

How to make CUBOTino Pocket: A small, 3D printed, Rubik's cube solver robot.

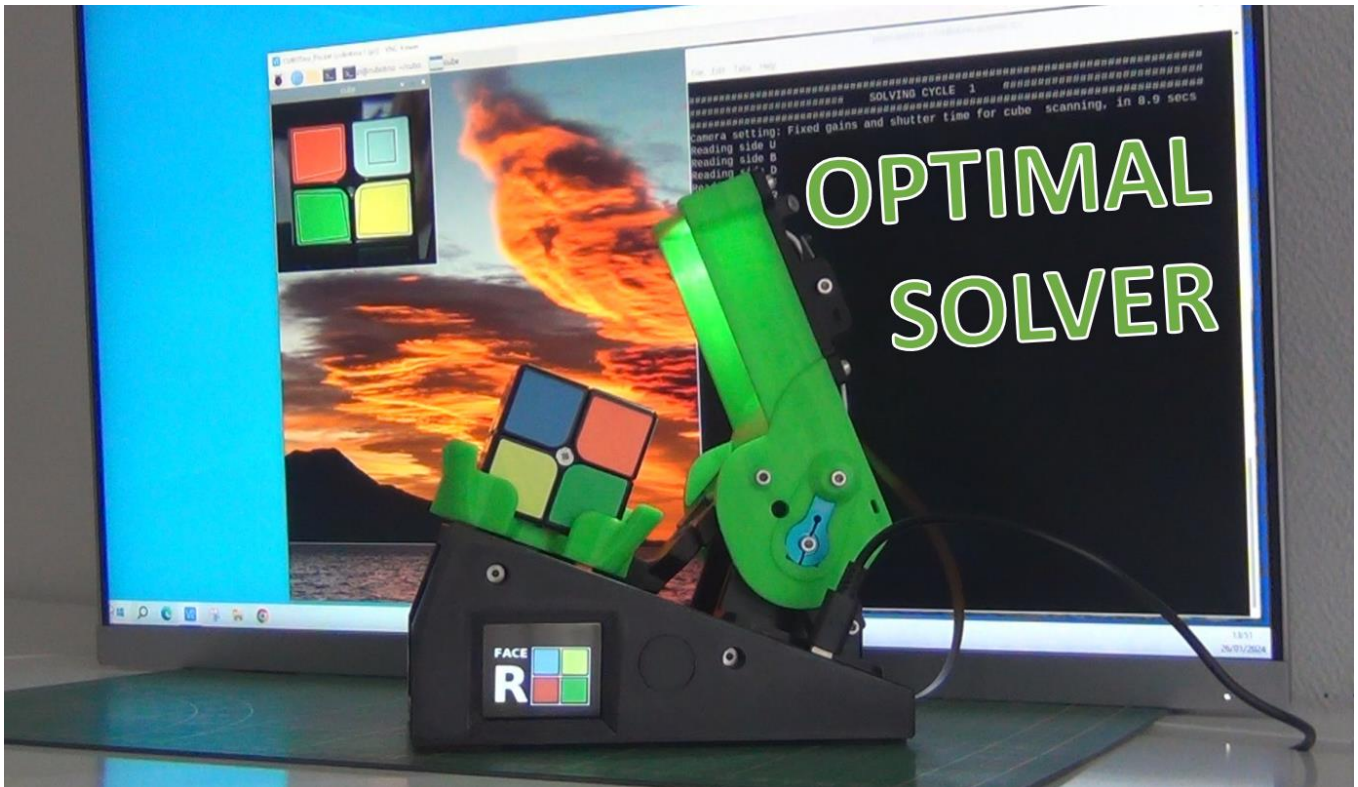
<https://www.instructables.com/CUBOTino-Pocket-Optimal-Solver-Robot-for-Rubiks-Cu/>

https://github.com/AndreaFavero71/cubotino_pocket

Andrea Favero, Groningen (NL)

29/03/2024 Rev. 2.1 (always check if a newer version is available)

Robot demonstration at YouTube: <https://youtu.be/wB4MwINGon0>



Highlights:

- It works with a 2x2x2 Rubik's cube, also known as 'Pocket'.
- It is fully autonomous.
- It delivers the optimal solution.
- It takes less than 60 seconds for scanning and solving a scrambled cube (95% CI).
- About 100 € of materials.
- All the needed info and files have been made available.

1. Instructions: Order and organization

The present document is organized in one main body of text and two appendixes.

The main body contains all the necessary information to build, set up and operate the robot.

Appendix 1 provides additional information, useful when the robot is built and working.

Appendix 2 is informative about the Cubotino project and its various embodiments; it also explains how the robot works and solves the Cube.

How to build, set up and operate the Cubotino Micro

1. Supplies

2. Initial preparation

- a. Make the Connections_board.
- b. Setup the Raspberry Pi.
- c. Test the display and connections.
- d. Test PiCamera and set the focus for the right distance.
- e. Test the servos range and set them to their 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:
 - 1) Servos.
 - 2) Camera.
- b. How to operate the robot.
- c. Automatic start and Raspberry Pi shut-off by the robot.
- d. Troubleshooting.

Appendix 1 – Additional Information

(not strictly necessary to build the robot yet useful)

- a. Updating the files.
- b. Set up the Python development interface (THONNY).
- c. Fine Tuning of the servos via Command Line Interface (CLI).
- d. Parameters and settings.

Appendix 2 – The Cubotino Project

(My preferred part 😊, yet not strictly necessary to build the robot)

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

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 at your own risk.

3. Manage expectations

Be prepared: the robot won't work magically right after assembly: **Tuning is mandatory!**

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

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

I strongly suggest to the interested maker, before anything else, to read thoroughly the present "How to" manual, then, after a cool-down time, to read it again...

... and now, let's start!

Summary

How to build, set up and operate the Cubotino Micro

1.	Instructions: Order and organization	2
2.	Safety	4
3.	Manage expectations	4
4.	Supplies.....	7
5.	The suggested cube	9
6.	Alternative and optional components.....	10
7.	Initial preparation	12
8.	Raspberry Pi Zero2 GPIO pinout	13
9.	Connections board.....	14
10.	Setting up the Raspberry Pi	24
11.	Display and connections test.....	40
12.	Servos check and set to neutral (mid) position	43
13.	Test the PiCamera.....	45
14.	Set the correct focus for the PiCamera	45
15.	3D printed parts.....	46
16.	Assembly steps	50
17.	Assembly details	51
18.	Raspberry Pi 2, 3 or 4.....	69
19.	Tuning	75
20.	Fine tuning the servos via GUI.....	84
21.	Tuning the camera via GUI	88
22.	How to operate the robot	92
23.	Automatic start.....	97
25.	Rpi shut down via Touch button.....	98
26.	Troubleshooting.....	99
27.	Collection of robot's pictures	111

Appendix 1 - Additional information










1.	Updating the files	117
2.	Files copied to Raspberry Pi.....	118
3.	Set Thonny IDE interpreter	120
4.	Fine tuning servos via CLI	126
5.	Parameters and settings.....	129


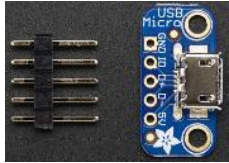
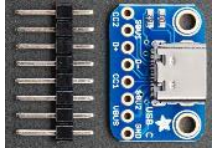
Appendix 2 - THE CUBOTINO PROJECT

1.	Introduction	135
2.	Project scope	135
3.	Robot name	136
4.	Models	136
5.	High level info	137
6.	Construction	138
7.	Computer vision part.....	139
8.	Color's detection strategy	146
9.	Optimal solution	148
10.	2x2x2 Rubik's cube and robot efficiency	149
11.	Robot solver moves	152
12.	Python main scripts, high level info.....	153
13.	Virtual solvers	158
14.	ChatGPT	160
15.	Commitment.....	161
16.	Credits.....	161
17.	Myself	162
18.	Revisions	163

Use the Summary links to quickly reach the chapters.

4. Supplies

Q. ty	Part	link to the shop I used	Cost (€)	Notes
1	Rubik's cube 2x2x2 (recommended MoYu RS2M 2x2x2)	See next page	6	 MoYu RS2M Magnetci
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?gateawayAdapt=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)	https://www.sossolutions.nl/	25 (2W version, Dec 2023)	
1	microSD HC 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 <i>s_mode</i> 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_s01?ie=UTF8&th=1	5	
1	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	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_od_dt_b_asin_title_s01?ie=UTF8&psc=1	4 (5 pcs)	

Qty	Part	link to the shop I used	Cost (€)	Notes
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)	
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:

Qty	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	Optional

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	MicroUSB 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).

5. The suggested cube

MoYu RS2M is a very nice cube:



In theory every 2x2x2mm Rubik's cube of ca 50mm will work, but this one is recommended.

The rationale to suggest this model:

1. Very low rotation friction
2. Cut corners mechanism.
3. Adjustable friction.
4. Robust construction.

As reference, a couple of shops (AliExpress and Amazon.com)

- https://www.aliexpress.com/item/1005005486654198.html?spm=a2g0o.productlist.main.3.63957390o7zaTX&algo_pvid=d091193b-eb86-411f-b00c-e44a1b3578d2&algo_exp_id=d091193b-eb86-411f-b00c-e44a1b3578d2-1&pdp_npi=4%40dis%21EUR%2119.31%215.6%21%21%21148.06%21%21%402103854017049066580241572e9423%2112000033274128968%21sea%21NL%21768246036%21&curPageLogUid=KkyuyfcHmahs&utm-url=scene%3Asearch%7Cquery_from%3A
- https://www.amazon.com/Evolution-Cube-Structure-Stickerless-Mini/dp/B09ZXBZRB6?ref=ast_sto_dp

Notes:

1. I'm neither affiliated with Amazon nor with MoYu; I'm just sharing my positive experience with this cube 😊.
2. The links are just used as reference, to find further information.

6. Alternative and optional components

Raspberry Pi ZeroW (instead of Zero2W)

In case Raspberry Pi Zero2W will suffer again for severe chip shortage, then Raspberry Pi ZeroW board is a valid alternative for this project:

Pro:

1. In 2022 and 2023 it had better availability than Zero2W, see notes below.
2. Hardware and Software compatibility.
3. Size.
4. Price.

Cons:

1. Performances deteriorates on OS10, yet acceptable results:
 - a. The Boot with script loading takes about 120secs, roughly double the time of Zero2W.
 - b. Cube status detection takes about 10 seconds more than Zero2W.
 - c. Solving time takes about 5% more than Zero2W.

On average the cube status detection and solving takes 90 seconds vs 70 seconds of a Zero2W.
2. Performances deteriorate too much on OS11, making the robot unpleasant to work with.

Purchasing a Raspberry pi Zero (Info valid at the moment of writing, 18 January 2024)

Both Raspberry Pi ZeroW and Zero2W are back in stock, notably in Amazon and AliExpress.

The Raspberry Pi official site (<https://www.raspberrypi.com/products/raspberry-pi-zero-w/>) displays the availability of the products through the official dealers in various countries. Select *Buy now*, enter your Country and check one by one the proposed shops for availability.

Notes:

- In some Countries / shops restrictions are applied: In the Netherland, where I live and placed my order, it has been periods in which orders were restricted (only one board per person per month).
- On December 2022 I spent 18.6€ (+ 3€ for shipment) and I got my first ZeroW in a couple of days. Yes, this is not the 'old' price, but competitors aren't cheap either....
- On February 2023 I spent 20.8€ (+ 10€ for shipment, ordered abroad) and I got my second ZeroW in a couple of days.
- On March 2023 I spent 21€ (included shipment) and I got my third ZeroW in a couple of days.
- On July 2023 I spent 22€ (+ 2.5€ for shipment) and I got a Zero2W from an official dealer in The Netherlands.
- In December 2023, a Raspberry Pi Zero2W without headers was 27,50€ through AliExpress, shipping included.

Alternative servos


It is possible to use servos having 270deg rotation combined with an extended Pulse Width (from 500µs to 2500 µs): When I started the CUBOTino platform project, I didn't know about the extended pulse width... a missed opportunity I would call it now!

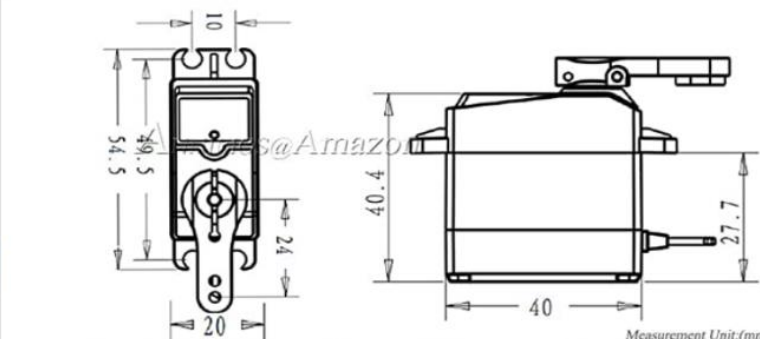
Rotation and Pulse Width ranges makes this servo a much better choice for this robot:

- 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 it seems widely available.

As reference this servo has these characteristics:

Qty	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	500~2500µsec	
Neutral position	1500µsec	
Rotating direction	Counterclockwise (when 500~2500 µsec)	

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.

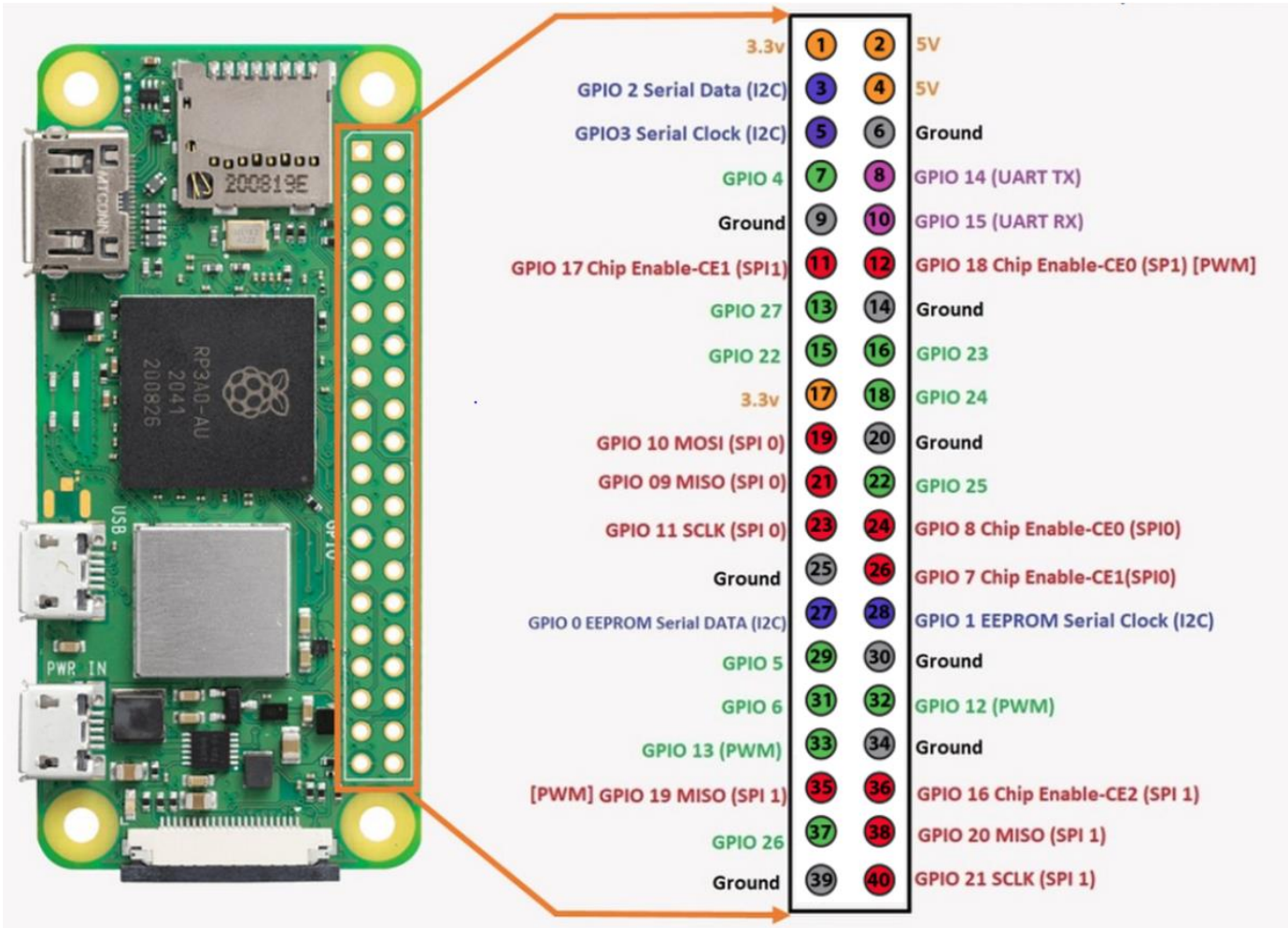
When moving from a 180° to a 270° version, the angle resolution should still be around 1.4°: Again, better than the servo I initially used and suggested.

7. Initial preparation

Before starting to assemble the robot, the following tasks must be completed:

1. Make the Connections_board (see specific chapter).
2. Setup the Raspberry Pi (see specific chapter).
3. Test the Connections_board, LED, servos, display (see preliminary test chapter)
4. Adjust the PiCamera focus, and test it (see preliminary test chapter)
5. **Check the servos rotation range and set them to their middle position** (see specific chapter).
6. Print the parts (see specific chapter).

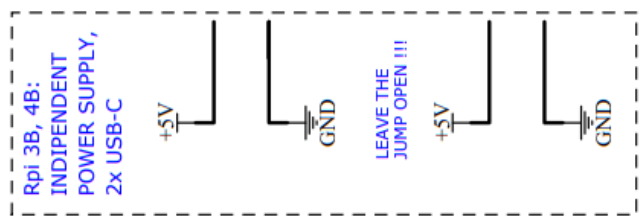
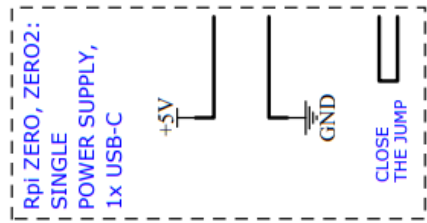
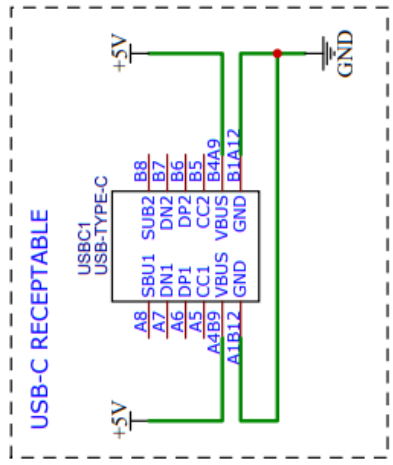
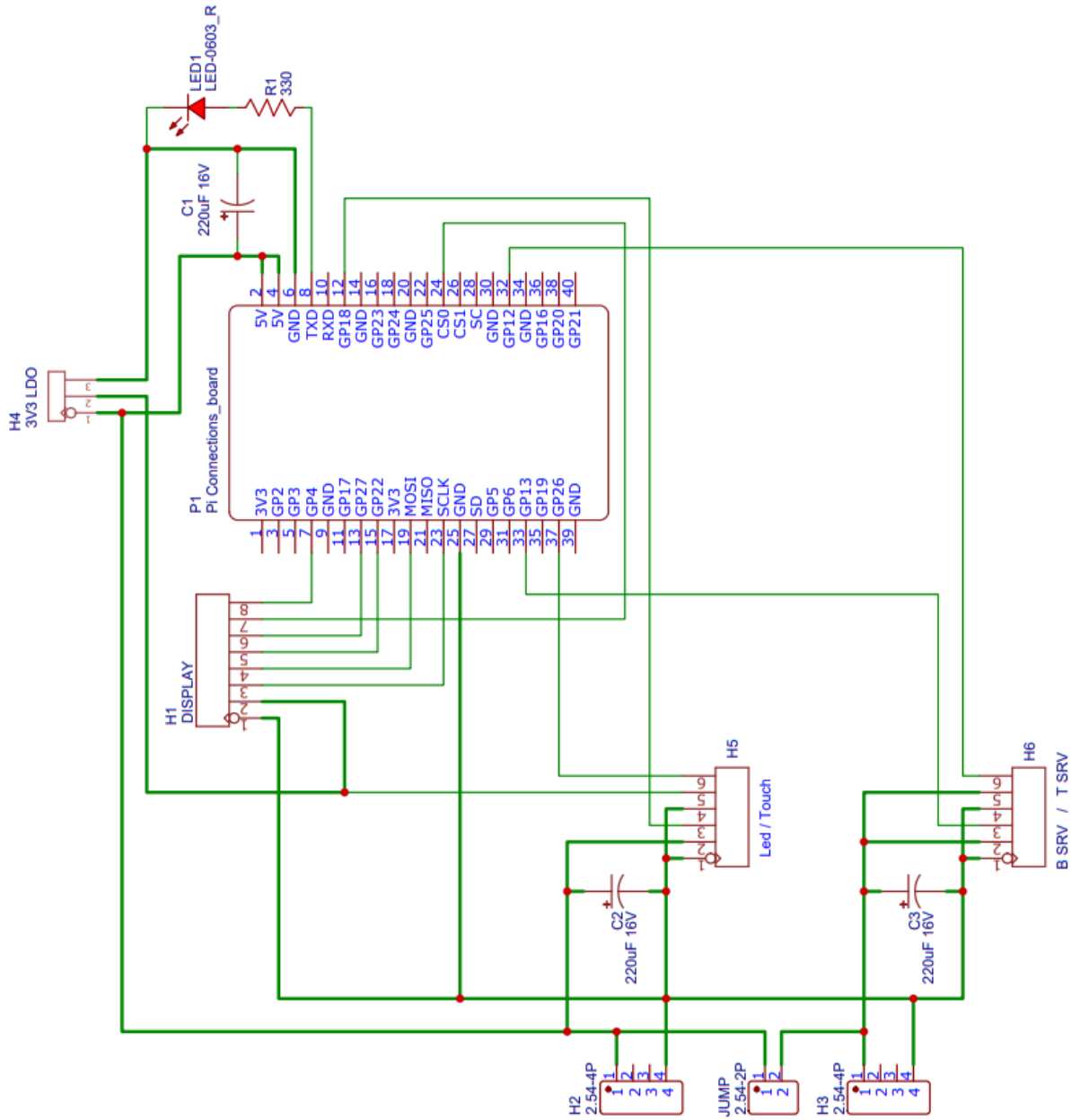
8. Raspberry Pi Zero2 GPIO pinout



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 SPI 0)	7 Display CS (Chip select)
19	GPIO 10 MOSI (SPI 0)	4 Display SDA (Serial Data Pin)
23	GPIO 11 SCLK (SPI 0)	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

9. Connections board



There are some options for the Connections board:

- A) Chad (Vanblokkland) has drawn and made available the Gerber files to order the pcb to a pcb manufactures; 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.

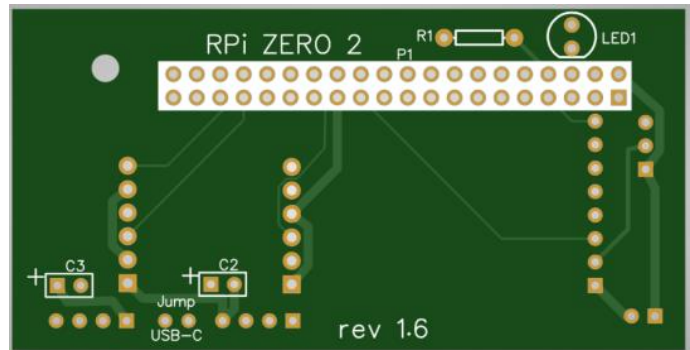
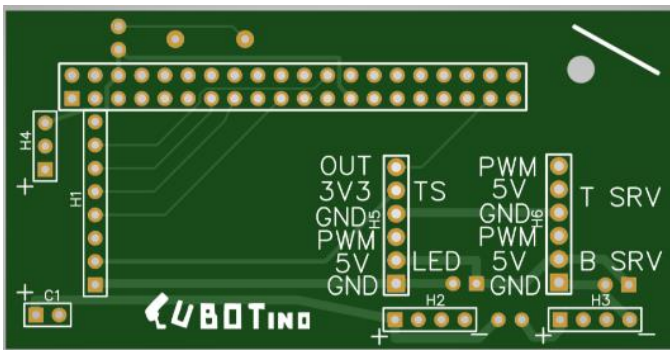
A) Notes for the printed circuit board:

- 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

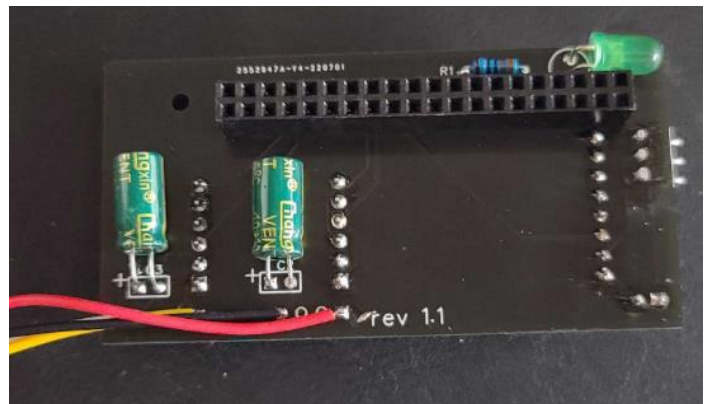
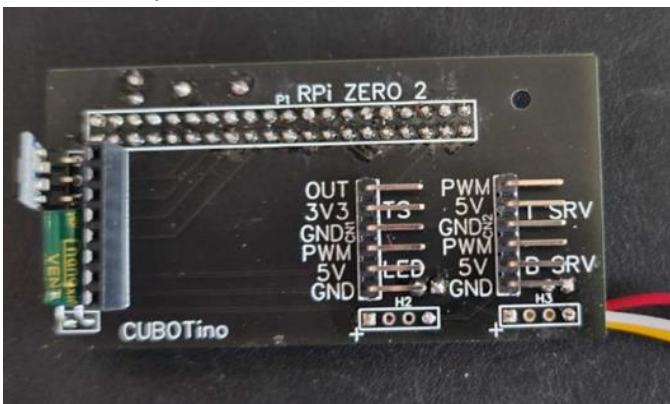
https://github.com/AndreaFavero71/cubotino_pocket/tree/main/connections_board/gerber

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.easyeda.com/en/PCB/Order-PCB>.
- 4. The boards I got from Chad, have been ordered at: <http://jlcpcb.com/>.



