can adjust some parameters and get your servos working fine; At Cubotino_T_servo_settings.txt change:

- b_min_pulse_width from 1 to 0.5 (meaning the min pulse width reduces from 1ms to 0.5ms)
- b_max_pulse_width from 2 to 2.5 (meaning the max pulse width increases from 2ms to 2.5ms)
- t_min_pulse_width from 1 to 0.5 (meaning the min pulse width reduces from 1ms to 0.5ms)
- t_max_pulse_width from 2 to 2.5 (meaning the max pulse width increases from 2ms to 2.5ms)

3. In case you've got servos with 270deg rotation, it shouldn't be of a problem.

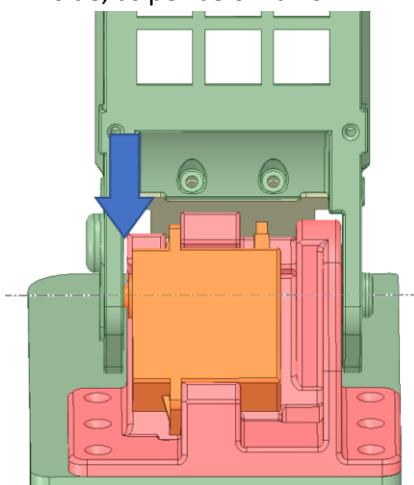
If the Pulse Width is from 500 μ s to 2500 μ s, then the angle resolution will be acceptable; In this case correct the Pulse Width parameters as per previous Troubleshooting point.

In case of Pulse Width from 1ms to 2ms I don't know if the angle resolution will be acceptable for this robot.

After adjusting the Pulse Width parameters, in case servos aren't from 1ms to 2ms, the default values for servos positions should be scaled by a 0.67 factor, to scale down from 270 to 180 degrees of rotation.

4. Servos don't move smoothly.

1. Don't use jumper wires or use quality jumper wires.
2. Don't use bread boards, make the Connections board instead.
3. Add the capacitors, to prevent voltage drops when servos are activated.
4. Use an at least 2A power supply for the servos.
5. Use a 20 to 25Kg/cm servo.
6. Minimize Top_cover rotation friction:
 - i. Ensure the hole for the M4 screw (pivot) has some gap on the Top_cover hole (\varnothing 4.1 to \varnothing 4.3mm).
 - ii. Rub some candle wax on the screw.
 - iii. Ensure the M4 screw head is not pushing toward the Top_cover; 1mm 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.

5. Cube layer (bottom, or central vertical) doesn't align nicely.:

This is probably the most difficult part of the tuning process.

Bear in mind CW and CCW notations are from the servos point of view: This means it will be the other way around for the person watching the Cube_holder.

1. Verify if the cube Holder makes an extra rotation, at both CCW and CW directions, before stopping. If this doesn't happen:
 - 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:
 1. Try to enlarge the Pulse Width range by 0.02 or 0.04 (increase b_max_pulse_width if the Cube_holder doesn't make enough rotation at CW location, decrease b_min_pulse_width if the Cube_holder doesn't make enough rotation at CCW location)
 2. 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.
3. In case the cube has very little friction between layers, it is possible to get the mid vertical layer misaligning by the Lifter while flipping the cube, more likely when consecutive flipping.
Most of Rubik's cube can be adjusted on the friction:
 - a) remove the cap at the centre facelets.
 - b) choose a proper and good screwdriver for the screws.
 - c) close each screw by half turn, to increase the friction.
 - d) evaluate if sufficient friction increment.

6. Top cover usage to flatten the cube:

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. Cube detection error:

It is returned when the interpreted cube status isn't coherent, meaning not having 9 facelets per colour or other inconsistencies. Possible causes:

1. Objects on the table (background); Objects on the table can form square like contour, interpreted as facelets by the cv. This can be solved by positioning the robot to a uniform-coloured surface, without cables and objects in front of 30cm around the robot. Another good way to solve this problem is to tune the cropping parameters.
2. Light reflection. Try to orient the robot with external light source (i.e. window) coming from the side or to use a cube with less glossy facelets.
3. Too little light conditions cannot be compensated by the LED light source.
4. In case the cube has some prints (i.e. brand), typically on the white centre, it is suggested to carefully scratch out.
5. In case a frameless cube type is used (facelets without the black frame around the facelets), while at Cubotino_T_settings.txt the frameless_cube parameter is not 'true' or 'auto'.
6. The setting 'auto' to detect the status on cubes with and without the frame works better with good light conditions; If this isn't possible, set the parameter to the specific type of cube associated to the robot.

8. Robot stuck on reading the same face, until timeout:

- a. When the frameless_cube is set 'false' (classic cube type), the cube status detection algorithm must find 9 facelets with given characteristics before changing face; If the robot doesn't change the cube face, it is because some of the pre-conditions aren't met (at least 9 facelets, areas of the facelets, distance between the facelets, etc)
- b. When the frameless_cube is set 'true' or 'auto', the cube status detection algorithm must find 5 to 7 facelets with given characteristics before changing face; The remaining facelets are estimated for the position, not the colour.
- c. When the ambient light is rather low, and the U face is rather clear: Increase the ambient light or change the cube orientation to allow the camera to set to a more "balanced" face.
- d. When the ambient light is rather high and the U face is rather dark: Decrease the ambient light or change the cube orientation to allow the camera to set to a more "balanced" face.

To troubleshooting is important to visualize what the camera sees; This is possible via below steps:

- 1) Connect to the Rpi via VNC Viewer.
- 2) If the robot has automatically started at the boot, two processes need to be killed as per "How to operate the robot" Step12.
- 3) Resize the terminal to no more than half screen, and move it to the right part of the screen.
- 4) Run the script from the terminal
 - 4_1) `cd ~cubotino/src`
 - 4_2) `source .virtualenvs/bin/activate`
 - 4_3) `python Cubotino_T.py`
- 5) Press start to let the robot working, and a windows will show what the camera sees.

A contour will be drawn, over the camera image, on every location interpreted as facelets (excess of contours are filtered out, lack of contours is critic...)

This should help to have an understanding on the reason, or reasons, the robot stuck on the first cube face.

Possible reasons for the facelets detection failure:

- A)** the camera doesn't see the complete top face of the cube: In this case change the camera orientation angle, via the 2 screws on the Camera_support, to have margin around the top cube face.
- B)** facelets on the back cube side are also detected (detected means that on the image contours are drawn on the back cube facelets): Apply the cropping as explained for "frame cropping" in the "Tuning" chapter
- C)** the critic face has a logo on the central facelets: Carefully scratch that out or cover it.
- D)** too low light conditions: Increase ambient light.
- E)** light reflection: Avoid localized light source from the ceiling, better from the side or even better if diffused. Consider the option to make matt the facelets.
- F)** frameless_cube parameter not matching with the used cube.
- G)** the setting 'auto' at frameless_cube parameter works best with good light conditions; If that isn't possible, then it is preferred to set the frameless_cube to the specific type of cube associated to the robot.

9. Cube's facelets and light reflection (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 facelets, I made the surface matt by using a fine grit sandpaper (grit 1000); This makes the cube status detection much more unsensitive to the light situations.

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



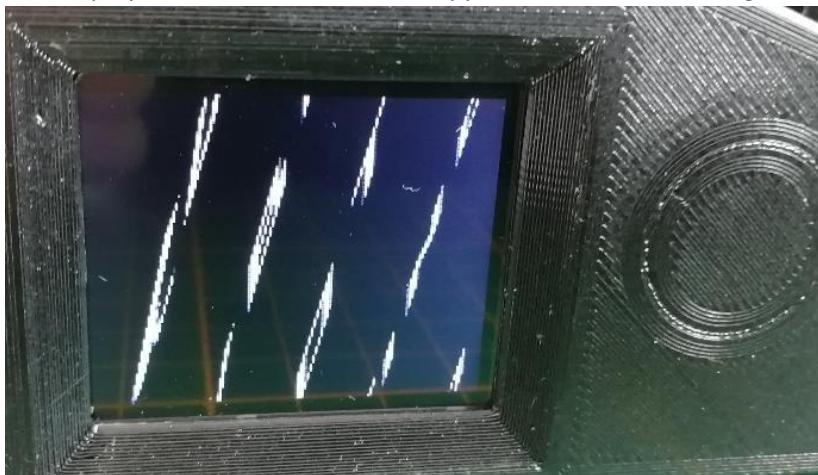
Cube with glossy stickers (after taking this picture I made matt these facelets too)



10. Display: Text and images are weird, simply un-readable:

Display parameters are set in a (Json) text file: Cubotino_T_settings.

The display I've received, from the supplies list, wasn't working as intended:



I took me some time to realize a sort of drifting.... And to find the fix.

By slightly changing the dimensions of just a couple of pixels on the display width parameter (in my case I had to set 128 instead of 130 pixels), the display was working.

Once that problem was solved, a second one became apparent: On two sides there were some 'tiny death bandwidth', again for just a couple pixels.

By checking the ST7735.py code, I discovered it provides offset parameters, for the two display directions, suggesting these issues to be well known.....

By playing with both the display dimensions and the offsets, the display start working simply fine.

I don't expect all the display to have the same issue, yet in case the parameters variables are already in the code, and ready to be properly tuned.

11. Program doesn't work as intended:

This is a difficult topic, as my coding skills are rather limited

A good starting point is to get some feedback from the script:

- a. Edit Cubotino_T.py
- b. At `__main__` change the Boolean "debug" to True. This variable is used by many functions to print out info to the terminal.
- c. Run Cubotino_T.py
- d. Check the prints.
- e. If the printout doesn't suggest much to you, share it at the Instructables chat.

When the problem seems more related to the servo program:

- a. Edit Cubotino_T_servos.py
- f. At about row 85 change the Boolean "s_debug" to True. This variable is used by many functions to print out info to the terminal.
- g. Run Cubotino_T_servos.py (activate the venv first and recall typing python in front). This code activates the servos like solving a predefined scrambled cube.
- h. Check the prints.
- i. Run Cubotino_T.py and let it call the Cubotino_T_servos.py.
- j. Check the prints.
- k. If the prints out don't suggest much to you, share it at the Instructables chat.

12. PiCamera focus.

The V1.3 PiCamera has fixed focus.

Because of the short distance between the PiCamera and the cube face, it is necessary to adjust the focus.

To verify the camera focus, activate the PiCamera as indicated above "Test the PiCamera

The camera will be activated, and its image sent via the VNC:

After 20 seconds the test ends, and the command can be repeated as long as needed.

Check if the focus is roughly ok at 6cm distance, if not proceed with the following instructions until the focus is satisfactory.

Reference tutorial I followed <https://projects.raspberrypi.org/en/projects/infrared-bird-box/6>

I changed the focus on four PiCamera so far; initially it wasn't easy, yet it seems I've now got the key learnings:

Remove **all** the glue, all around, **before** trying to rotate the lens.

- Use a cutter with a new blade, to have a very sharp thin tip.
- To reduce the focus distance, the lens must be rotated CCW (unscrew it).
- To prevent the lens from detaching from the camera, proceed in small steps (i.e., 1/3 of revolution per time).
- To get the cube in focus, I had to rotate the lens for about one full revolution.
- To check the camera focus result, prior to the robot assembly, consider the cube face will be at 55~65mm distance from the camera lens (mid-point at ~60mm).
- After correcting the focus, try to evaluate if the lens cover has enough friction, to avoid adding glue. I did not have to glue the lens, as there was still quite a bit of friction, preventing the lens from getting loose.



13. Cube misalignments after updating the scripts.

On 21/08/2022 I changed Cubotino_T_servos.py code.

I've been informed this has led to cube misalignment on a case, wherein the robot tuning was performed with an older script version.

In my view the misalignment should be solved by tuning again the parameters, as the new script has a more logical (and wanted) solution.

In case the update, from before 21/08/2022 to after that date, gives you problems:

Search this date on the new Cubotino_T_servos.py, uncomment the original code and comment the new one.

This date is added as comment on 4 locations (2 times for the old command and 2 for the new one).

14. Low Voltage Throttle

In case you're using a full size Rpi board (model 3 or 4), and you experience erratic behaviour or LVT (Low Voltage Throttle / Warnings), there are good chances your power supply is not good enough.

The Low voltage warning is displayed on the (VNC) screen:



The Low voltage warning can also be checked on the terminal; In this case type `vcgencmd get_throttled`

```
(.virtualenvs) pi@cubotino:~/cubotino/src $ vcgencmd get_throttled
throttled=0x0
```

Be noted the Raspberry Pi Zero range has not the low voltage detection circuitry

(<https://www.raspberrypi.com/documentation/computers/raspberry-pi.html#power-supply>).

I believe the low voltage warning won't be visible on the raspberry Pi Zero series.

`vcgencmd get_throttled` return must be interpreted as per below table

(<https://www.raspberrypi.com/documentation/computers/os.html#vcgencmd>):

Bit	Hex Value	Meaning
0	1	Under-voltage detected
1	2	ARM frequency has been capped
2	4	Currently throttled
3	8	Soft temperature limit is active
16	1000	Under-voltage has occurred
17	2000	ARM frequency capping has occurred
18	4000	Throttling has occurred
19	8000	Soft temperature limit has occurred

Section4: Tuning and robot operation

Example of throttled: 0x50000

Converted to binary: **0101 0000 0000 0000 0000**
19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

On this example bits 16 and 18 are showing up, meaning the low voltage has been throttled.

Tips to solve the Low Voltage Throttle:

- Use the official Raspberry Pi power supply; The one for model 4 comes with the USB Type-C connector.
- Use USB Type-C receptacles (nominal current is higher than microUSB).
- Use at least 20AWG wires (18AWG for the 4b model) between the USB receptacles and the Connections_board.
- Eventually connect the GPIO connector (pin 4 positive, pin 6 negative) directly to the USB receptacles.

15. Updating the Cubotino software

Cubotino is a hobby project, that kept improving and growing thanks to the feedback from the Makers like you.

This means there might be firmware updates to solve bugs or to add functionalities.

See "Setting up Raspberry Pi" Step 8 for the details on how to check / update your system.

16. Raspberry Pi freezing (memory management)

In case of a Raspberry Pi with 512Mb of RAM (ZeroW, Zero2W,etc) there might be situations requesting for more memory, and not fitting with the default of 100Mb swap_size memory.

Rpi green light flashing, with an irresponsible microprocessor, might suggest a large amount of data is write to the microSD (a potential out of memory recovery).

The swap_size memory can be enlarged, to prevent this type of issue.

See "Setting up Raspberry Pi" Step 16 for the details.

17. Raspberry Pi disconnects from VNC (Wi-Fi stability)

When the VNC connection drops for long time inactivity, it can be set again without problems.

Differently, when the connection suddenly drops, and it isn't possible to re-establish a connection, then it's necessary to search for the potential cause:

4. Check if the power supply at Rpi is ok.
5. Check if the Rpi green light isn't flashing; This indicates the Rpi processor being busy/freezing (see out of memory info above) and not capable to handle the VNC connection.
6. Check the network at your PC is up and running.

If the problem isn't related to the above listed causes, then you might want to try different Wi-Fi settings.

See "Setting up Raspberry Pi" Step 17 for the details.

18. Errors during the Raspberry Pi installation process.

If the installation process stops before getting to an end, scroll the printout searching for error messages.

I've been informed about consistent installation errors, disappearing by using different microSD card.

Another possible reason is the change of available "Legacy OS", from Legacy OS Buster until middle of October 2023 to Legacy OS Bullseye.

This can be solved by flashing Raspberry Pi OS (Legacy) ported from Debian Bullseye, as indicated at chapter "Setting up raspberry Pi".

19. Error "Total corner twist is wrong" after scanning the cube.

If the cube scanning ends with a message "An error occurred" and a couple of rows above there is a message "Solver return: Error: Total corner twist is wrong", then one or more cube's corner has been twisted making the cube unsolvable.

In this case you might consider disassembling the complete cube, and to reassemble it fully solved: Whatever scrambling you'll apply to a normal cube allows the solver to find a solution; Whatever scrambling you'll apply to a cube with twisted corners, will remain an unsolvable cube 😊.

20. Display and/or Top_cover_led flickering right after the boot.

On Raspberry Pi Zero2 with Bullseye, pigpiod server might give this problem.

Via the Cubotino installation, pigpiod is set to automatically start at the boot, as this is the logical approach is most of the cases ... yet not always working fine.

The pigpiod can be delisted from the services to run at boot (as reference

<https://raspberrypi.stackexchange.com/questions/70568/how-to-run-pigpiod-on-boot>):

sudo systemctl disable pigpiod

sudo systemctl stop pigpiod

Differently, the flickering is stopped once one of the Cubotino_Tpy script is started; When the Robot automatically starts at the boot, the flickering hasn't enough time to become apparent.

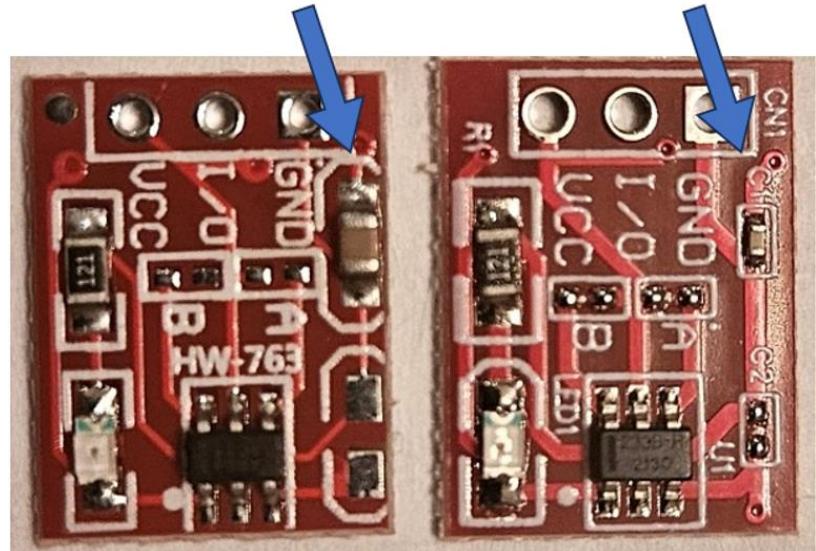
21. Touch sensor issue.

One of the CUBOTino makers has experienced issues with the touch sensor: About half of the received boards weren't working and those that were working weren't sensitive enough to sense from behind the PCB_cover.

After buying the "same" sensors from another vendor, everything started working properly.

The board on the left, with a larger C1 capacitor, is the one from the second vendor that works as expected.

The board on the right, with a smaller C1 capacitor, is the one that wasn't sensitive enough.



In case you experience issues with the touch sensor, please consider:

- a. The sensor sets its zero reference at power up: Sensor movements from the plastic wall, after the zeroing, are detrimental for proper sensing.
- b. Verify if the touch sensor is really powered at 3.3V; If not, you might refresh the soldering especially at the LDO (mine have pins with material difficult to solder); If the voltage is still not ok, you might exchange the LDO.
- c. Verify the sensing surface touches properly the PCB_cover. In case the soldered wires are protruding too much, cut them shorter.
- d. Force the sensor to stay against the PCB_cover. For a test, add a little spacer over the sensor and fix the sensor with tape; If this solves the problem, apply some hot glue instead of the tape.

If these checks do not bring positive results, and the board has a small C1 capacitor, you might consider buying sensors from another vendor.

19. How to operate the robot

1. Before starting:

At the robot start, the Top_cover will ‘suddenly’ open: Do not let kids to stick their nose right on top of robot!

2. Use a uniform-coloured table:

Part of the table around the robot will be captured on cube face images.

Keep some free space around, and use a uniform-coloured base, to prevent cables or other objects to be eventually detected as facelets.

3. Power up the servos (when independent power supply) :

This considers an independent power supply for the servos

Connect a 5V power source to the microUSB connector where the servos are connected.

In my case the servos work flawless with a phone charger rated 2A, but I had inconsistent movements when using a phone smart-charger rated 2A, and with a normal phone charger rated 1A.

Notes

- a. Do not energize the servos once the SBC is up and running, if you aren’t sure whether the cube holder is positioned to home.
- b. If the servos are already connected (or at least energized before the Raspberry Pi boots),

Cubotino_T.py script takes care to position the servos according to a pre-defined order.

4. Power up Raspberry Pi Zero 2 (when independent power supply):

Connect a 5V power source to the remoted microUSB connector where Raspberry Pi is connected.

Do not connect phone smart-charger, those that can deliver a voltage higher than 5.1V.

In my case the SBC works flawless with a phone charger rated 1A (note the official documentation suggest higher current).

Section4: Tuning and robot operation

6. Run the Cubotino_T.py script (if not automated at Rpi boot)

- a. Access src folder: `cd cubotino/src`
- b. Activate the venvs: `source .virtualenvs/bin/activate`
- c. Run the script: `python Cubotino_T.py`
- d. Below arguments can be added:

Short (-)	Argument (--)	Description
-h	-- help	Provides get the list of possible arguments with a little description
-v	--version	Script version
-f	--twosteps	Enables a movement interruption from Flip-Up to Close_cover (two steps instead one), giving the cube more time to stop bouncing.
	--no_animation	Disables facelets animation on display and screen, after the cube being solved
	- -cv_wow	Shows on screen the image processing steps (it requires a FHD screen/setting)
-F	- -F_deg	To use Fahrenheit degrees instead of Celsius
-c	--cycles	See "remote" usage, for automated scrambling and solving cycles.
-p	--pause	
-s	--shutoff	
	- -timer	Visualize a timer after the scrambling function. It starts with 15s for cube status inspection, followed by an incremental timer. Timeout 1 hour
	- -slow_t	Adds a pause after each servo movement, for demo purpose. The parameter must be followed by an integer of the time in tenth of second use (i.e., 10 means 1.0 sec)

Only for test purpose

Short (-)	Argument (--)	Description
-d	- -debug	Printout info and variables for debug purpose
	--picamera_test	Activates the PiCamera for 30s and plots the image on the screen, also VNC (test purpose)
	--no_btn	Starts the solving cycle without using the touch button (test purpose)
	--silent	Deactivate servos, for 'quite' debug of non-servos related aspects (test purpose)

Notes:

- Argument can be added into the bash file for automated run at Raspberry Pi boot.
- Arguments don't have an order.
- Most of them can be combined (when - -cycles then - -timer is skipped as not compatible).

Examples:

```
python Cubotino_T.py - -debug - -F_deg
python Cubotino_T.py - -cv_wow - -F_deg
python Cubotino_T.py - -timer
```

Section4: Tuning and robot operation

6. Start a solving cycle:

- a. Position the cube on the cube holder; any cube orientation is accepted.
- b. Cube layers should be reasonably aligned.
- c. Shortly touch the PCB_cover in front to the touch sensor (the circle suggests the touch sensor location).
- d. The robot reacts by energizing the LED, and by indicating CAMERA SETUP on the display.
- e. The solving process can be interrupted at any time, by pressing the touch sensor for about 1 second.