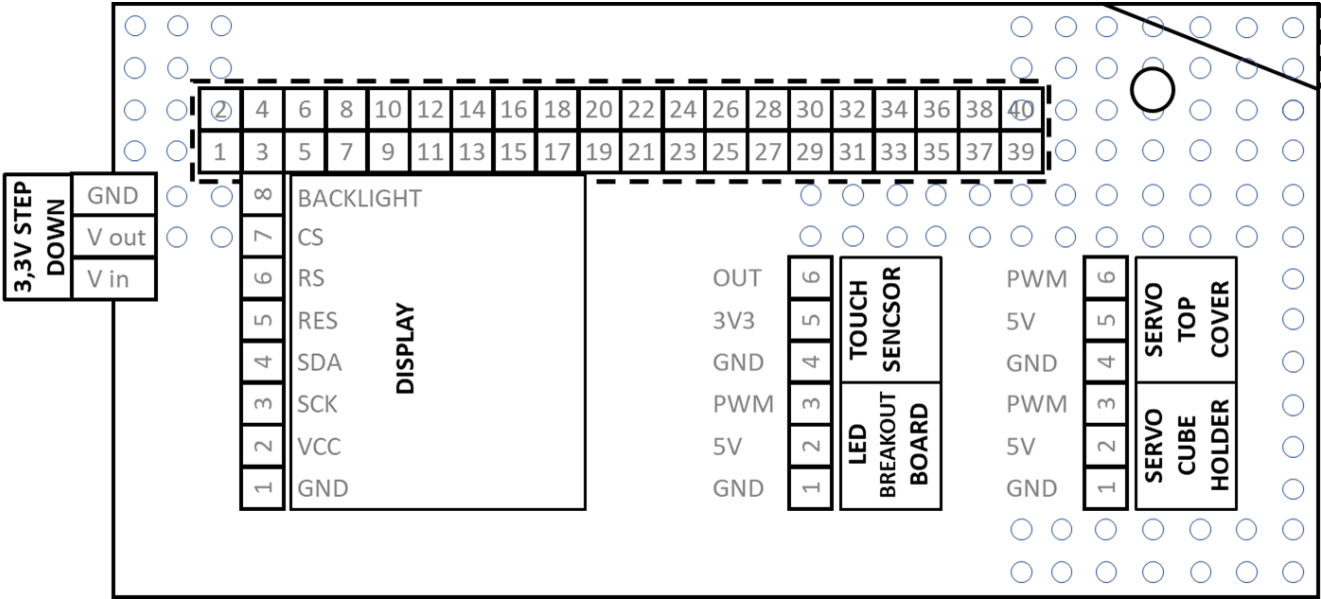
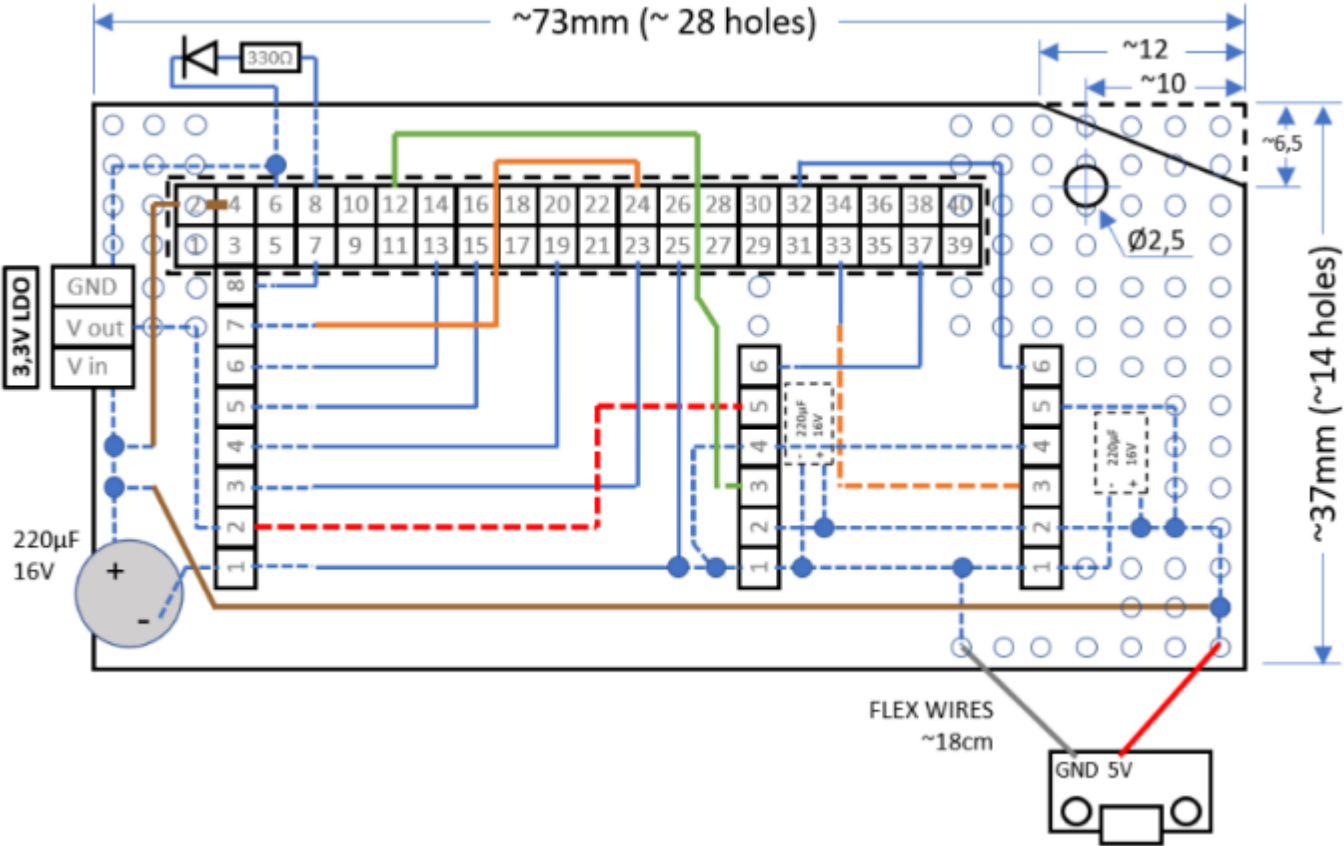
Just as example:



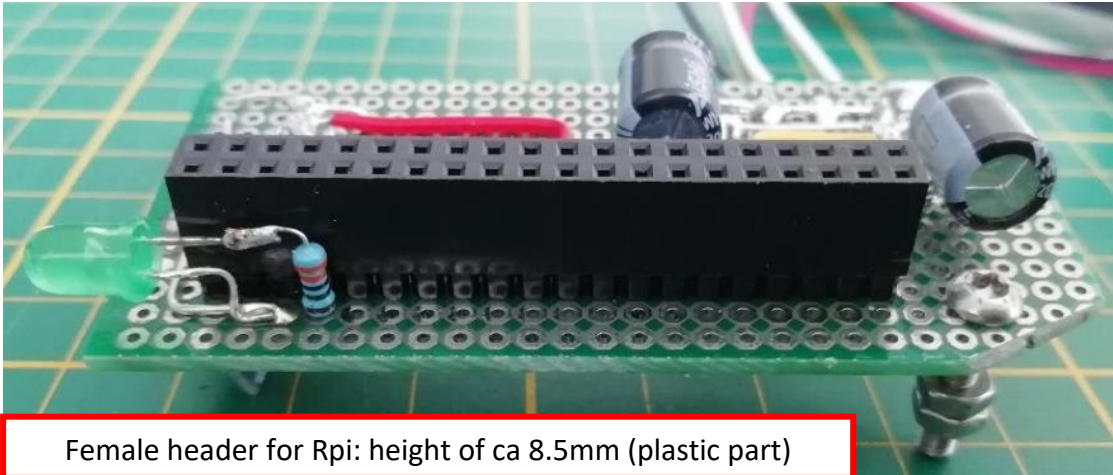
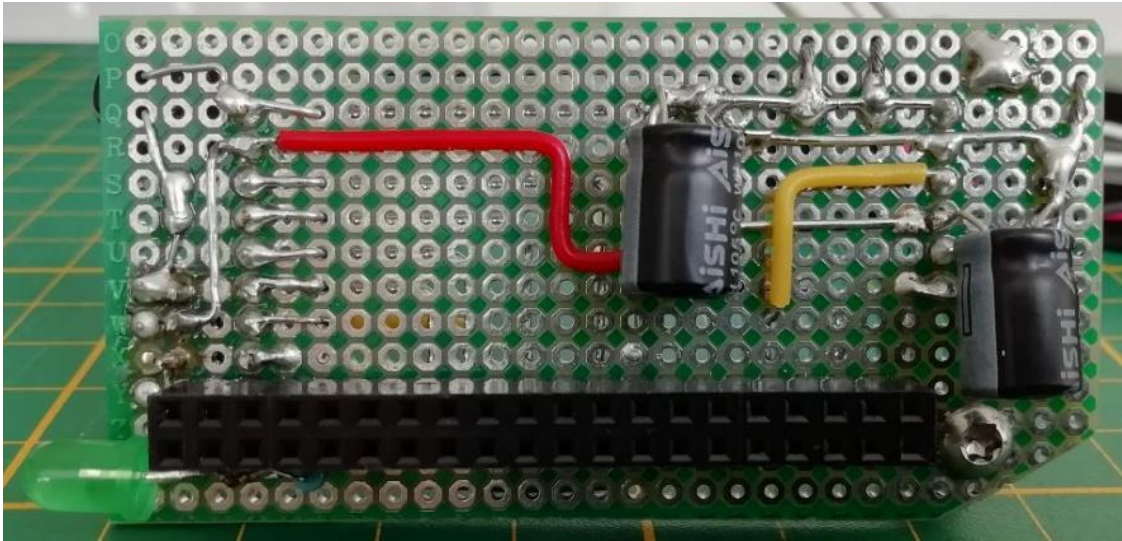
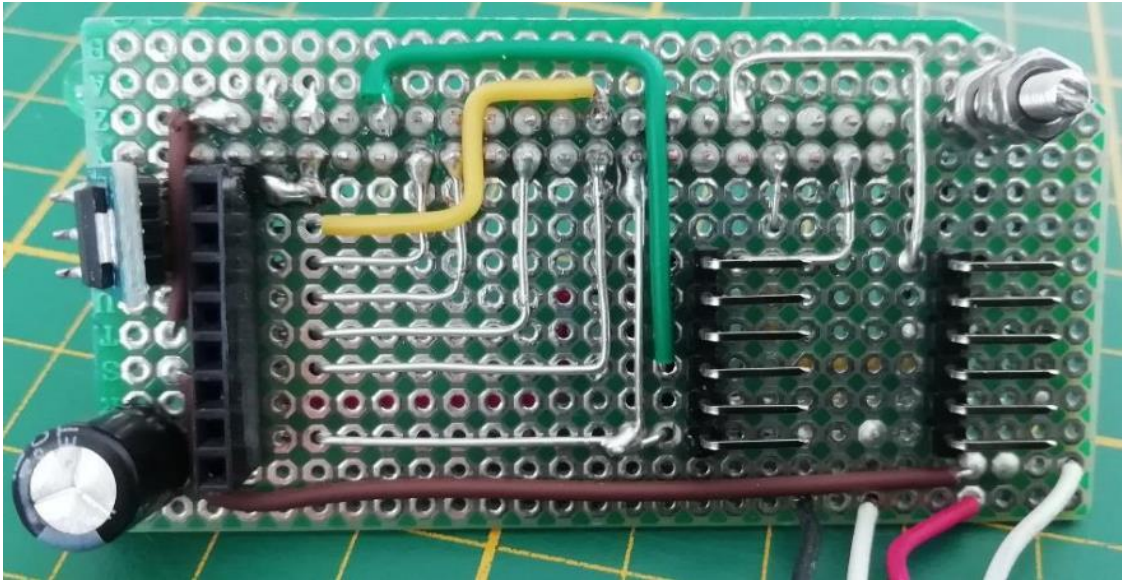
B) Notes for the prototyped board:

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. $\varnothing 2,5\text{mm}$ hole is for a M2,5mm bolt (and 3 nuts) to support the display. The highest nut should be at $\sim 10.5\text{mm}$ from the board surface.
12. Add the last 2 capacitors, close to the servo connectors, once the board has been tested.

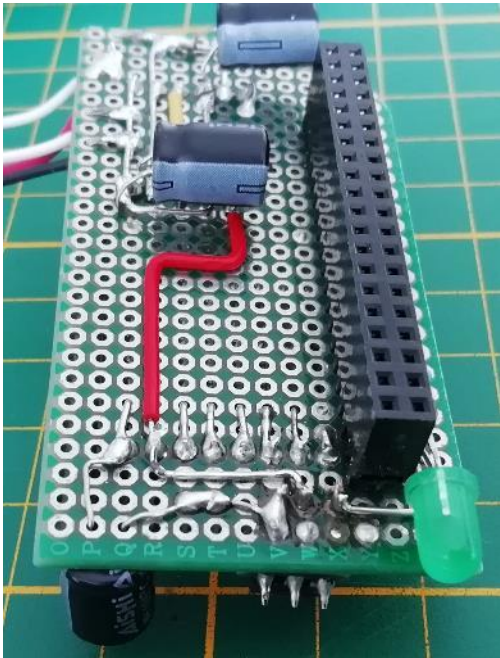
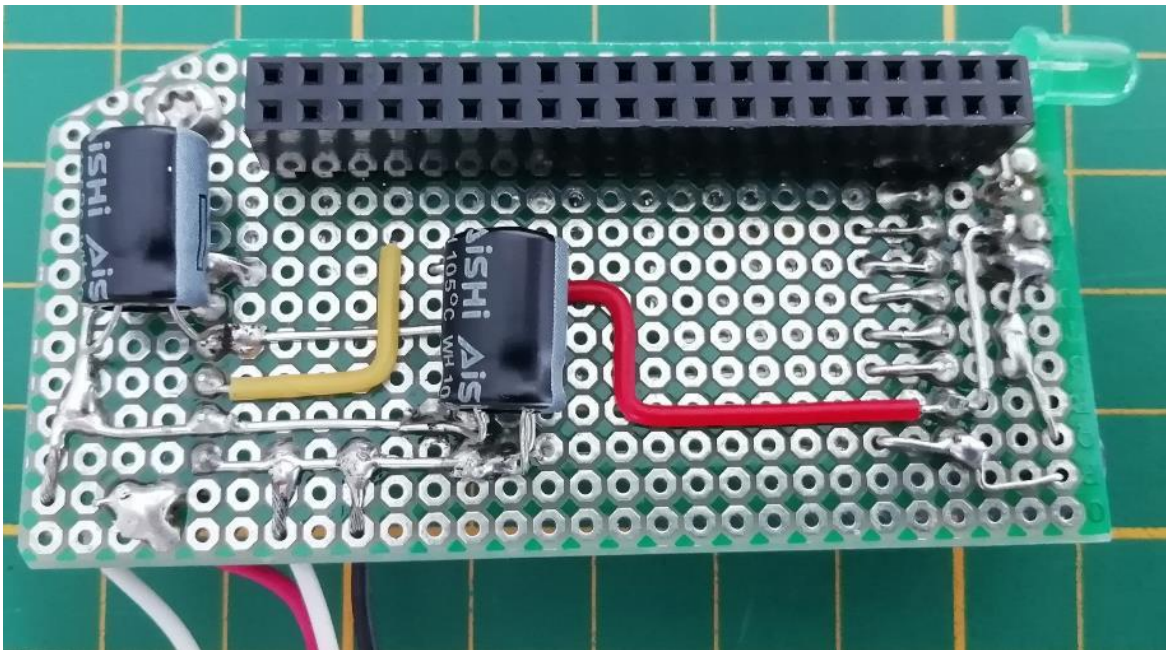
Front view (the 2x20 header is on the opposite side):



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



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



Wire soldering orientation to the USB breakout board. Recommended the USB-C type.



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.

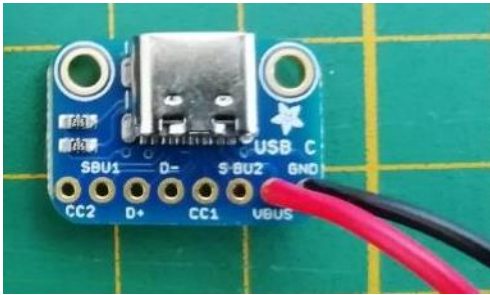
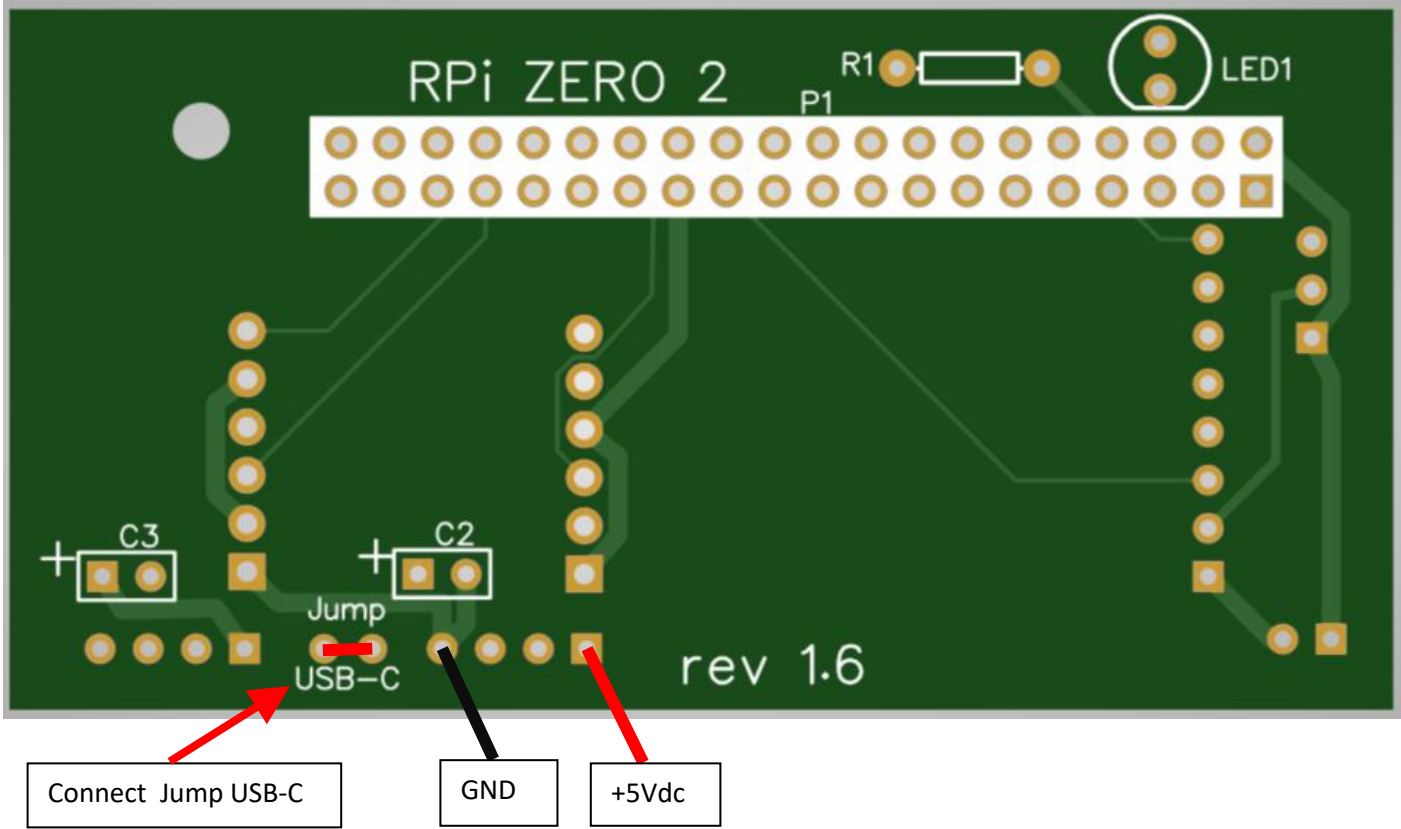
In case of Low Voltage Throttle check the tips at troubleshooting.

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:

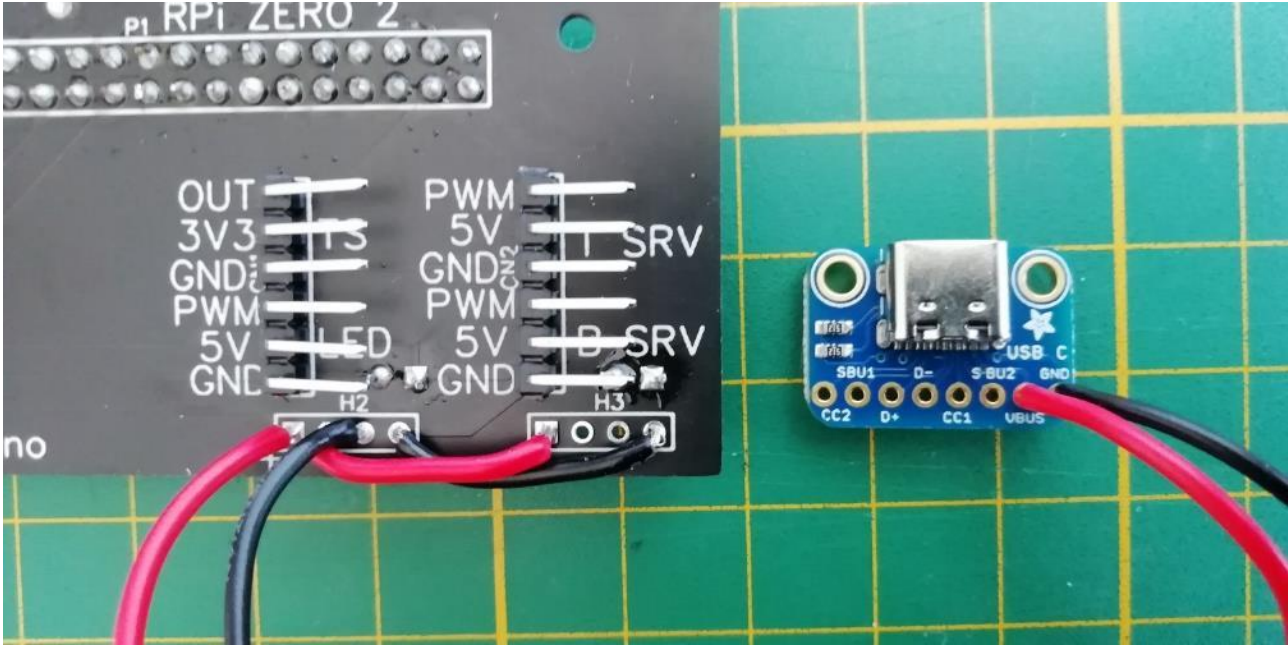
- 1. Connect the USB-C breakout board to either H2 or H3.
- 2. Connect the "Jump USB-C" pads.



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

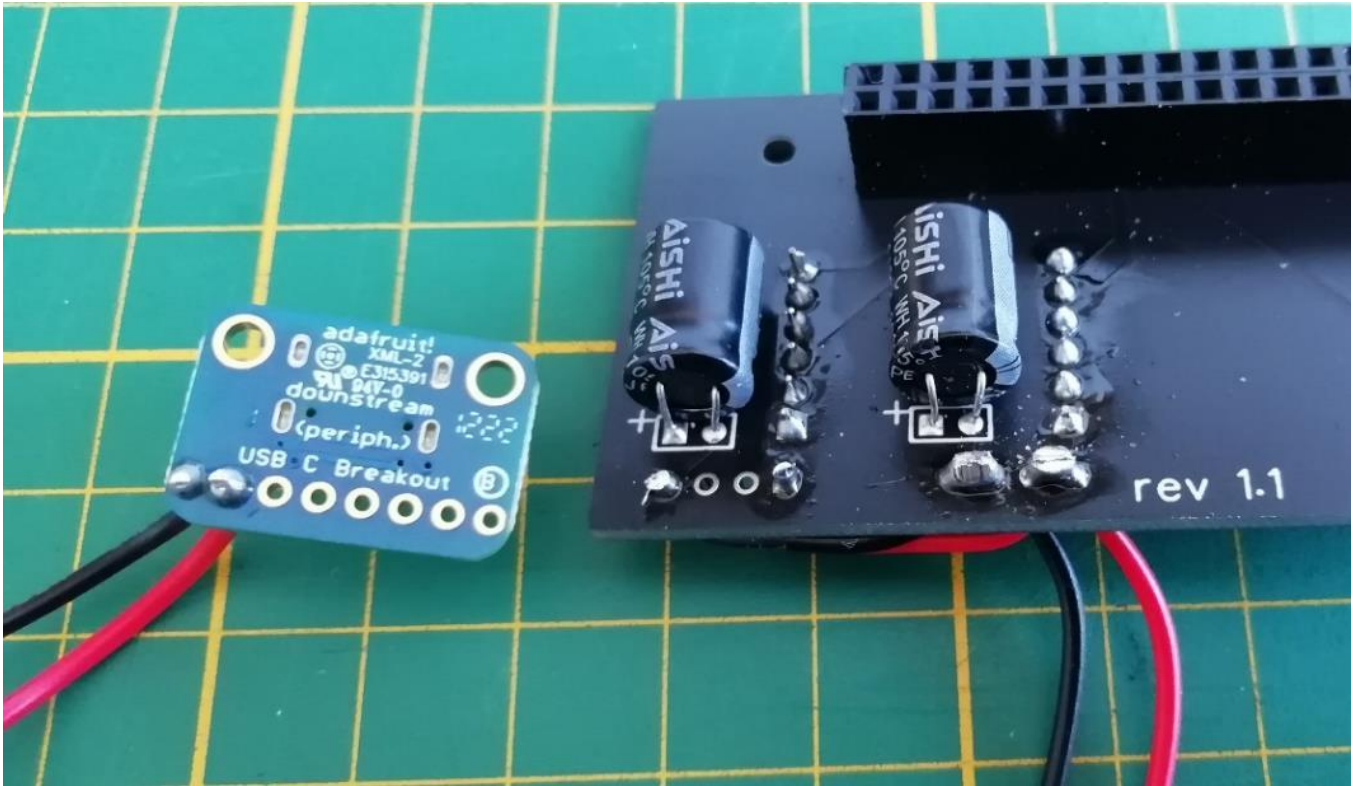
USB type-C connector is rated 3A, therefore possible to use a single power supply connections for the robot.

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:



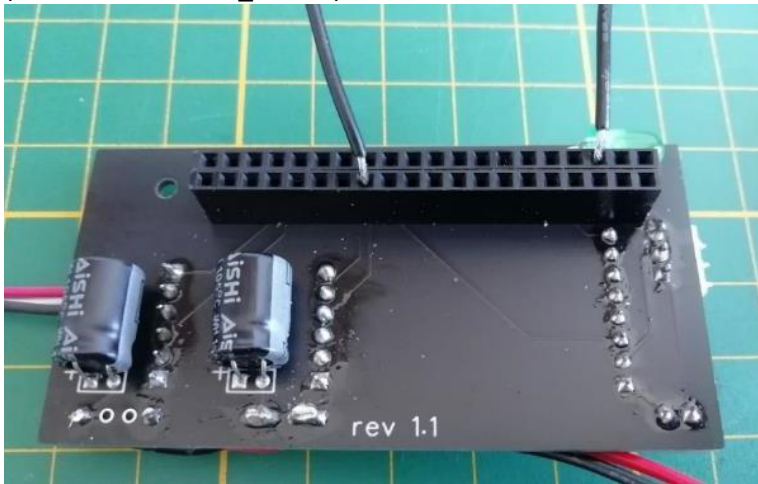
Testing the Connections_board before connecting it to the Raspberry Pi:

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

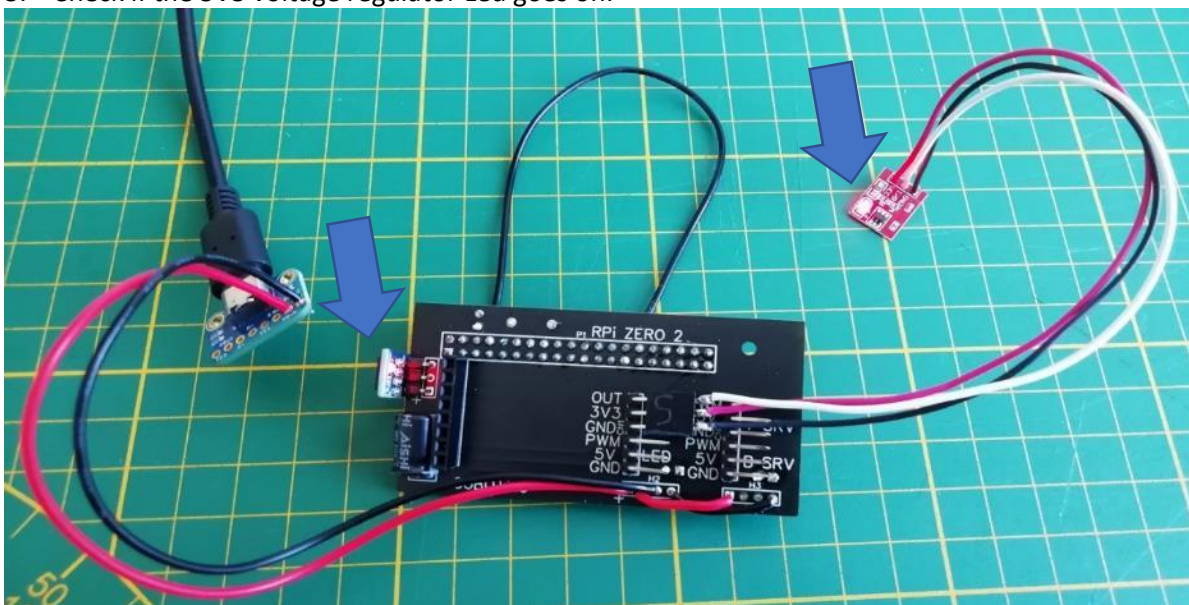
1. Connect the Touch_sensor breakout board.
2. Energize the board, via the USB type-C break out board or via the micro-USB breakout board connected to H2.
3. Touch the Touch_sensor pad and check if the breakout board Led goes on.

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 break out board or via the micro-USB breakout board connected to H2.
3. Check if the 3V3 voltage regulator Led goes on.



10. Setting up the Raspberry Pi

Setting up the Raspberry Pi essentially means:

1. Entering your personal information to the Imager (info like Wi-Fi SSID, Wi-Fi password, username for the robot, Country code, etc.).
2. Flashing the Raspberry Pi Operating System.
3. Install the needed libraries and files (all the needed files are stored in a GitHub repository).

This robot has been developed to work with OS 10 (Buster) and with OS (Bullseye); The largest difference is that Buster and Bullseye use a different camera library (picamera2 on Bullseye and picamera on Buster).

My suggestion is to flash OS 11 (Bullseye), as it's more recent and it will have more support going forward: This instruction covers the OS11 installation.

The OS must be downloaded from the Raspberry Pi official site; Since the release of Bookworm, the Buster OS is no longer listed by the Raspberry Pi Imager (also not as Legacy).

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

Step 2: Flash the OS to the microSD:

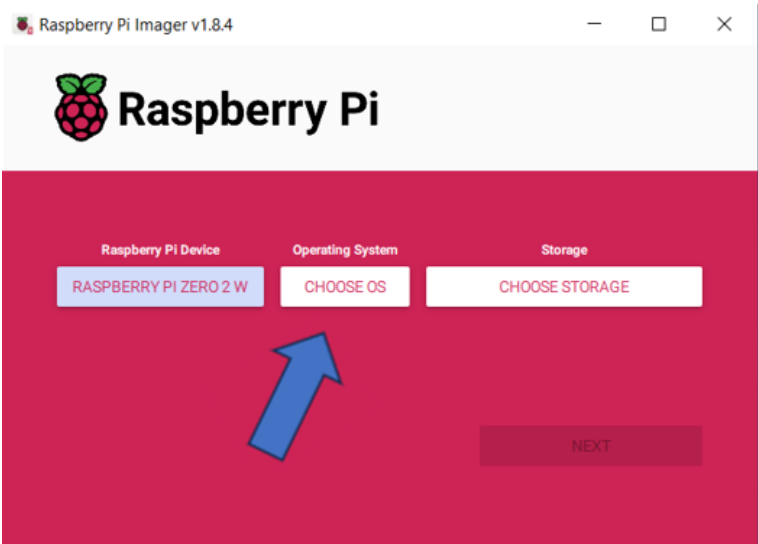
- a. Select **CHOOSE BOARD**:



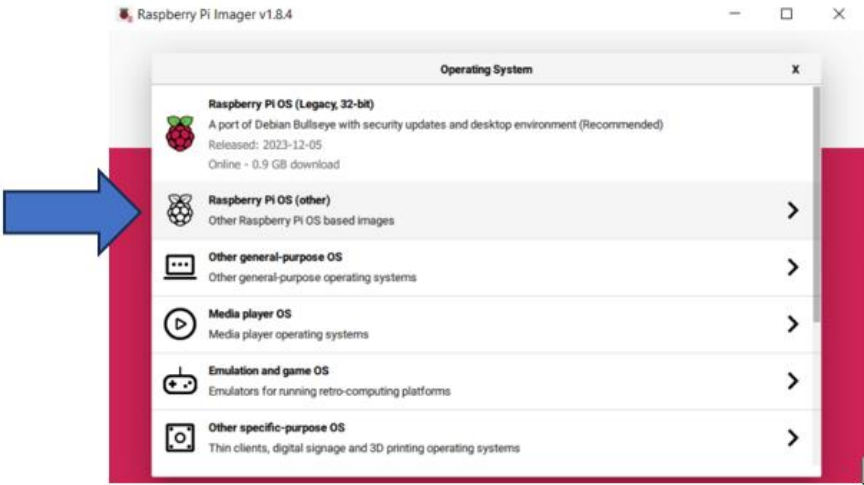
and select the board type you are using:



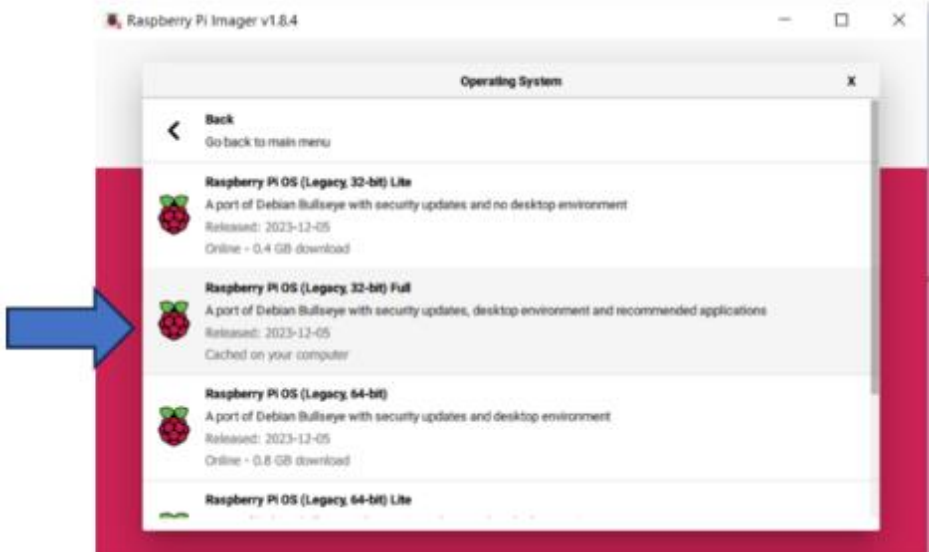
b. Select **CHOOSE OS**:




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



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



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



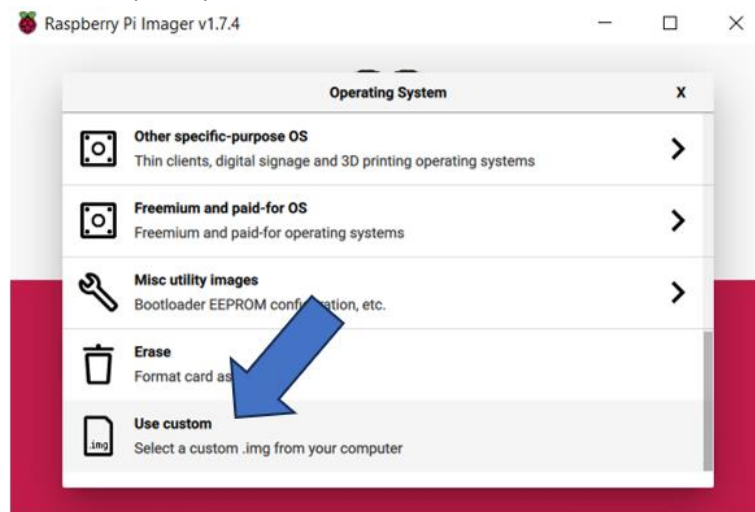
Raspberry Pi OS (Legacy, 32-bit) Full
 A port of Debian **Bullseye** with security updates, desktop environment and recommended applications
 Released: 2023-12-05

Step2d alternative.

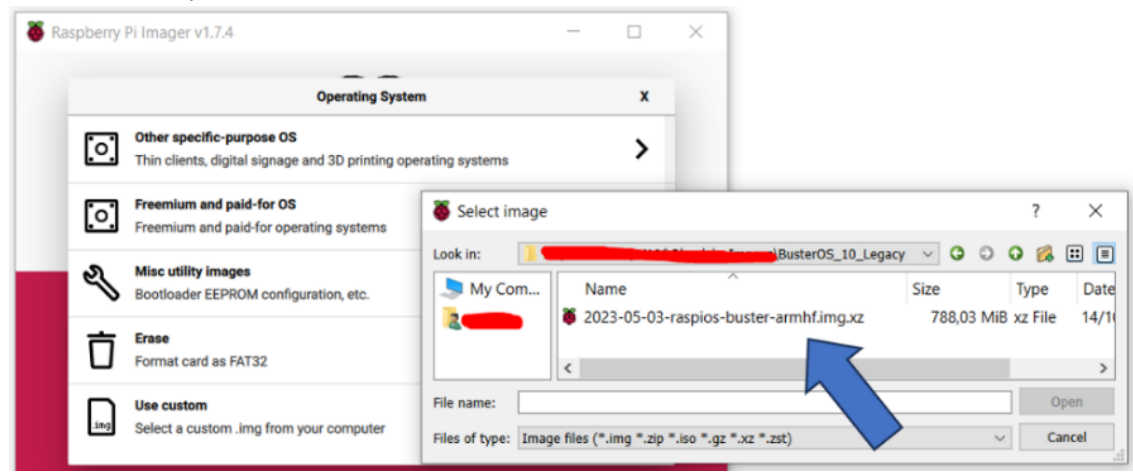
In alternative to the previous step, or as needed approach once Bullseye won't be listed into the Imager. In this case the OS image needs to be first downloaded in your PC; The official image for the Bullseye OS is:

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.

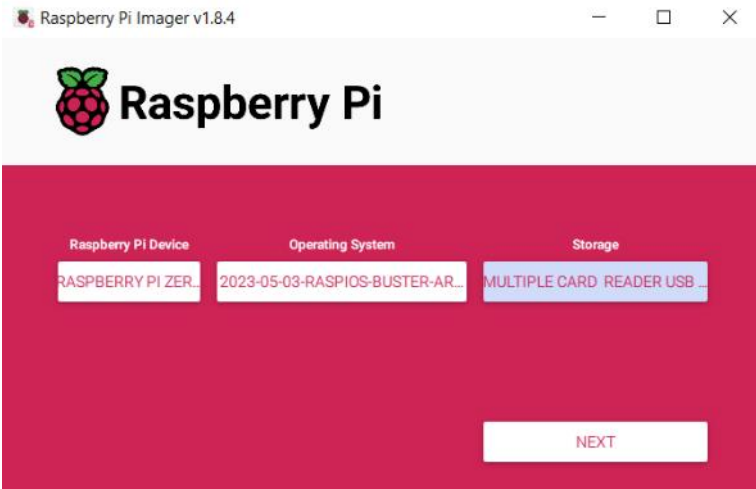


From here onward, the remaining steps are the same.

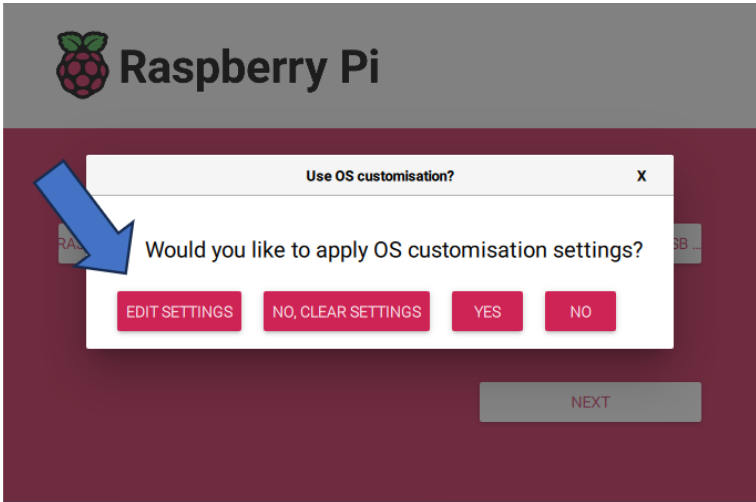
e. Select the **Choose Storage** disk (microSD card):



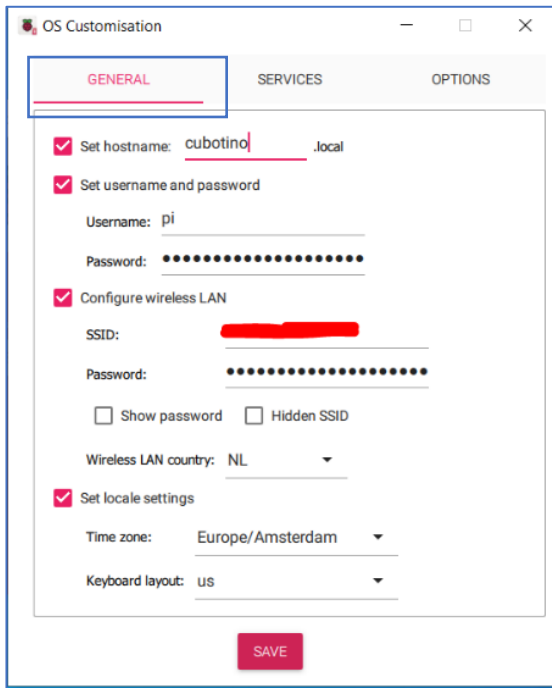
f. Select **NEXT**:



g. Select **EDIT SETTINGS**:



On settings, GENERAL tab, enter:



The screenshot shows the 'OS Customisation' window with the 'GENERAL' tab selected. The following settings are visible and configured:

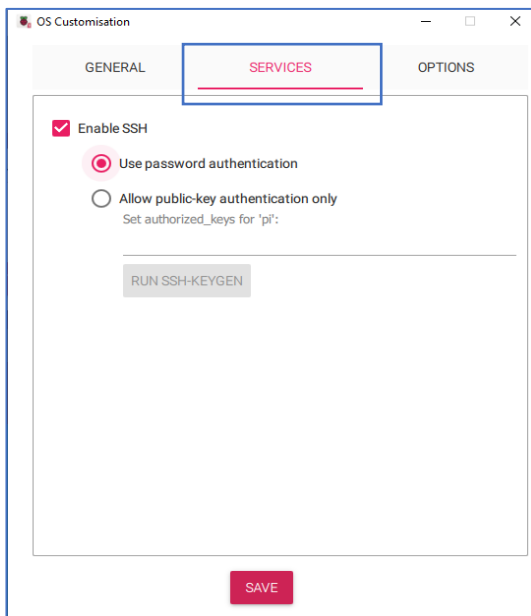
- Set hostname: `cubotino`.local
- Set username and password
 - Username: `pi`
 - Password: [masked]
- Configure wireless LAN
 - SSID: [redacted]
 - Password: [masked]
 - Show password Hidden SSID
 - Wireless LAN country: NL
- Set locale settings
 - Time zone: Europe/Amsterdam
 - Keyboard layout: US

A red 'SAVE' button is located at the bottom center of the window.

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

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

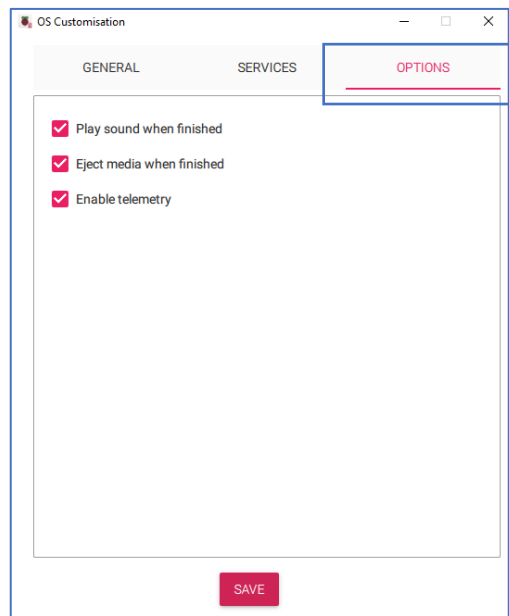
Double check **Enable SSH** is selected



The screenshot shows the 'OS Customisation' window with the 'SERVICES' tab selected. The following settings are visible:

- Enable SSH
 - Use password authentication
 - Allow public-key authentication only
 - Set authorized_keys for 'pi':
 - `RUN SSH-KEYGEN` button

A red 'SAVE' button is located at the bottom center of the window.



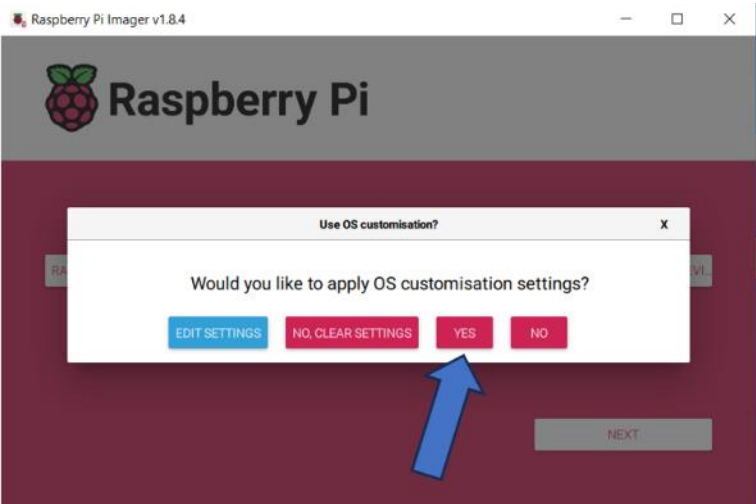
The screenshot shows the 'OS Customisation' window with the 'OPTIONS' tab selected. The following settings are visible:

- Play sound when finished
- Eject media when finished
- Enable telemetry

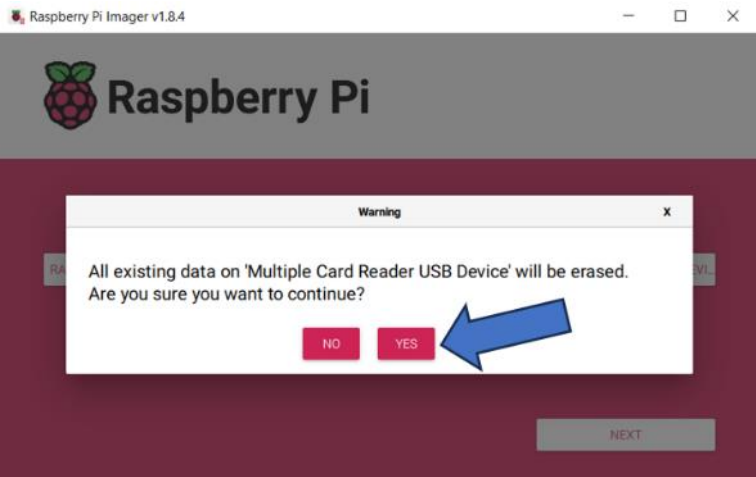
A red 'SAVE' button is located at the bottom center of the window.

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

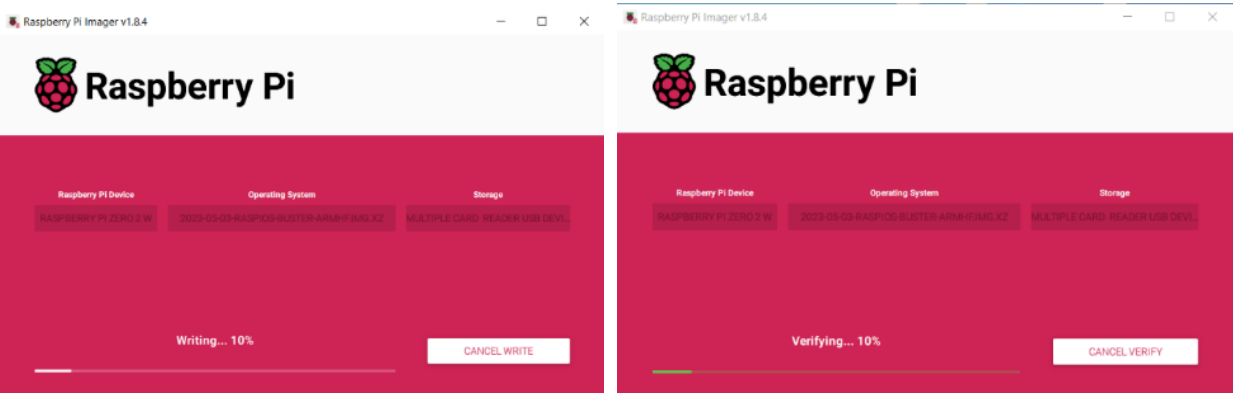
h. Select **YES** to use the settings:



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



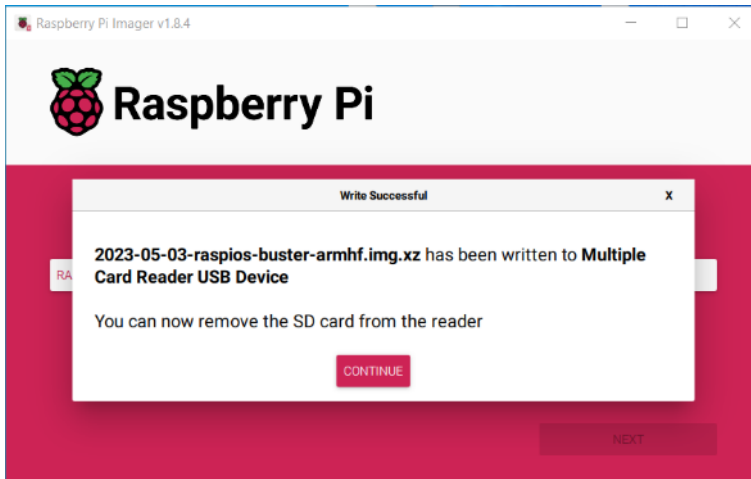
Raspberry Pi Imager starts writing the OS in the microSD, and it's followed by a verification:



Writing and verifying takes around 10 minutes.

Section2: Initial preparation

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



Note: In case you'd like to have multiple Wi-Fi possibilities:

- 1) Eject the card
- 2) Re-insert the card
- 3) jump to **Step 11**.
- 4) Once done come back to Step3.

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

The first boot takes longer, up to two minutes on Zero2W and up to 4minutes on ZeroW.

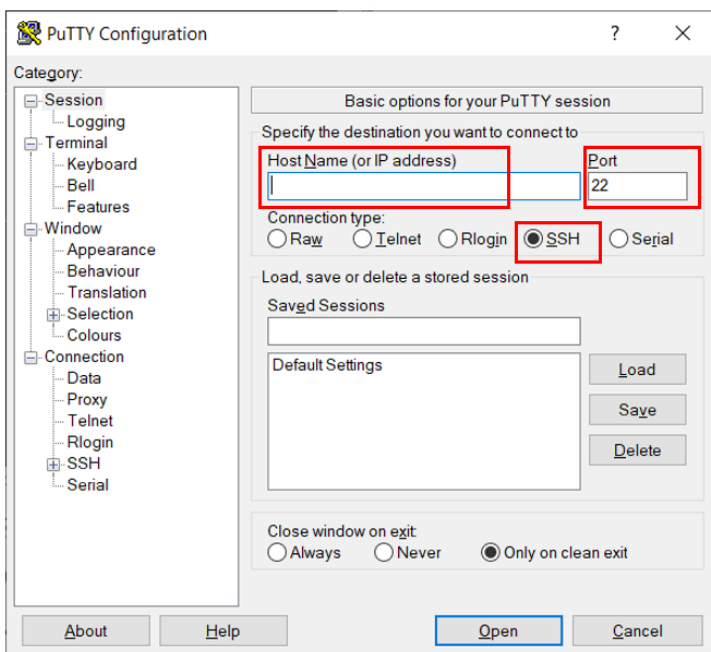
Wait until the LED stops blinking.

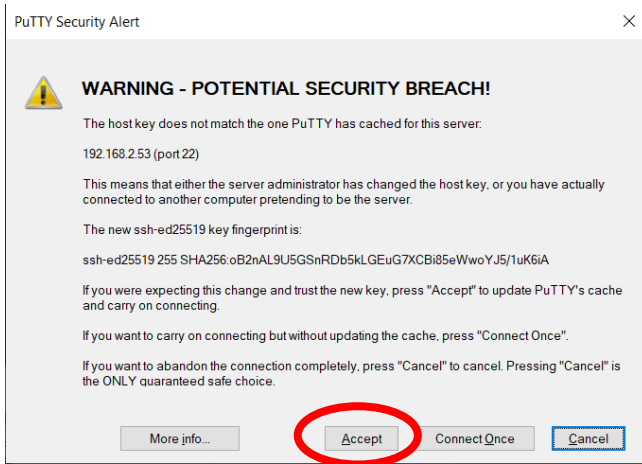
Step 4: from a command prompt try to connect to the Raspberry Pi via `ssh pi@cubotino.local`; insert the password and jump to Step 6

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....





Login as: *pi*

You'll be prompted to enter a password, "pi@xxx.xxx.x.xx's password:" enter *your_password* (*raspberrypi* in my case). Below image shows how the connection to Raspberry Pi looks like via SSH:

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

Step 6: Clone the repository; From the root (pi@cubotino:~ \$) type (or copy - paste) *git clone https://github.com/AndreaFavero71/cubotino_pocket.git*

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

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

Step 7: Start the installation:

- Enter cubotino_pocket/src folder from the root type: `cd cubotino_pocket/src`
- Change the setup.sh file property: `chmod +x install/setup.sh`
- Start the bash file that takes care of the installation: `sudo ./install/setup.sh` (attention to the dot).
- In about 10 minutes, depending on the internet connections speed, a Raspberry Pi Zero2W will complete the setup.
- After the installation ends, a Raspberry Pi reboot is proposed. If this is not your case, it means something went wrong. Scroll the terminal text and try to solve what went wrong.
Afterward, rerun: `sudo ./install/setup.sh`
- Once requested confirm the reboot with a `y` and press enter; In case the prompt is returned then type `sudo reboot`
- When the Raspberry Pi boot ends, the integrated green LED will stop blinking.
- As reference check the printouts of the installation (/doc/ the Installation_printout.pdf)

The installation enables the necessary Raspberry Pi interfaces: Camera, SPI.

At this point the installation is pretty much completed, HURRA 😊!

From `/home/pi/cubotino_pocket/`, the main robot folder, below folders will be made at Raspberry Pi:

<code>connections_board</code>	info for the Connections_board
<code>doc</code>	How_to_build... .pdf file
<code>extra</code>	link to the Instructables page of this robot
<code>images</code>	Cubotino logo image for the display
<code>stl</code>	all the robot stl files
<code>stp</code>	all the robot stp files (not locally downloaded)
<code>src</code>	all the robot specific code files
<code>src/solver2x2x2</code>	lookup tables for the Kociemba solver
<code>src/install</code>	setup.sh bash file: Do not use this file!