7. Start a scrambling cycle:

- a. Position the cube on the cube holder.
- b. Cube layers should be reasonably aligned.
- c. Double touch (within 1 second) the PCB_cover in front to the touch sensor (the circle suggests the touch sensor location).
- d. The robot reacts by energizing the LED already at the first touch; If the second touch is detected within 1 second CUBE SCRAMBLING appears on the display.
- e. The scrambling process can be interrupted at any time, by pressing the touch sensor for about 1 second.

8. Stop a scrambling or a solving cycle:

The scrambling or solving process can be interrupted at any time, by pressing the touch sensor for about 1 second.

In case the push button is maintained pressed longer, a warning is presented to the display.

Very long touch time (5 to 6 secs) of the touch button, is interpreted as intention to quit Cubotino-T.py script.

9. Raspberry Pi shut down:

Please be noted Raspberry Pi, like normal PC, cannot be unpowered when it is working.

To shut it down, there are a few possibilities:

- a. Connect to the Raspberry Pi via SSH, and type `sudo halt -p`

The SBC closes the open applications and files

When the SBC activity is almost done, the led at Connections board goes off

Wait additional 10 seconds and the power supply can be safely removed (the hole on the Structure, above the Raspberry Pi, allows to sneak and check if the RPI “working” led is off.

- b. If the robot has proved to work without errors, un-comment last row at Cubotino_T_bash.sh file (`halt -p`); This means the SBC will automatically shut down when the `Cubotino_T.py` file ends.

To quit the `Cubotino_T.py` script, keep touched the Touch_button long enough (ca 6 seconds) until the ‘SHUT DOWN’ appears on display; The SBC will close the open applications and files.

If the touch sensor is released as soon as the display shows ‘SURE TO QUIT?’, then the robot will consider the request as a stop request instead of a shut-down request.

When the SBC shut-down process is almost done, the led at Connections board will goes off.

Through the Structure hole check if the green led of the Raspberry Pi Zero is off; This typically takes additional 10 seconds, afterward the power supply can be safely removed.

If the cover_self_close parameter has been changed to ‘true’, the top cover will be closed automatically at the Raspberry Pi shutdown (at python script closure).

This action is anticipated by some info on the display: Please be aware this might pose a risk to your kids if they have their hand on the way while the cover closes!

10. Un-power the servos:

This considers an independent power supply for the servos.

Servos can be unpowered at any moment.

Un-power the servo before shutting down Rpi, if you experience strange servo behaviour when Rpi goes off; The script takes care to set the servo PWM related GPIO to a fix level (low), at the shut-down, but it does not work on 100% of the times.

11. Running the robot from VNC Viewer:

Here is explained how to run the robot from VNC Viewer, when the python script has been started via the Bash file at Raspberry Pi boot, and ‘halt -p’ command is uncommented in the bash file.

Under these circumstances:

- it is not an option to quit the script from the robot, by keeping the touch button pressed long, as the Raspberry Pi will shut down
- it is not possible to run a ‘new’ script over the first one , as will conflict with Camera resources; It is necessary to quit the running python scrip first.

Steps to do:

- a. Connect VNC Viewer to the robot
- b. Open a terminal
- c. Folder is not relevant
- d. List all the running processes via *ps aux*

USER	PID	%CPU	%MEM	VSZ	RSS	TTY	STAT	START	TIME	COMMAND
pi	624	0.0	0.2	4488	808	?	Ss	16:24	0:00	/usr/bin/ssh-ag
pi	638	0.9	0.3	8760	1296	tty1	S+	16:24	0:00	-bash
pi	642	0.2	0.9	43400	3692	?	Ssl	16:24	0:00	/usr/lib/gvfs/g
root	657	0.0	0.2	7676	1048	?	S	16:24	0:00	bash -l /home/p
pi	664	0.2	0.9	56752	3384	?	S1	16:24	0:00	/usr/lib/gvfs/g
root	674	33.2	27.3	360008	102152	?	Rl1	16:24	0:13	python Cubotino
pi	684	1.2	2.1	62392	8132	?	S	16:24	0:00	openbox --confi

- e. Search for python Cubotino process and note the ID (674 on the above example); This is by far the process that takes more CPU and memory resources, making easier to find it.
- f. Search for bash command, from root user, located above python Cubotino, and note the ID (657 on the above example)
- g. First kill the bash process *sudo kill -9 ThePIDNumberForBash* (by using the above example the command will be *sudo kill -9 657*)
- h. After kill the python process *sudo kill -9 ThePIDNumberForPythonCubotino* (by using the above example the command will be *sudo kill -9 674*)

```
pi@raspberry:~ $ sudo kill -9 657
pi@raspberry:~ $ sudo kill -9 674
pi@raspberry:~ $
```

Note: by reversing the order on steps **g** and **h**, the Raspberry Pi will shut off right after the Cubotino python process is killed: Not the wanted result 😞

Section4: Tuning and robot operation

12. “Remote” usage, for automated scrambling and solving cycles:

In case you’d like to run many cycles, for statistical purpose, or you’d like to place your robot in a shop window, it is possible to automatically scramble and solve the cube for a given quantity of cycles.

Run the robot from VNC Viewer (see previous bullet point), and add below arguments to Cubotino_T.py:

argument	parameter	notes
-- cycles	Int > 0	<ul style="list-style-type: none"> Quantity of consecutive scrambling and solving cycles If not provided, the robot will wait for commands from the touch button
-- pause	Int > 0	<ul style="list-style-type: none"> Wait time in between the automated scrambling cycles (time in seconds) If not provided, there won’t be waiting time between cycles It does work only if the --cycles is also provided
-- shutoff	No parameters	<ul style="list-style-type: none"> If provided, the RPI will be shut off at the end of the automated cycles. If not provided, after the automated cycles, the robot will wait for commands from the touch button It does work only if the --cycles is also provided

Examples:

- `python Cubotino_T.py --cycles 10` will scramble and solve the cube 10 times.
- `python Cubotino_T.py --cycles 15 --pause 300` will scramble and solve the cube 15 times, by applying a pause of 300 seconds in between the automated cycles
- `python Cubotino_T.py --cycles 20 --pause 3600 --shutoff` will scramble and solve the cube 20 times, by applying a pause of 3600 seconds in between the automated cycles; After the last cycles the python script will be quitted, and RPI will shut off (see Rpi shut down via Touch button, to automate the shut-off).

Notes:

1. Active cycle and the total cycles plotted on robot screen.
2. Remaining time for next cycle plotted on robot screen.
3. When “--pause” is provided, on the robot screen is indicated the remaining time to start the next cycle.
4. When “--cycles” is provided without “--pause” argument, there won’t be pause between cycles.
5. On PC connected to the robot (via VNC) it is shown the top cube face after each solving cycle. The image remains on screen until the next scrambling cycle starts.
6. Image on PC is completed with last performed cycle and total cycles, additionally to date and time.
7. The image is not saved.
8. During the waiting time in between automated cycles, feedback is also provided to the CLI: The percentage of the waiting time and the left time in seconds.



```
#####
END SOLVING CYCLE 1 #####
Next cycle: [.....] 29.5% 14s
```

20. Automatic robot start

It is possible to have the robot starting up automatically, when the Raspberry Pi boots.

From the root or from the venv: `sudo crontab -e`

The first time you'll be asked to choose an editor, use 1 for nano

```
(.venv) pi@cubotino:~/cubotino $ sudo crontab -e
Select an editor. To change later, run 'select-editor'.
 1. /bin/nano      <---- easiest
 2. /usr/bin/vim.tiny
 3. /bin/ed

Choose 1-3 [1]: |
```

Un-comment the last row

```
'MAILTO=""'
@reboot su - pi -c '/usr/bin/vncserver :0 -geometry 1280x720';
#@reboot /bin/sleep 5; bash -l /home/pi/src/cubotino/Cubotino_T_bash.sh >
/home/pi/cubotino/src/Cubotino_T_terminal.log 2>&1'
```

Note: Eventual arguments (see “How to operate the robot”) you would like to get at the automatic robot start, can be added to the `Cubotino_T_bash.sh` file.

Example: change `python Cubotino_T.py` to “`python Cubotino_T.py - -timer - -F_deg`”

21. Rpi shut down via Touch button

The Raspberry Pi shut down can be initiated via the touch button (long press), by enabling that function:

From the folder `/home/pi/cubotino/src`, edit the file with `sudo nano Cubotino_T_bash.sh` and uncomment the last row

```
# 'halt -p' command shuts down the raspberry pi
# un-comment 'halt -p' command ONLY when the script works without errors
# un-comment 'halt -p' command ONLY after making an image of the microSD
#halt -p
```



Note: Do not uncomment the last row (`halt -p`):

1. Before being sure the robot code runs without errors; A little indentation error sometimes happened when changing a parameter.
2. Before having made an image of the microSD.

The following chapters where initially at the instruction start, introducing and explaining the project bases.

Since October 2022, I changed the order to prioritize the robot construction.... so, if you find weird the instructions order, at least you know the reason.

I still hope the below information will be useful and appreciated 😊

22. Introduction

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

That robot is based on a Raspberry Pi 4B (2Gb ram) with a PiCamera, it reads the cube status via a 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>

That robot works simply fine, I had lot of satisfaction from it, I learned a lot on different areas....yet that robot has clear drawbacks:

- The cost, as there are about 150euro of components.
- Another limiting factor is the box size, a bit too large for most of the domestic 3D printers.

23. 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:

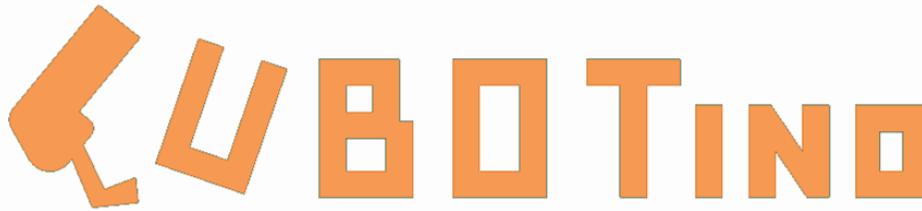
- The Base version must be as cheap as possible
- The mechanical part should be the same for all the versions
- The robot should be scalable (in automation, and consequently in complexity/materials cost)
- The robot should not require changes to the cube for gripping
- Compact design
- Fully 3D printable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

24. 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 **C**U**B**OT and **i****n****o** (**i****n**o** is 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:



25. Models

This project considers the robot to be scalable.

The idea is to develop three robot versions, 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	<ul style="list-style-type: none"> • Cube status entered on the GUI, via mouse or PC webcam • Cube solution (Kociemba) generated at PC 	Finalized (April 2022) Link
Medium	PC dependent, for cube solution	<ul style="list-style-type: none"> • Cube status detection at the robot • Cube solution (Kociemba solver) generated at PC 	Stand-by, see notes below
Top	Autonomous	<ul style="list-style-type: none"> • Cube status detection at the robot, via a vision system • Cube solution (Kociemba solver) generated at the robot 	Finalized (June 2022) Link
Micro	Autonomous	<ul style="list-style-type: none"> • This essentially is scaled down from the Top version. 	Finalized (March 2023) Link
Pocket	Autonomous	<ul style="list-style-type: none"> • For the "pocket" Rubik's cube (2x2x2 version) 	Finalized (January 2024) Link

Notes for the Medium version:

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 has proved to don't be sufficient; 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 the Medium version....

Considering the Base version has been “published” on early April 2022: I'm pretty sure in little time people will design their own version ... and probably the Medium version will come to live in this way 😊

26. High level info (Top version)

1. Re-uses most of parts from the base version; Only the PCB_cover must be re-printed for this Top version.
2. The robot is based on a Raspberry Pi Zero2 (WH) with a 16Gb microSD.
3. A PiCamera (Ver 1.3) is used to detect the cube status; Camera is placed at an angle with respect to the cube.
4. Python CV2 library is used for the computer vision part.
5. A led module is used to reduce the influence from the ambient light conditions.
6. A small display provides feedback on the robot task/progress.
7. All coded in Python.
8. 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
9. 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
10. Cube's orientation considers the Western colour scheme (<https://ruwix.com/the-rubiks-cube/japanese-western-color-schemes/>):

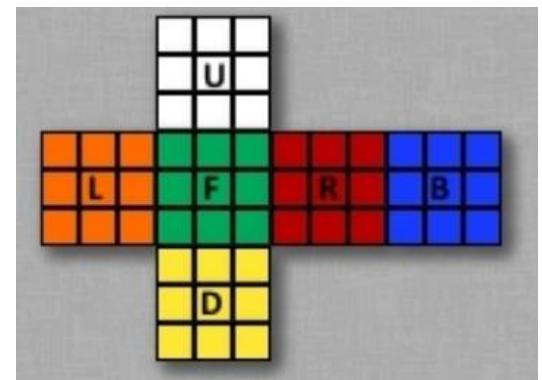
The Western colour scheme (also known as BOY: blue-orange-yellow) is the most used colour arrangement used not only on Rubik's Cubes but on the majority of cube-shaped twisty puzzles these days.

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



11. 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>
12. When rotations are express as CW, or CCW, it is meant by facing the related cube face. For the servo the same rule is applied: CW and CCW are meant from the motor point of view.
13. 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.
14. The robot detects the cube status on cubes with and without the black frame around the facelets.

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

27. Construction

The robot mechanical targets remain the same of the base version:

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 centre 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 the camera+led 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. This robot has 2 pivots total, both at the servo's axes; I believe Tilted Twister has a total of 8 pivots....
5. 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 from Hans on 2008 and 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.

28. 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 little robot, the computer vision part is achieved by combining the below elements:

- **Raspberry Zero 2 SBC** (the computer part)
- **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).
- **PiCamera** (a camera module, highly integrated with Raspberry Pi)

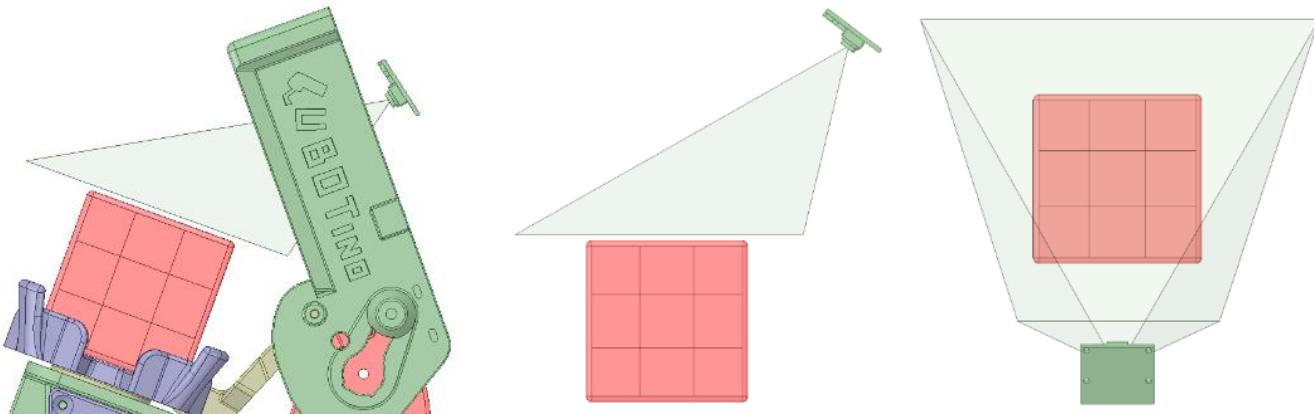
In which the python script '**Cubotino_T.py**' is responsible for the interaction with these elements.

Below listed aspects, are presented on the next pages:

1. Camera positioning
2. Taking consistent images
3. Image analysis
4. Contour analysis
5. Colour retrieved
6. Is all of this really needed?

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

- A. To get images, everything starts with positioning the camera on the right location:



The initial idea was to position the camera parallel to the cube upper face, yet I ended up with the solution depicted by above pictures. The reasons are:

1. The flex cable for Raspberry Pi Zero is max 30cm long; This prevented the possibility to mount the camera on an extension of the Top_cover (like I've done on my first robot).
2. Need to move the camera as far as possible, to have sufficient Field of View (FOV); Obviously a complete cube face must be fully visible by the camera

This construction gives some drawbacks:

1. the Top_Cover easily produces shadow on the cube; This affects the facelets colour uniformity, therefore the robustness to always read (and assign) the correct colours.
2. the cube facelets have a relevant perspective; This makes more difficult to evaluate if a detected contour is really a facelet, by evaluating if fitting a square shape.

To solve the above problems:

1. A controllable LED has been placed close to the camera; This removes the shadows generated by the Top_cover, and it reduces the overall sensitivity to the ambient light conditions.
2. The cube image is artificially warped, prior the facelets edge analysis, below an example

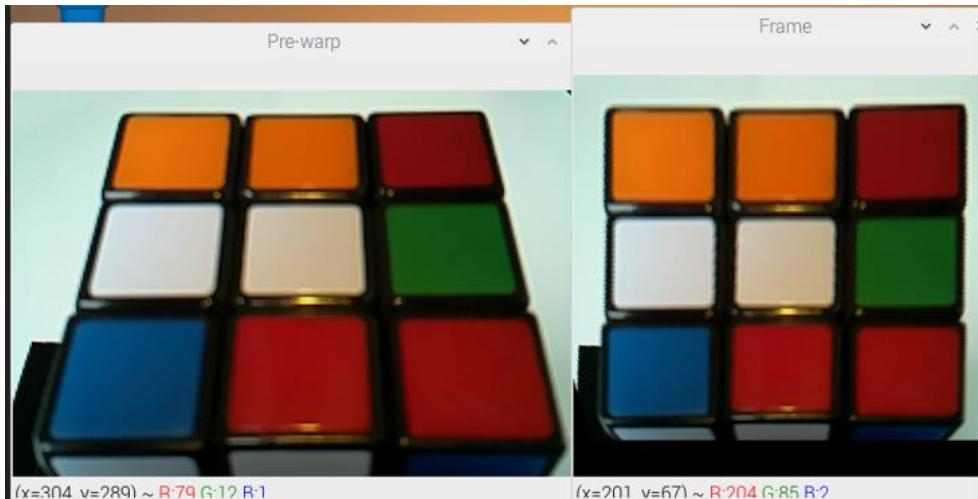


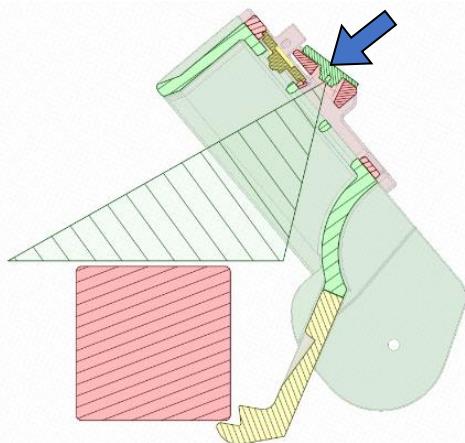
Image warping requires some computation power, but it does not affect the overall timing.

B. Taking consistent images

This is a crucial aspect for proper colour analysis.

The light source addition is a good way to mitigate the environment light conditions, typically out of our control.

On below an image it is visible the position of a 3W LED light source:



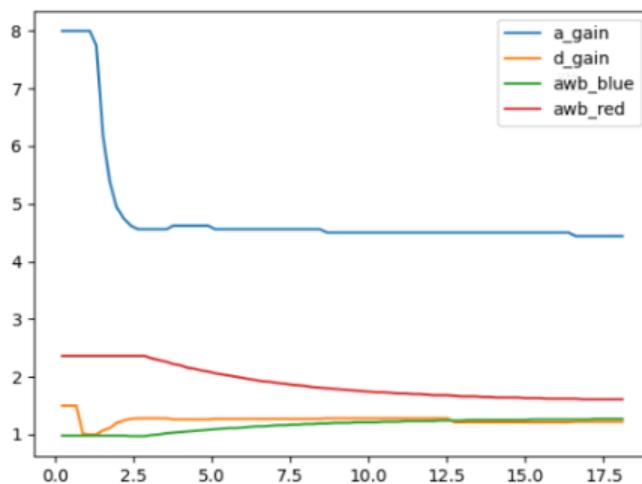
Notes on the chosen LED module:

- It has 120-140 deg angle
- It is controllable by the MCU
- Used only a fraction of the 3W max

When the robot is requested to detect the cube status, the LED and Camera are both activated.

The camera is initially set in auto mode and inquired on a series of parameters: Analog gain, Digital gain, AWB (Auto White Balance) and Exposure time.

Below the variability of these parameters in time (X axis is in secs), based on measurements made on the robot:



PiCamera gains (range 0 to 8), and AWB, are plotted versus time (secs).

In this case the cube was placed after 2 secs from pressing robot start-button; This means the camera was initially adjusting the gains on the black cube support, and right after it had to adjust on the cube (with some white facelets): **It's clear that AWB adjustment takes quite some time to get stable**

Differently, if the cube is placed on the cube support few secs before pressing the button, then the gains are already well set.

To cover these situations, a so called 'warm-up' function is implemented in Cubotino_T.py script: Once all these parameters are within 2% from the average value (average is calculated from the function start), then the camera is switch to manual mode and the average parameters values are set to the camera; This process takes typically a couple of seconds, but it can take up to 20 seconds if large parameters variation occurs.

This procedure is only done on the first cube face, and it gives a first good estimation about the ambient light conditions.

Afterward, the cube is flipped four times and the Exposure time measured on each of these 4 sides.

The camera is then set to fix shutter time, with the average value detected on 4 out of 6 faces; Of course, it will be even better to measure the Exposure time on all 6 cube faces, but only 4 faces are quick to get because of the robot construction.

The camera is now set to take consistent images.

Having the LED and camera angled, has turned out to be beneficial to reduce the light reflection.

C. Image analysis:

The approach uses a similar technic as explained at <https://medium.com/swlh/how-i-made-a-rubiks-cube-color-extractor-in-c-551ccea80f0>

1. The warped image is converted to gray scale: `gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)`
2. The grayscale image is filtered with a low pass filter to reduce noise:
 - on classic cube types (frameless_cube set ‘false’): `blurred = cv2.GaussianBlur(gray, ...)`
 - on frameless cube type (parameter_cube set ‘true’): `blurred = cv2.bilateralFilter(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,...)`
 - when parameter_cube is set ‘auto’ the canny filter is applied on both blurred images, and the two results are (OR) combined in a single canny `canny = cv2.bitwise_or(canny_01, canny_02,...)`
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,...)`
 Anyhow I preferred to use a different kernel than Dilate, and keep rather thick edges
6. The Eroded binary image is now used to find contours: `cv2.findContours(image, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)`

D. 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 Facelets.

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 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
 - b) Max area deviation, from the median one
 - c) Max sides length difference, from a pre-defined threshold
 - d) Max diagonals length difference, from a pre-defined threshold
 - e) Max distance from the central one; This step includes ordering the 9 contours, according to their centre coordinates.
 - f) Quantity of contours left, after discharging those not ok
5. The first 9 contours, passing through this process, are then used as masks; These masks are applied on the coloured warped image, as guidance for the facelets position.
 In case the frameless_cube is set ‘true’ or ‘auto’, as soon as 7 contours are detected the remaining two are estimated for their position.
6. The ‘accepted’ contours are plot over the coloured warped image, as visual feedback.