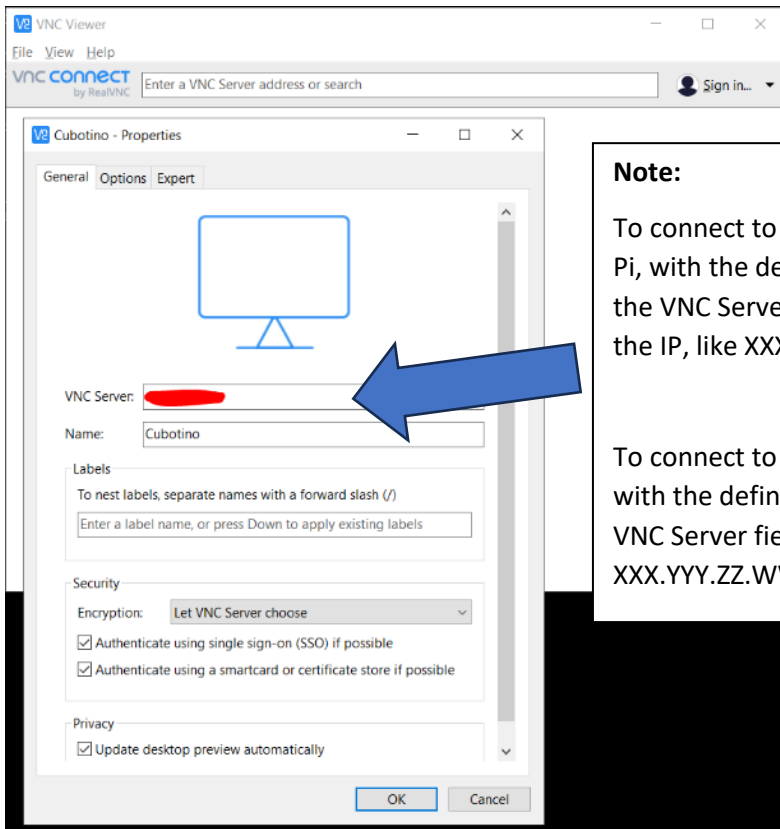
At the next connections with the Raspberry Pi, it will be convenient to use VNC Viewer, because it provides a useful graphical support.

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

You must then register (for free) with RealVNC and create an account, to be able to set up a new connection with the Raspberry Pi board.

Step 8: Create a new connection in VNC Viewer

Create a new connection: File, New connections, insert the IP address of your Raspberry Pi into the VNC Server frame, and a name, Cubotino, as an example, in the Name frame.

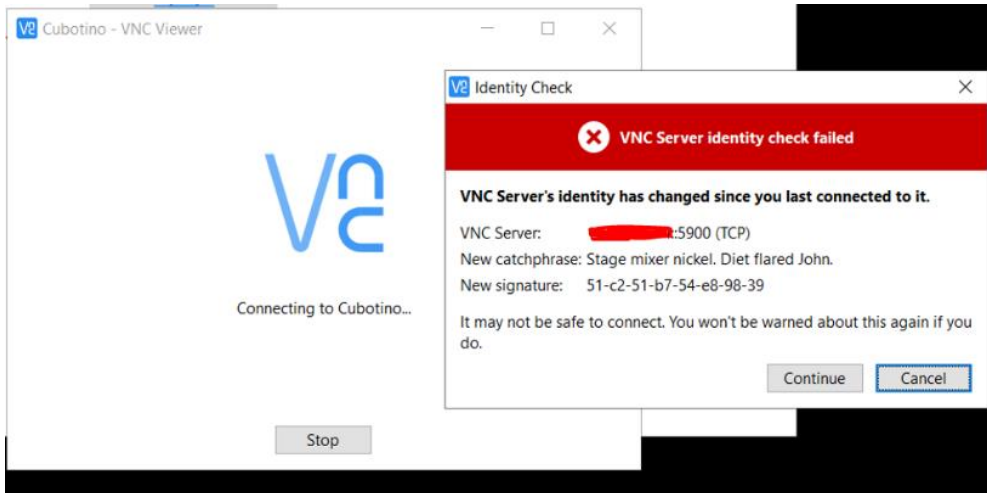


Note:

To connect to a “**Bullseye**” based Raspberry Pi, with the defined command at *crontab -e*, the VNC Server field requires the “:1” after the IP, like XXX.YYY.ZZ.WW:1

To connect to a “**Buster**” based Raspberry Pi, with the defined command at *crontab -e*, the VNC Server field only needs the IP, like XXX.YYY.ZZ.WW

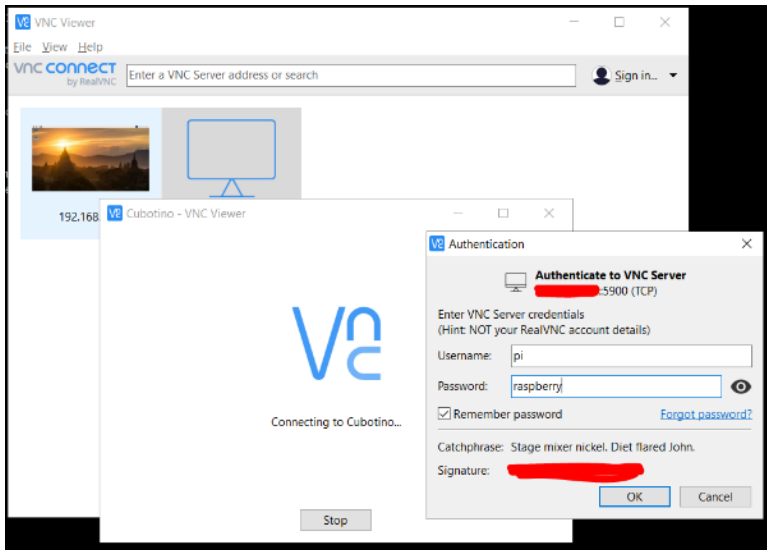
Accept the warning by clicking on “Continue”:



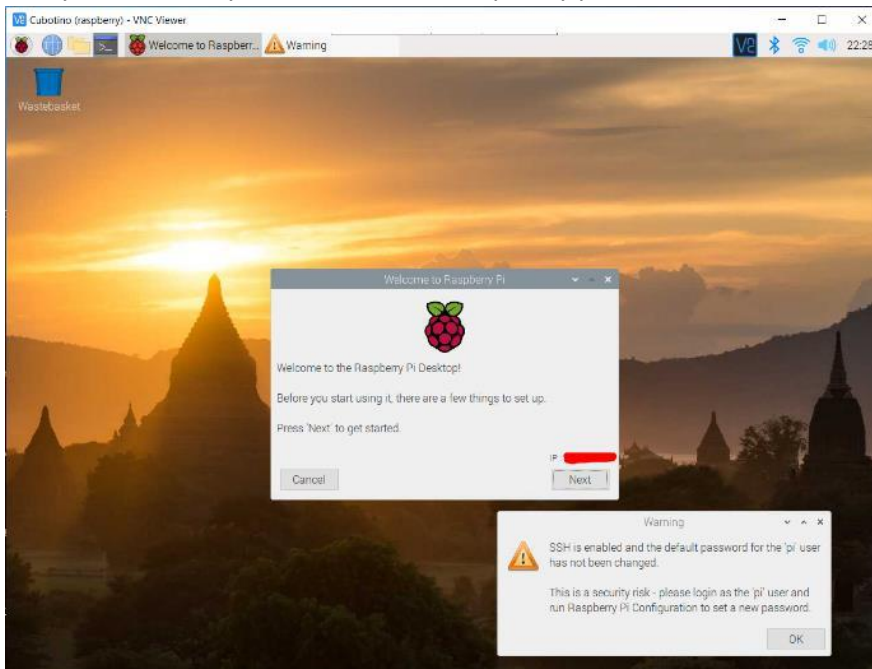
Section2: Initial preparation

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

Check the option “Remember password:”



And you're virtually connected to the raspberry pi monitor:

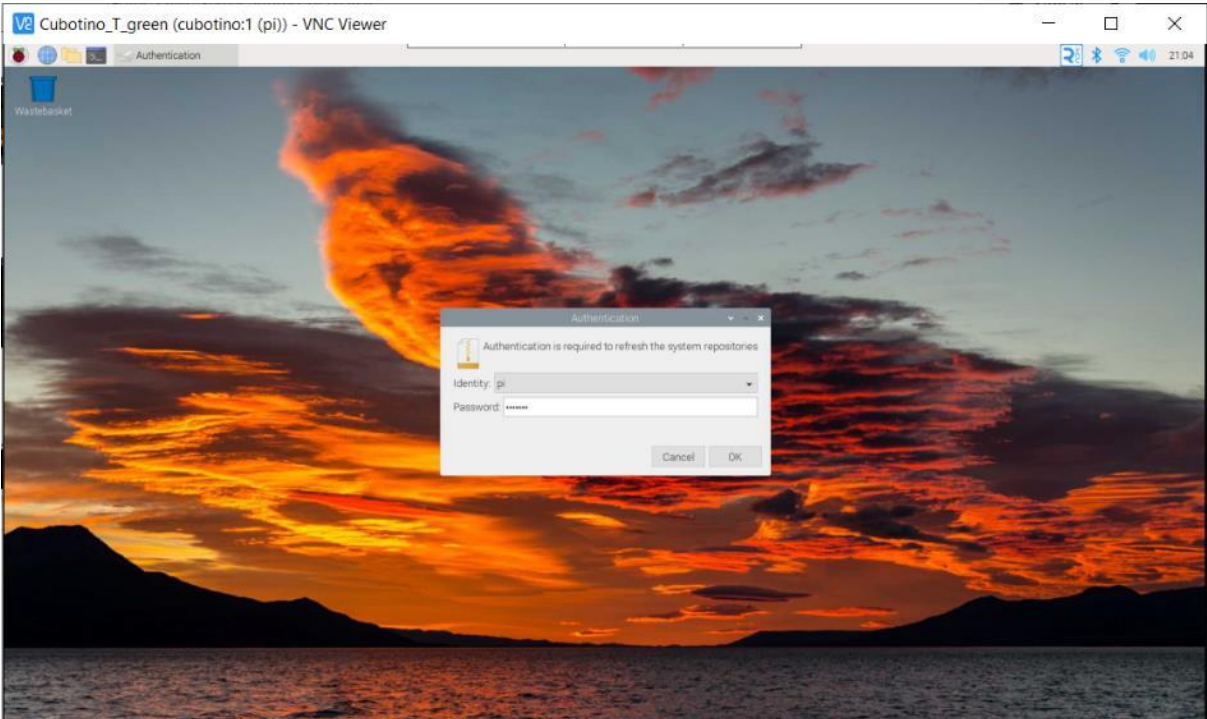


Check, and accept 😊, the warning.

Complete the settings (only if not already provided the info at Raspberry Pi Imager Settings)

- Set Country.
- Change password (in my case again *raspberrypi*).
- **Ignore** the Setup screen and click on “continue” (see below).
- **SKIP** Select Wi-Fi Network.
- **SKIP** Update Software.

Bullseye" example:



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

Step 9: Eventually change the monitor size.

From the root or from the Virtual Environment (venv), type *sudo crontab -e*

```
pi@cubotino:~/cubotino_pocket/src $ sudo crontab -e
```

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

```
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]:
```

The Crontab content will look like:

```
pi@cubotino: ~/cubotino_pocket/src
File Edit Tabs Help
GNU nano 5.4 /tmp/crontab.lx5vKW/crontab *
MAILTO=""
@reboot su - pi -c "/usr/bin/vncserver-virtual :1 -randr=1920x1080"
#@reboot /bin/sleep 5; bash -l /home/pi/cubotino_pocket/src/Cubotino_P_bash.sh > /h>
^G Help      ^O Write Out  ^W Where Is   ^K Cut        ^T Execute    ^C Location
^X Exit      ^R Read File  ^\ Replace    ^U Paste      ^J Justify    ^_ Go To Line
```

To tune the servos via the Graphics interface (GUI), it is necessary to use a large screen, roughly 1920x1080 pixel. that is the default setting proposed.

In case you'd like to change it, edit those values.

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

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

Step 10: Wi-Fi stability:

VNC connection drop is a common problem with the default settings of the Raspberry Pi. Before experiencing any worrisome loss of dialog, it is suggested, as a preventive measure, to change the following settings:

1. remove the Wi-Fi power management: *sudo iwconfig wlan0 power off*
(to verify the Wi-Fi status: *iwconfig wlan0*)
2. edit the file /etc/ssh/sshd_config: *sudo nano /etc/ssh/sshd_config*
3. add at the end: *IPQoS cs0 cs0*

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

Step 11: multiple Wi-Fi settings:

Adding a second (or more) Wi-Fi connection, for instance to add your phone Wi-Fi 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 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 Fair, 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 WIFI's 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.

As an example:

```
CTRL interface=DIR=/var/run/wpa_supplicant GROUP=netdev  
update_config=1  
country=FR  
network={  
ssid=Jules  
psk=Cesar  
priority=10  
}  
network={  
ssid=Bonjour  
psk=Andrea  
priority=20  
}
```

Step 12: Rpi memory management settings:

(I'm not a real Raspberry Pi expert, the description below 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.

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 irresponsive 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 swapping 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`, the `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 caching.

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

In Rpi boards with limited RAM (512Mb), 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`

Step 13: Make a backup image of the microSD:

This is the perfect moment to secure the time 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.

The well-known software PowerIso allows you to navigate in both Linux and Windows partitions in the backup iso file.

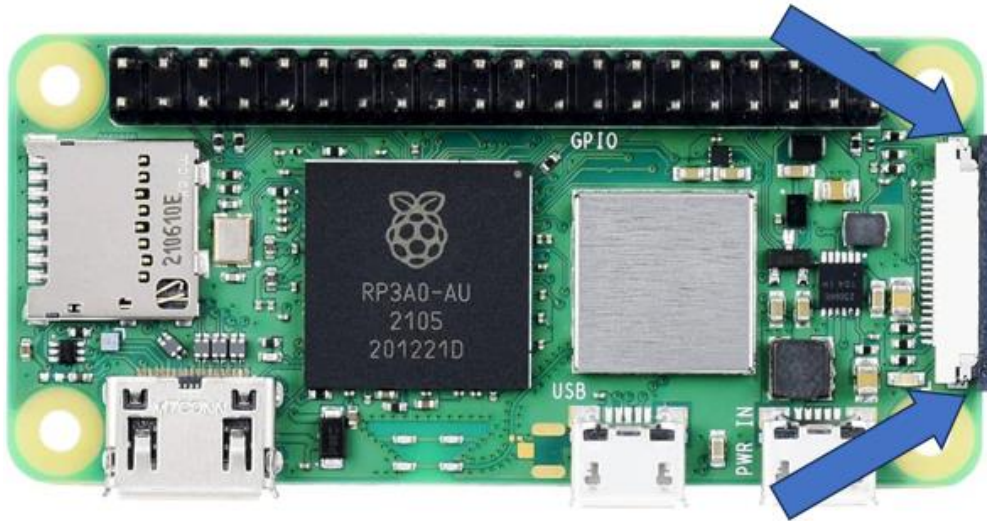
A second backup will take place later, when the settings for the servos and the camera will be finalized.

11. Display and connections test

At this point it is possible to make a first bench test of the electrical parts on the table.

Connect:

1. The camera to the Raspberry Pi



To unlock the ribbon cable connector of the Raspberry Pi Zero, push very gently the black part of the connector forward, in the direction of the arrows.

Insert the narrow end of the ribbon in the Pi, making sure the connections side of the ribbon is facing down. The ribbon should enter about 3mm into the connector.

Push very gently the black part in its original place, to lock the ribbon.

Make sure that the ribbon cable is fully seated in its connector before and after locking it.

Do the same on the camera side, with the wide end of the ribbon. The ribbon should enter ~4mm into the PiCamera connector, as it's a larger connector.

2. The servos to the Connections_board.
3. The Touch button to the Connections_board.
4. The Led module to the Connections_board.
5. The Connections_board to the Raspberry Pi.
6. The display to the Connections_board

Energize the circuit and wait until the Raspberry Pi green light stops blinking. This is visible from the top hole on the Structure.

Connect to the Raspberry Pi via SSH (i.e., putty). If the installation went well, the connection should be possible.

Display test:

The display can be tested:

- a. enter the cube folder. From root, type `cd cubotino_pocket/src`
- b. activate the virtual environment: type `source .virtualenvs/bin/activate` Pay attention to the dot before virtualenvs.
- c. run the script `python Cubotino_P_display.py`

```
pi@cubotino: ~/cubotino_pocket/src
File Edit Tabs Help
pi@cubotino:~/cubotino_pocket/src $ source .virtualenvs/bin/activate
(.virtualenvs) pi@cubotino:~/cubotino_pocket/src $ python Cubotino_P_display.py

Activated pigpiod

Display initialized

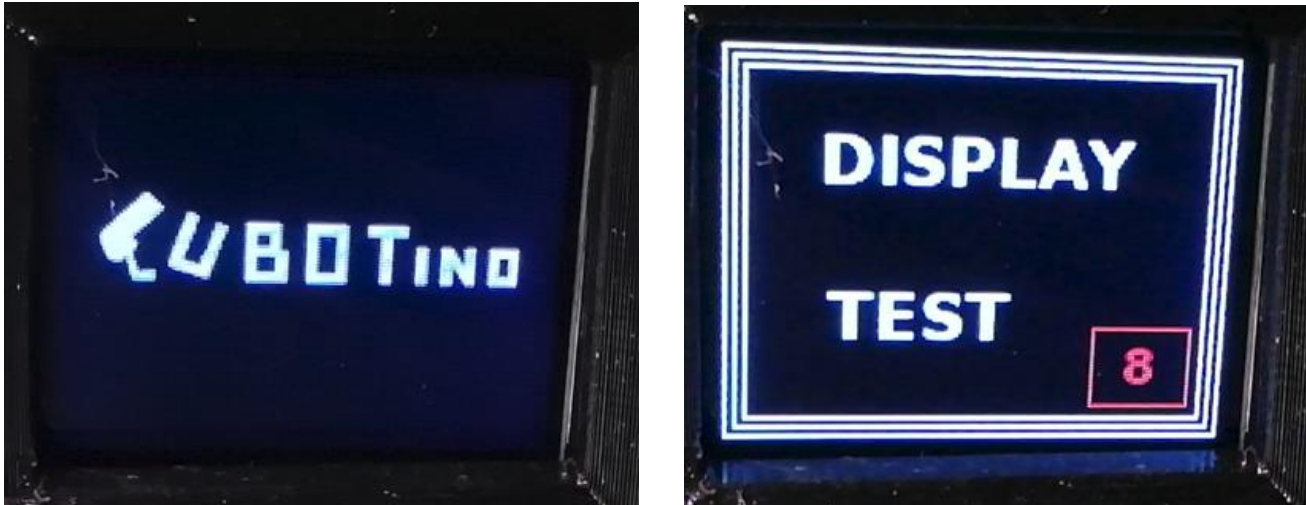
Display test for 20 seconds
Display shows rectangles, text and Cubotino logo
Display test1 finished

Display URF colours sequence of Western Rubik's
white
blue
yellow
green
red
orange
Display test2 finished
```

Section2: Initial preparation

This test is split in two parts, the screen “centering” and the colors:

For 20 seconds the display will alternate between the Cubotino logo and 3 rectangles that should be fully visible:

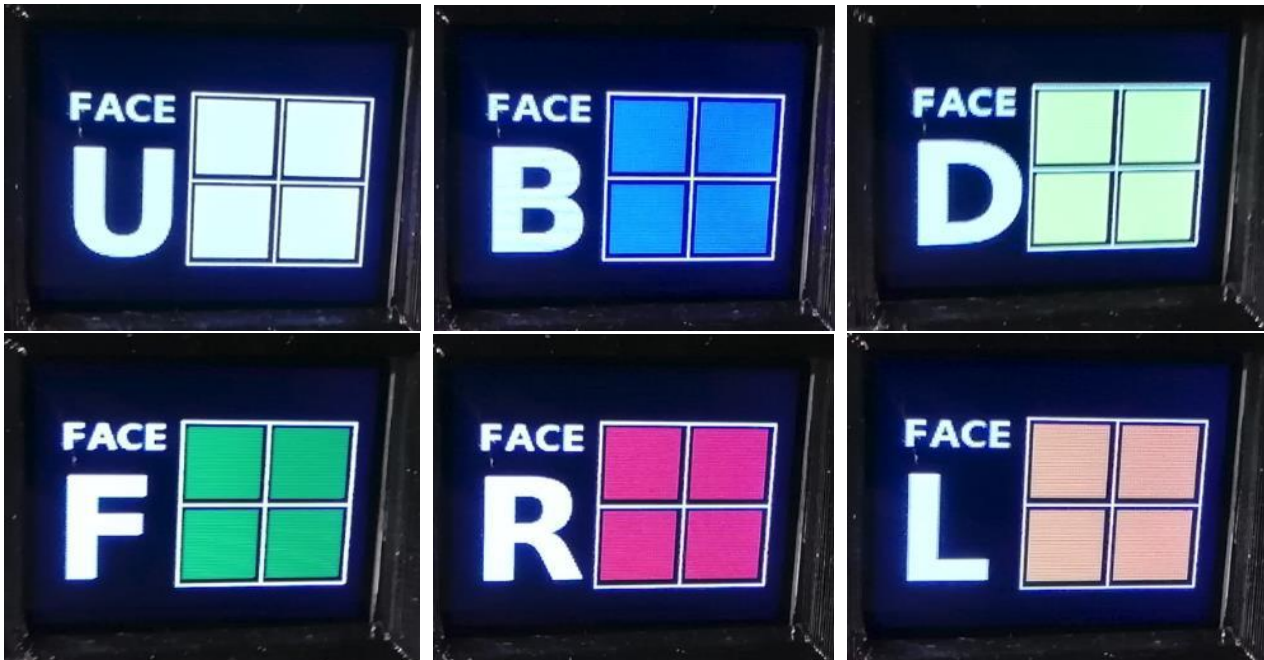


In case the display has a shift, part of those rectangles won't be visible.

In case the rectangles aren't complete, correct the X (`disp_offsetL`) and/or Y (`disp_offsetTdeac`) offset at `Cubotino_P_settings.txt`. See “Parameters” in “Appendix 1, Additional Information”.

The time count-down is plotted in the little red square.

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).



After the testing, type `sudo halt -p` to shut The Raspberry Pi off and wait 10 s before removing the power supply from it.

12. Servos check and set to neutral (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 has about 190° rotation, to be used for the cube holder.
- **Set both the servos in their neutral position, prior to the robot assembly.**

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

1. Connect a connections arm to the non-assembled servos.
2. Enter the cube folder: `cd cubotino_pocket/src`
3. Activate the virtual environment: `source .virtualenvs/bin/activate`
4. Set the servos to the mid position: `python Cubotino_P_servos.py --set 0` (pay attention to the double '-' without space in between, and the space before the zero).

```

pi@cubotino: ~/cubotino_pocket/src
File Edit Tabs Help
(.virtualenvs) pi@cubotino:~ $
(.virtualenvs) pi@cubotino:~ $ cd cubotino_pocket/src
(.virtualenvs) pi@cubotino:~/cubotino_pocket/src $ source .virtualenvs/bin/activate
(.virtualenvs) pi@cubotino:~/cubotino_pocket/src $ python Cubotino_P_servos.py --set 0

servos to: MID POSITION

enter a new PWM value from -1.00 to 1.00 (0 for mid position, any letter to escape): █

```

5. 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.

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.

6. Once verified the servos have 180° or larger rotation and decided which one has the larger range (if any), **set both the servos to their neutral (mid) position by entering 0.**

Type any letter to quit:

```

enter a new PWM value from -1.00 to 1.00 (0 for mid position, any letter to escape): q

Quitting Cubotino_P_servos.py

(.virtualenvs) pi@cubotino:~/cubotino_pocket/src $ █

```

Section2: Initial preparation

An alternative method is to use a servo tester. This little device, used for Radio Controlled models setup, works with all kinds of servos (analog, digital), checks the full rotation range, and sets the servos to neutral. Here is a picture of a sophisticated model (ca. 10 €) but base models can be found for less than 3€ (AliExpress).

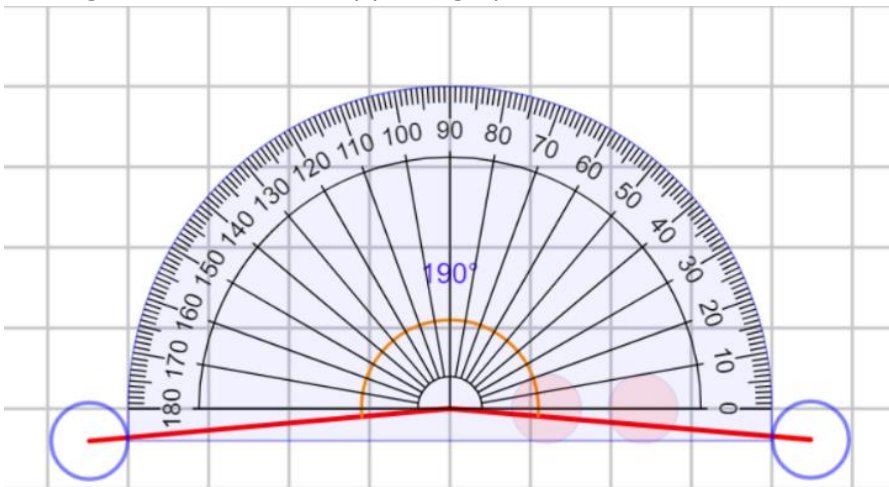


Position the servo arm as per below picture:



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!

13. Test the PiCamera

1. Connect via VNC
2. Enter the cube folder: `cd cubotino_pocket/src`
3. Activate the virtual environment: `source .virtualenvs/bin/activate`
4. Run the camera test: `python Cubotino_P.py --picamera_test`
5. 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 9cm distance, if not proceed with the following “Set the correct focus” instructions.

14. Set the correct focus for the PiCamera

The V1.3 PiCamera has fixed focus.

Because of the short distance between the PiCamera and the cube face, it's suggested to adjust the focus.

In some cases, the PiCamera will be able to detect the cube edges and color correctly, even without adjusting its focus, but the PiCamera images won't be very nice to be seen.

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:

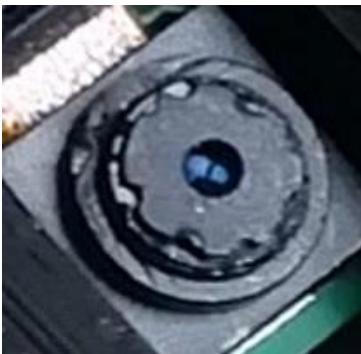
After 20 seconds the test ends, and the command can be repeated as long as needed.

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 a 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 reasonably 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.



15. 3D printed parts

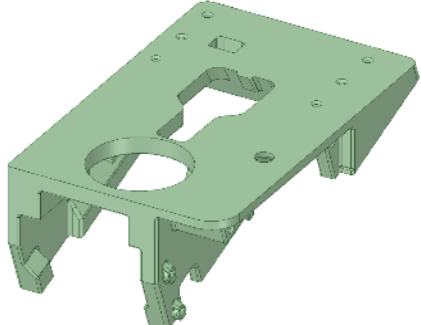
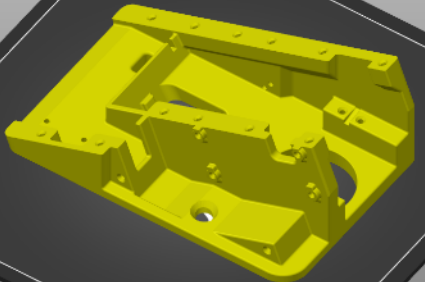
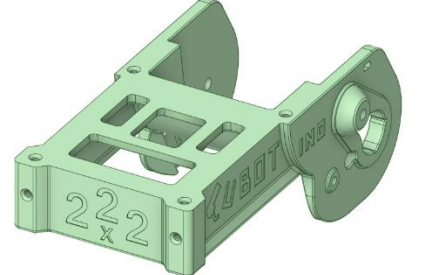
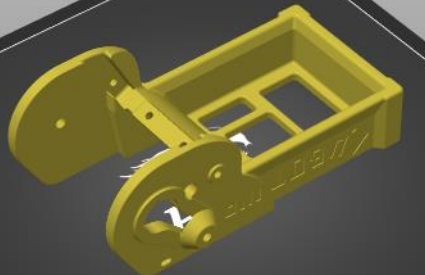
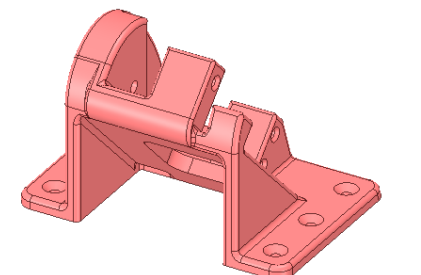

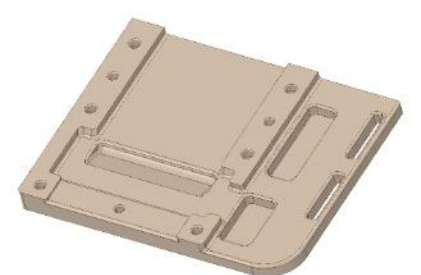
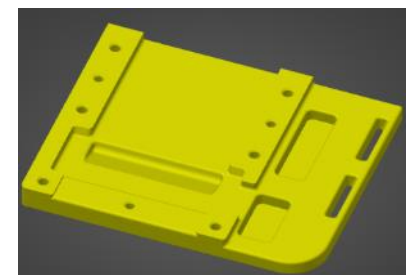
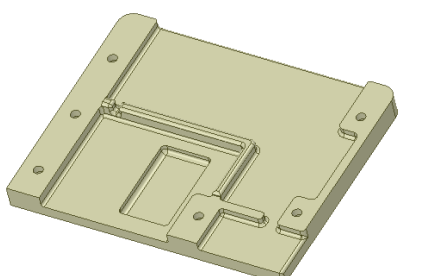
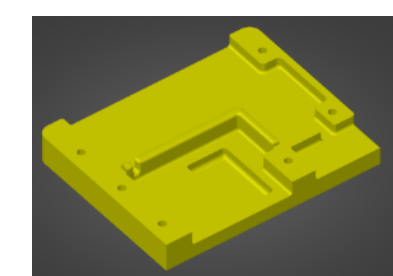
See notes below for filament quantity, and printing time ...

Ref	Part	Filament		Printing time	Version specific parts
		Meters	Grams		
1	Structure	46.5	142	14h30m	Pocket version
2	Top_cover	27.8	85	9h27m	Pocket version
3	Hinge	14.9	45.5	5h08m	Pocket version
4	Baseplate front	12.2	36	3h33m	
5	Baseplate back	12.6	37.3	3h36m	
6	Cube_holder	9.8	28.8	3h20m	Pocket version
7	Lifter	5.5	16.2	1h48m	
8	Servo_axis_sup (or its alternative)	2.3	7.1	0h54m	Pocket version
9	Servo_axis_inf (or its alternative)	1.7	5.2	0h40m	Pocket version
10	PCB_cover_display (or its alternative)	14	41.4	4h13m	
11	PiCamera holder	2.2	6.5	0h50m	
12	PiCamera holder frame	5.0	15.1	1h56m	Pocket version
13	Personaliz_plate (or its alternative)	1.5	5	0h30	
TOTAL		~ 160m	~ 500g	~ 50h	

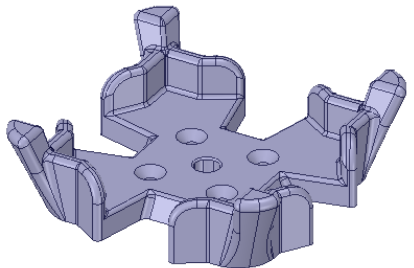
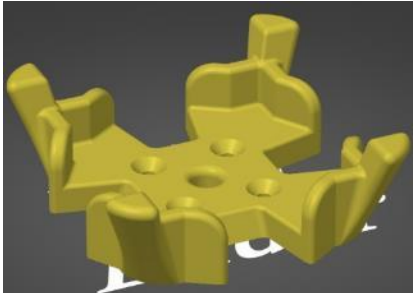
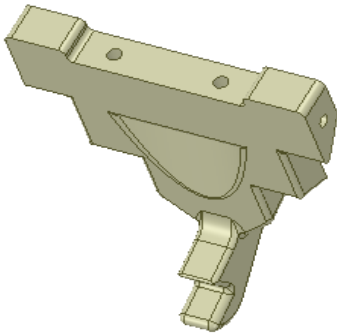

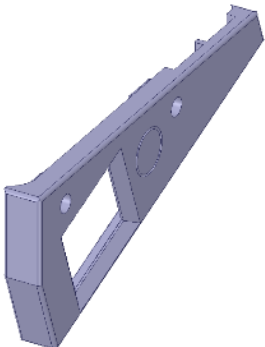
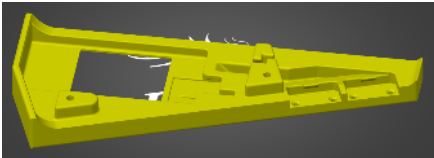
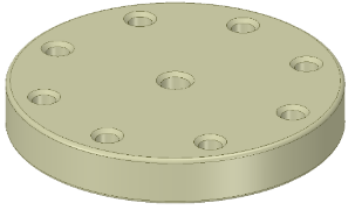
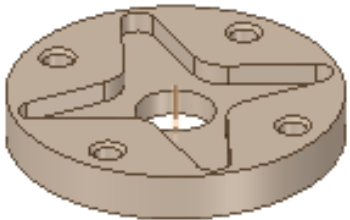
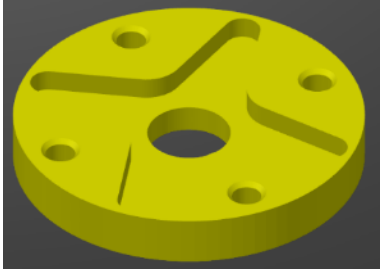
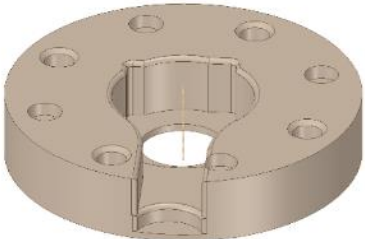

Notes:

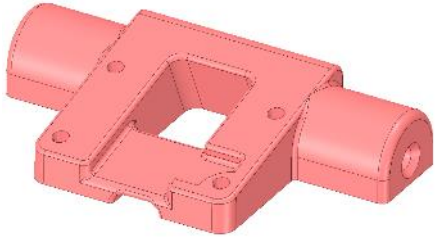
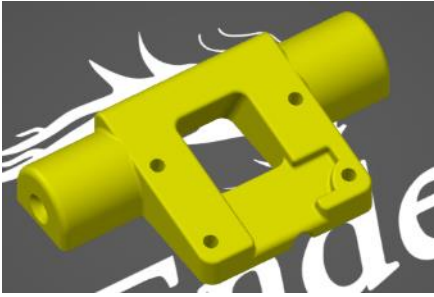
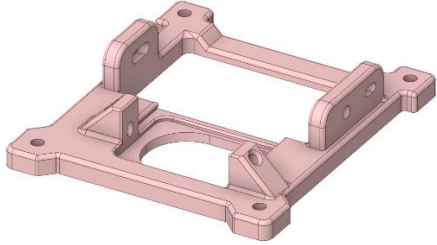



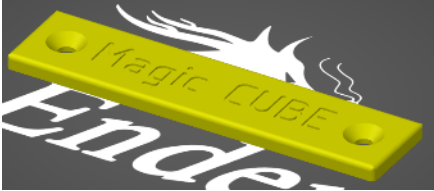
- Filament length is based on $\varnothing 1.75\text{mm}$.
- Filament weight is based on PETG density (1.23g/mm^3), and printing settings I've use on my Ender 3 printer, for accurate result:
 - 0.2mm layers
 - Low speed (between 25 to 40mm/s for the external parts and 1st layer)
 - 4 layers on vertical walls
 - 5 layers on horizontal walls
 - 30% filling
 - 8mm brim
 - 5 mm retract length
- All parts have been designed to be **printed without supporting** the overhangs (**no support needed**).
- Some parts have been split, for easier and better 3D printing.
- The suggested part orientation for the 3D print is shown on below Table.
- The **stl** files are available at: https://github.com/AndreaFavero71/cubotino_pocket/tree/main/stl STL files get downloaded to the Raspberry Pi when cloning the repository.
- The **step** files are only available at https://github.com/AndreaFavero71/cubotino_pocket/tree/main/stp Step files do not get downloaded to the Raspberry Pi when cloning the repository.

Section3: 3D print and assembly

Part name	image	3D print orientation
Structure_Pocket		
Top_cover_Pocket		
Hinge_Pocket		
Baseplate_front		
Baseplate_rear		

Section3: 3D print and assembly

<p>Cube_holder_Pocket</p>		
<p>Lifter</p>		
<p>PCB_cover_display (or its alternative)</p>		
<p>Servo_axis_sup_Poc ket (or its alternative)</p>		<p>Symmetrical</p>
<p>Servo_axis_inf_Poc ket</p>		
<p>Alternative _Servo_axis_inf_Poc ket</p>		

<p>PiCamera_holder</p>		
<p>PiCamera_holder_frame_Pocket</p>		
<p>Personaliz_plate Or Personaliz_plate01</p>	 	

16. Assembly steps

Before assembling the robot:

1. Make the Connections_board.
2. Setup the Raspberry Pi.
3. Position the two servos output gear to their middle position (see Servos test and set to mid position chapter)

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



List of the tasks to be performed to assemble the robot:

1. Screw the bottom servo to the structure.
2. Prepare the sandwich Servo_axis_inf / servo lever / Servo_axis_inf.
3. Assemble the Cube_holder to the Servo_axis assembly.
4. Assemble the Cube_holder assembly to the bottom servo.
5. Assemble the Hinge to the Structure.
6. Assemble the "T25" servo arm to the upper servo, only after knowing the servo is on its middle position.
7. Insert the Top_servo assembly into the Top_cover slot.
8. Assemble the Top_cover assembly to the Hinge.
9. Complete the top servo assembly to the Hinge.
10. Assemble the Lifter to the Top_cover.
11. Assemble the PiCamera holder frame to the Top_cover.
12. Assemble the LED breakout board to the Top_cover.
13. Position the cables and assemble the Baseplate_rear.
14. Connect the PiCamera flex cable to Raspberry Pi Zero2 board and fix it to the Structure.
15. Fix the Touch_sensor to the PCB cover.
16. Assemble the USb breakout board to the PCB_cover.
17. Connect the servos, LED breakout board and Touch sensor.
18. dress the cable and connect the display.
19. Assemble the PCB_cover.
20. Assemble the Baseplate_front to the Structure.
21. Stick the PiCamera to its board, via the self-adhesive tape underneath the camera.
22. Assemble the PiCamera module to the Picamera_holder.
23. Assemble the PiCamera holder to the PiCamera holder frame.
24. Connect the flex cable to the PiCamera module.
25. Personalize the formal Stop plate, transformed to a personalization plate in this robot version.

Each of these steps are described in detail in the following pages.

17. Assembly details

Info: All the assembly steps are as per CUBOTino Autonomous

Step1 (Mount the bottom servo to the structure):

4x M3x10mm (or M3x12mm) cylindrical head

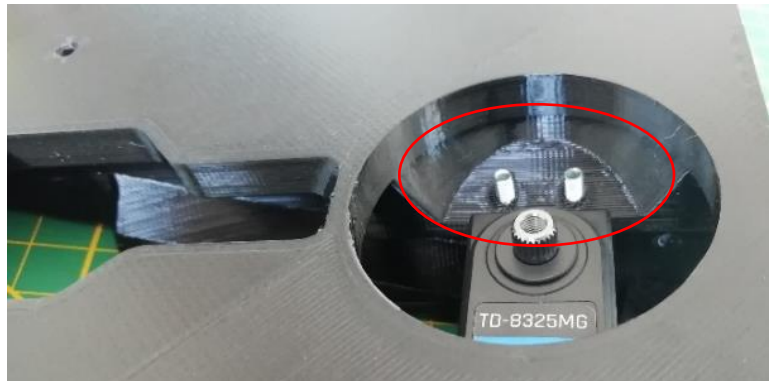
Couple of washers

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.



Step2 (sandwich Servo_axis_inf / servo arm / Servo_axis_inf):

4x M3x12mm conical head

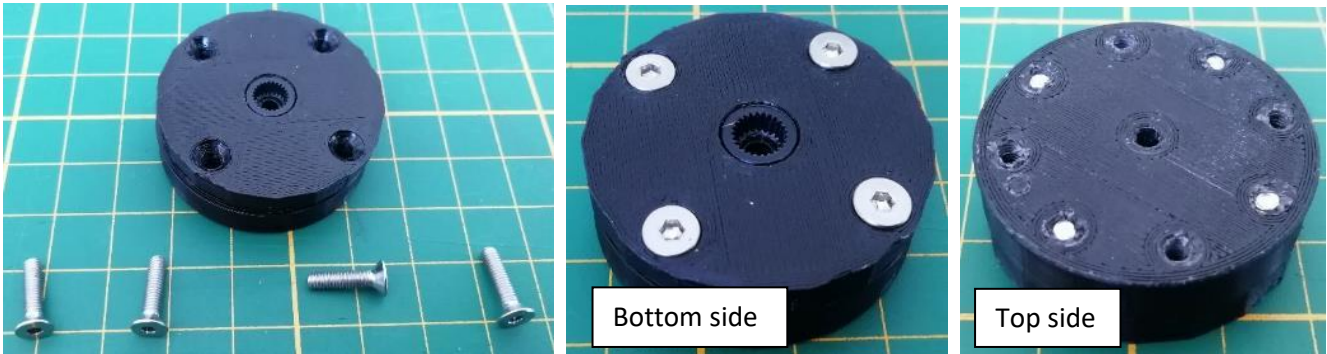


Check if the supplied X arm fits with the indentation of Servo_axis_inf part:



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



Step2a 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



Step3 (Assemble the Cube_holder to Servo_axis assembly):

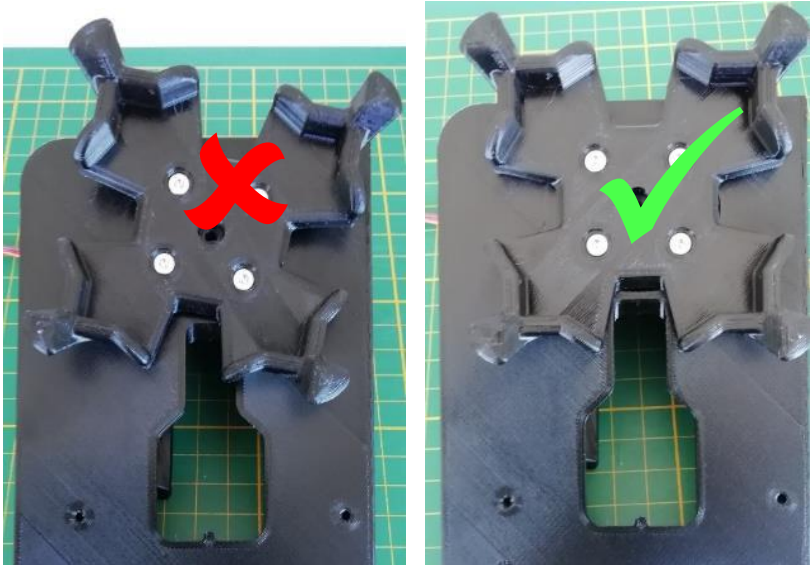
4x M3x12mm conical head



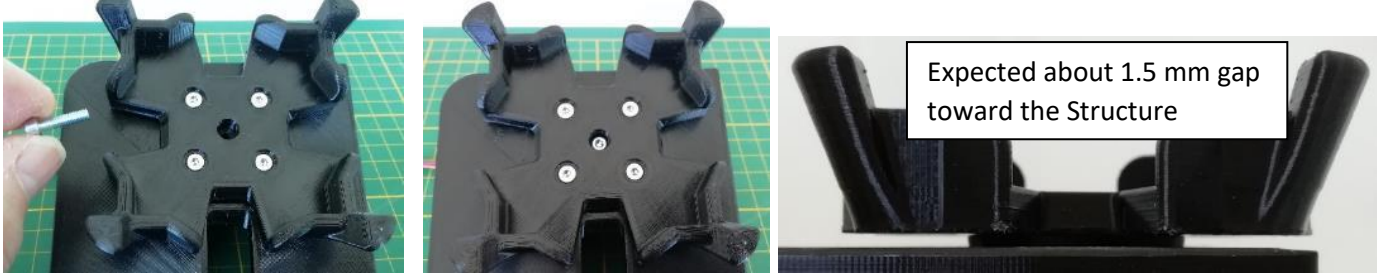
Step4 (Assemble the Cube_holder assembly to the bottom servo):

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 orientation with good coupling.

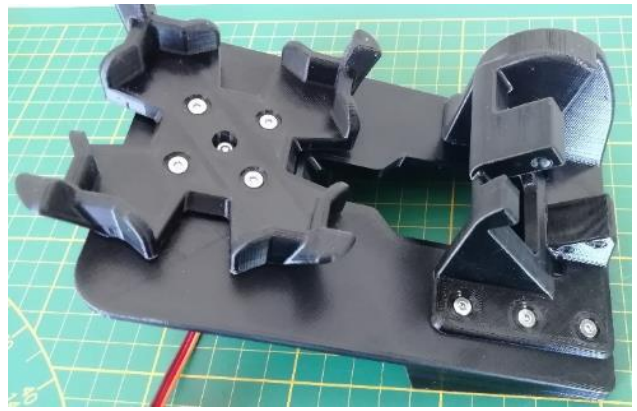
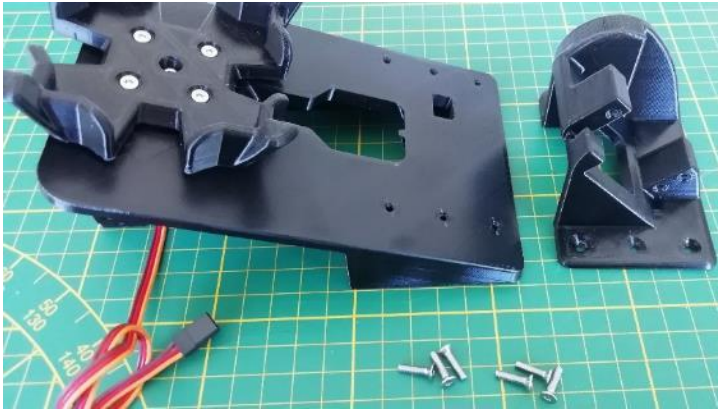


1x M3x12mm cylindrical head

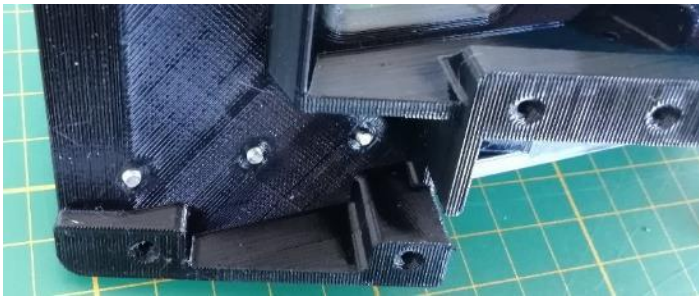


Step5 (Assemble the Hinge to the Structure):

6x M3x12mm conical head



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



Step6 (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

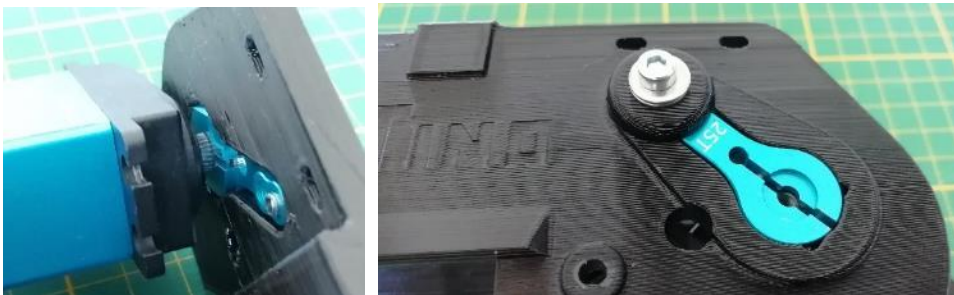
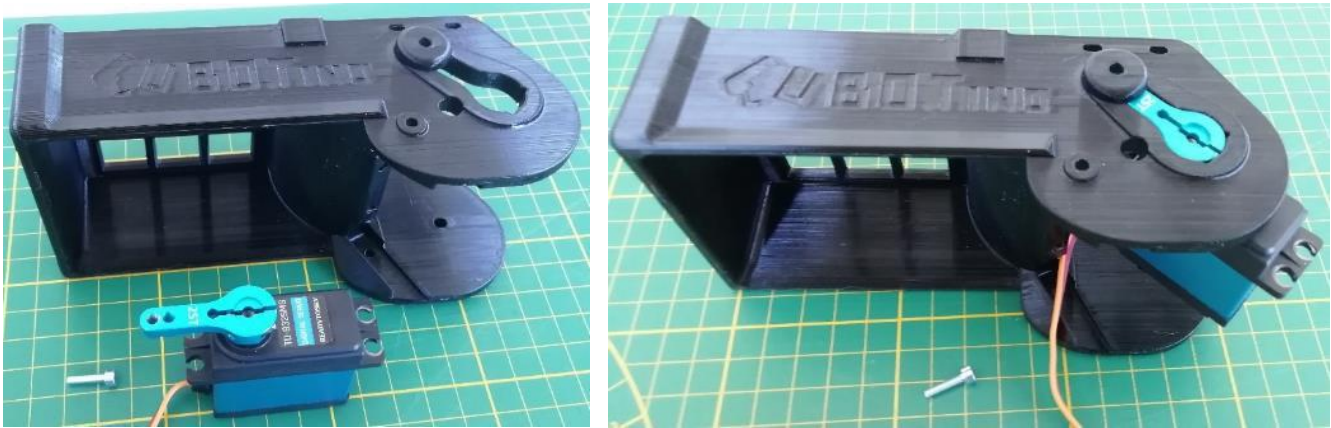


Step7 (Insert the Top_servo assembly into the Top_cover slot):

1x M3x12mm cylindrical head

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

Ensure the hole for the M4 screw doesn't not constrain the screw (it should have $\varnothing 4.1$ 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

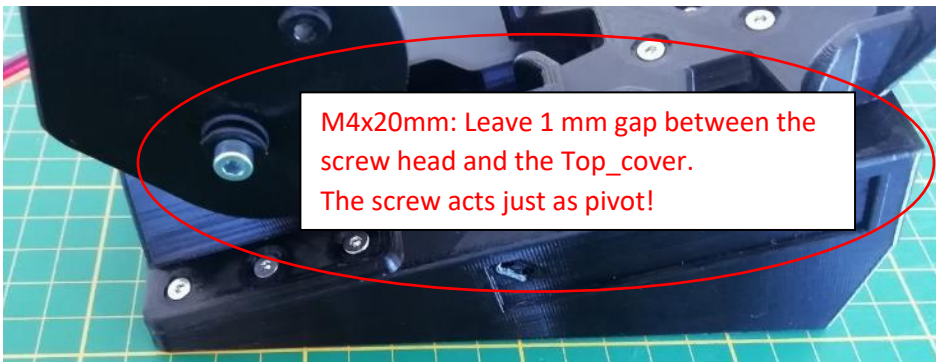


Step8 (Assemble the Top_cover assembly to the Hinge):

1x M4x20mm cylindrical head

3x M3x12mm cylindrical head

Ensure the hole for the M4 screw doesn't not constrain the screw (it should have $\varnothing 4.1$ to $\varnothing 4.3$ mm)

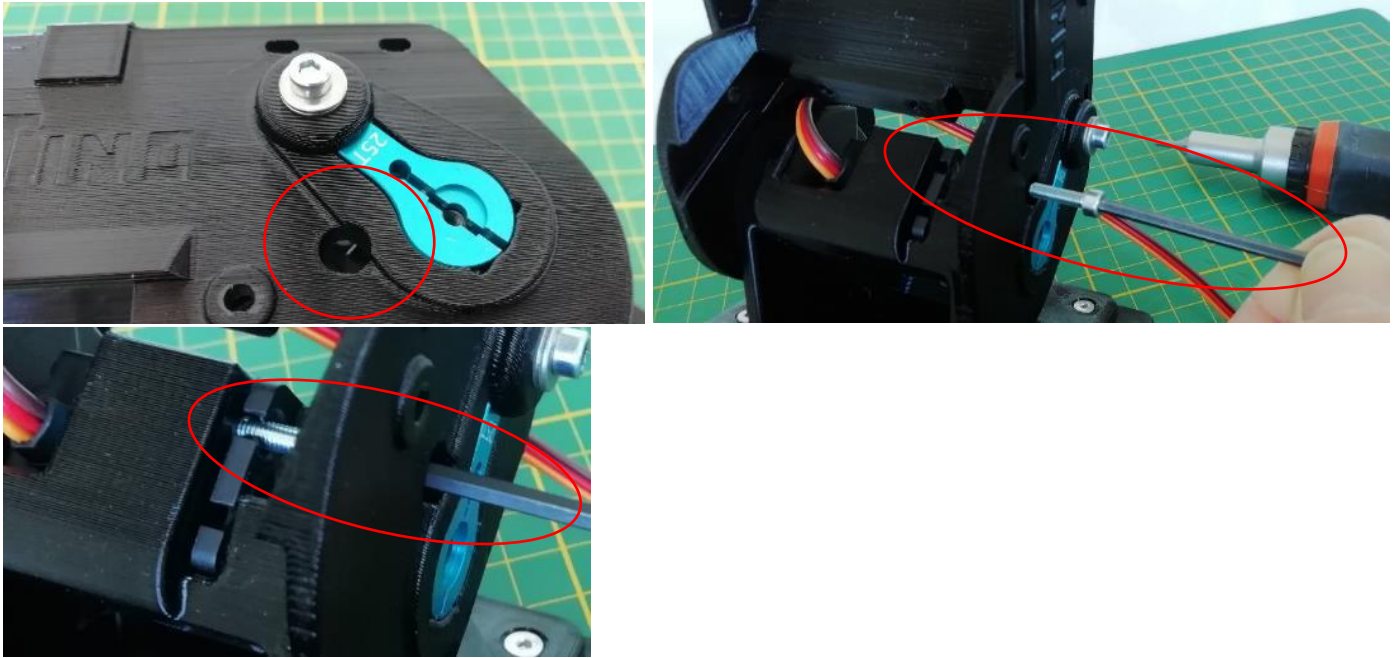


Do not fully tighten the two M3x12mm, until also the third screw is positioned

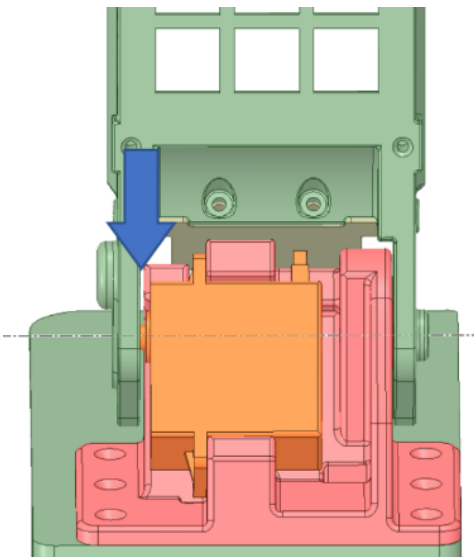
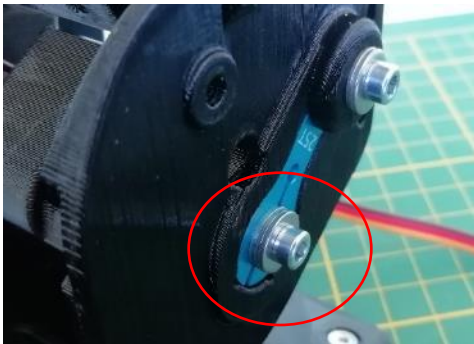


Step9 (Complete the top servo assembly to the Hinge):

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



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



Verify 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.