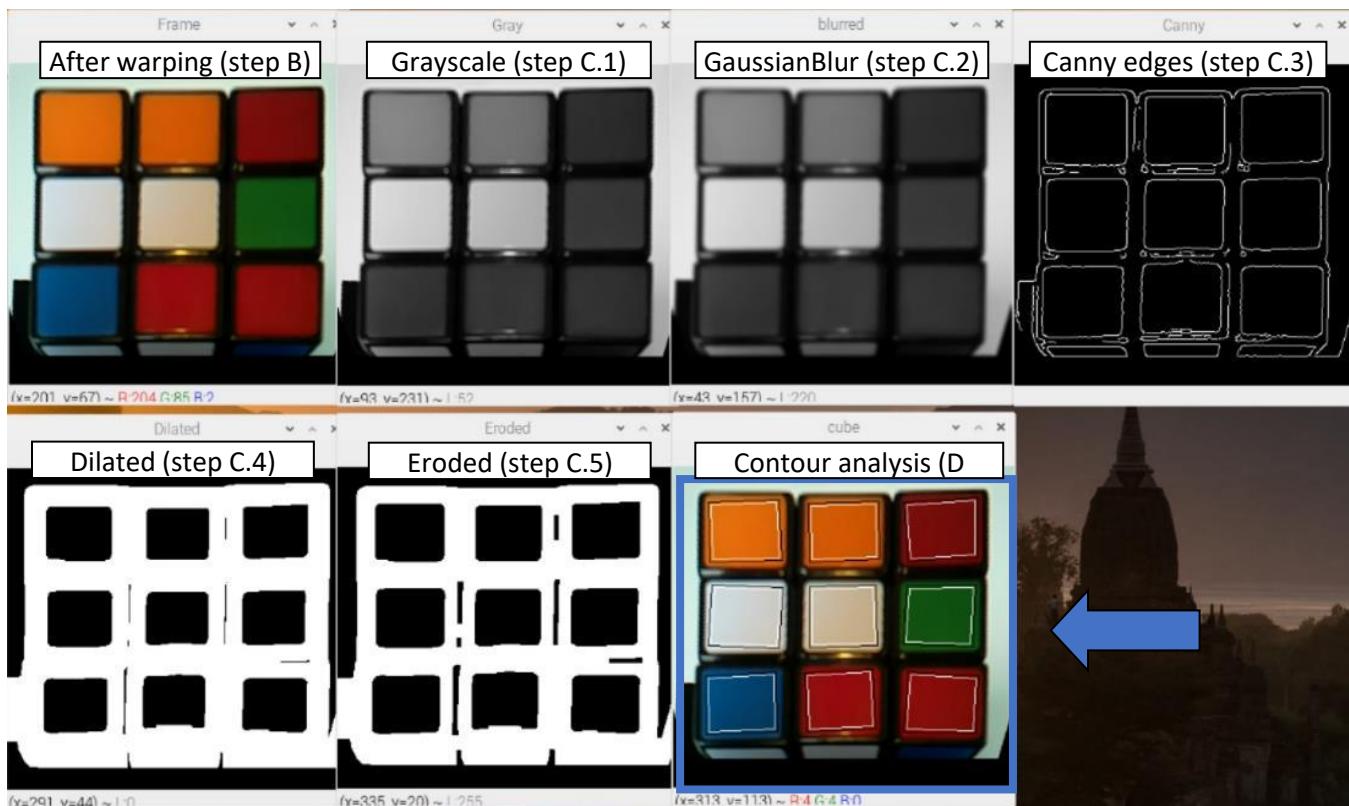
E. Colours retrieved:

On each Facelets, are the retrieved 2 main info:

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

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

(If you'd like to see these images on screen, run `python Cubotino_T.py -cv_wow`, a FHD screen/setting needed)

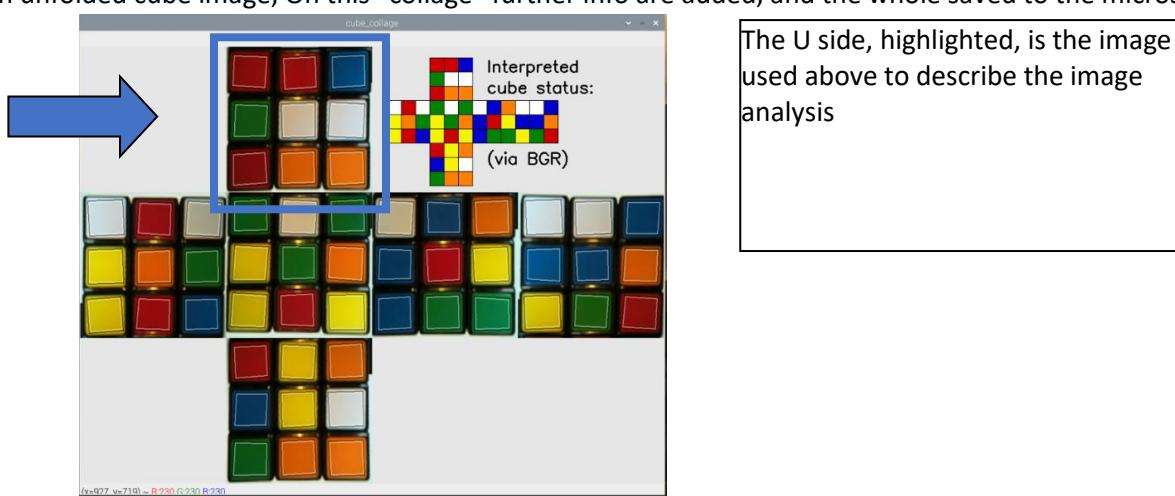


Not all images are oriented as per user point of view: Sides UBDF are 180deg rotated.

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

The last processed image of each side, the one with the ‘accepted’ contours, are stored in RAM.

Once the full cube status is detected, these images are further cropped (based on the detected contours) to generate an unfolded cube image; On this “collage” further info are added, and the whole saved to the microSD.



F. Fix Coordinates System (FCS):

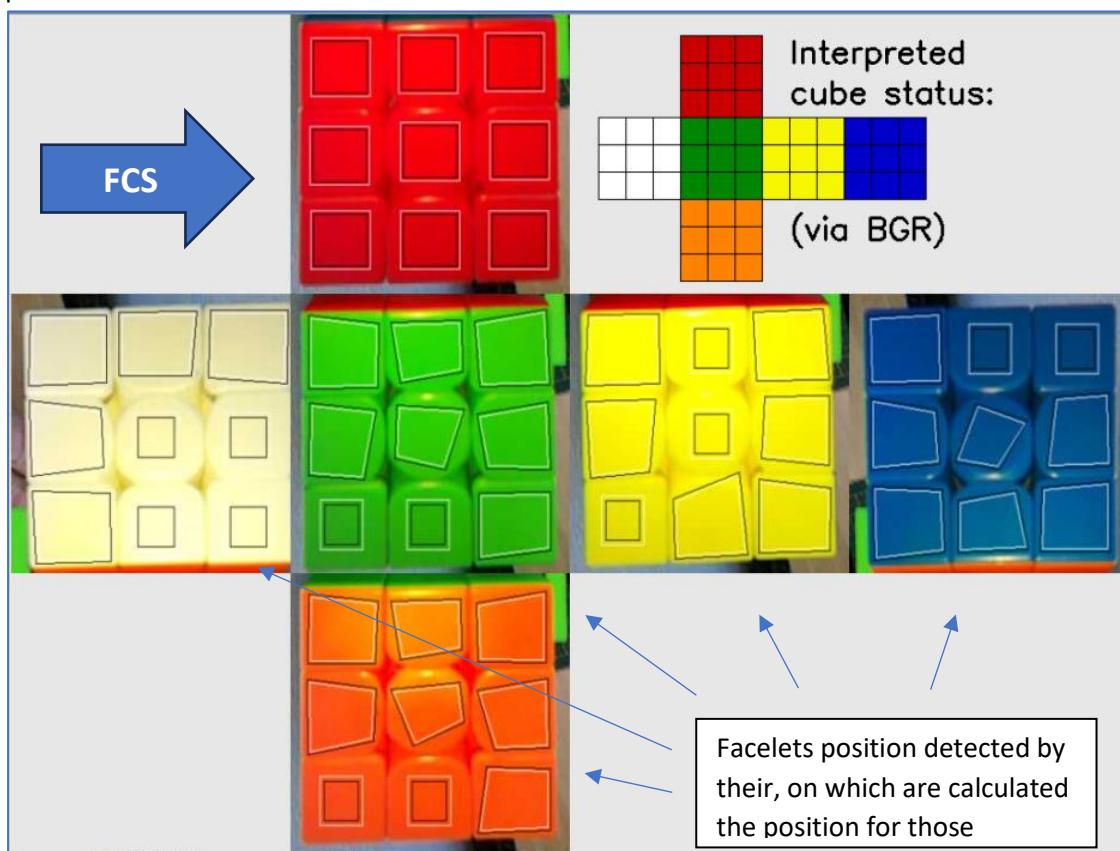
Starting from 21st January 2024, facelets position can be estimated via an additional method (FCS), below the main aspects:

- The FCS algorithm is started:
 - On any cube's face.
 - After the fcs_delay time elapses (priority is given to the edge detection method).
 - If at least one facelets is seen (this happens rather easily, yet not if a monochrome piece of paper is placed instead of the cube).
- As the wording suggest, predefined coordinates are used to point the cube face and retrieve the colour information. This implies that the algorithm will proceed even when there isn't any cube on the Cube_holder.
- Fix coordinates are automatically generated 😊.
- The coordinates are generated every time a scanning cycle successfully completes, by detecting the facelets position via the edges detection system (in other words, when FCS is not called).
- New generated coordinates are appended to the Cubotino_T_coordinates.txt file.
- The coordinates are loaded from the file once the Cubotino_T.py is launched.
- Coordinates are averaged from the last "n" values.
- When the Cubotino_T_coordinates.txt file becomes too long, the older values are removed.
- By lowering the fcs_delay to 0, then the FCS algorithm will start immediately. This can make the cube scanning faster, but it won't update the coordinates. As result, it might lead to errors in case of mechanical drift or new servos settings.

On the below example, the facelets detection of the U face were done via the FCS.

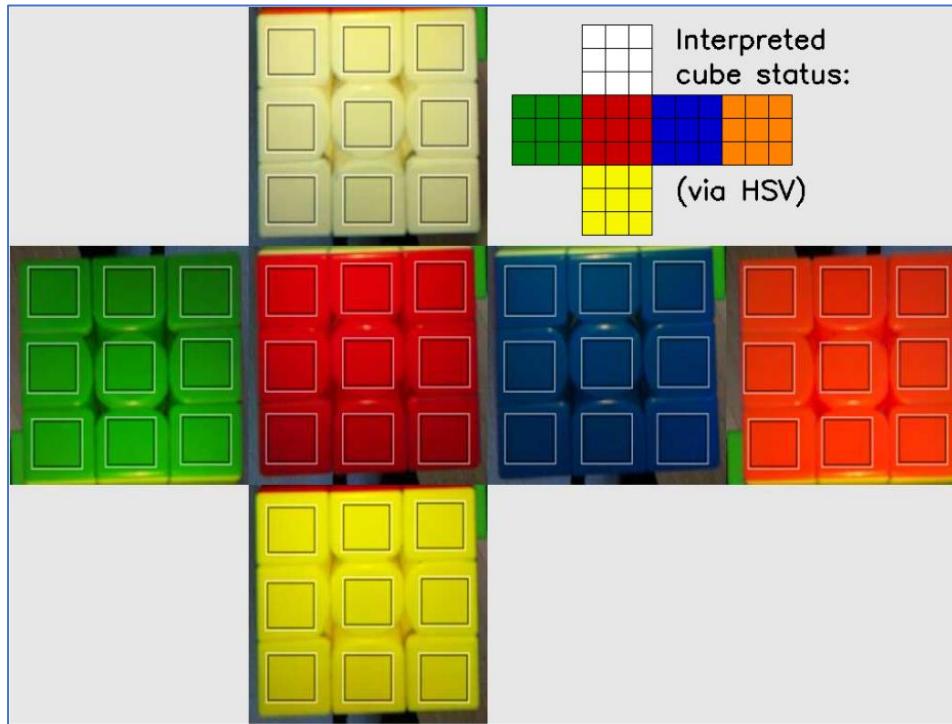
It is recognizable because all the contours have the same dimensions and are positioned as per a regular array.

On all the other faces most of facelets are detected via their edges, and the remaining are estimated in their position.

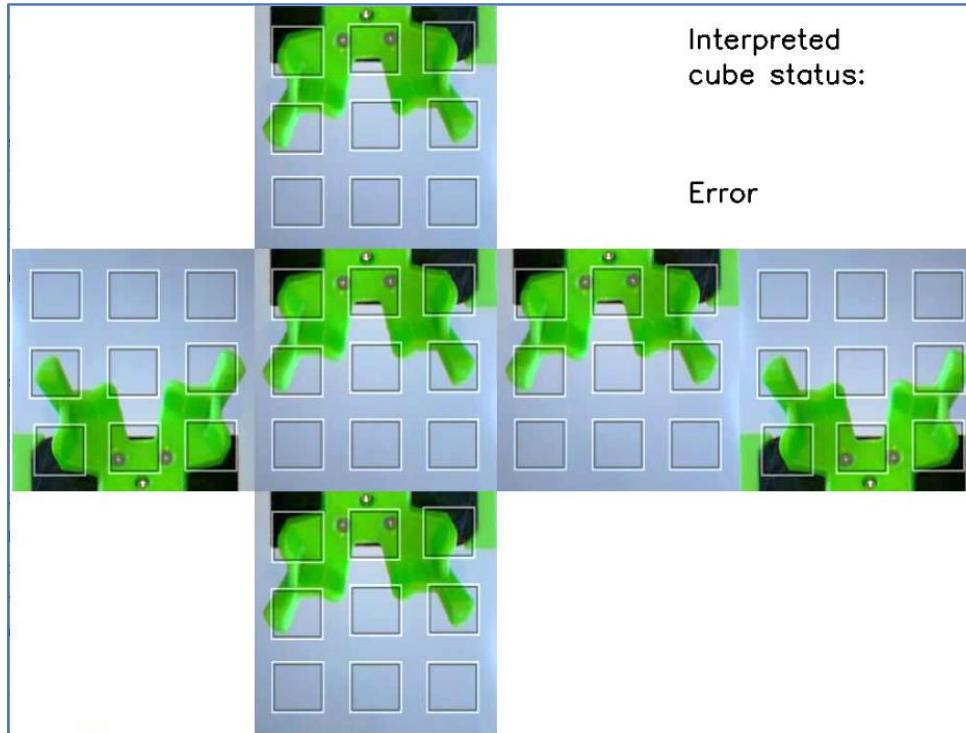


If the fcs_delay is set to 0 (zero), it means no delay time to start using this approach. As result, priority will be given to the Fix Coordinates System over the edge's detection one.

If a cube is on the Cube_holder, then it works and it might gain some speed



If the cube is not on the Cube_holder, the algorithm just proceeds until an error is returned:

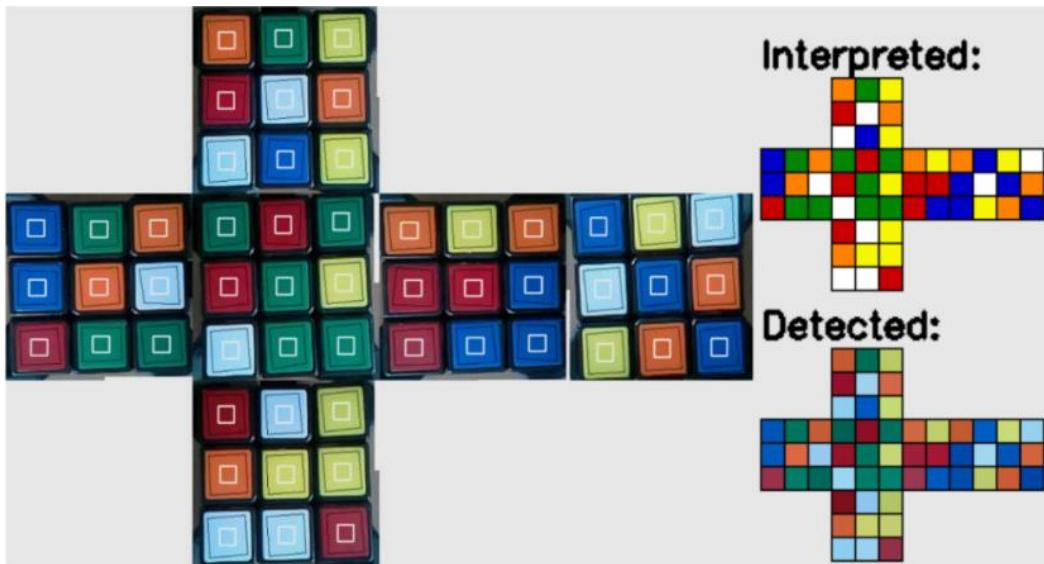


Colour's detection strategy

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

Based on the identified contours:

- 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 centre coordinates.
- The inner one, in white on below picture, depicts a smaller square area centred on the outer contour; This smaller area is used to:
 - Measure the BGR average value, used for the colour interpretation according to the 1st method (BGR colour distance)
 - calculate the HSV average colours, used for colour interpretation according to the 2nd method (Hue value).



- Properties of the faces centre facelets:

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

- These facelets don't move (fix facelets number).
- These facelets have (obviously) 6 different colours.
- 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 centres, is used to determine which colour is located on the 6 centres:

- White facelets 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.
- Remaining 4 centres are evaluated according to their Hue, and the Hue at opposite face.
- 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.
- Out of the two remaining centres, 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 centres) 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.

6. Facelets colour interpretation is made, by using two methods, via a tentative approach:
 - a. The first method compares the average RGB colour of each facelets, in comparison with the one at the 6 centres, and the colour decision is based on the smallest colour distance. The Euclidian distance of RGB per each facelets if calculated toward the 6 centres.
 - b. In the second method the Hue value of each coloured (non-white) facelets are compared to the Hue of the 5 reference centres; White facelets are retrieved according to 3 parameters (Hue, Saturation, Value), in comparison to the white centre 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.

7. Facelets position estimation

The “frameless” parameters defines whether the algorithm searches the contour for all the 9 facelets or for at least 5 of them.

Frameless	Default	Description
false	Yes	The algorithm searches for all the 9 facelets contours
true	No	The algorithm searches for at least 5 facelets contours, that should cover 3 different rows and columns of the array; More commonly 7 contours are found. The position and the area of the remaining facelets will be estimated based on those detected.
auto	no	The algorithm uses the “Frameless false” approach on the first image of each cube face; In case the 9 contours aren’t found, then the algorithm changes to “Frameless true” from the second image onward.

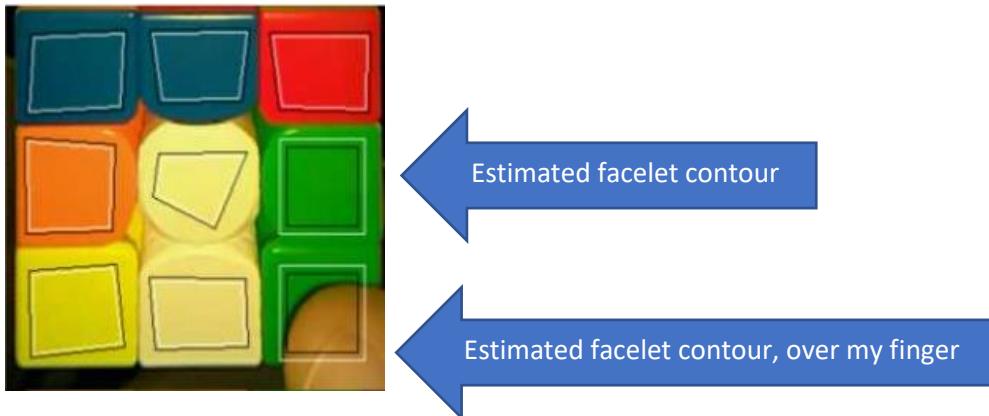
You might argue on the reason ‘auto’ is not the default choice for the cube facelets detection method.

The reason is that the used strategy for the frameless cube is less robust.

When frameless_cube is set ‘true’ or ‘auto’, as soon as there are at least 5 detected facelets, without an empty row or column, the remaining facelets will be estimated on their position.

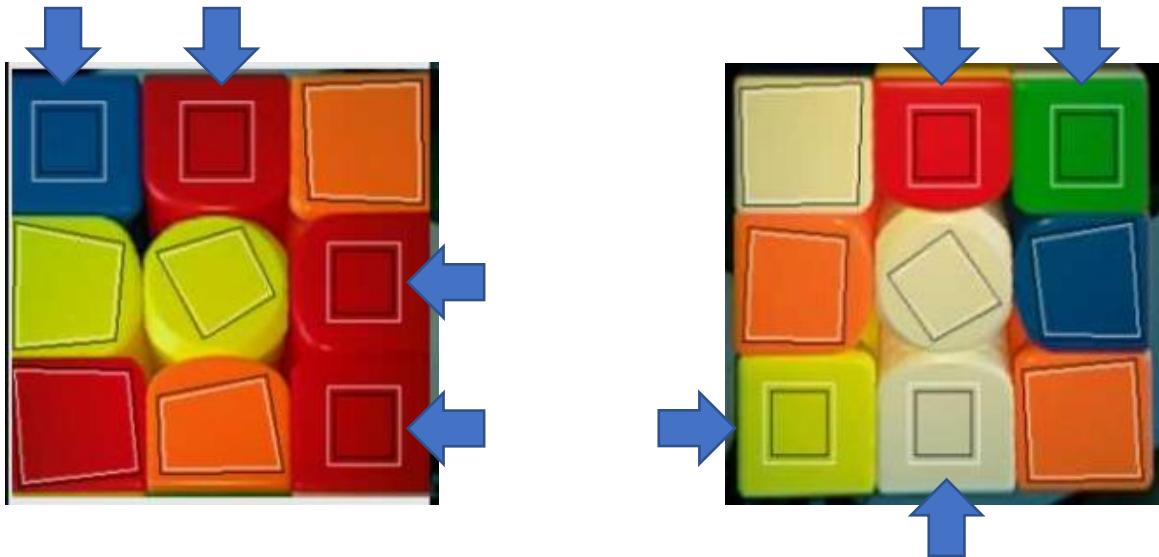
This approach helps when adjacent facelets have the same colour, but it doesn’t prevent your finger (or cube logo) to get captured: See below image, with my finger not been evaluated by the algorithm, as it was on the “last two” facelets, and later leading o wrong colour detection from that facelets (likely cube status detection error)

Estimated facelets have the white contour placed outside the black one.



Examples of estimated facelets:

On below examples 4 facelets have been estimated; To be noted all the rows and column have at least one detected facelets:



Be noted one of the estimated facelets goes to the central white one with the logo.

When the facelets have a printed logo, a defect, or some pollution, it will make more difficult to be detected as facelets, therefore those facelets will end on those estimated.

In this case it will be rather possible to get a detection error, as the average colour retrieved from that facelets isn't very similar to other white facelets.