Step10 (Assemble the Lifter to the Top_cover):

4x M3x12mm cylindrical head

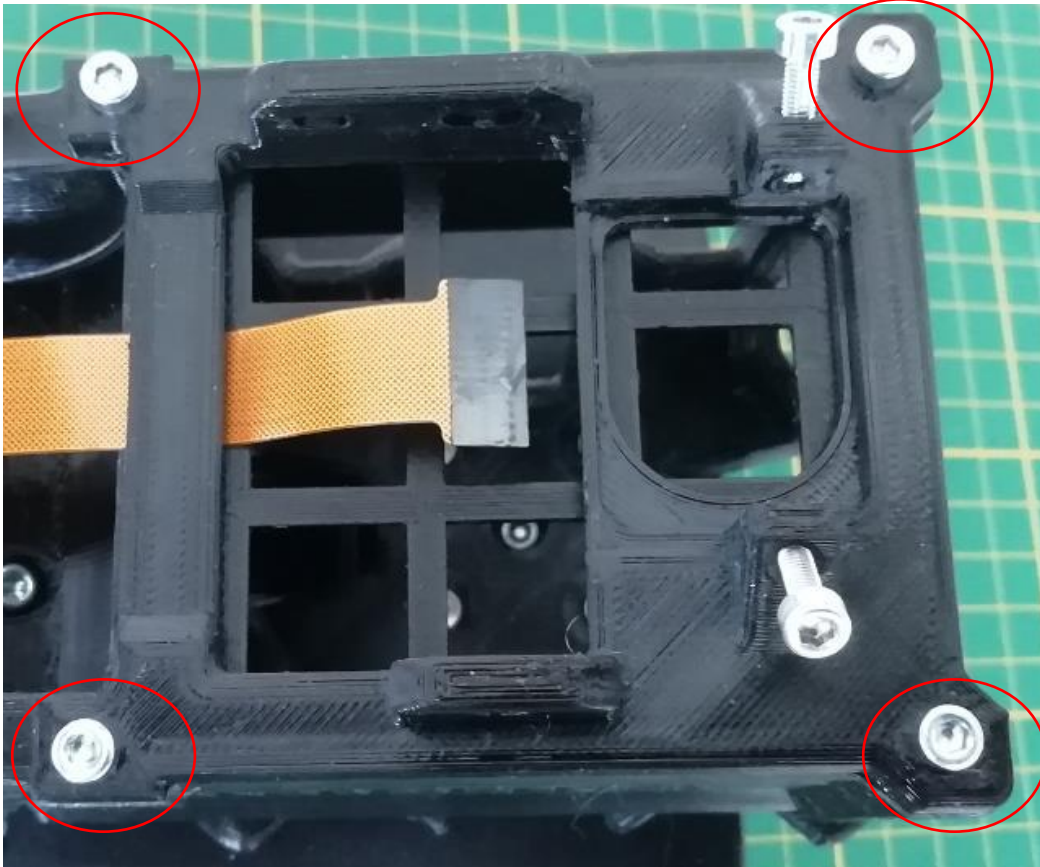
Slide the Lifter into the Top_cover slots.



Step11 (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

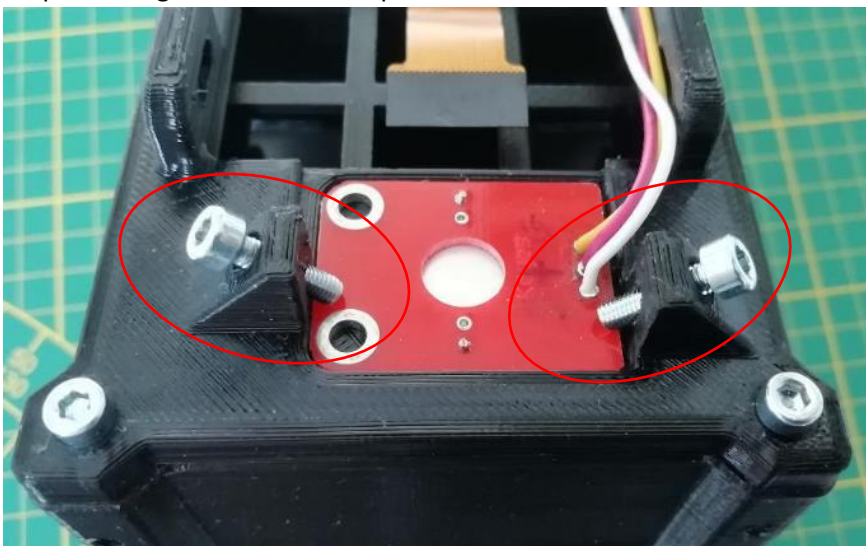


Step12 (Assemble the LED breakout board):

Add 2x M3x12mm cylindrical head

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

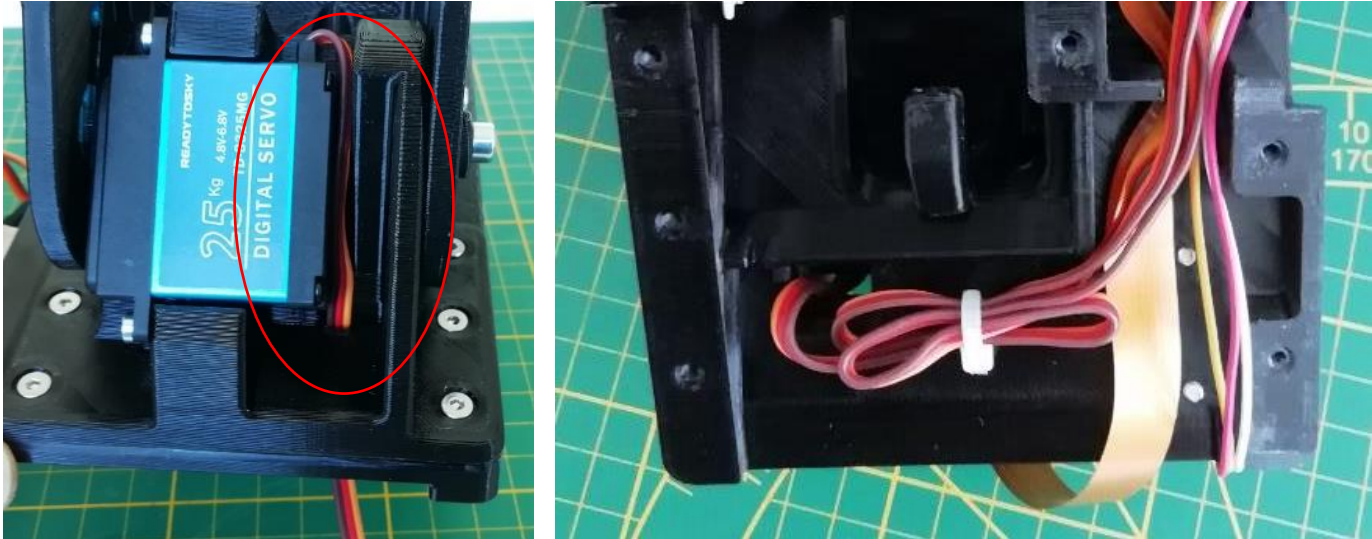
Stop screwing once the screw tip touches the board



Step13 (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 of cable length as per below picture, as there is very little space left at the board location

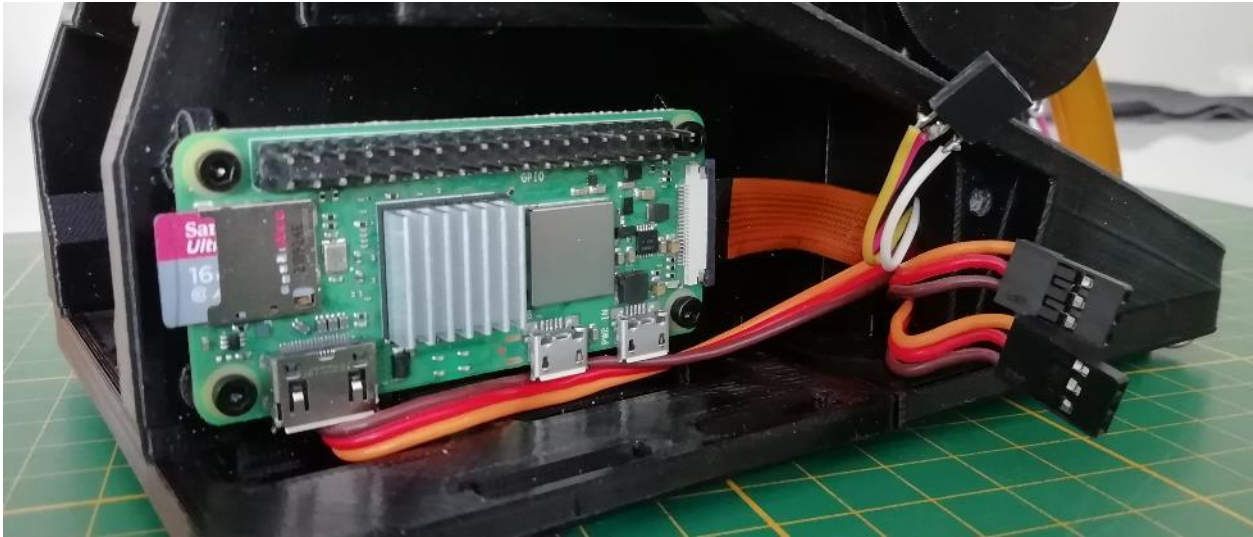
Dress the cables and close the Baseplate_rear.



Section3: 3D print and assembly

Step14 (Connect the PiCamera flex cable to Raspberry Pi Zero2 board, and fix it to the Structure)

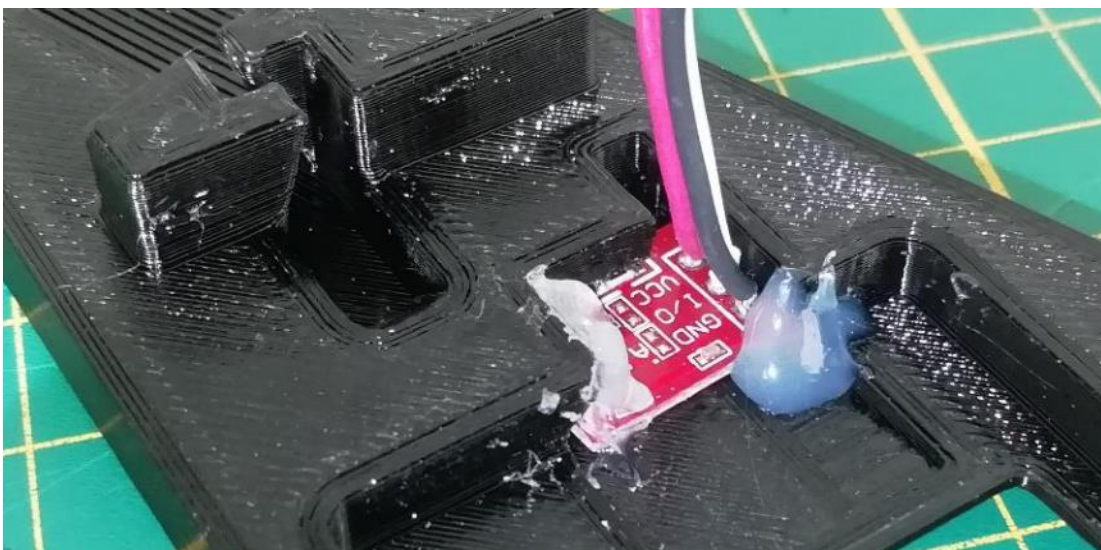
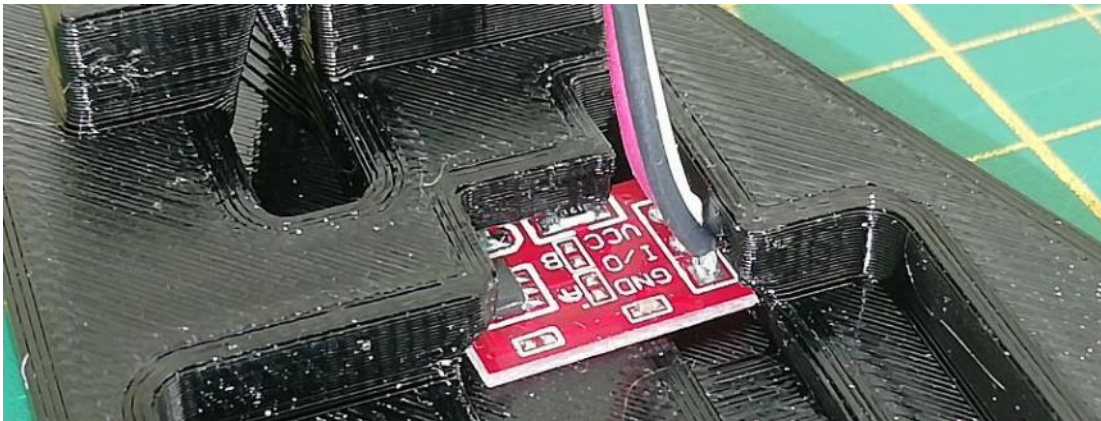
4x M2.5x4mm (or to M2.5x max 12mm) cylindrical head



Step15 (Fix the Touch_sensor to the PCB cover):

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

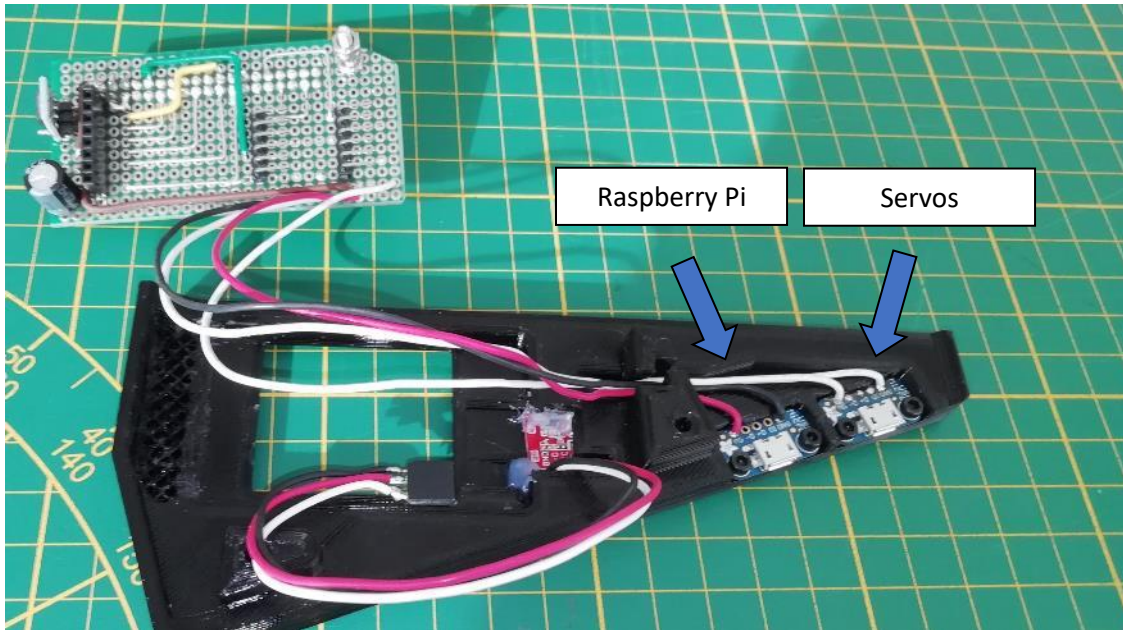
If the board centre part doesn't touch the PCB_cover, reduce the wire protrusion underneath the board.



Step16 (Fix the microUSB breakout board to the PCB_cover):

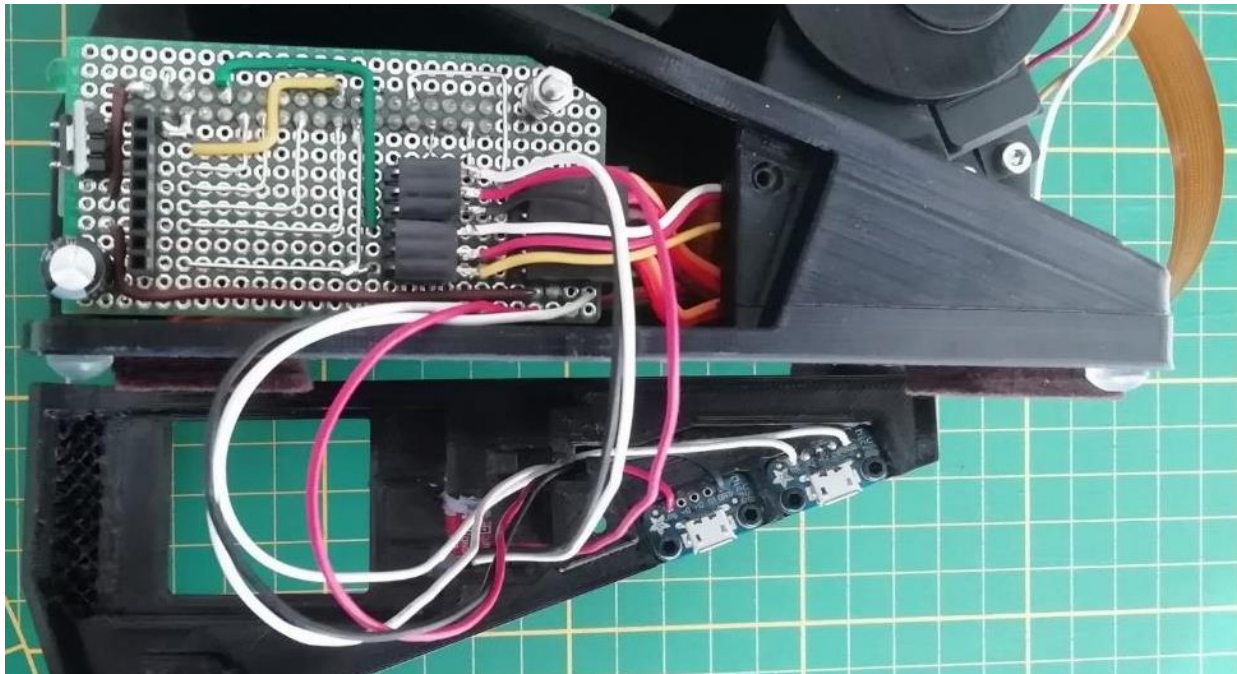
2x M2.5x4mm cylindrical head

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



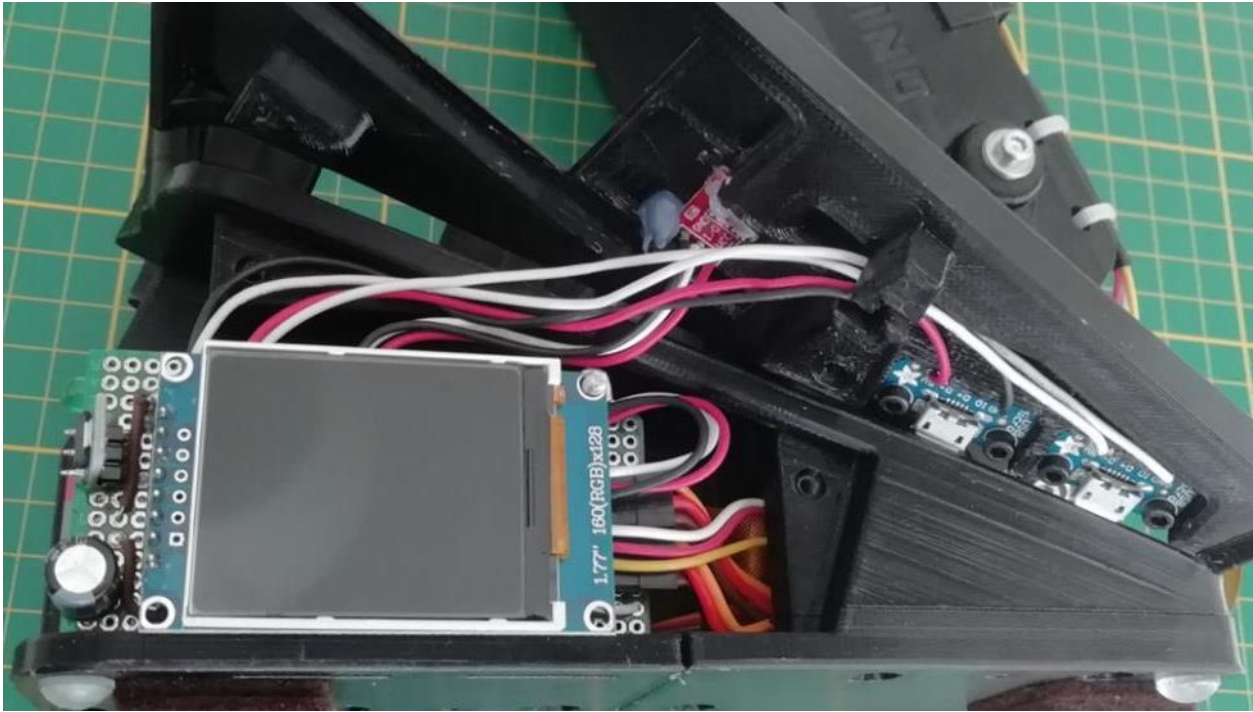
Step17 (Connect the servos, LED breakout board and Touch sensor):

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



Step18 (dress the cable and connect the display):

Cables at the microUSB breakout boards have to be well positioned into the groove



Step19 (Assemble the PCB_cover):

2x M3x12mm conical head

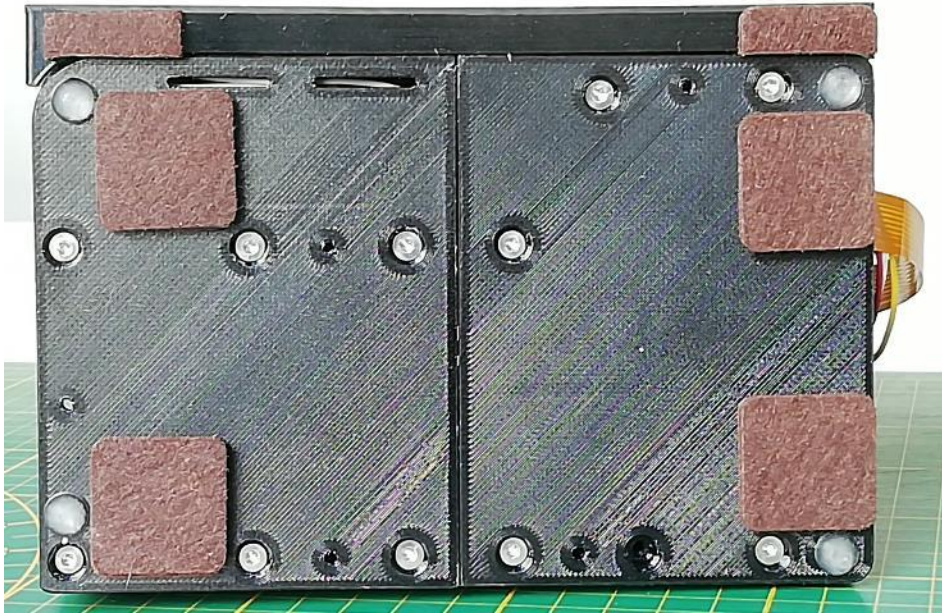
Attention to don't squeeze cables against the Structure



Step20 (Assemble the Baseplate_front to the Structure):

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

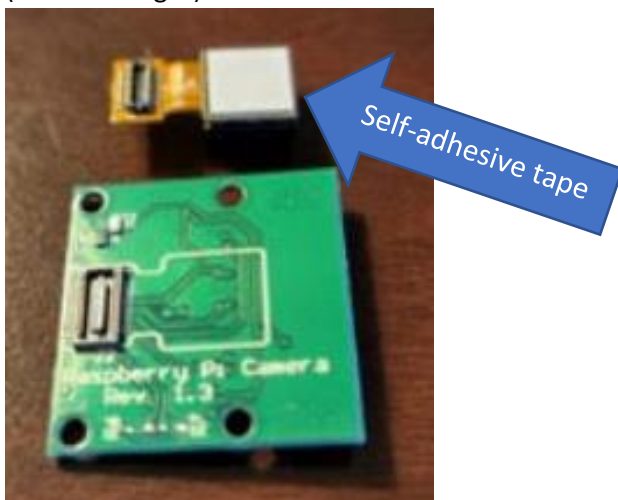
Stick self-adhesive rubber feet (or felt pads) to the baseplate; Those at the back should be placed as close as possible to the corners



Step21 (Stick the PiCamera to the board):

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

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



Step22 (Assemble the PiCamera module to the Picamera_holder):

Necessary some little pieces of filament $\varnothing 1.75\text{mm}$.

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



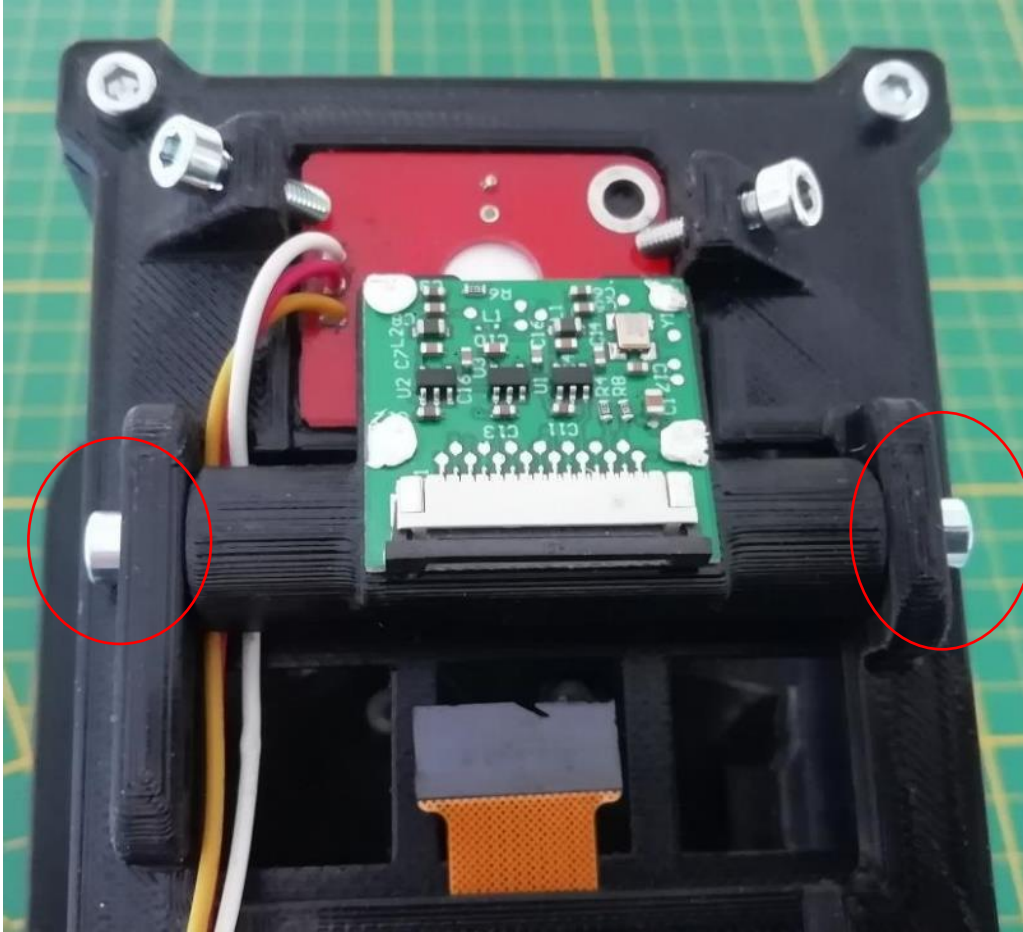
Alternatively, 4x M2x3mm screws can be used or 4x M2.5x4mm screws can be used.

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

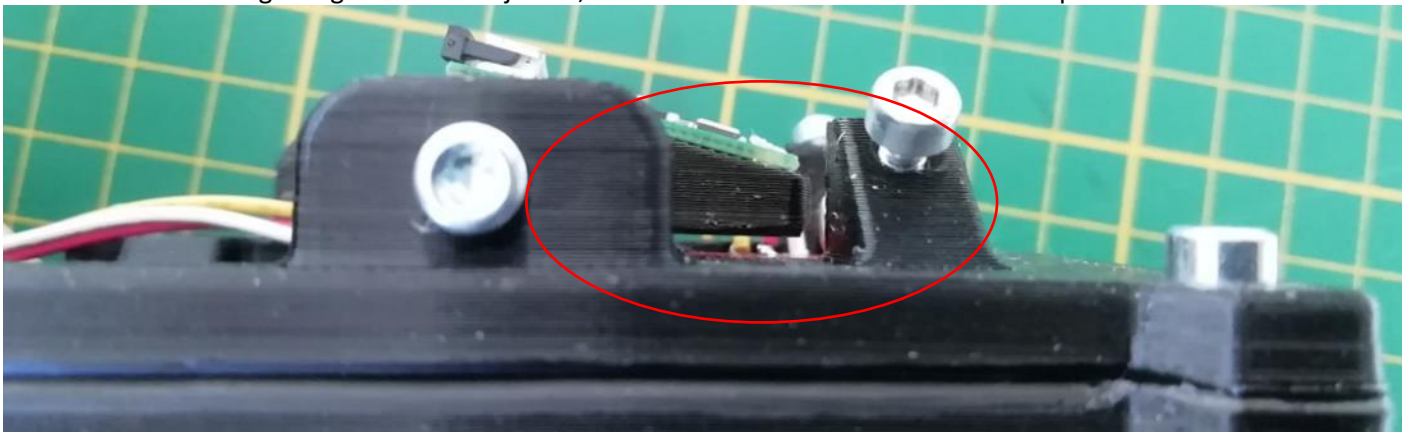


Step23 (Assemble the Picamera_holder to the Picamera_holder_frame):

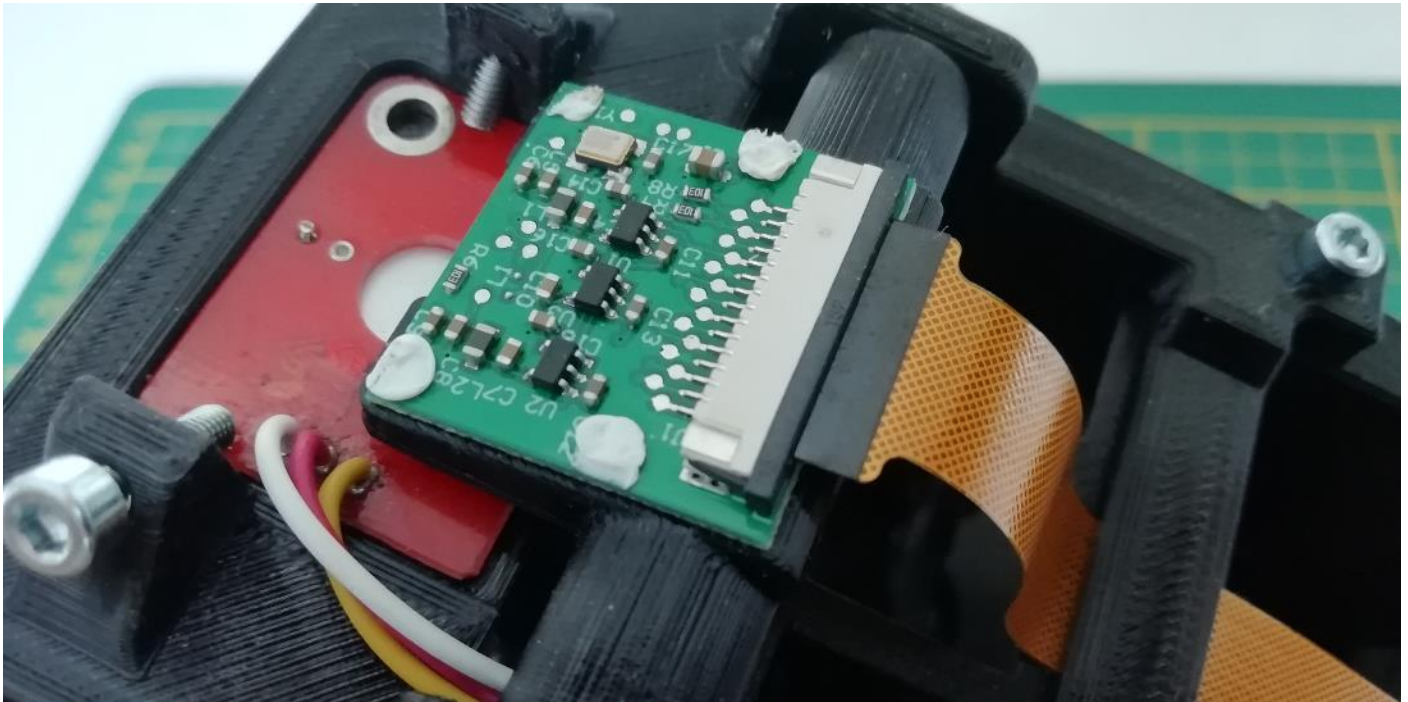
2x M3x12mm cylindrical head



Keep the PiCamera holder parallel to the Top_cover, while tightening the screws:
Be considered this angle might be later adjusted, to orient the camera to the the cube top face



Step24 (Connect the flex cable to the PiCamera module):



Step25 (Personalize the formal Stop plate, transformed to a personalization plate to this version):

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 😊.

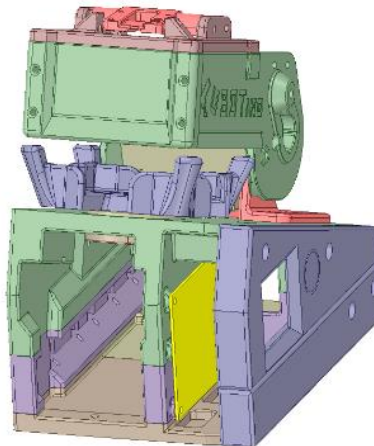
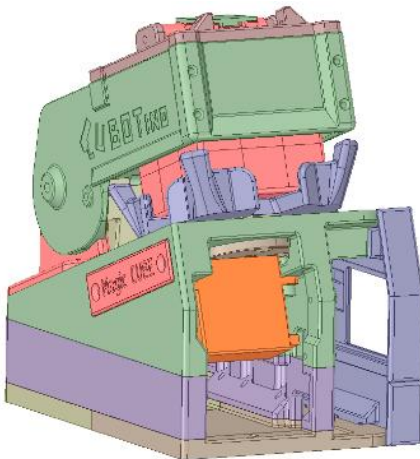
18. Raspberry Pi 2, 3 or 4

CUBOTino platform has been designed in late winter 2021, by considering the Raspberry Pi Zero2 an easy to source and relatively cheap (ca 15\$) component; Unfortunately, the market situation has changed and the Raspberry Pi Zero2 has suffered for quite a shortage and price increment.

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:

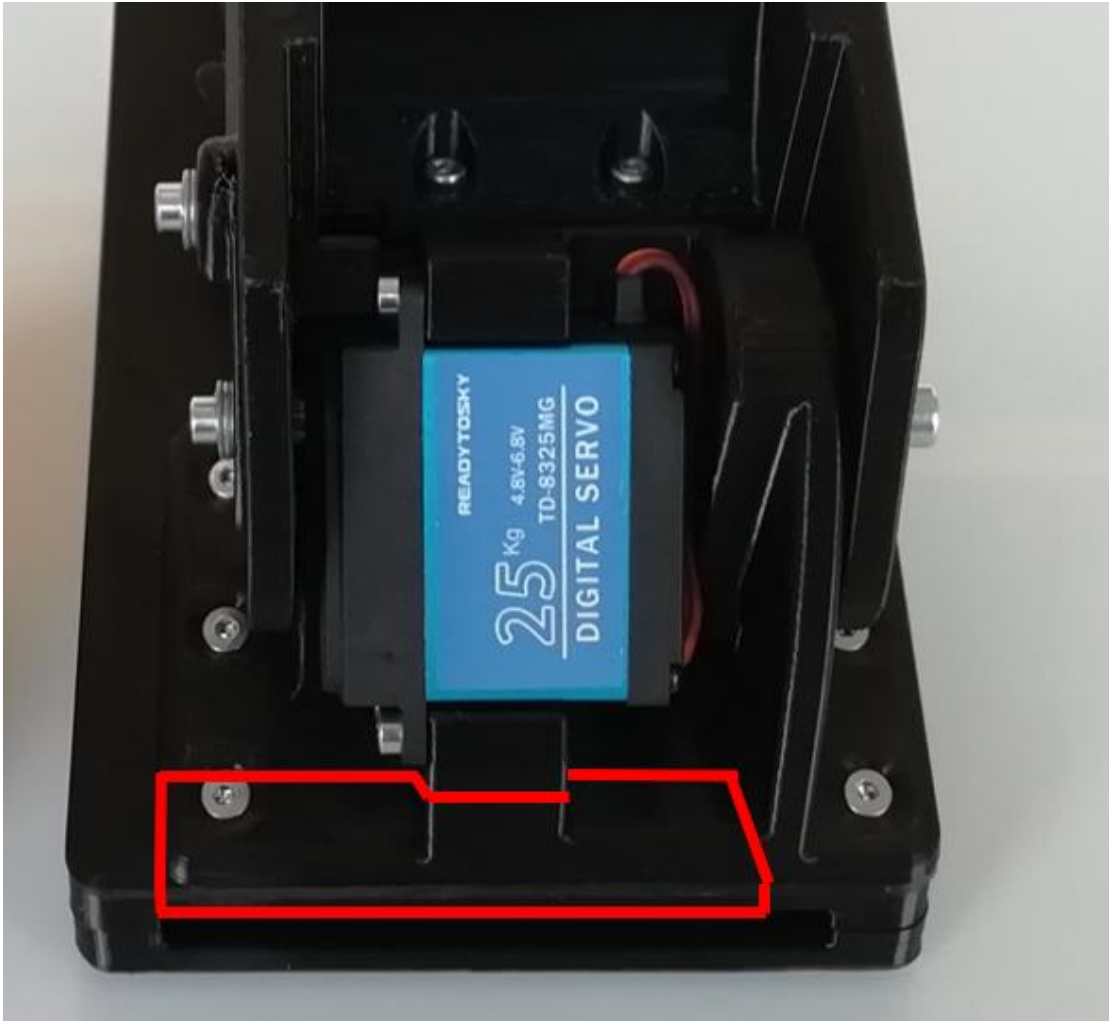
- The Connections_board need to be slightly smaller, to prevent interference with some of the raspberry Pi 3 or 4 connectors.
- Thicker wires are needed for these SBCs, especially for Raspberry Pi 4b; See Connections_board chapter.
- A 3A power supply is needed (preferably the Raspberry Pi original one 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


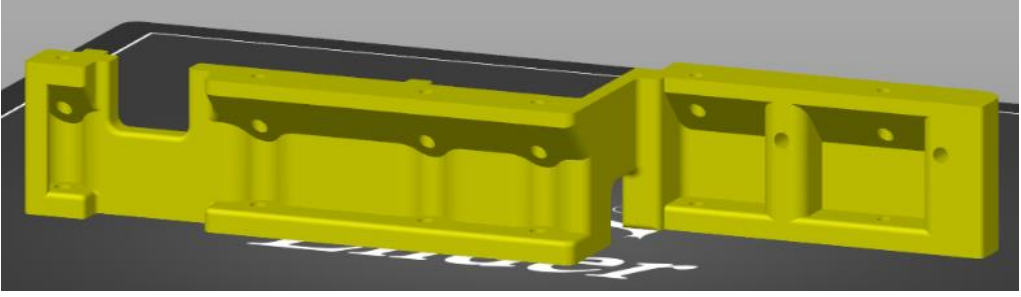
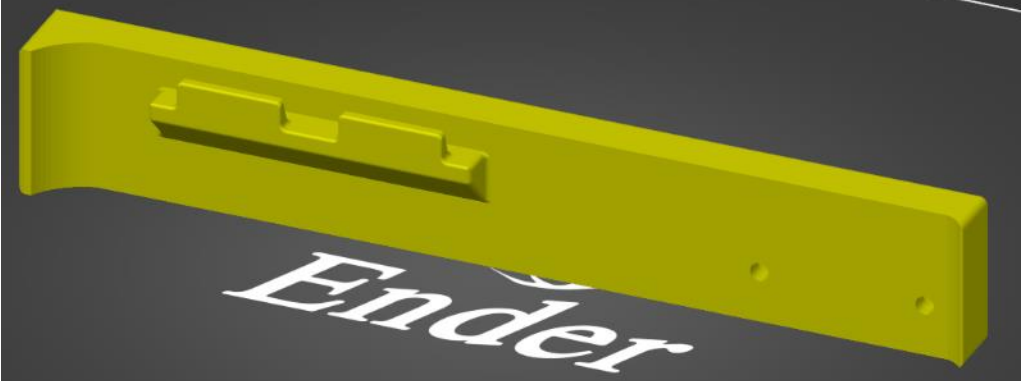
Stl files for the Extensions are (back) available since 22/10/2022 at Instructables and GitHub:

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:

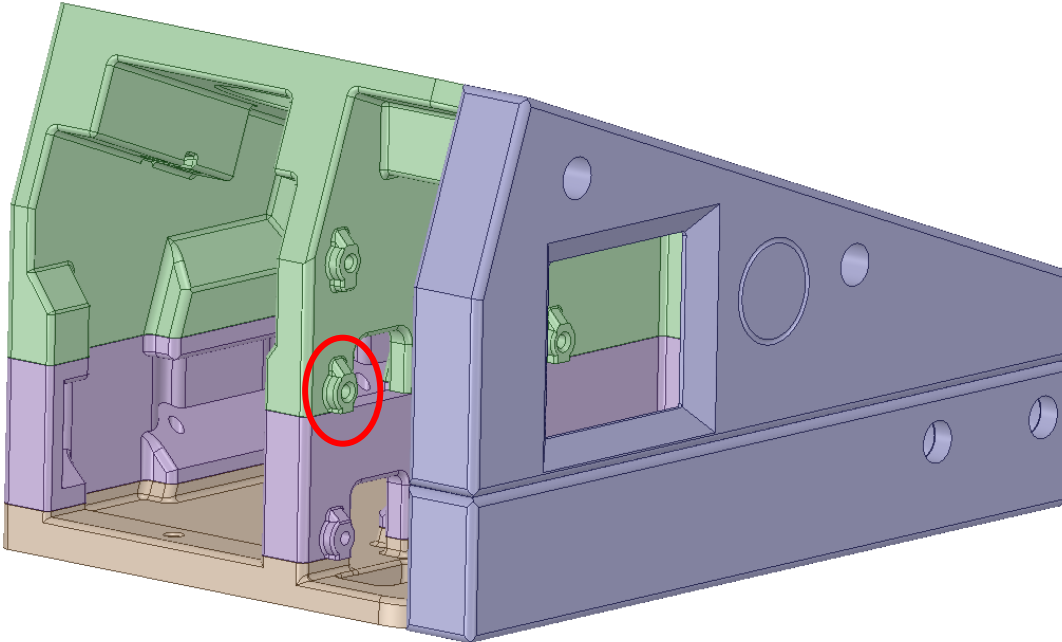
1. Still valid the notes on chapter 3D printed parts (i.e.,no support needed).
2. Above Filament consumption and printing time are based on a quality printing setting; If these parts are really meant to be temporary in your case, a much lower quality could be considered.
3. The suggested part orientation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

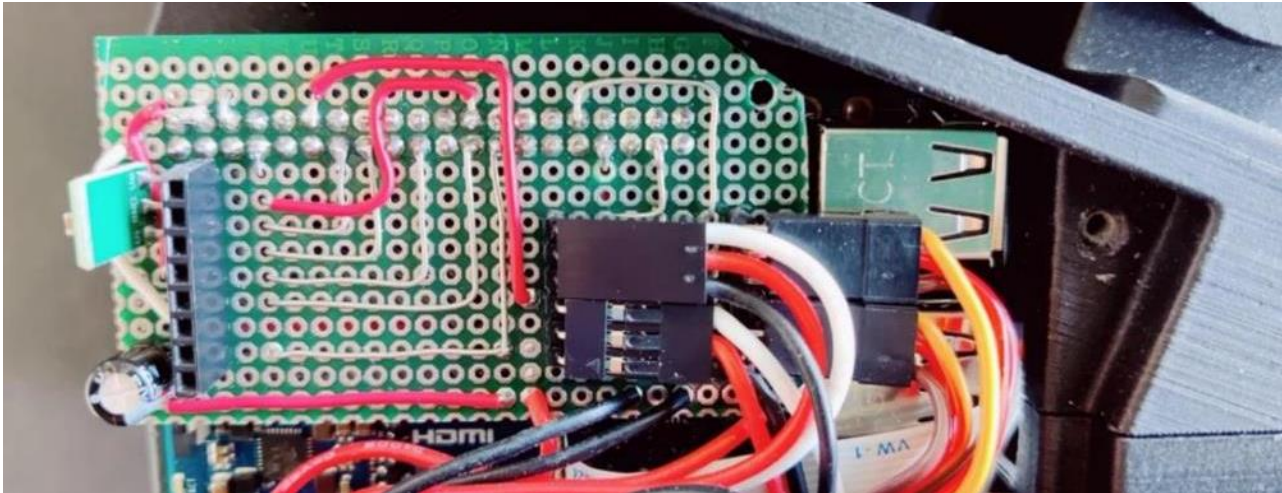
- The three additional parts are connected by screws.
- Use M3x12mm cylindrical, to connect the Extension_left and the Extension_middle toward the Structure.
Note: In case you don't have Allen keys with "spherical" head, to reach the screw under an angle, it might be necessary to use screws with conical 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 Baseplate_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 has to be assembled before assembling the Baseplate_front and before assembling the PCB_cover_display.

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

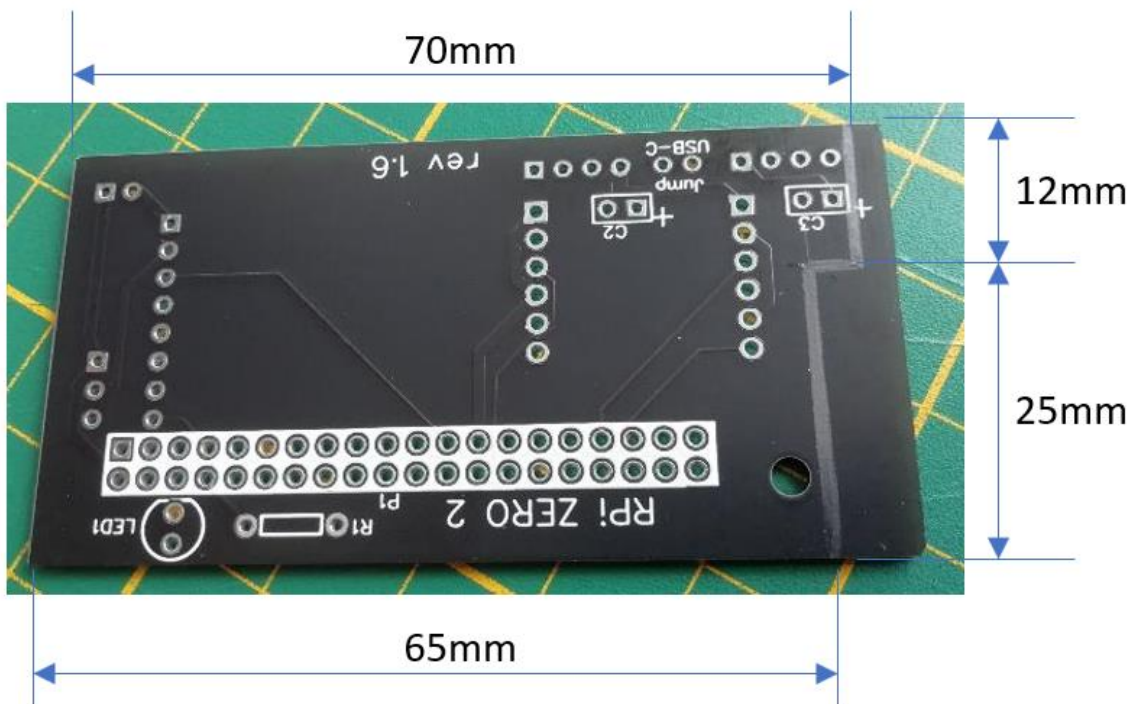
If you make the Connections_board, out of a proto-board, the important changes are:

- 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 **nrdlrd**, 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:



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.

19. Tuning

As anticipated, the robot won't magically work right after assembly: **Tuning is needed!**

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

- Servos.
- Arm position relative to the servo.
- Cube dimensions.
- Print quality.
- Assembly.

By considering the Cubotino Autonomous version has been made by many other Makers, I'm very confident you will successfully tune your own Cubotino Micro too 😊

1. General:

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

These parameters are grouped into two (json) text files: See Parameters and settings chapter in Appendix 1.

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 suggested at supplies are rated for 180° of rotation, that can be extended to ~200°: This is sufficient for this robot.

I don't suggest buying 270° servo as this will affect the angle resolution, and likely the torque too.

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 (20, 21 or 25 teeth, meaning many coupling possibilities). This means the reference angles set on Cubotino_P_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!**

The rotation angle of the servos is controlled via a PWM signal (https://en.wikipedia.org/wiki/Servo_control)

The servos at supplies list, like most servos used for radio-controlled models, for example, are rated for a Pulse Width signal from 0.5ms to 2.5ms, wherein 1.5ms is the mid angle, corresponding to the neutral (zero) position of the servo.

The Cubotino_P_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: In essence, this library normalizes the rotation range from -1 to +1.

It's possible to slightly extend the range, by extending the Pulse Width range: In my case from ca 0.3 to 2.7 (+20% on the Pulse Width range, meaning 20% gain in the rotation range 😊).

Detailed info on servo management:

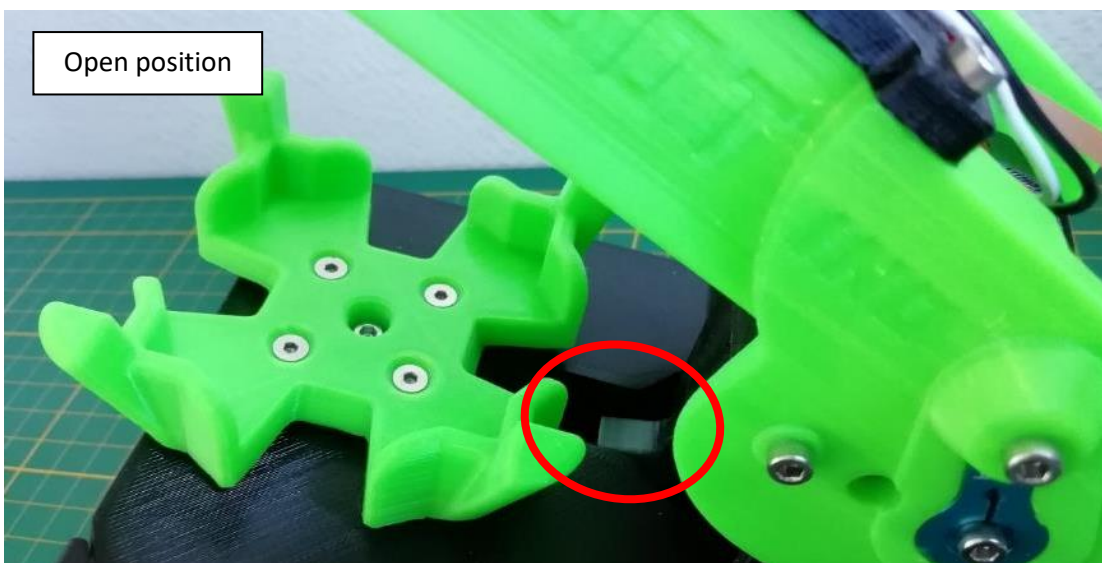
1. On Parameters and settings chapter are listed the involved variables and the default values.
2. The gpiozero library is used to control the servos, with a (float) parameter ranging from -1 to 1.
3. The float value entered on the '--set' argument (see Servos test and set to mid position chapter), represents a normalized rotation.
4. Value -1.0 is the max CCW servo rotation; The library sends the minimum Pulse Width to the servo.
5. Value 1.0 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 results to a CW rotation of the servo outlet.
8. On servos with a Pulse Width ranging from 0.5 to 2.5ms, every 0.02 step in the "--set" argument determines a rotation of 1.8 degrees: a variation of -0.02 rotates the servo outlet of 1.8deg CCW, while a variation of +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 constraints on the servo rotation.
10. In case your servos don't make 180 degrees rotation, when the '--set' parameter is changed from -1.0 to 1.0: 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).