Apart from the printed logo, below example shows again 4 adjacent estimated facelets.



29. 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_T_moves.py file.

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

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

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

The robot solver follows instead the below approach:

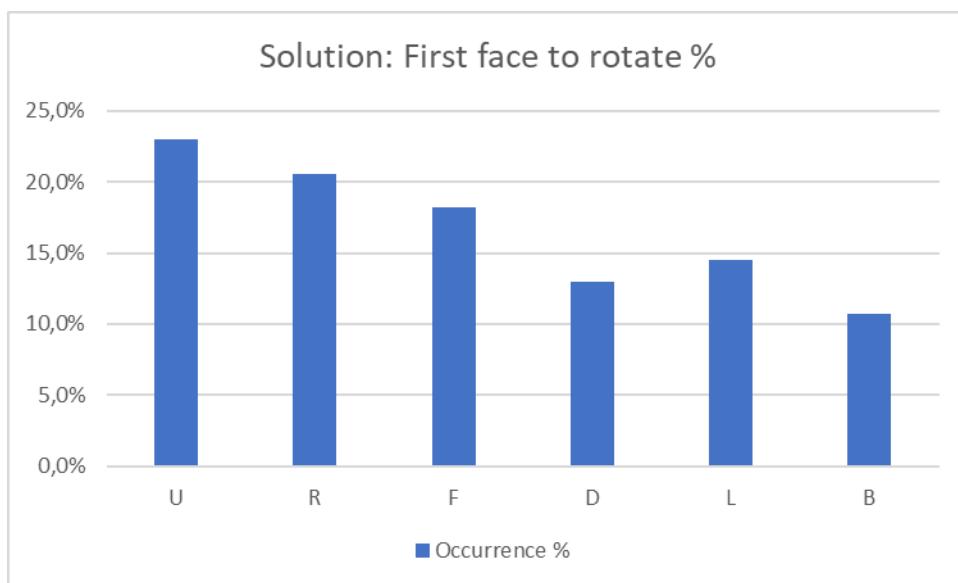
1. All the 18 possible cube moves (U1, U2, U3, , B1, B2, B3) the solver can return, are used as keys in a dictionary; There are 18 sets of robot movements (hard coded) associated to these keys. These robot movements consider the cube as ideally positioned: U side facing up and F side facing front.
2.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

On November 2022 I took some time to look back at the information saved in the log files of my 3 robots.

I had a total of about 1300 manually scrambled cubes, after removing those with less than 10 rotations.

On below table the occurrences of the first rotation face, returned by the Kociemba solver:

First face move	Cubotino_black	Cubotino_green	Cubotone	Tot	Occurrence %
U	42	87	173	302	23,0%
R	56	73	141	270	20,6%
F	40	71	128	239	18,2%
D	36	41	93	170	13,0%
L	36	63	91	190	14,5%
B	30	32	78	140	10,7%
			Tot	1311	



Based on these cases, the Kociemba solver proposed a solution having higher chances URF to be the first move; Furthermore, the chances the first move will be on U face is higher than all the other faces.

To be noted the time given to the solver is very limited, therefore rather expected the URF faces having the large percentage.

After scanning the cube, my original and simplistic approach has been to move the cube to the “Initial position”: This means the U face is facing upward, and D face is laying on the Cube_holder.

Based on the above analysis, there were only 13% chances to apply the first move (face rotation) on D face, therefore, to apply a face rotation without flipping the cube first.

Starting from 27/11/2022, after scanning the cube, it is not anymore moved to the “Initial position”.

This means the cube is laying on its R face (20% chances to be the first face to rotate) and with a single flip the U face will be on the Cube_holder.

With this new approach it should be possible to save some robot movements

30. Python main scripts, high level info

1. *Cubotino_T.py* is the main python script on the robot. This script imports other custom files.
2. *Cubotino_T_settings_manager.py* (from 10th March 2024) is the only file interacting with the settings file. It serves as a Class to which the other scripts refer to get/modify/save the settings. Settings are organized in two Json files, for easier management, setting and communication purpose.
3. When the script *Cubotino_T.py* is started (eventually automatically at the Raspberry pi boots), the script checks if there are monitors connected. The monitor can also be via VNC, i.e. with VNC Viewer. The presence/absence of a monitor is needed to use/skip commands requiring graphical screen communication. This prevents errors, further than having a better experience.
4. Kociemba solver is tentatively imported from different locations; venv, active folder and ‘twophase’ sub-folder under active folder.
5. The script uses a “tentative” approach, on a couple of analysis:
 - a. (See Colour detection strategy chapter for more info) When the image is analysed, it returns contours of facelets and many unwanted ones; This happens in the function *get_facelets()*. Afterward, consecutive filters are applied to only keep contours having cube facelets requisites. This process ends when 9 facelets, all matching the filters criteria, are retrieved from a single image
 - b. (See Computer Vision chapter for more info) When determining the cube status, according to the facelets colour; The analysis starts with a first method determining each (side and corner) facelets colour, based on the colour distance from the colours of the 6 centres. In case the cube status obtained with this first method is not coherent, then a second method is called. The second method uses the Hue value of each (non-white) facelets, by comparing it to expected (predefined) Hue ranges, adapted upon the Hue measured on the 6 centres. In case also the second method doesn’t provide a coherent cube status, then an error message is returned, and relevant info logged in a text file.
6. Kociemba solver:

Kociemba solver is uploaded at the start; In case of multiple cubes solving, no need to reload it

The detected cube status, with URF notations, is sent to the Kociemba solver.

The solver, with the chosen parameters, returns the best-found solution within the time-out; The solver doesn’t provide the absolute best solution, as it is too computational (and time) expensive, yet it typically returns a solution with 20 movements or less. Very rarely, the solution has 21 movements, mostly because of the chosen time-out of ‘only’ two seconds.

The solver returns an error if the cube status is not coherent; This info is then used to attempt the second colour assignment method, or to stop by providing error feedback to the display.
7. From cube solution to robot movements:

(see Robot solver algorithm chapter for more info) Cube solutions, in Singmaster notation, sent to *Cubotino_T_moves.py* that returns a (long) string with the sequence robot movements. Movements are Spin, Rotate, Flip.
8. From cube robot solution to robot movements:

Robot solution string, in Cubotino notation, is sent to *Cubotino_T_servos.py* that operates the servos to actuate all the intended movements.

9. Data logged:

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. 20220428_213439
FramelessCube	Setting of the parameter frameless_cube at Cubotino_settings.txt
ColorAnalysisWinner	The approach that has returned a coherent cube status; Possible strings are 'BGR', 'HSV' and 'Error' (when both approaches did not provide a coherent cube status)
TotRobotTime(s)	Time, in seconds, from pressing the start button, until the cube is solved. From 27/11/22 this time is forced to zero if the robot is stopped
CameraWarmUpTime(s)	Time, in seconds, from pressing the button, until the robot ends all the camera settings
FaceletsDetectionTime(s)	Time, in seconds, from pressing the button
CubeSolutionTime(s)	Time, in seconds, used by the Kociemba solver to return the solution
RobotSolvingTime(s)	Time, in seconds, to solve the cube from when the cube solution is available
CubeStatus(BGR or HSV or BGR,HSV)	Dictionary with the average colours per each facelet, according to the colour space of the winner detecting method; In case both the detecting methods have failed, then the average colour returned by both the colour spaces are reported
CubeStatus	Cube status, in URF notation: i.e. RLFDUUDBBLFRURRBDDDRDDFFLURULDRFDLFLUFDLBRLRFLBUBFUBBLBU
CubeSolution	Cube solution string, in Singmaster notations: i.e. D2 L2 F2 R3 F2 L1 D2 F2 R3 U2 F1 L1 F3 R1 B2 F3 D3 L2 F2 (19f)
Screen	Since rev.3.17 Indicates if a graphical desktop (i.e. VNC viewer) was connected. Possible strings are "screen" or "no screen" (no data).
Flip2close	Since rev.5.0 Indicates if the Top_Cover (and Lifter) moves from the Flip position to the Close position with a single movement (default) or in two steps. When two steps it means there is a short stop at the Read position, that gives some time to the cube to seat into the Cube_holder before closing the Top_cover. Possible values are 1 or 2
OS_ver	Operative System version (10 or 11).
FCS	0 means the facelets were detected via their edges. Value > 0 indicates how many faces the Fix Coordinates System was used.

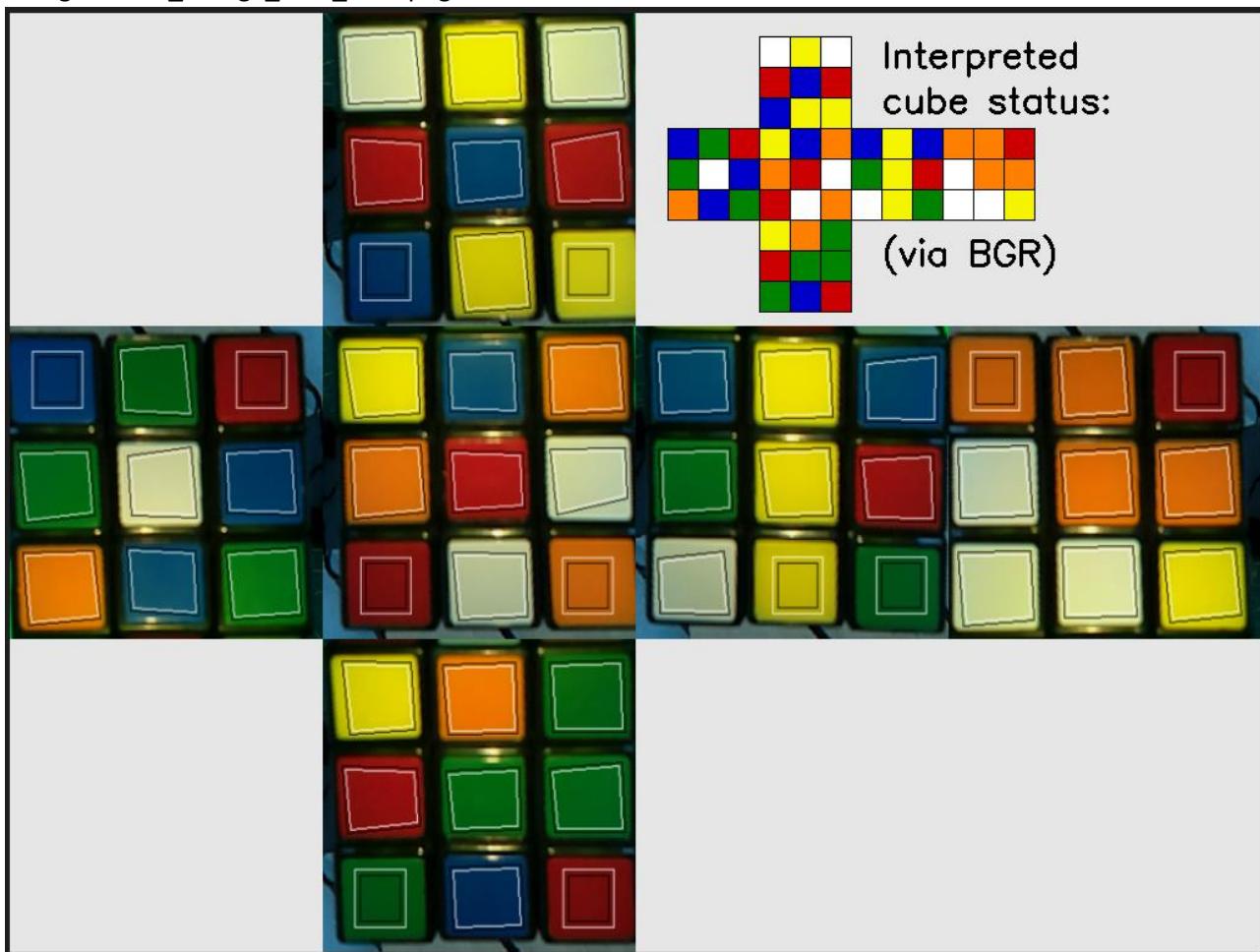
Notes:

1. The folder `Cube_data_log` is made from the folder where `Cubotino_T.py` is running.
2. The logged data is saved in the `Cubotino_solver_log.txt` file.
3. Text file uses tab as separator.

Further than saving data in the text file, a picture of the unfolded cube status is also saved

Folder: CubesStatusPictures

Images: cube_collage_date_time.png



10. Animation:

After solving a cube, a simple animation is plot on the display.

The animation reproduces all the cube facelets movements that the robot has applied.

At the animation end, a solved cube appears, with the same orientation the real cube has on the Cube_holder.

The animation:

- can be prevented via the argument '`--no_animation`'.
- It is not executed if the cycle gets interrupted.
- It is also plot to the screen, with much bigger size, in case a screen is connected (also via VNC).

11. Date, and especially time, are used by the robot:

Raspberry pi doesn't have an integrated RTC, therefore when the robot isn't connected to a PC and/or internet, this info could be inaccurate.

If the robot establishes a connection to the Wi-Fi, the system time gets updated, yet this will alter the robot time calculation if the update comes when the robot is solving a cube.

To prevent this problem from happening, the robot script checks at the start-up if there is an internet connection, and in that case, it waits until the system time is updated before proceeding.

In case there aren't internet connections, the robot simply proceeds with the non-updated system time.

In my view this approach is sufficient for reliably timing the robot performances.

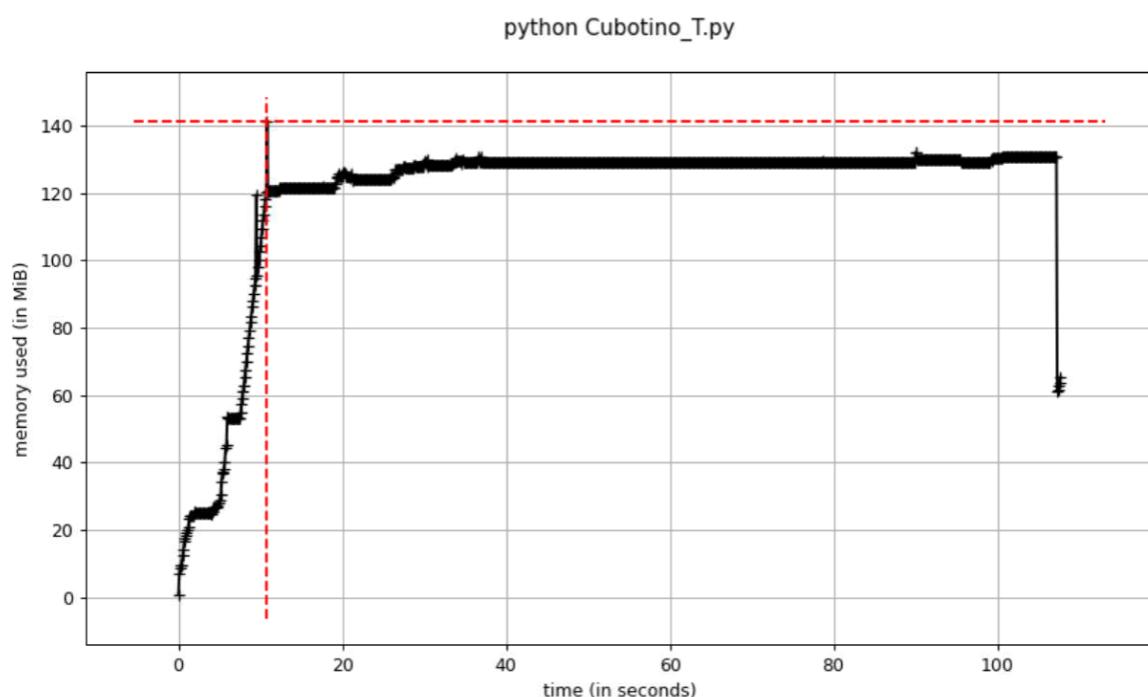
12. Memory profiling

Before running Cubotino_T.py, the available memory is ~ 180Mb (with some little swapping):

```
(cv) pi@raspberry:~/cube $ free -h
              total        used        free      shared  buff/cache   available
Mem:       364Mi       114Mi      133Mi        21Mi       116Mi      179Mi
Swap:      99Mi        86Mi       13Mi
(cv) pi@raspberry:~/cube $
```

Without VNC Viewer there is somehow lower memory usage; Below plot includes a full cycle:

- import libraries
- read and solve a scrambled cube
- quit the script



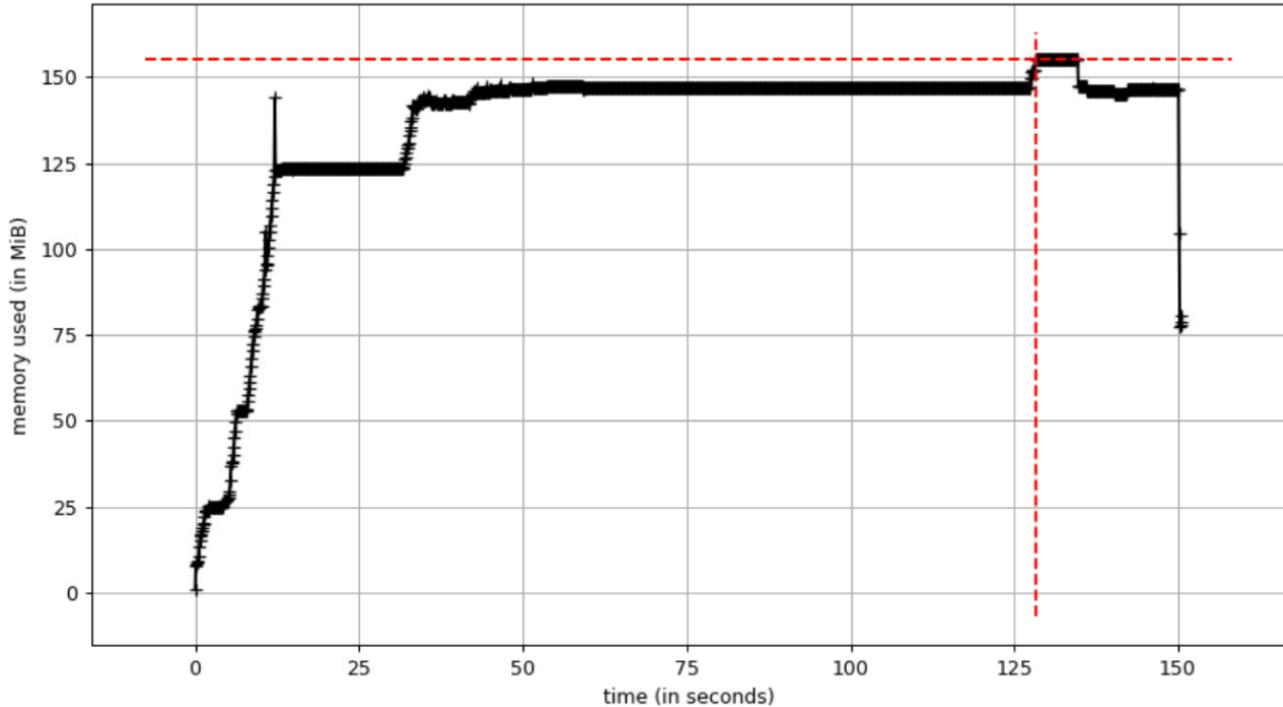
Results:

- 1) There is a peak of about 140Mb at the end of the library import
- 2) Free memory will be on the order of 40 to 50Mb

With VNC Viewer there is somehow larger memory usage; Below plot includes a full cycle:

- import libraries
- sharing of graphical information on PC screen, via VNC
- read and solve a scrambled cube
- quit the script

python Cubotino_T.py



Results:

- 1) There is a peak of about 155Mb, when the unfolded cube picture collage is shared on screen
- 2) After uninstalling mprof, the memory situation at the unfolded cube picture collage isn't critical, because of the swap memory:

```
pi@raspberry:~ $ free -h
              total        used        free      shared  buff/cache   available
Mem:      364Mi       198Mi       26Mi       13Mi      139Mi      102Mi
Swap:     99Mi        80Mi       19Mi
```

- 3) The swap memory is left on its default value of 100Mb
- 4) the swapiness is also left to its default value of 60.

The project has been developed / tested with:

- Linux raspberry 5.10.103-v7+ #1529 SMP Tue Mar 8 12:21:37 GMT 2022 armv7l GNU/Linux
- Python version: 3.7.3 (default, Jan 22 2021, 20:04:44) [GCC 8.3.0]
- CV2 version: 4.1.0
- VNC Viewer (6.20.529 r42646 x64), connected via SSN, to interact with the Raspberry Pi; This also includes file sharing.

31. Virtual solver

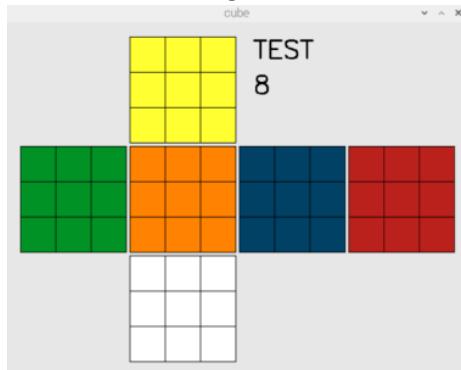
Starting from 1st April 2024, CUBOTino has a virtual solver: **Cubotino_T_test_random.py**

The virtual solver:

10. Uses the Kociemba solver to generate a random cube status.
11. The random cube status it sent to the solver to get the solution.
12. The solution is sent to the robot solver to get the necessary moves
13. The robot moves are individually, virtually, applied.
14. After each movement the cube status is updated.
15. After the virtual manipulation of the cube, the final cube status is achieved: The final cube status is verified if conforming to a solved cube.
16. It saves the relevant data in a text file (tab separated data).
17. Considers the servos' timers to estimate the solving time.
18. Returns the expected servos time, servos movements, and how often the optimizations (1 and 2) are used.

The quantity of tests can be passed with ‘-- runs’ argument (i.e., ‘-- runs 1000’), otherwise it stops after 100 tests.

There are other arguments available, like –plot to visualize the solving on the screen:



Below how the outcome looks like, after testing 1'000 random cube status:

```
#####
Tested 1000 random cube's status with 0 failures
The test took 810.34 seconds
Average time to analyze each cube_status: 810.3 ms

Average quantity of robot movements: 96.2
Average estimated solving time (servos): 79.5
std on estimated solving time (servos): 5.287
Min estimated solving time (servos): 59.6
Max estimated solving time (servos): 95.1

Optimization type 1 being used: 595
Optimization type 2 being used: 25

Saved the cube depth and estimated servo time, of the 1000 runs
File saved at: /home/pi/cubotino/src/TestRandom_1000cubes_20240401_201240.txt
```

The virtual solver is a tool for CUBOTino development, to verify the effectiveness of different moves strategies 😊.

A variant of it has been used to compare the servos' time from Paul's code vs the original code (180 deg. Rotation instead of 2x90deg), by using the same random generated cube status, thus a fair comparison (Paul's code is ca 13% faster than the original, with standard deviation of 5.8%).

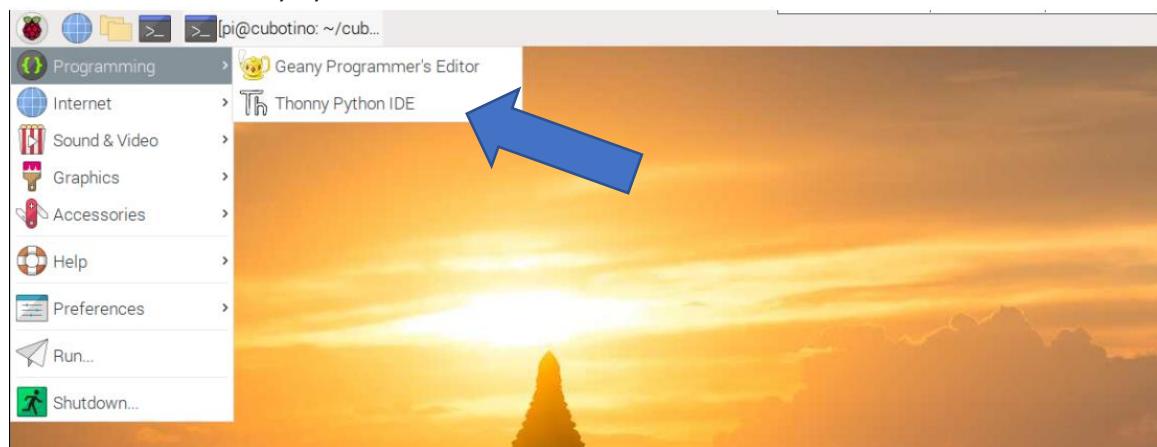
32. Set Thonny IDE interpreter

In case you'd like to see or change the python script, it will be handy to use Thonny as it also offers the possibility to run the script and test your changes.

from Wikipedia: Thonny is an integrated development environment for Python that is designed for beginners. It supports different ways of stepping through the code, step-by-step expression evaluation, detailed visualization of the call stack and a mode for explaining the concepts of references and heap.

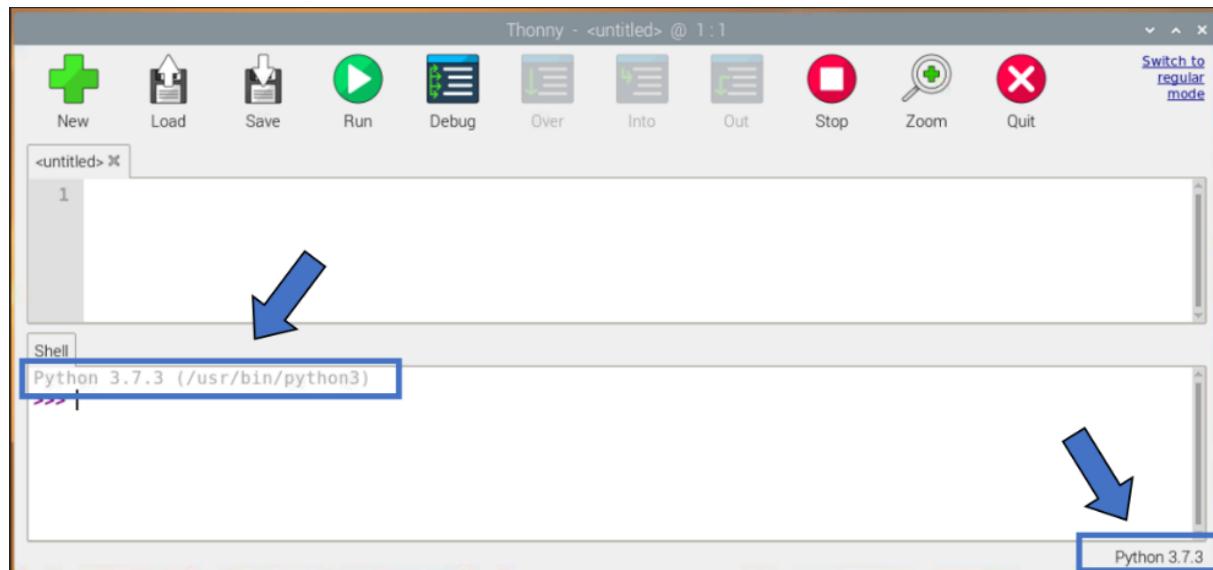
Thonny is part of the Raspberry Pi installation (according to the 'Setting up Raspberry Pi' procedure):

1. Access the Raspberry Pi via VNC, for instance via VNC Viewer
2. At Raspberry Pi, open the applications menu
3. Select Programming
4. Choose Thonny Python IDE

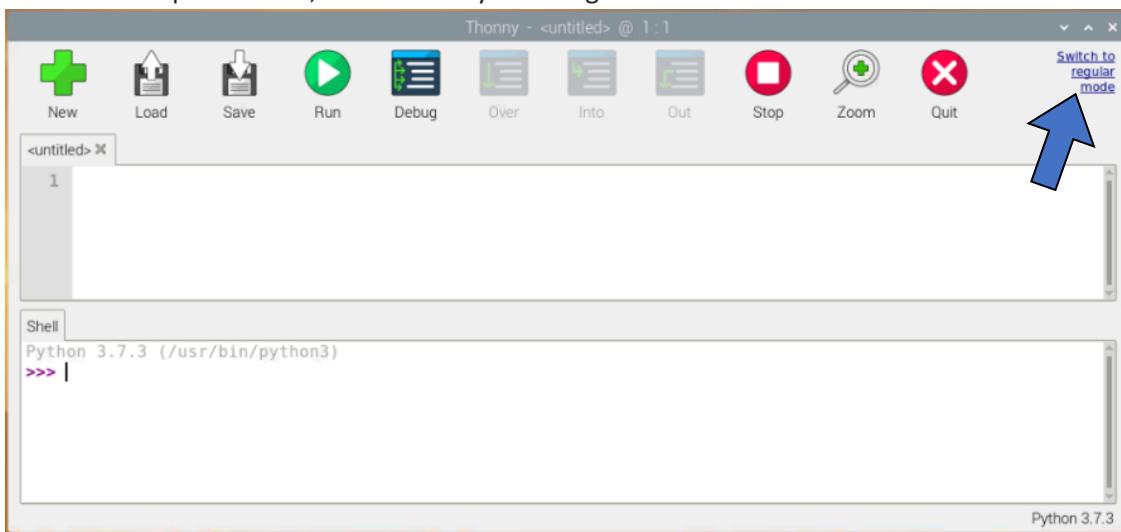


Setting up Thonny IDE interpreter, to work with the venv, it will be handy to tune the parameters hard coded in the scripts.

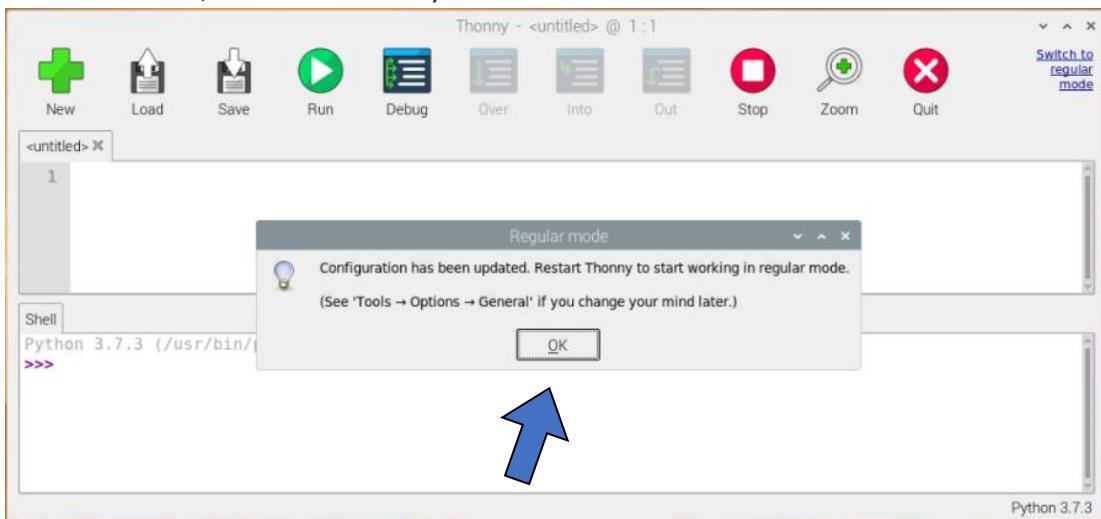
Thonny opens with the standard interpreter (/usr/bin/python3), and if you run Cubotino_T.py it won't find the libraries....



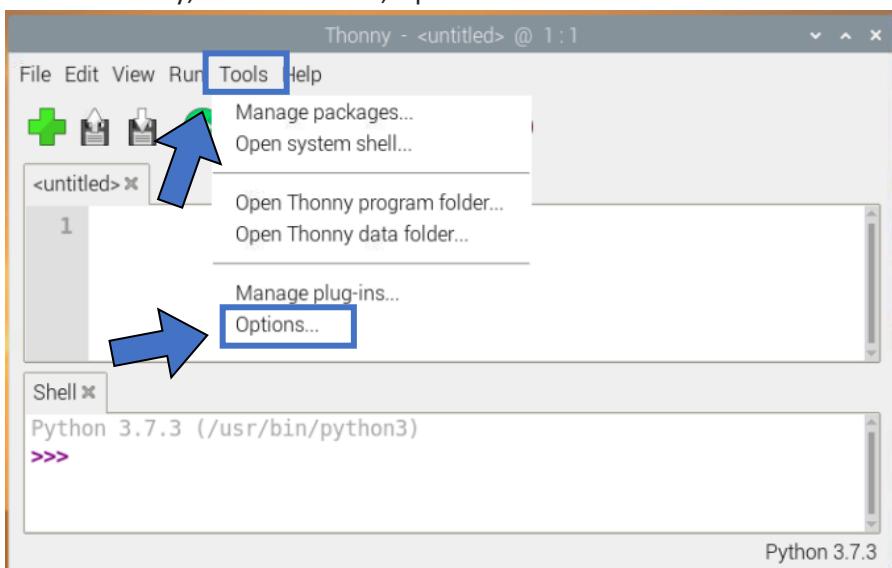
To have the Option menu, it is necessary to change the mode:



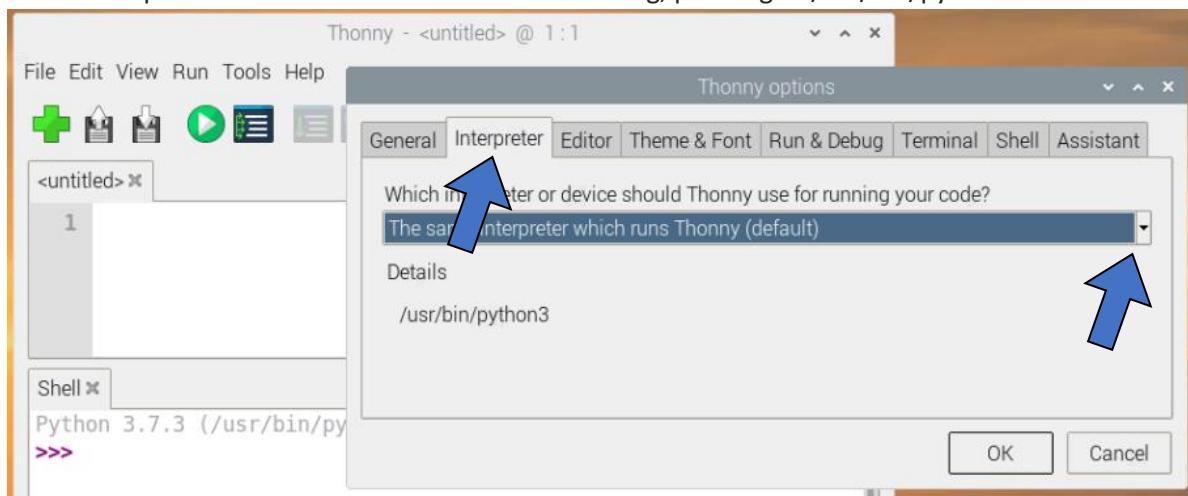
Confirm the info, and restart Thonny



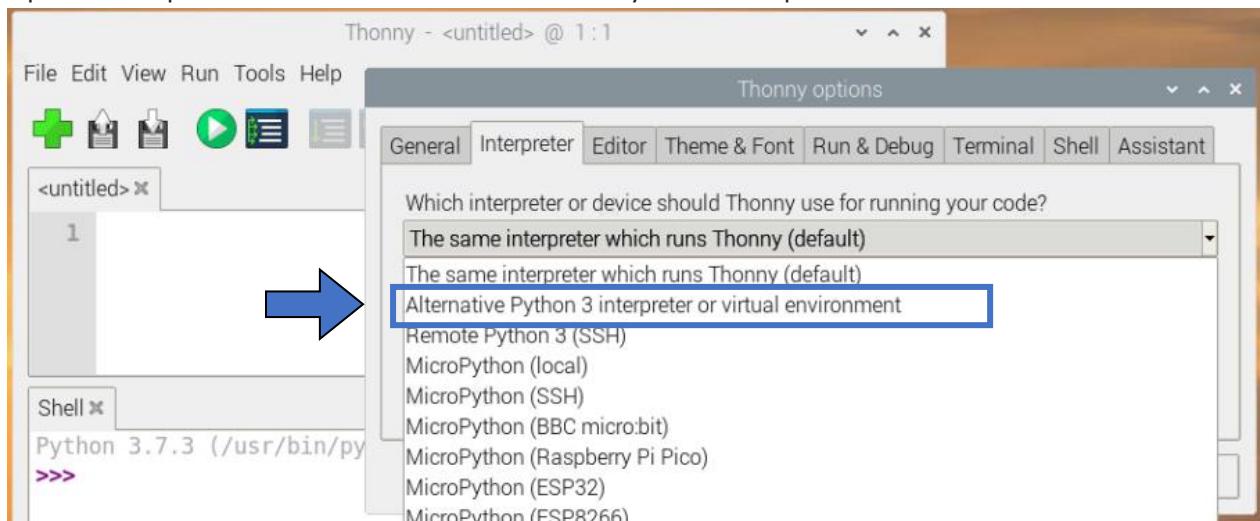
Restart Thonny, and select Tool, Option



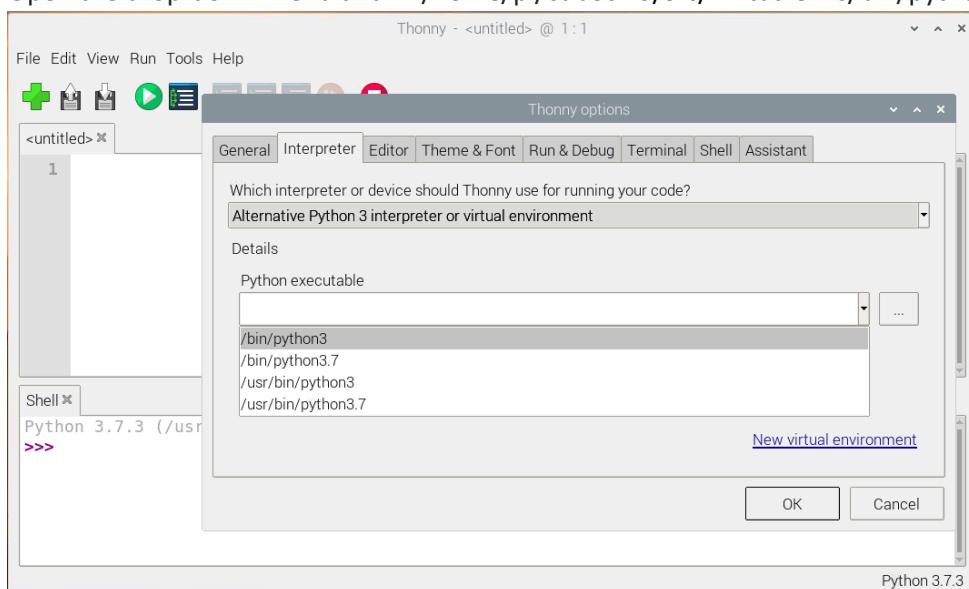
Select Interpreter where it is shown the default setting, pointing to /usr/bin/python3.



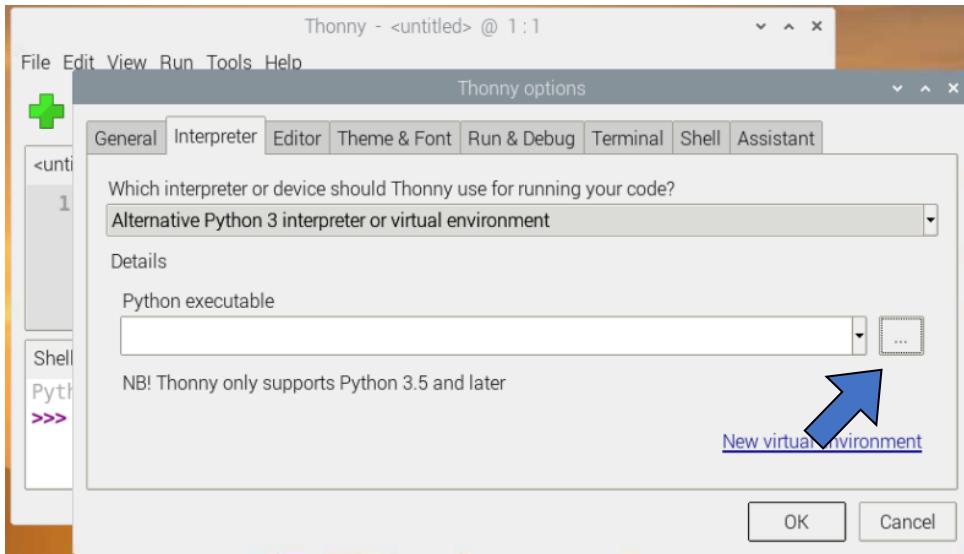
Open the drop-down menu and select 'Alternative Python 3 interpreter or virtual environment':



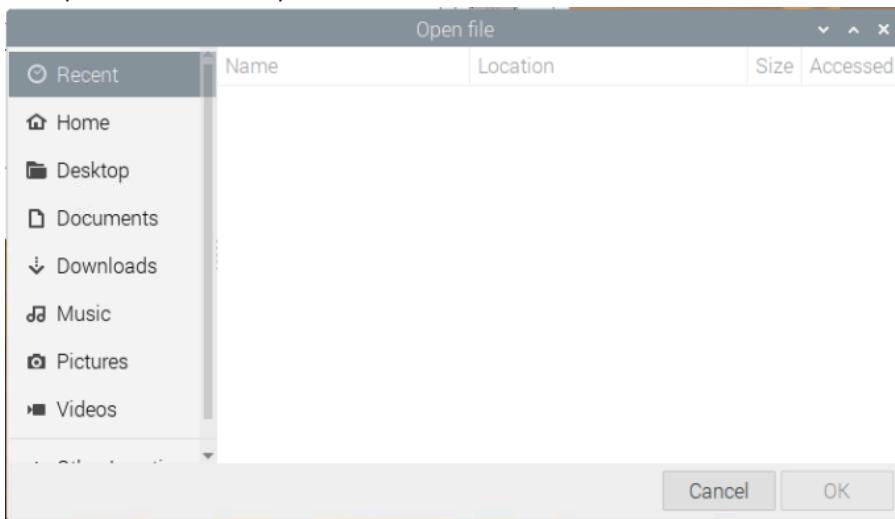
Open the drop-down menu and if '/home/pi/cubotino/src/.virtualenvs/bin/python3' is listed just select it



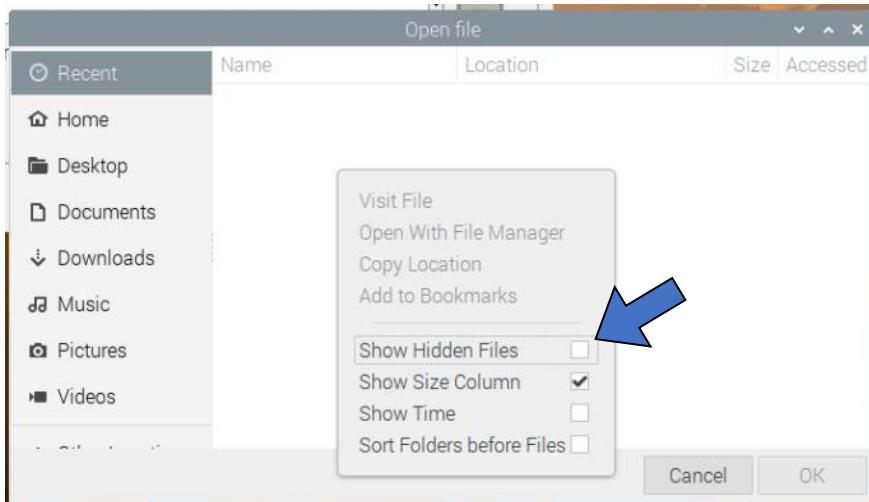
If ‘/home/pi/cubotino/src/.virtualenvs/bin/python3’ is not listed, select the browse button:



An Open file window opens:



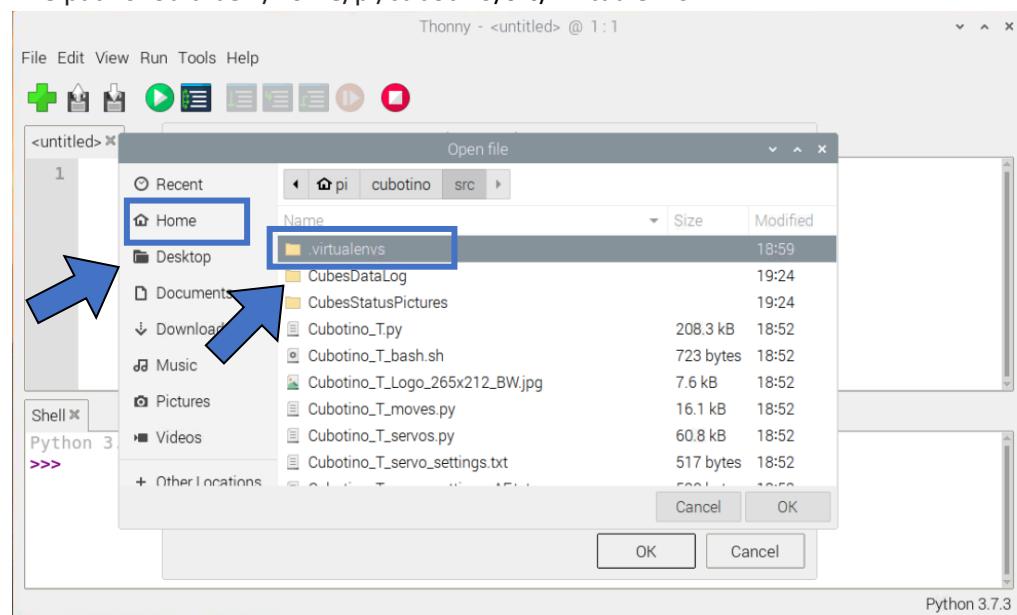
Right click on the empty window part, and check ‘Shows Hidden Files’:



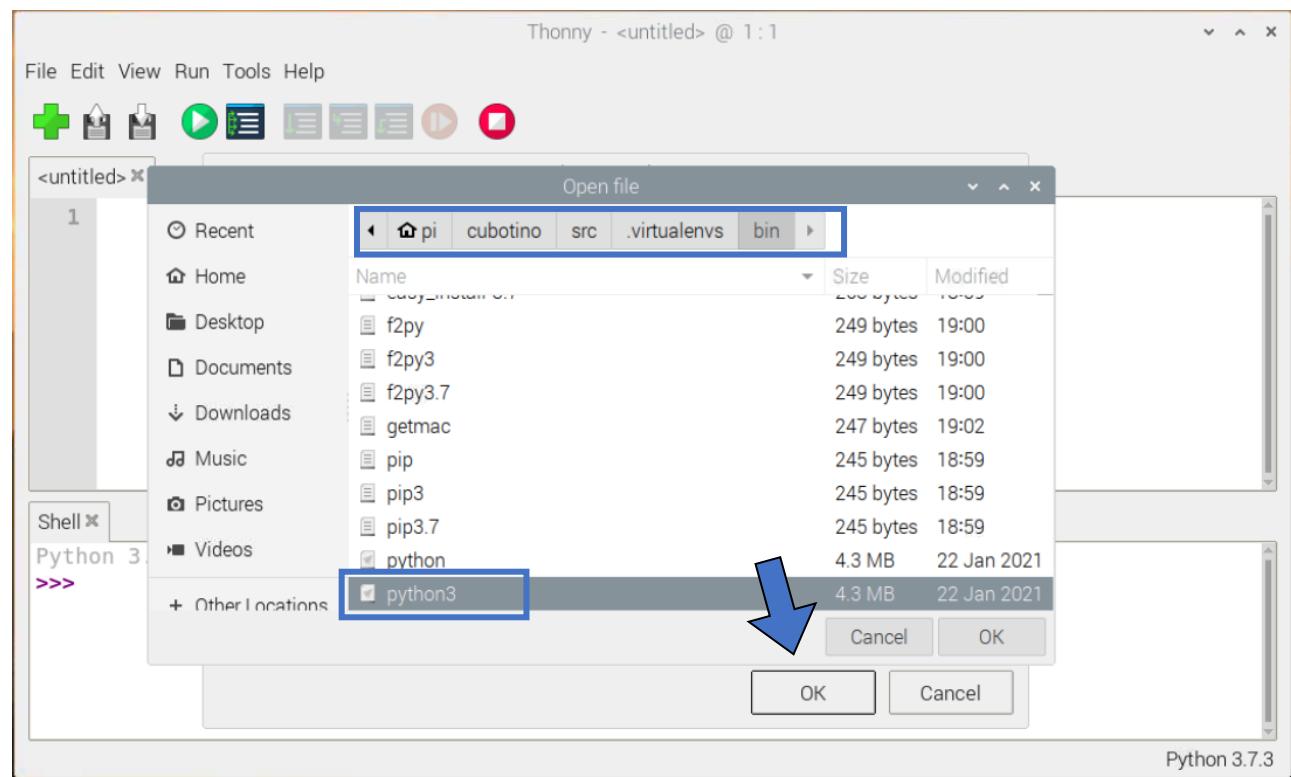
Section5: Info

Select Home, .virtualenvs should appears (Note: all folders and files starting with a dot are hidden type)

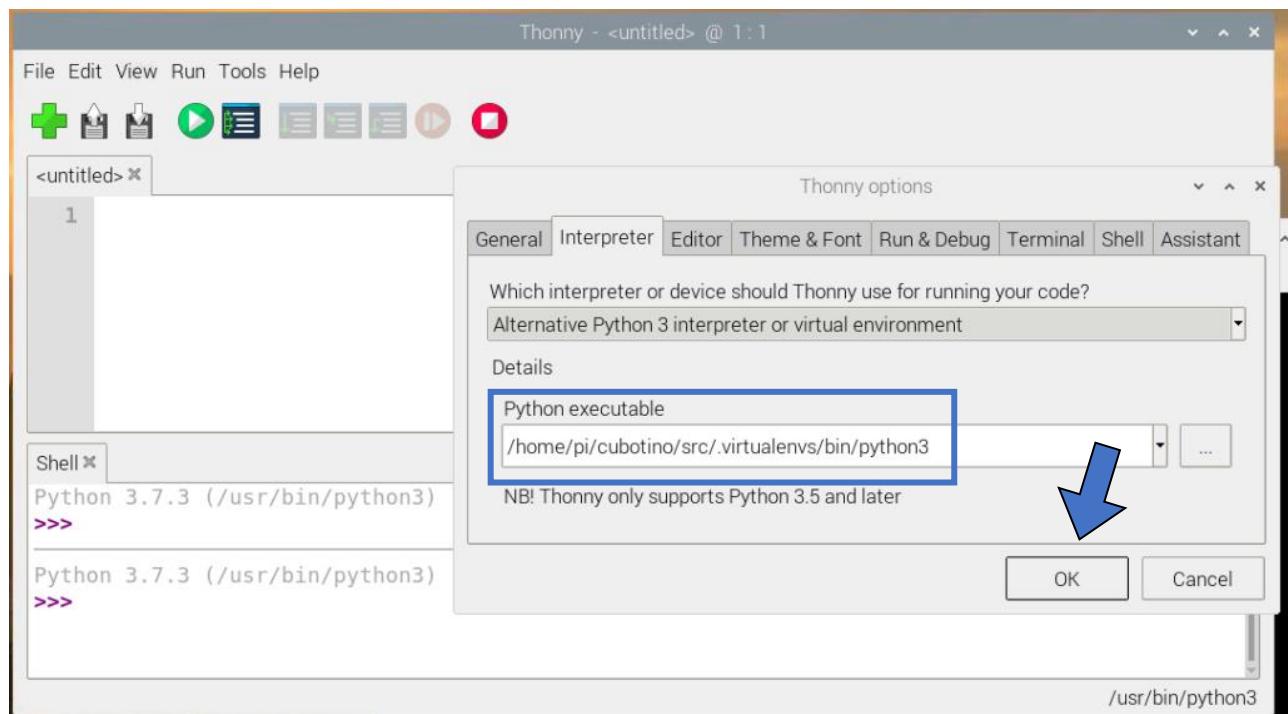
The path should be “/home/pi/cubotino/src/.virtualenvs”



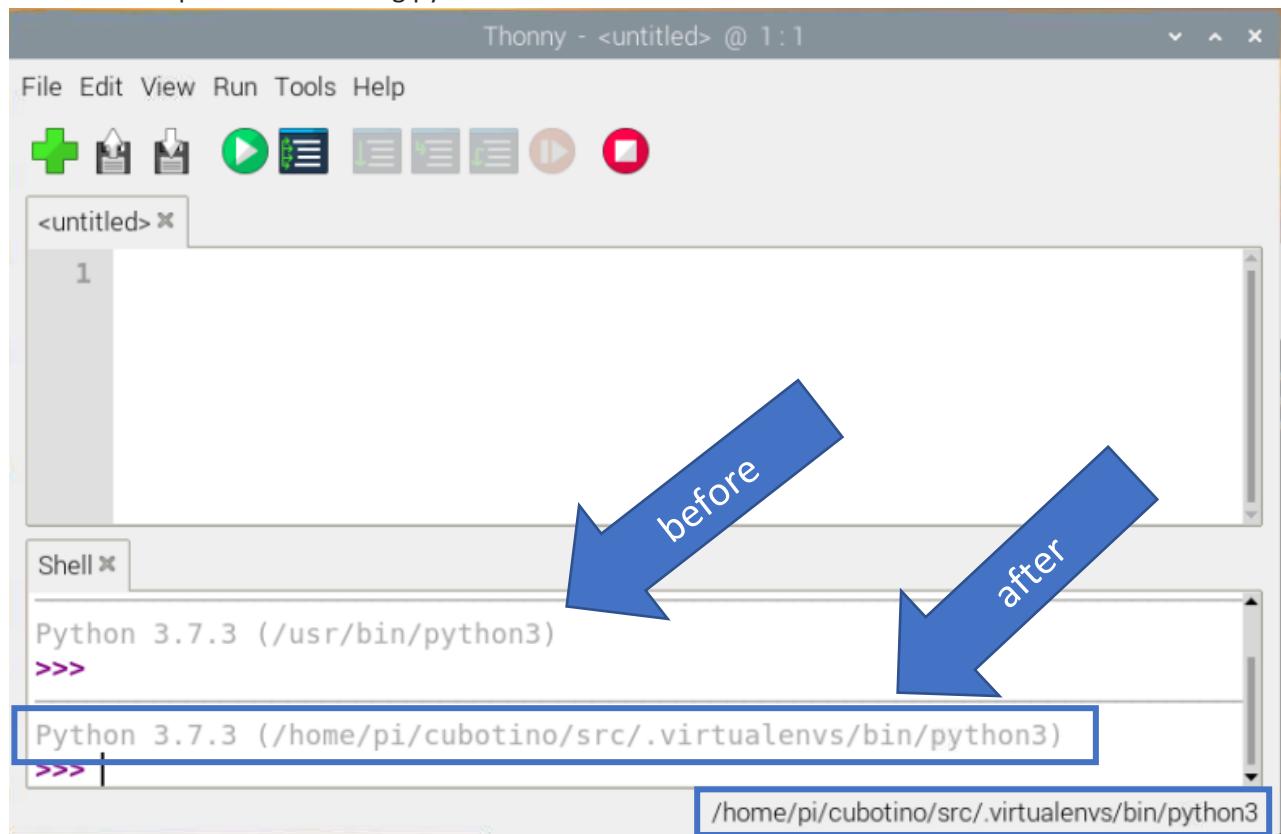
Select python3 from the path: ‘/home/pi/cubotino/src/.virtualenvs/bin/python3’ and confirm a couple of times



Confirm one more time



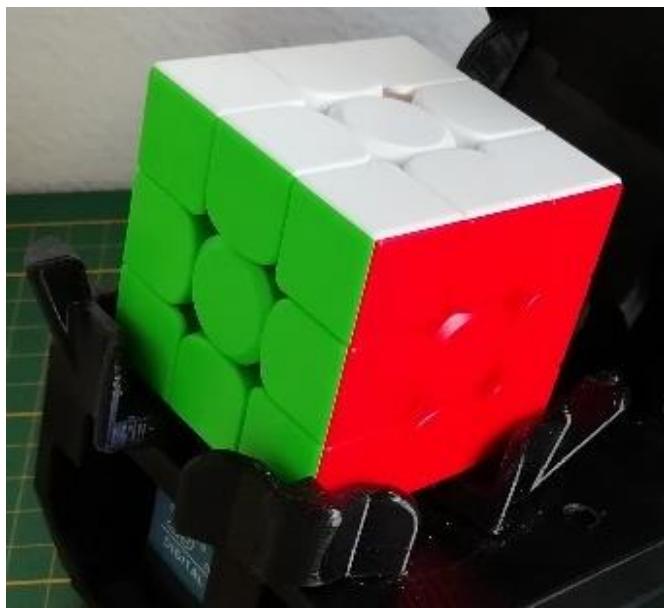
Note the interpreter is now using python3 from the venv



Notes, to get this change proposed as default:

- Do not open any python file
- Close and re-open Thonny

33. A convenient cube



Thanks to a comment from Art, I got to know the existence of “frameless” cubes

Thanks to a comment 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 I 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.

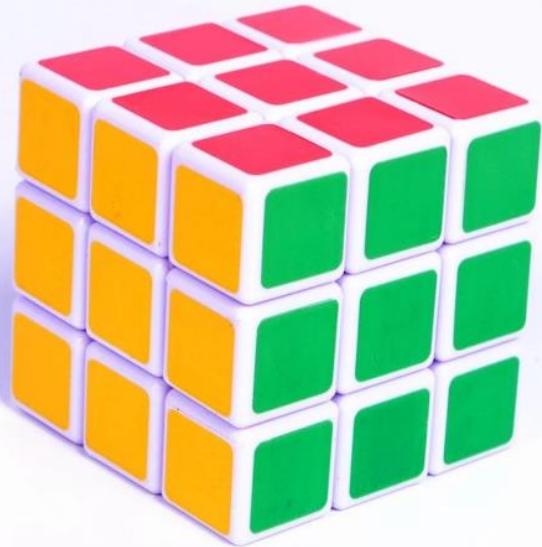
The logo on the white face is very large and must be scratched away.

When using this cube type, without the black frame around the facelets, it's necessary to set “frameless_cube” as true in the Cubotino_T_settings.txt.

Notes:

I'm not affiliated with Amazon, nor any other Company, I'm just sharing my positive experience with this cube 😊.
The Amazon link is just used as reference, to find further information.

Another interesting cube is a cube with the white frame.



I've bought one, to check the facelets detection, and the robot works very well.

This cube is mechanically not smooth, and it doesn't allow cut-corners, yet it is very inexpensive (less than 3.5 € shipment included). I have lubricated it

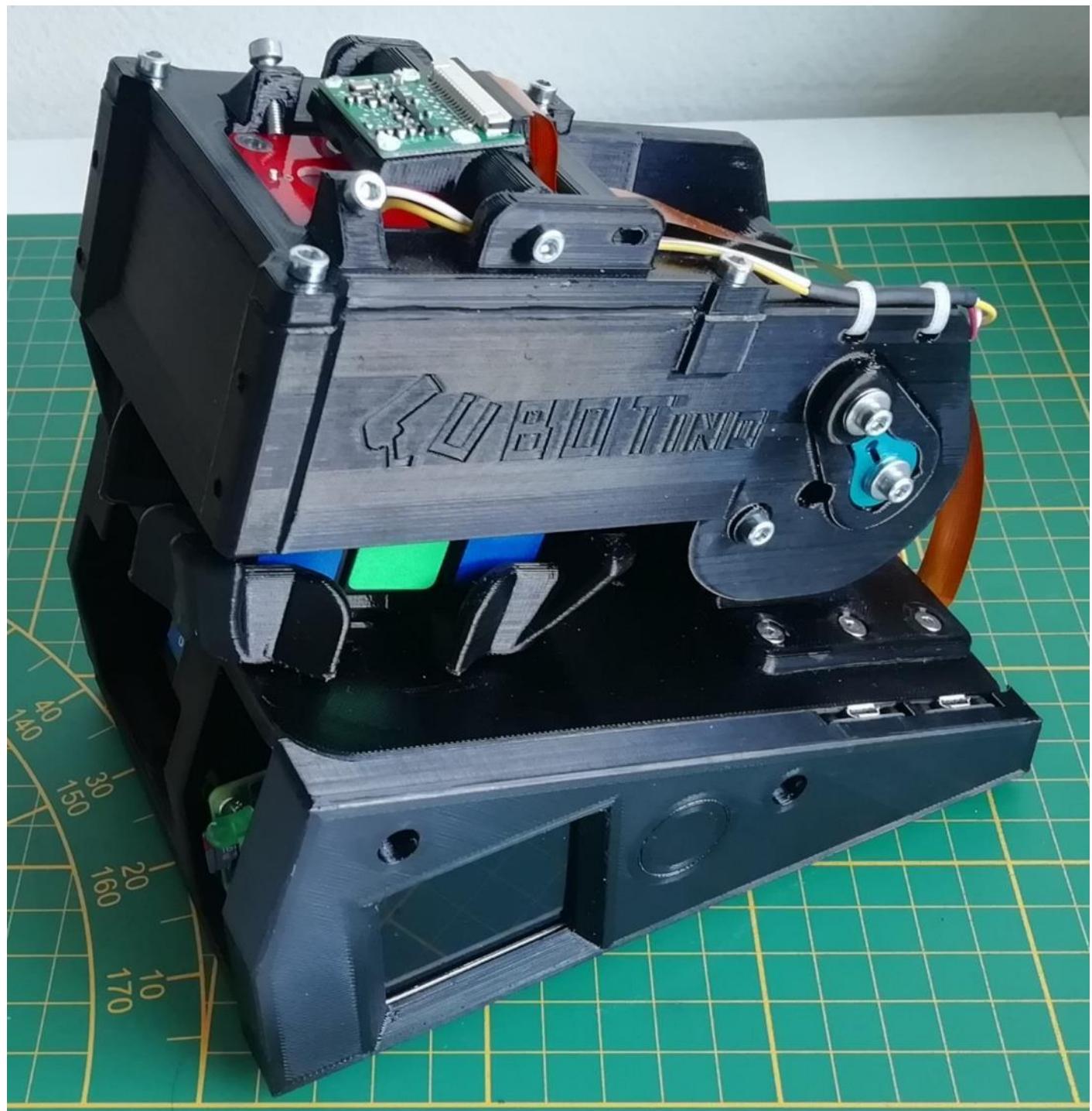
https://www.aliexpress.com/item/1005004029333733.html?spm=a2g0o.order_list.0.0.15281802R1ixih

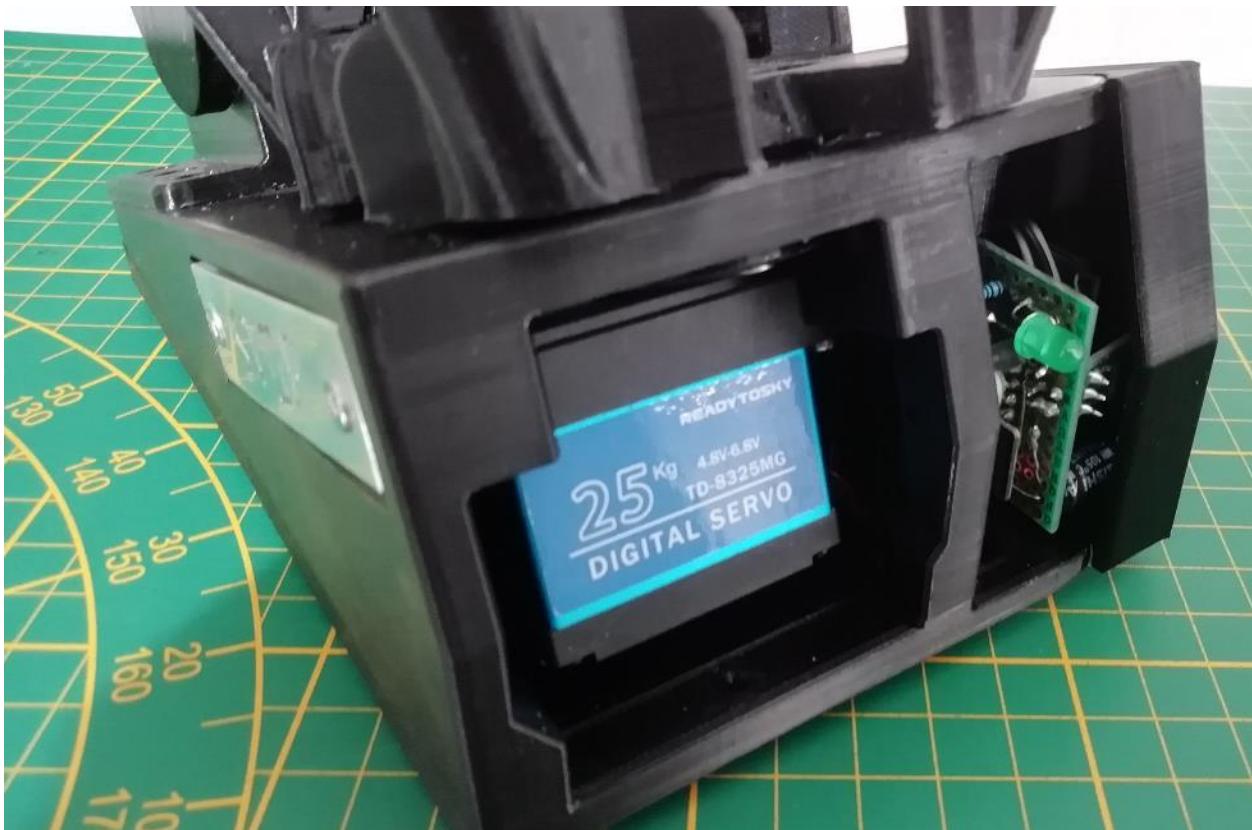
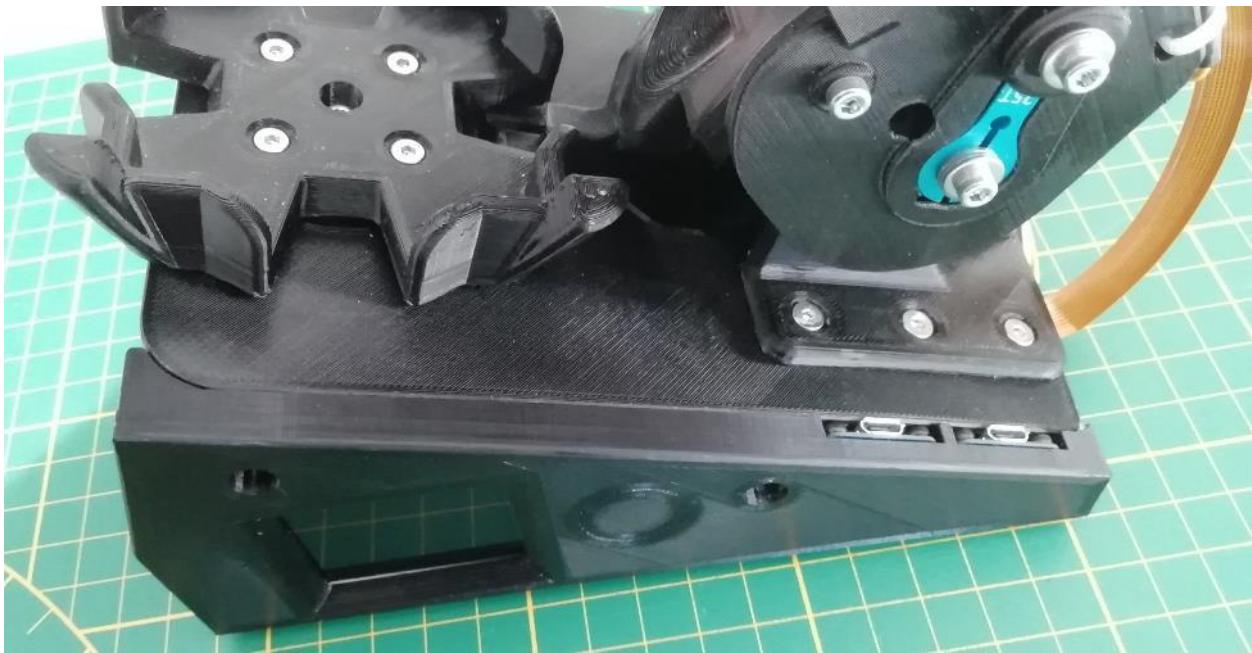
So, why this is an interesting cube? The robot detects well the status also in a completely dark room (of course the light provided by the LED).

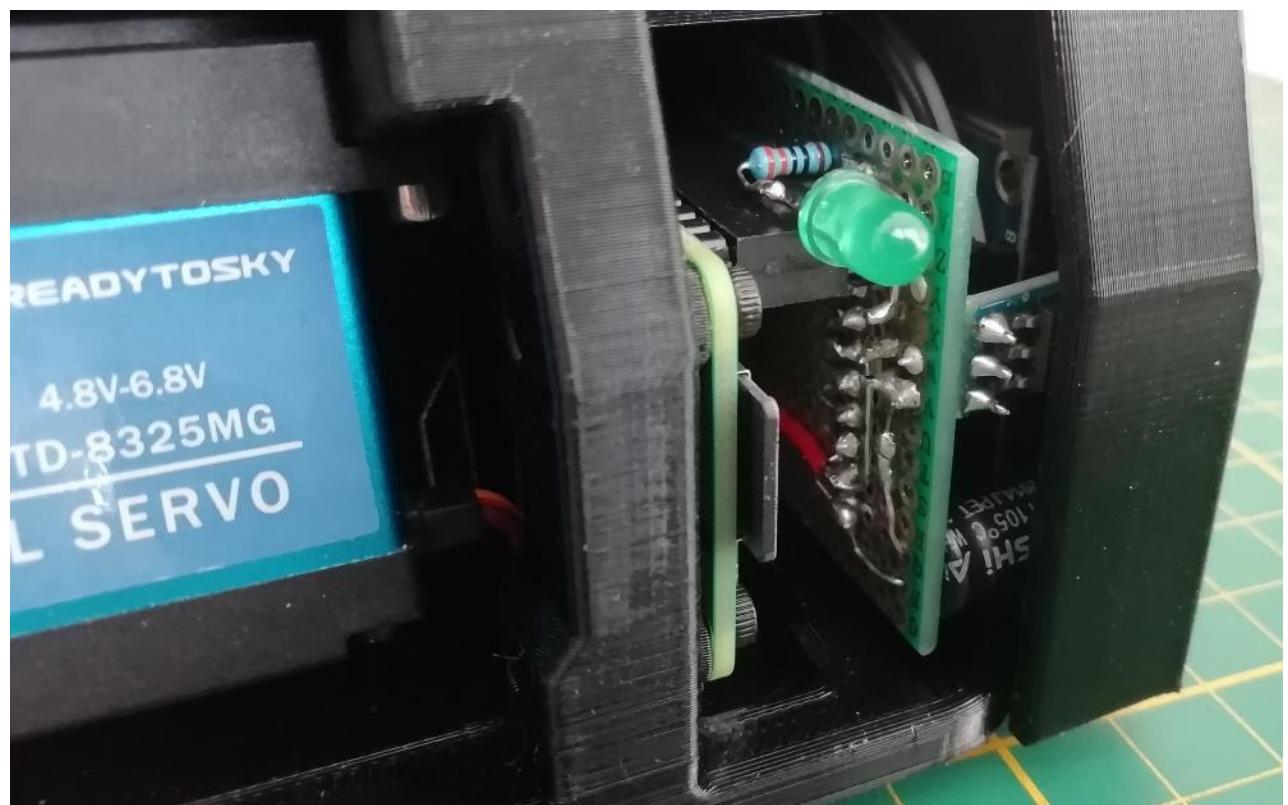
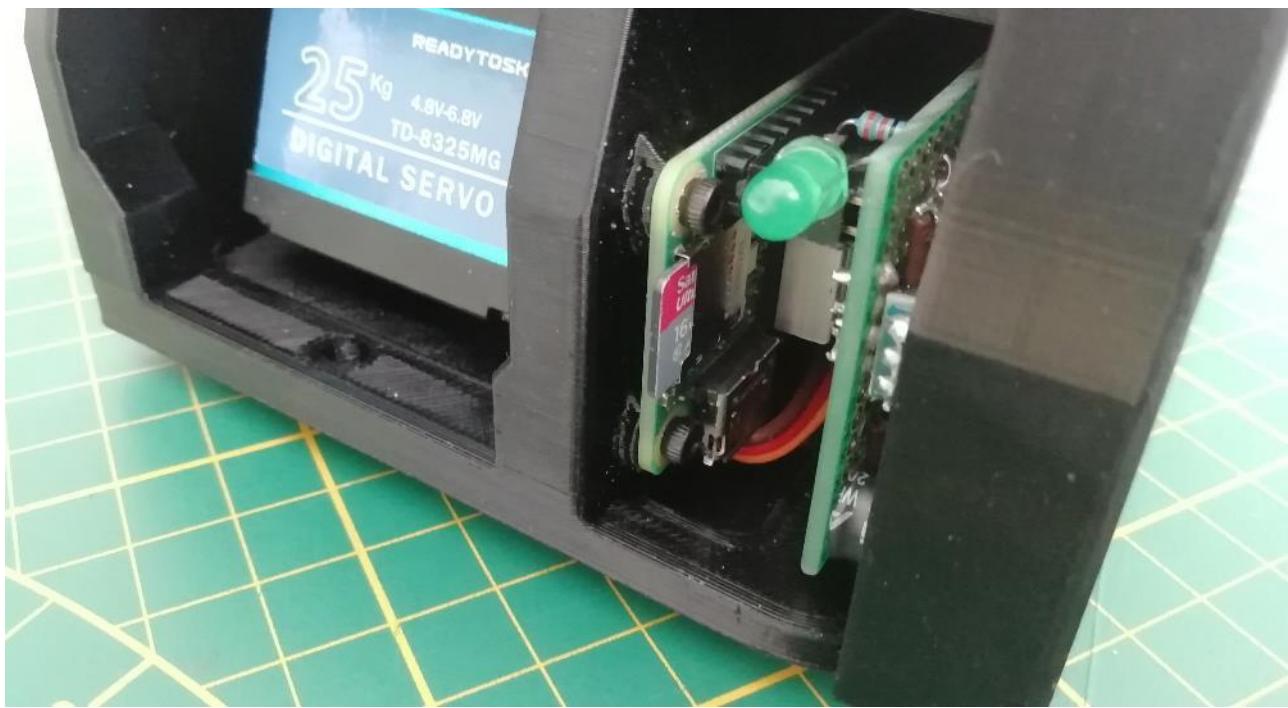
Notes:

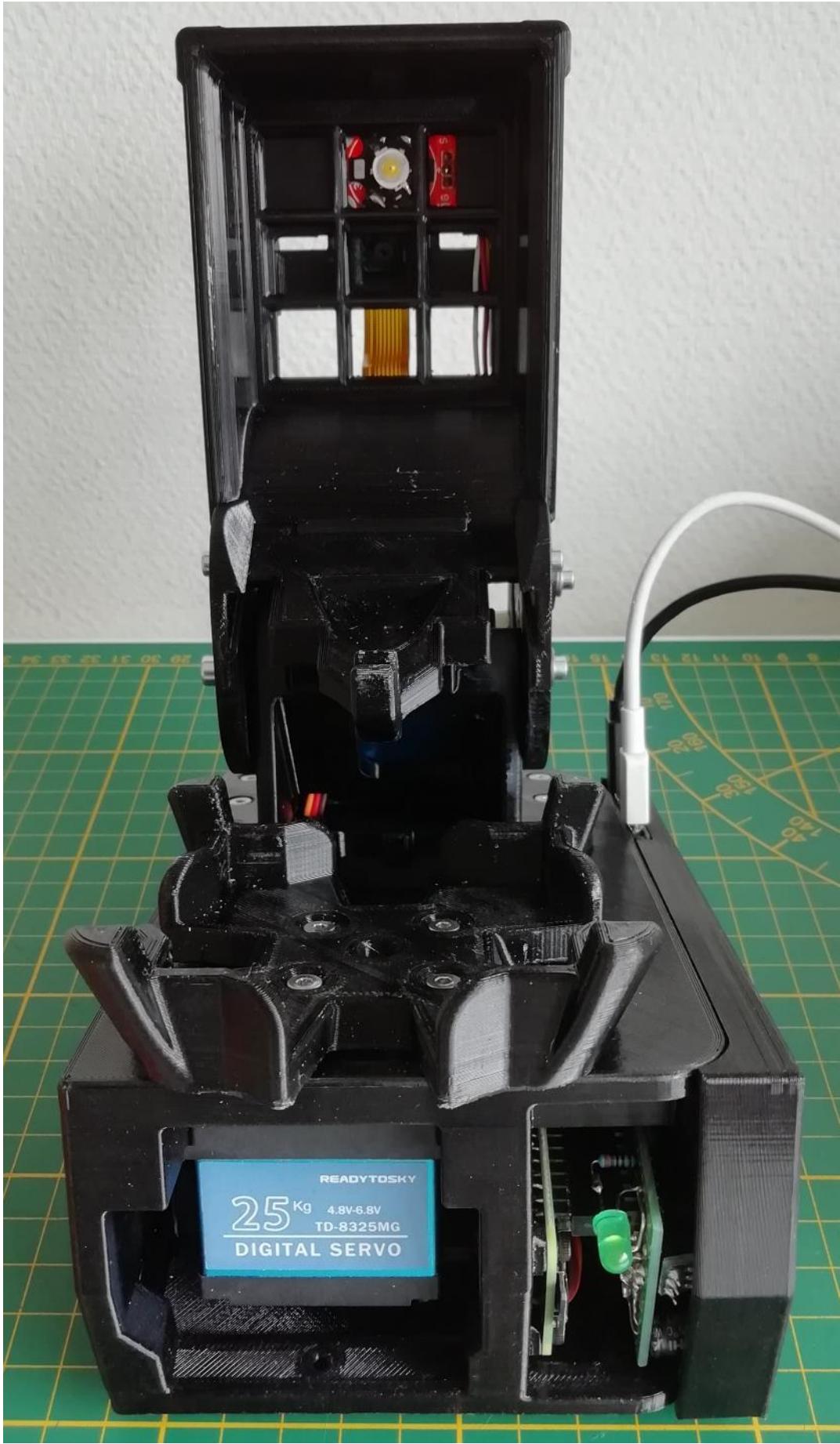
I'm not affiliated with Amazon, nor any other Company, I'm just sharing my positive experience with this cube 😊.
The AliExpress link is just used as reference, to find further information.

34. Collection of robot's pictures

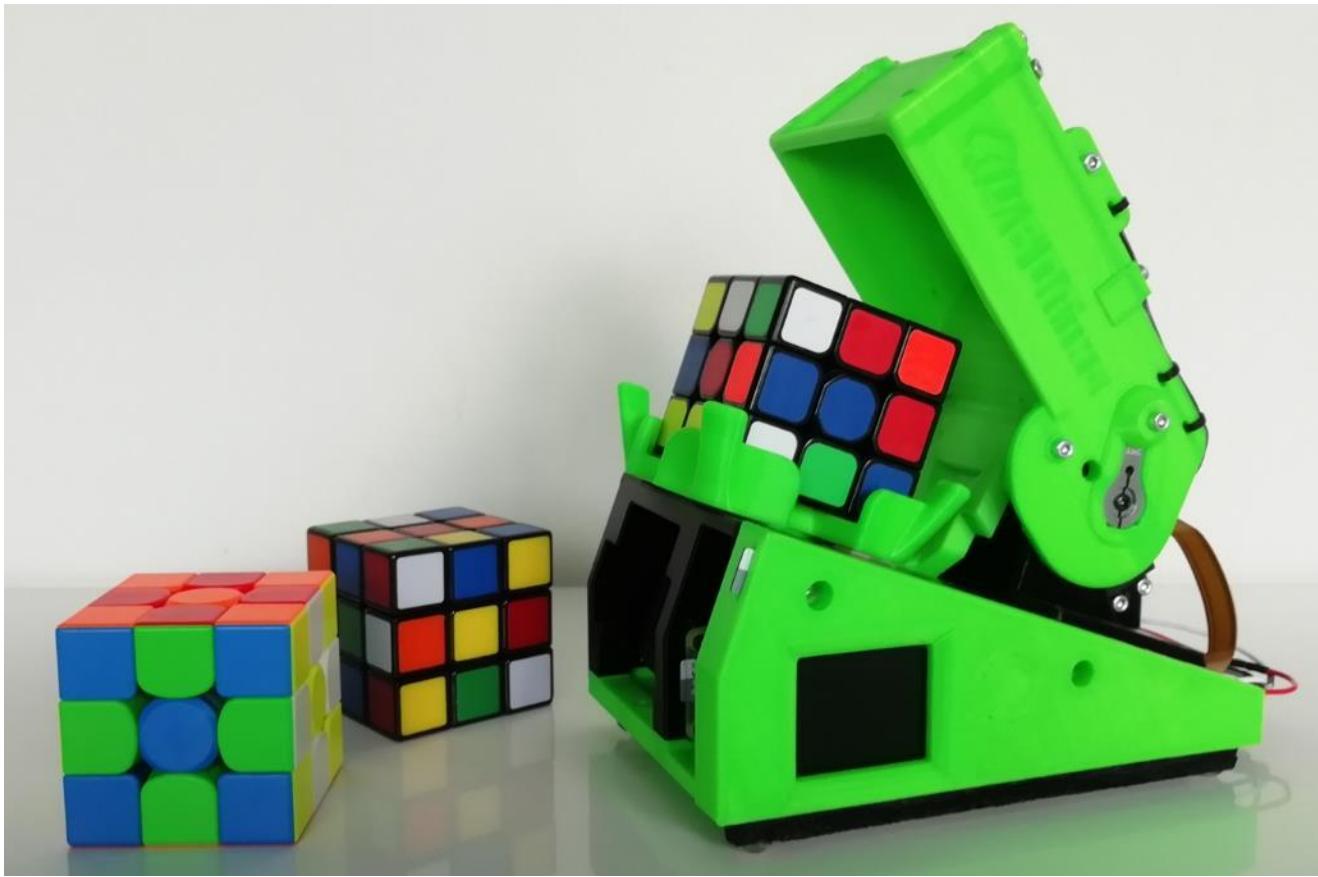


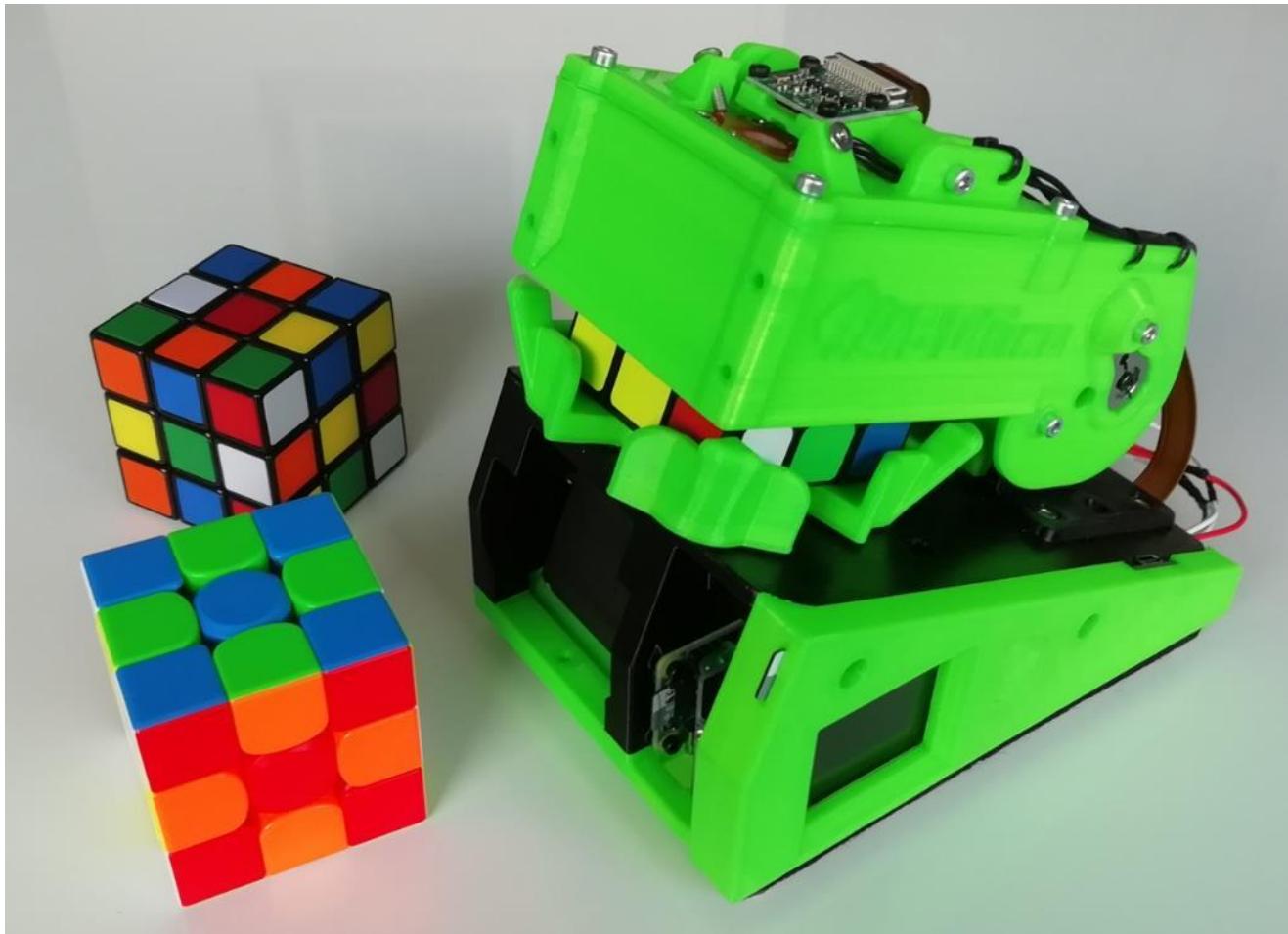














35. Conclusions

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

Pros:

1. The new way to manoeuvre the cube has proved to be effective.
2. Robot dimensions are very small.
3. The same base mechanic works fine on these two versions.
4. Base version specific:
 - a. is relatively cheap (about 30 to 40 euro of material, on early 2022, depending on where you live).
 - b. GUI works simply fine.
 - c. Rather easy to set the robot parameters via the GUI.
5. Top version specific:
 - a. Raspberry Pi Zero2 (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, or a single USB Type-C (3A).

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. Micropython documentation is somehow limited.
 - b. 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.
 - c. Single power supply inlet not robust on all the builds
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

36. Next steps

I'm not sure I'll work out the Medium robot version.

37. 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 feedback or thumbs up on YouTube (<https://youtu.be/dEOLhvVSBCg>), or at the Instructables 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-Autonomous-Small-3D-Printed-Rubiks-Cube-R/>).

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

38. Credits

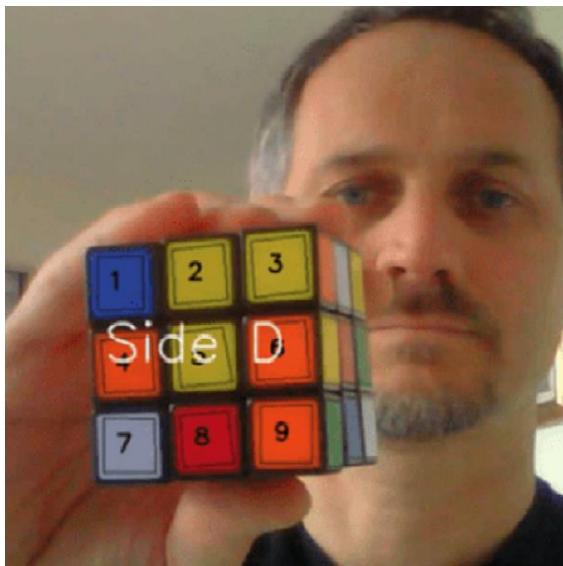
- to Mr. Kociemba, that further than developing the two-phase-algorithm solver, he also wrote a python version of it
- Hans Andersson, with his Tilted Twister ([Tilted Twister 2.0](#)) Lego robot, so inspiring: Very simple yet effective mechanic concept.

Credits to all the people who have provided feedback particularly those I've had contact with along this project:

- Jacques, who triggered me on the colour's sensors direction.
- Richard and Scott, for their keen check of this document and precious feedback.
- Yannick, and his initiative to simplify the installation via GitHub repo, further than other improvements to the code.
- Chad, and his initiative to draw the connections board in CAD and sharing the Gerber files with us.
- Kevin, the STEM evangelist, and his extensive availability to test the nightly versions.
- Martin, and his cross testing with other servo types and feeding all of us back.
- John for his sharp's observations and good feedback.
- Credit to Paul, for proactively add the support on alternative displays, to create the 180deg rotations instead of 2x90deg, further than several bugs identification.

I might forget the name of other contributors, please accept my apologies in that case.

39. Myself



I am Andrea Favero.

I was born in Italy in 1971.

I am married to Raffaella, and we have a son (Luca) and a daughter (Alice).

Since 1994 I have been working as an engineer, and since 1997 in R&D for small kitchen appliances Companies.

Since 2015 we have been living in Groningen, The Netherland.

On 2019 I had the opportunity to attend a Python class course, and I felt in love with coding.

On 2021 I decided to learn computer vision and Raspberry Pi, by giving myself the target to build a Rubik's cube solver robot.

My first Rubik's cube solver robot has been a test for myself, yet I learned so many things from others that I wanted to share it back via the Instructables site; The positive reactions and criticisms, led me thinking on how to make an easier version, with an eye to costs: CUBOTino project started on January 2022.

Contacts:

- I am not into social media.
- I can be reached via email: andrea.favero71@gmail.com

40. ChatGPT

January 2023

About 10 minutes after making an account to [chatGPT](#) (for info <https://en.wikipedia.org/wiki/ChatGPT>), I decided to test it with some specific questions (searching for help on one of my next projects...)

I must admit I got really impressed by those answers, and decided to ask a more general question...

Question: “*how to make a 3d prionted rubik's cube robot solver*” (...yes, also a typo on my question)

Answer: *Building a 3D printed Rubik's cube solving robot would involve several steps:*

1. *Designing the robot: Use a 3D modeling software like SolidWorks or AutoCAD to design the robot's mechanical parts, including the frame, the servo motors, and the moving parts that will manipulate the cube. It's important to keep in mind the size and weight of the robot, as well as the size of the cube it will be solving.*
2. *3D printing the parts: Once the design is complete, use a 3D printer to print out the parts. Make sure to use the appropriate material for the parts, such as ABS or PLA plastic.*
3. *Assembling the robot: Assemble the robot by attaching the servo motors, the moving parts, and the frame together. Use screws and other hardware to secure the parts in place.*
4. *Programming the robot: Write the code to control the servo motors and solve the cube using a microcontroller like Arduino or Raspberry Pi. The code should include instructions for moving the servo motors and solving the cube using an algorithm such as Kociemba's, IDA* or any other you prefer.*
5. *Calibrating the robot: Use the code to calibrate the robot to the cube, so that it can move the cube's individual faces and solve it.*
6. *Testing and fine-tuning: Test the robot and fine-tune the code and mechanical parts as needed.*

Please note that this is a general guide, and the details of each step will depend on the specific design of the robot and the 3D printing technology used. It is a complex project and requires a good understanding of mechanical engineering, robotics, and programming.

The generic question got a generic answer, yet it is quite a remarkable summary with proper order and suggestions.

I have also asked other questions, like:

- *how to increase the rotation range of a servo*
- *is it better PET or PLA for a 3d printed robot*
- *how to detect the rubik's cube facelets via OpenCV*

Please try it yourself if you are curious about chatGPT capabilities.

Please also be critical, as on different questions I got wrong answers and/or broken links when asking for evidence.

I do have the feeling I will use chatGPT when searching for coding help 😊

41. Revisions

Rev	Date	Notes
0	01/06/2022	First release.
1	06/06/2022	<p>Rather large revision, in particular:</p> <ul style="list-style-type: none"> • Modified Structure.stl by adding a hole (to see Raspberry Pi integrated LED) • Modified Cubotino_T.py and Cubotino_T_servos.py to upload settings from Json files • Integrated the list of files to be copied to Raspberry Pi • Added a few chapters on this document <ul style="list-style-type: none"> ◦ how to set Thonny ◦ Parameters and settings • Improved the Tuning chapter • Corrected a mistake at 'setup Raspberry pi interfaces' • Made more complete the 'how to use the robot' • Modified Rpi setting to include mac address
1.1	07/06/2022	Correct an error on this doc, for the display pin numbers
2.0	14/6/2022	<p>Modified Cubotino_T.py</p> <ul style="list-style-type: none"> • to tentatively import Kociemba solver from multiple locations • to visualize/save the manipulated images to detect the facelets <p>Added:</p> <ul style="list-style-type: none"> • Kociemba installation chapter • Memory usage info <p>Updated:</p> <ul style="list-style-type: none"> • Troubleshooting, with more details for the Top_cover < -- > Hinge gap and friction
2.1	17/6/2022	Corrected typo on the Connections_board
2.2	18/6/2022	<p>Added:</p> <ul style="list-style-type: none"> • 3 stl files, acting as spacers to use Raspberry Pi 3b or 4b instead of Zero 2 • Info related these new (optional) parts
2.3	22/6/2022	<ul style="list-style-type: none"> • Removed the additions made with rev 2.2 • Modified Cubotino_T.py due to a bug, throwing an error under some rare, yet possible, conditions
3.0	25/6/2022	<p>Since today, is available a simplified setup for the Raspberry Pi</p> <p>With this approach, few other changes were necessary, resulting in:</p> <ul style="list-style-type: none"> • Modified Cubotino_T.py to better managing permissions on folders and files creation • Updated instruction
3.1	30/06/2022	<p>GitHub:</p> <ul style="list-style-type: none"> • Moved the robot specific files into a (src) folder; The folder structure is maintained at Raspberry Pi for easy future updates. <p>Updated files:</p> <ul style="list-style-type: none"> • Cubotino_T.py due to a bug and to make backups of the settings files • Cubotino_T_bash.sh to point the right folder • setup.sh <p>Instructions:</p> <ul style="list-style-type: none"> • Removed the Manual installation part (to setup the Raspberry Pi) • Further explained the Simplified installation via GitHub
3.2	06/07/2022	Modified Cubotino_T.py to be used with Raspberry Pi Zero (not 2) without crashing. Added some minor improvements

Rev	Date	Notes
3.3	09/07/2022	<p>Modified:</p> <ul style="list-style-type: none"> • Cubotino_T_servos.py to upload the settings when used with the argument '- -set', to cover servos having pulse width different from the common 1 to 2ms • Baseplate_front.stl for taller servos (up to 31mm protrusion under the flange) <p>Instruction:</p> <ul style="list-style-type: none"> • Better covered the usage of servos with different pulse width range • Corrected typos on Cubotino_T_servos.py (the 's' was missed) • Better explained how to set the servos positions <p>Added notes on the max protrusion from the flange, when 'taller' servos</p>
3.4	13/07/2022	<p>Instructions:</p> <ul style="list-style-type: none"> • Corrected typos • Better explained the servos settings
3.5	25/07/2022	<p>Modified Cubotino_T_servos.py to check and play with the servo's positions directly from the command line (additionally from the REPL, as it was so far possible) At the instructions, better explained how to verify the servos positions.</p>
3.6	06/08/2022	<p>Modified Cubotino_T.py:</p> <ul style="list-style-type: none"> • display is initialized once (thanks Yannick for the nice solution) • frameless cubes can now be detected <p>Modified Cubotino_T_settings.txt and Cubotino_T_settings_AF.txt, by adding</p> <ul style="list-style-type: none"> • frameless_cubes parameter • cover_self_close parameter <p>Improved or modified the instructions:</p> <ul style="list-style-type: none"> • Alternative servos that can be used • Gerber files for the connections board (thanks to Chad) • frameless_cube parameter • cover_self_close parameter • Troubleshooting for servos • Troubleshooting for cube status detection • Display init and test • Sticking the PiCamera with its self-adhesive tape to the board • Location for the step files
3.7	11/08/2022	<p>Modified Instructions:</p> <ul style="list-style-type: none"> • Corrected typo on screw protrusion for display support <p>Modified Cubotino_T.py:</p> <ul style="list-style-type: none"> • Corrected bug on frameless cube detection • Improved the facelets estimation, and increase up to 4 the estimated ones • Enlarged the permits at text files creation (*_backup.txt, *Log.txt)

Rev	Date	Notes
3.8	21/08/2022	<p>Modified Cubotino_T.py to import settings for a second Andrea's robot (based on mac address)</p> <p>Modified Cubotino_T_servos.py:</p> <ul style="list-style-type: none"> • to import settings for a second Andrea's robot (based on mac address) • to include the b_extra_sides setting when spinning toward CW or CCW <p>Single power supply, via USB type-C: Added an alternative PCB_cover stl and stp files, that also includes a slot to store the microSD clone.</p> <p>Modified Instructions:</p> <ul style="list-style-type: none"> • Additional alternative servos (cheaper) • USB type-C breakout board reference • Connections for a single power supply • Tests of the Connections_board, before connecting it to Rpi • Setting for more Wi-Fi connections (i.e. for the phone Wi-Fi hotspot) <p>Modified PCB_support stl file, the one for two micro-USB break out boards, to better couple with the display (reduced by 2.5mm the gap toward the display).</p>
3.9	19/09/2022	Updated instructions to the V1.6 Connections_board version
3.10	28/10/2022	<p>Updated instructions:</p> <ul style="list-style-type: none"> • Re-ordered the content, with initial part fully focused on the robot making process • Added info/pictures on how to increase the servo rotation range. • Added info on how to embed a Raspberry Pi 3 or 4 • Added info related to the PiCamera focus • Better explained the parameters listed in chapter 13 <p>GitHub / Instructables:</p> <ul style="list-style-type: none"> • Added the stl files for the “extensions”, needed when Rpi 3 or 4 is used • Modified the Hinge.stl (and .stp) to accept 43mm servos <p>Cubotino_T.py:</p> <ul style="list-style-type: none"> • Changed from the word “exposition” to “exposure” on the display
3.11	31/10/2022	Correct the Instructions layout error, that lead to more than 260 pages instead of 118.
3.12	13/11/2022	<p>Modified Cubotino_T.py:</p> <ul style="list-style-type: none"> • Added the scrambling function • Improved the cycle stopping part <p>Modified Cubotino_T_servos.py:</p> <ul style="list-style-type: none"> • Differentiated the progress for scrambling from the one for solving. <p>Modified Cubotino_T_display.py for the scrambling progress bar for scrambling.</p> <p>Updated instructions:</p> <ul style="list-style-type: none"> • Explained how to start the scrambling • Added info on female header height for the Connections_board • Minor corrections along the document
3.13	15/11/2022	Added the references for a “convenient cube” (I was convinced I had written this before 😊)

Rev	Date	Notes
3.14	27/11/2022	<p>First steps into remote usage.....</p> <p>Modified Cubotino_T.py:</p> <ul style="list-style-type: none"> • Improved shut-off function access: It can be reached again when Rpi is idling • After scanning the cube is not moved to the initial position • If the cube solving is interrupted, the solving time is forced to zero, also at log file. • Added the possibility to automate scrambling plus solving cycles • Cleaned the __main__ function <p>Modified Cubotino_T_moves.py:</p> <ul style="list-style-type: none"> • The cube starting position is the last left when scanning the 6th face <p>Modified Cubotino_servo.py:</p> <ul style="list-style-type: none"> • Placed back, yet commented out, the change made on ver3.8 (21/08/2022) <p>Instructions:</p> <ul style="list-style-type: none"> • How to use the robot for the shut-off • How to use the robot to remote/automate cycles • Reasons to not move the cube to initial position after the scanning (see Robot solver algorithm) • Troubleshooting the cube misalignment due to update before/after 21/08/2022
3.15	27/12/2022	<p>Modified Cubotino_T.py for remote usage, by adding:</p> <ul style="list-style-type: none"> • - -shutoff argument (shuts off the RPI after the last automated cycle) • cube image on (VNC) screen after each solving process • progress bar on (VNC) screen during the pause between automated cycles <p>other argument added:</p> <ul style="list-style-type: none"> • - -debug (print out info to terminal for debug purpose) • - -c_wow (shows on VNC screen the image manipulation steps) • - -F_deg (Fahrenheit degrees instead of Celsius) • - - timer (shows a timer after the scrambling function) <p>Modified Cubotino_T_settings.txt and Cubotino_T.py:</p> <ul style="list-style-type: none"> • Added s_mode parameter to for easier setting of PiCamera version (V1.3 or V2). <p>Modified Cubotino_T_moves.py: Removed 2 Flips under a specific movements sequence.</p> <p>Instructions; Added info about:</p> <ul style="list-style-type: none"> • s_mode parameter (for PiCamera V2) • wire size for the power lines (especially for Rpi model 3 and 4) • Low Voltage Warnings and related troubleshooting • Myself
3.16	29/12/2022	Modified Cubotino_T_moves.py twice due to bugs (28/12/2022 and 29/12/2022)

Rev	Date	Notes
3.17	14/01/2023	<p>Most of these changes have been triggered by testing Cubotino with Rpi ZeroW 😊</p> <p>Modified <code>Cubotino_T.py</code>:</p> <ul style="list-style-type: none"> • Removed bug when <code>--cv_wow</code> and Raspberry Pi ZeroW. • Raspberry Pi ZeroW has now all the functionalities used in other Rpi models. • Shut off calling function adapted for Rpi ZeroW responsiveness. • Added '<code>vnc_delay</code>' parameter, for better robot to VNC graphical desktop synchronization. When 'older' <code>Cubotino_T_settings.txt</code>, without this parameter, 0.5secs will be used (when a VNC with graphical desktop). • Most of the arguments can be entered by their first letter. • When <code>--pause</code> (automated cycles) its value is rounded to upper multiple of 4 (secs). • When <code>--pause</code> the countdown time management is improved. • Cleaned and improved the <code>cubeAF</code> function. • Added graphical desktop screen status at the <code>Cubotino_T_log.txt</code> • Removed non-ascii characters. <p>Modified <code>Cubotino_T_settings.txt</code> by adding '<code>vnc_delay</code>' parameter</p> <p>Modified <code>Cubotino_servos.py</code>, for the '<code>-tune true</code>' part:</p> <ul style="list-style-type: none"> • Fixed a bug on 'init' command, now it properly reloads the last saved settings • Added 'print' command, to print the latest saved settings. • Arrows from the keyboard can be used to recall/edit previously entered commands. <p>Instructions, added info on:</p> <ul style="list-style-type: none"> • Robot performances with Raspberry Pi ZeroW vs Zero2W • Memory management (at "Setting up Raspberry Pi" and at the troubleshooting) • Wi-Fi stability (at "Setting up Raspberry Pi" and at the troubleshooting) • When '<code>git pull -v</code>' returns an error
4.0	20/01/2023	<p>Modified</p> <p><code>Cubotino_T.py</code> :</p> <ul style="list-style-type: none"> • Method to upload settings from <code>Cubotino_T_settings.txt</code> • Script version can now be retrieved. <p>GitHub: gitignore file to ignore <code>Cubotino_T_settings.txt</code> when updating local repos.</p> <p>Instructions: How to update the local repo via git.</p>

Rev	Date	Notes
5.0	18/04/2023	<p>Large update, bringing the learning from the Cubotino_micro</p> <p><u>Modified files</u></p> <p>Setup.sh: added ST7789 display driver for installation compatibility with Cubotino_micro.</p> <p>Cubotino_T_bash.sh : Exit error code management.</p> <p>Cubotino_T.py :</p> <ul style="list-style-type: none"> Main parameters for PiCamera are settable via GUI. Interpreted colours are plot to the display. Settings files, filename management. Exit error code. Script interruption without shutting the raspberry Pi off. Interpreted colours of each face are plot to the display: <p>Cubotino_T_display.py :</p> <ul style="list-style-type: none"> Interpreted colours of each face are plot to the display: Settings file, filename management. Display test. <p>Cubotino_T_servos.py :</p> <ul style="list-style-type: none"> Main parameters for servos are settable via GUI. Settings files, filename management. <p><u>New files</u></p> <p>Cubotino_T_servos_GUI.py : GUI for main servos and PiCamera parameters settings.</p> <p>get_macs_AF.py: reads macs_AF.txt file, to simplify settings files filename management.</p> <p>macs_AF.tx: text file with the AF's robot's mac addresses.</p> <p>Instructions:</p> <ul style="list-style-type: none"> GUI chapter to tune the main servos and PiCamera parameters. How to increase friction to a too loose cube (at Troubleshooting). Specific python libraries list (Setting up Raspberry Pi). ChatGPT.
5.1	10/05/2023	<p>Modified the <i>Cubotino_T_servos.py -- tune</i>, by adding:</p> <ul style="list-style-type: none"> Sweep function to verify the max servos rotation range. Possibility to quickly change the pulse width range, to get confidence on the bought/received servos. <p>Instructions:</p> <ul style="list-style-type: none"> Improved explanations on how to set the VNC resolution, i.e. for the GUI usage (See "Setting up Raspberry Pi" Step 10). Amended with info for the servos sweep (see Tuning via CLI). Corrected many typos.

Rev	Date	Notes
6.0	12/10/2023	<p>Update for “Bullseye” OS compatibility, forcing the library picamera2. Updated scripts keep compatibility with “Buster”, while aren’t compatible with the just released “Bookworm” OS.</p> <p>New files <i>Cubotino_T_camera_os10.py</i>, to use the PiCamera library with “Buster” OS. <i>Cubotino_T_camera_os11.py</i> to use the picamera2 library with “Bullseye” OS. <i>Cubotino_T_pigpiod.py</i> to stop and start pigpiod once Cubotino starts; This is a workaround for some issues with pigpiod in Bullseye, I couldn’t solve in a more elegant way.</p> <p>Modified files</p> <p><i>setup.sh</i>:</p> <ul style="list-style-type: none"> • Sets SWAP memory to 512Kb in case it’s smaller. • installs libraries as per installed OS, that implies different OpenCV versions. <p><i>Cubotino_T.py</i>:</p> <ul style="list-style-type: none"> • Moved camera functions from this script to the <i>Cubotino_T_camera_osXX.py</i> • Calls <i>Cubotino_T_pigpiod.py</i> once, at the start. • Added the OS version to the log data file (<i>Cubotino_solver_log.txt</i>). • Corrected minor bugs. <p><i>Cubotino_T_servos_GUI.py</i> :</p> <ul style="list-style-type: none"> • Small change to better work with updated <i>Cubotino_T.py</i> • Calls <i>Cubotino_T_pigpiod.py</i> once, at the start, when this script is used to tune the servos/camera. <p><i>Cubotino_T_servos.py</i>:</p> <ul style="list-style-type: none"> • Small change to better work with updated <i>Cubotino_T.py</i> • Calls <i>Cubotino_T_pigpiod.py</i> once, at the start, when this script is used to set/tune the servos. <p><i>Cubotino_T_display.py</i>:</p> <ul style="list-style-type: none"> • Small change to better work with updated <i>Cubotino_T.py</i> • Calls <i>Cubotino_T_pigpiod.py</i> once, at the start, when this script is used to test the display. <p>Instructions:</p> <ul style="list-style-type: none"> • Added the <i>Connections_board</i> schematic. • Corrected screws: Quantity, screws size. • Corrected assembly steps numbers. • Troubleshooting, added: <ul style="list-style-type: none"> ◦ Errors during the Raspberry Pi installation process. ◦ Error “Total corner twist is wrong” after scanning the cube. ◦ Display and/or <i>Top_cover_led</i> flickering right after the boot. • Added info on the Bullseye OS, to be chosen from now on. • Added some specific info on aspect where Bullseye differ from Buster.

Rev	Date	Notes
6.1	15/10/2023	<p>Modified files</p> <p><i>Cubotino_T.py:</i></p> <ul style="list-style-type: none"> • Solved a bug when more manual cycles. • Addition: OS and Script versions are displayed at the start. <p><i>Cubotino_Tservos.py:</i></p> <ul style="list-style-type: none"> • Small improvement on Top_cover position, before solving the cube. <p>Instructions</p> <ul style="list-style-type: none"> • Changed the link to the browser for downloading github folders. • Setting up Raspberry Pi, updating the system when error “untracked working tree files would be overwritten by merge”.
6.2	03/11/2023	Instructions: Added troubleshooting info about the touch sensor.
6.3	21/01/2024	<p>Modified files</p> <p><i>Cubotino_T.py:</i></p> <ul style="list-style-type: none"> • Camera setting up (formal camera warm-up function) uses 4 faces, instead of 1, for the AWB further than the Exposure time (that was already on 4 faces). • Added the FCS (Fix Coordinates System) algorithm, to retrieve facelets colour based on fix coordinates (when the facelets are not detected within fcs_delay time). • Added FCS (Fix Coordinates System) to the info to log at Cubotino_solver_log.txt. <p><i>Cubotino_T_camera_os11.py :</i></p> <ul style="list-style-type: none"> • Camera “warm up” moved to Cubotino_T.py. • Added the exposure_shift. <p><i>Cubotino_T_camera_os10.py</i> and</p> <ul style="list-style-type: none"> • Camera “warm up” moved to Cubotino_T.py. <p><i>Cubotino_T_servos_GUI.py:</i></p> <ul style="list-style-type: none"> • Added the LED PWM slider. • Added the exposure_shift slider (it is effective only when OS11). <p>Instructions</p> <ul style="list-style-type: none"> • Better explained how to adjust the PiCamera focus. • Explained how to find the OS image when not listed at the Raspberry Pi Imager. • Corrected board maker jlcpbc webpage address. • Explained how the FCS (Fix Coordinates System) works.
6.4	22/01/2024	<p>Modified files:</p> <p><i>Cubotino_T.py</i> : Corrected small bug on FCS (Fix Coordinates System).</p> <p><i>Cubotino_T_display.py</i> : Improved the build_by part.</p> <p><i>Cubotino_T_settings.txt</i>: Added built_by, built_by_x and built_by_fs parameters</p> <p>Instructions:</p> <ul style="list-style-type: none"> • Explained the GUI additions of Ver 6.3 (Led PWM and Exposure shift). • Added expo_shift, built_by, built_by_x and built_by_fs parameters on the “Parameters and settings” table.

Rev	Date	Notes
6.5	25/01/2024	<p>Modified files: <i>Cubotino_T.py</i>:</p> <ul style="list-style-type: none"> • Solved a bug when OS11 and ZeroW. <p>Instructions:</p> <ul style="list-style-type: none"> • Added link to download the OS 10 (Buster) image.
6.6	02/02/2024	<p>Modified files: <i>Cubotino_T.py</i>:</p> <ul style="list-style-type: none"> • Solved a couple of bugs on FCS
6.7	10/02/2024	<p>Modified files: <i>Cubotino_T_settings.txt</i></p> <ul style="list-style-type: none"> • 128 instead 130 on “disp_width”. • Added the “made by” parameters. • Added the “fcs_delay” parameter. <p><i>Cubotino_T.py</i></p> <ul style="list-style-type: none"> • Solved bag when coordinates file not yet generated. • Increased robustness vs coordinates file with empty lines. • FCS now reacts also when no one contour has been detected yet. <p><i>Cubotino_T_servos_GUI.py</i></p> <ul style="list-style-type: none"> • Before saving all the parameters are converted to string. <p>Instructions: Corrected the parameters table</p>
6.8	29/02/2024	<p>Modified files: <i>Cubotino_T.py</i></p> <ul style="list-style-type: none"> • Added a facelets color animation on the display, after a successful run. <p><i>Cubotino_T_display.py</i></p> <ul style="list-style-type: none"> • Added the animation plotting function. • Improved efficiency on Cubotino logo management. <p>Instructions: Added the argument ‘--no_animation’ at the “How to operate the robot” chapter</p>

Rev	Date	Notes
7.0	10/03/2024	<p>Rather large change to rationalize the settings management, and to prevent the local settings being overwritten by 'git pull'. This change is hardly noticeable to the Makers, while improving the code for maintenance.</p> <p>New file: <i>Cubotino_T_settings_manager.py</i></p> <p>Modified files (scripts using settings from the *settings.txt files): <i>Cubotino_T.py</i>, <i>Cubotino_T_servos.py</i>, <i>Cubotino_T_servos_GUI.py</i>, <i>Cubotino_T_display.py</i>, <i>Cubotino_T_camera_os10.py</i>, <i>Cubotino_T_camera_os11.py</i></p> <p>Renamed files:</p> <ul style="list-style-type: none"> From: <i>Cubotino_T_settings.txt</i> → to: <i>Cubotino_T_settings_default.txt</i> From: <i>Cubotino_T_servo_settings.txt</i> → to: <i>Cubotino_T_servo_settings_default.txt</i> <p>Other changes (not related to the settings management): <i>Cubotino_T.py</i></p> <ul style="list-style-type: none"> The animation is shown at the display or at the PC screen (if connected). The animation can be interrupted via the Touch sensor. <p>Instructions: Explained how the settings files are managed, and corrected their name at the Parameters and settings</p>
7.1	17/03/2024	<p>Modified files: <i>Cubotino_T_display.py</i> (added support for 240x320 display, st7789 driver). Instructions: Added info related to alternative displays options, and related settings.</p>
7.2	01/04/2024	<p>This release is about the usage of 180deg rotation, instead of 2 x 90deg (servos solving time reduced by 13% average) and the Virtual solver.</p> <p>New file: <i>Cubotino_T_test_random.py</i> → Virtual solver to test CUBOTino moves.</p> <p>Modified files:</p> <ul style="list-style-type: none"> <i>Cubotino_T.py</i> → Changed the animation for the 180 degrees rotation. <i>Cubotino_T_moves.py</i> → Added the 180 degrees rotations. <i>Cubotino_T_servos.py</i>: <ul style="list-style-type: none"> Added the 180 degrees rotations. Split parameters loading from servos initialization (for the virtual solver). <i>Cubotino_T_servos_GUI.py</i> → Slider 'Warp slice' from int to float (bug solving) <p>Instructions:</p> <ul style="list-style-type: none"> Info about the Virtual Solver (new chapter). More info on OS flashing.

Notes:

- Instructions and files will be updated whenever needed/possible.
- Please feedback in case of mistakes or parts requiring a better explanation.