Top_cover (t_servo) angles:

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

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

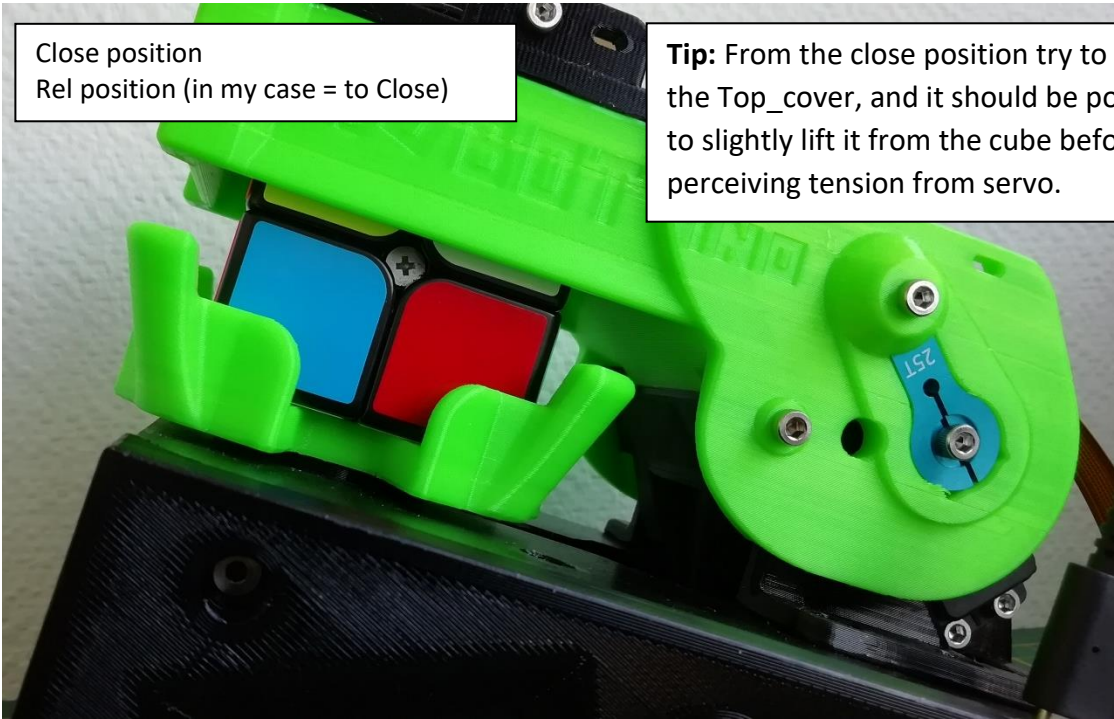
- a. Close: position to constrain the top and mid cube layers
- b. Rel: position to release tension, from cube, at Close position
- c. Open: position without interferences with the cube and Cube_holder
- d. Read: position for camera reading, with the Lifter almost touching the cube (and unfortunately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



Section4: Tuning and robot operation

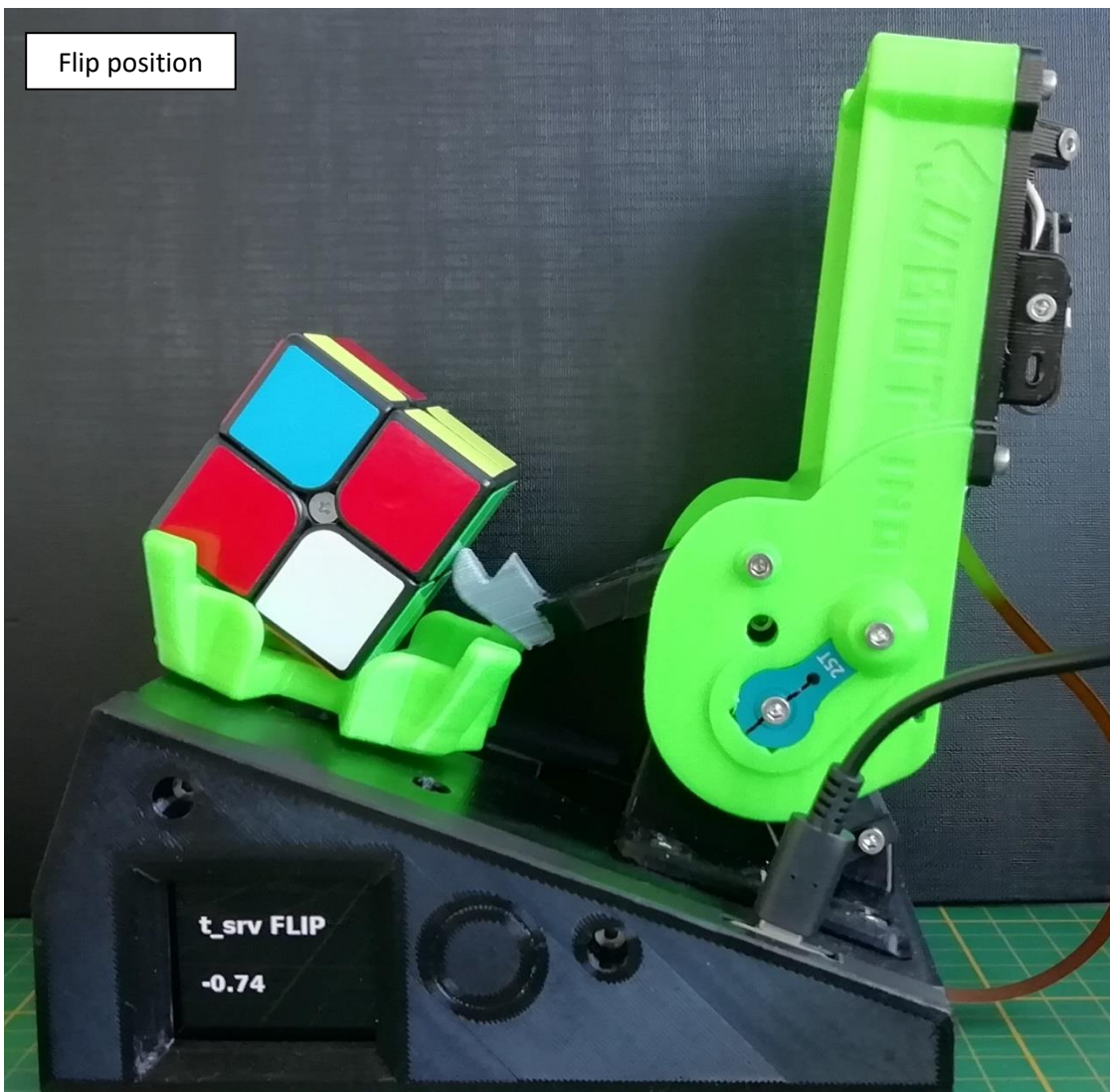
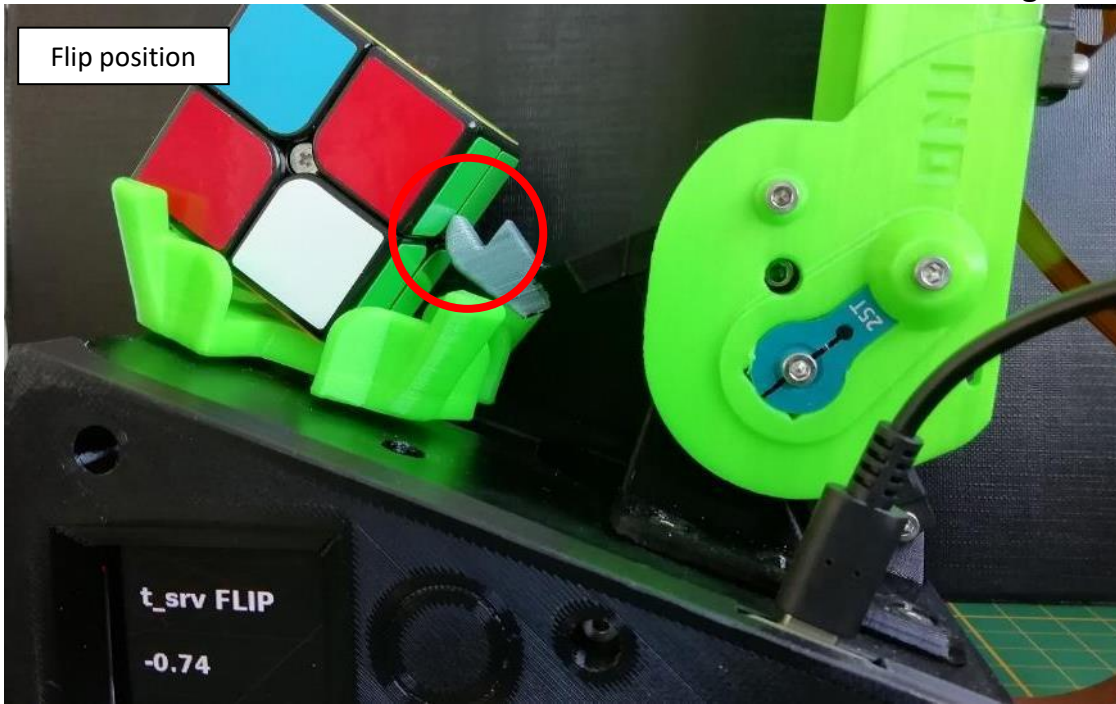
Close position
Rel position (in my case = to Close)

Tip: From the close position try to raise the Top_cover, and it should be possible to slightly lift it from the cube before perceiving tension from servo.



Read position



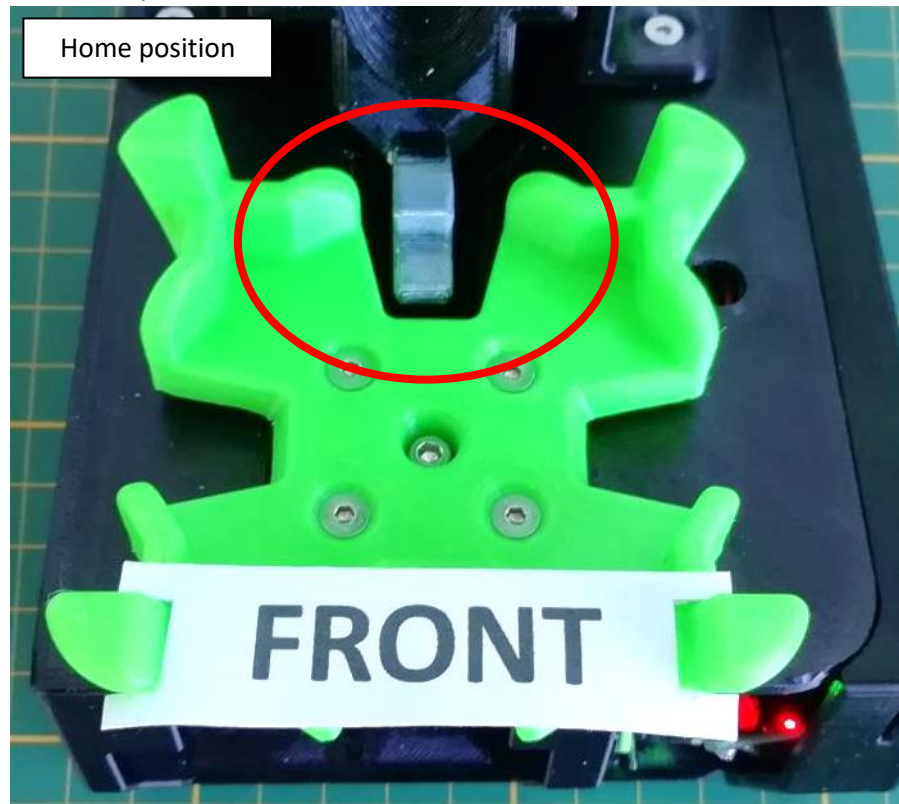


Cube_holder (b_servo) angles:

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 has to be well centered.



In order to center the Home position, it might be necessary to adjust the below parameters:

- `b_min_pulse_width`
- `b_max_pulse_width`
- `b_home`

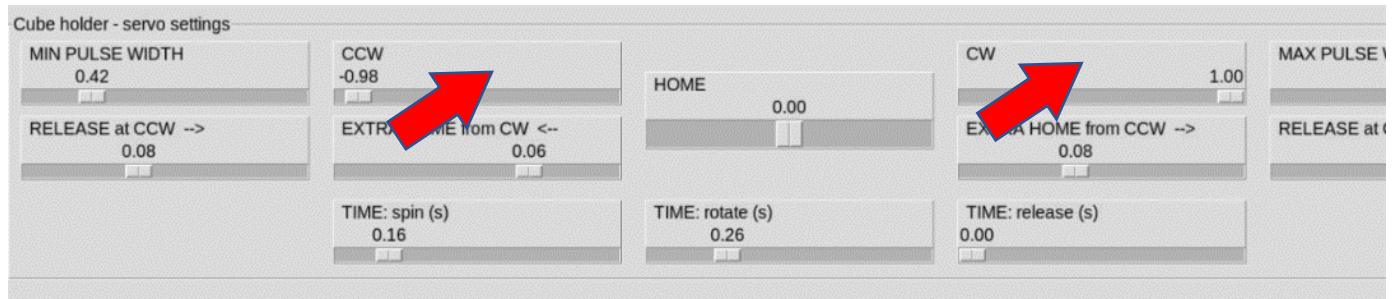
Section4: Tuning and robot operation

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

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



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:

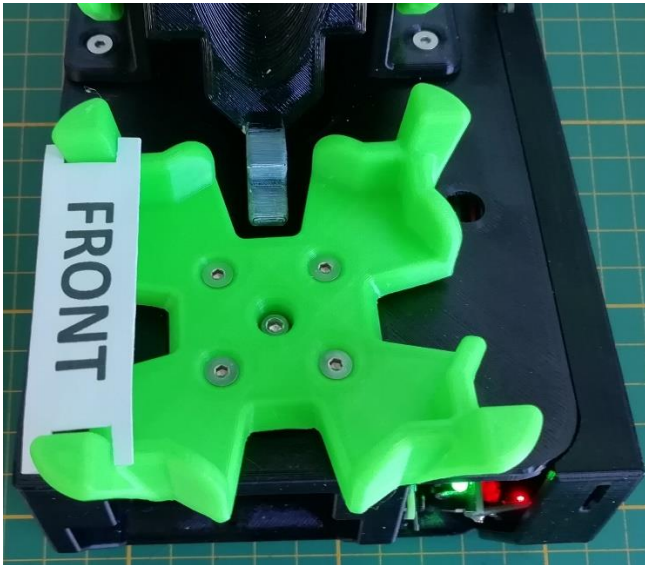
- This releases tension between the cube and the Cube_holder, and between the cube and Top_cover.
- It aligns the Cube_holder for cube flipping at CCW and CW.

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.

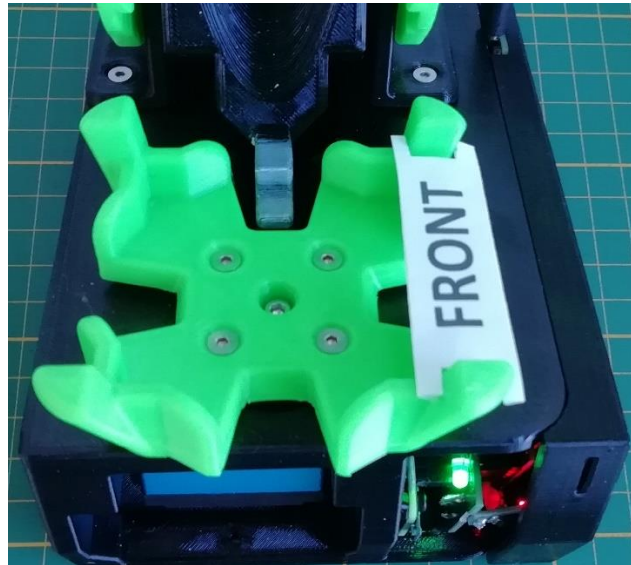
When using the GUI to tune the servos, the CCW and CW positions with the tension release are visible by pressing the CCW:



CCW position, after Release_CCW:



CW position, after Release_CW:



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_P_servo_settings.txt



The bottom line of sliders in the GUI displays the different adjustable timers.

Spin refers to the movement of the robot rotating the whole cube, i.e. a rotation of the Cube_holder with the top cover **open**.

Rotate refers to the movement of the robot rotating the bottom layer of the cube only, i.e. a rotation of the Cube_holder with the top cover **closed**.

Release refers to the small movement of the top cover going from the CLOSE position (when some downwards pressure is exerted on the cube) to the RELEASE position (when no pressure is exerted on the cube).

20. Fine tuning the servos via GUI

Servos need to be tuned first. We will then tune the camera.

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 by typing: `cd cubotino_pocket/src`
4. Activate the virtual environment by typing: `source .virtualenvs/bin/activate`
Do not forget the dot before virtualenvs.
5. run the script by typing: `python Cubotino_P_GUI.py`
6. the GUI window will open:



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 on the sliders.	The buttons for Cube_holder 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

Servos tuning high level order:

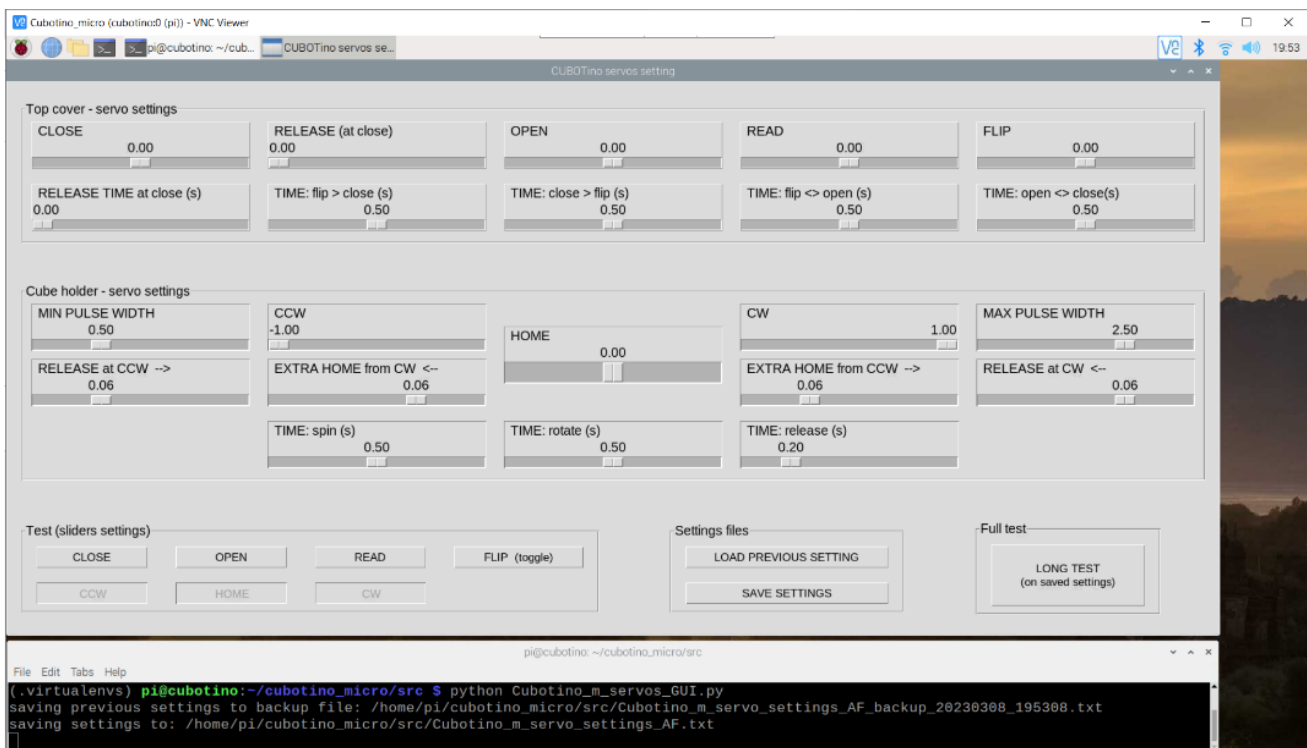
- Do not care about the timers, those are the very last thing to tweak.
- Start with the Top_servo positions, by using the sliders and the buttons.
- Maximize the Bottom_servo rotation range.
- Set the Bottom_servo positions, by using the sliders and the buttons.
- Verify a few times with the LONG TEST
- 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.
 7. I doubt the RELEASE from close being a setting really needed, on this Cubotino micro version.



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:
 - 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.
 - center the Cube_holder to allow Flipping at “CCW”, “Home” and “CW” positions.
 - When the **Top_cover is in open position**, the positions “CCW”, “Home” and “CW” are without making any extra-rotation; The only target is:
 - center the Cube_holder to allow Flipping at “CCW”, “Home” and “CW” positions.

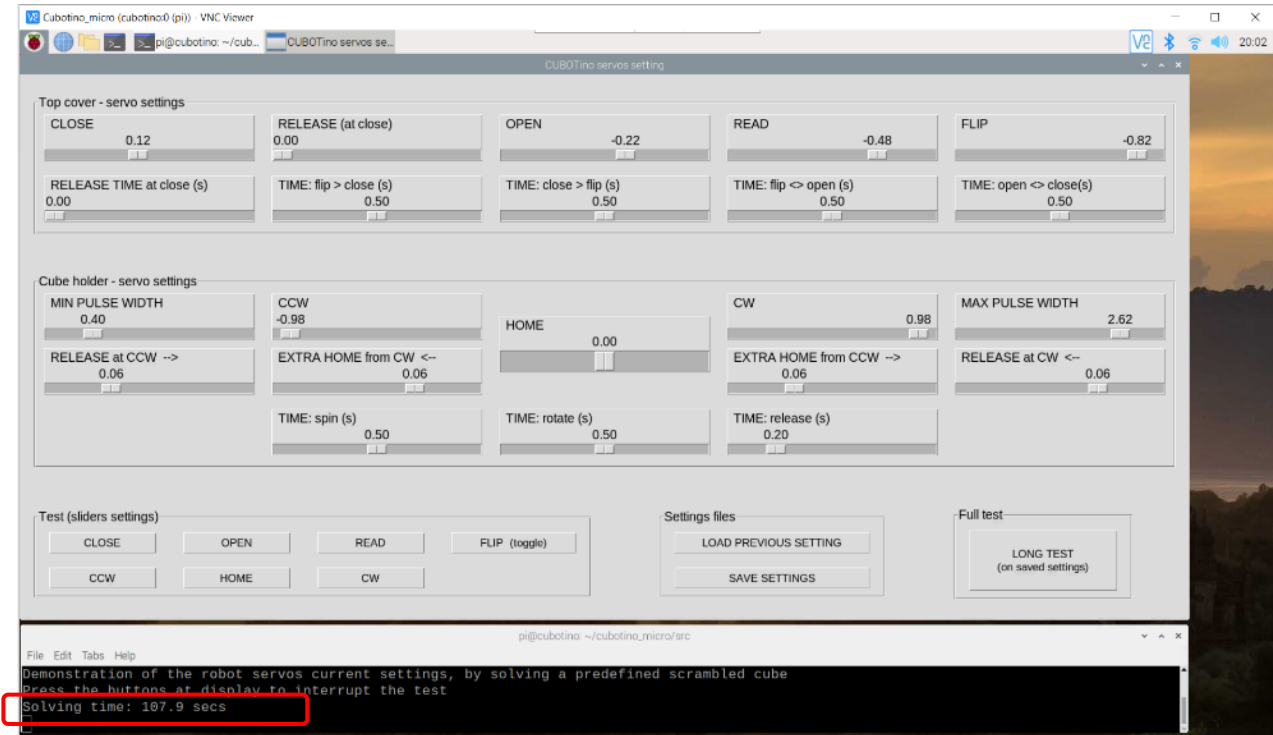
Suggested sequence for the Cube_holder settings:

1. Press OPEN button to position the Top_cover to the open position: This activates the Cube holder – servo settings widgets.
2. Search the min 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 the 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 the 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 centered.
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:

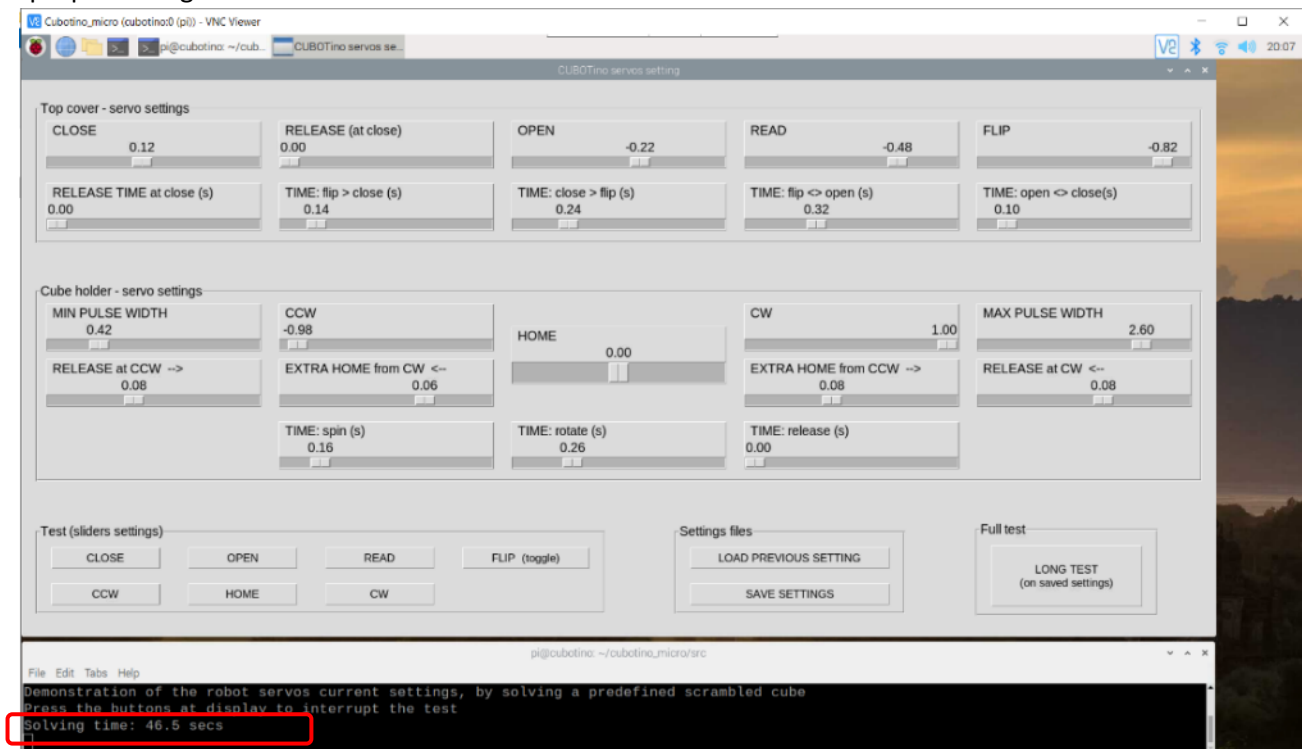
Section4: Tuning and robot operation

- The LONG TEST is based on the last saved settings.
- During this test the GUI is not anymore active.
- Use the buttons at the robot (display) to eventually interrupt the test.
- Once the LONG TEST is finished, the GUI is back active.

LONG TEST: With the timers set as per default, the solving test time is ca 108 seconds (Raspberry Pi Zero 2):



With proper tuning it can be lowered to ca. 47 seconds.



21. Tuning the camera via GUI

Now that servos are tuned, it is necessary to adjust the settings for the camera.

1. Different kinds of cubes:

On the market there are 2x2x2 Rubik's cubes with and without the black frame around the colored facelets. The cube facelets detection algorithm considers the possibility to use cubes with and without the frame. Below Rubik's cube images are explicative for the most common types of facelets:



At Cubotino_P_settings.txt there is a "frameless_cube" parameter, with below options:

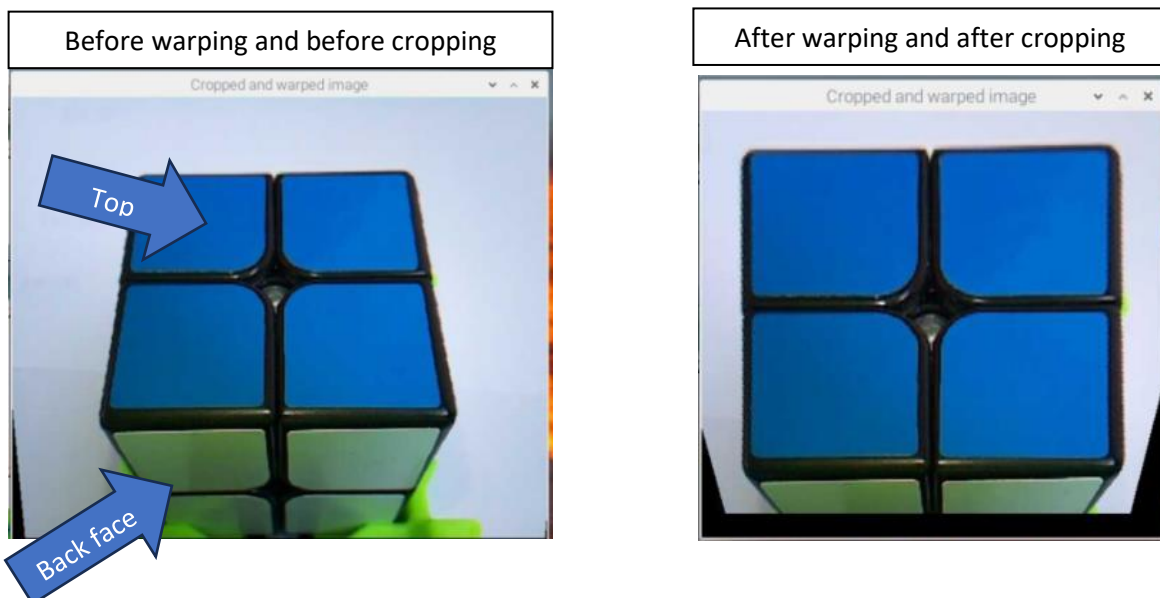
- 'true' for the frameless type.
- 'false' for the classic cube type.
- 'auto' for both types, (default value).

The 'auto' mode is computationally more demanding, therefore it will take slightly longer (perhaps one to two seconds more for the six cube faces).

2. Frame cropping:

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

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



Section4: Tuning and robot operation

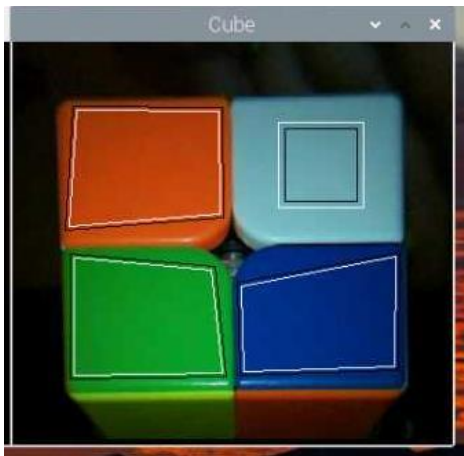
Image cropping must be tuned, to reduce the image to the region of interest (ROI).

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 of having some facelets at the back of the 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:



The cropping parameters identify the bandwidth, in pixels, to be removed from the four image sides.

4. Setup the camera with the GUI:

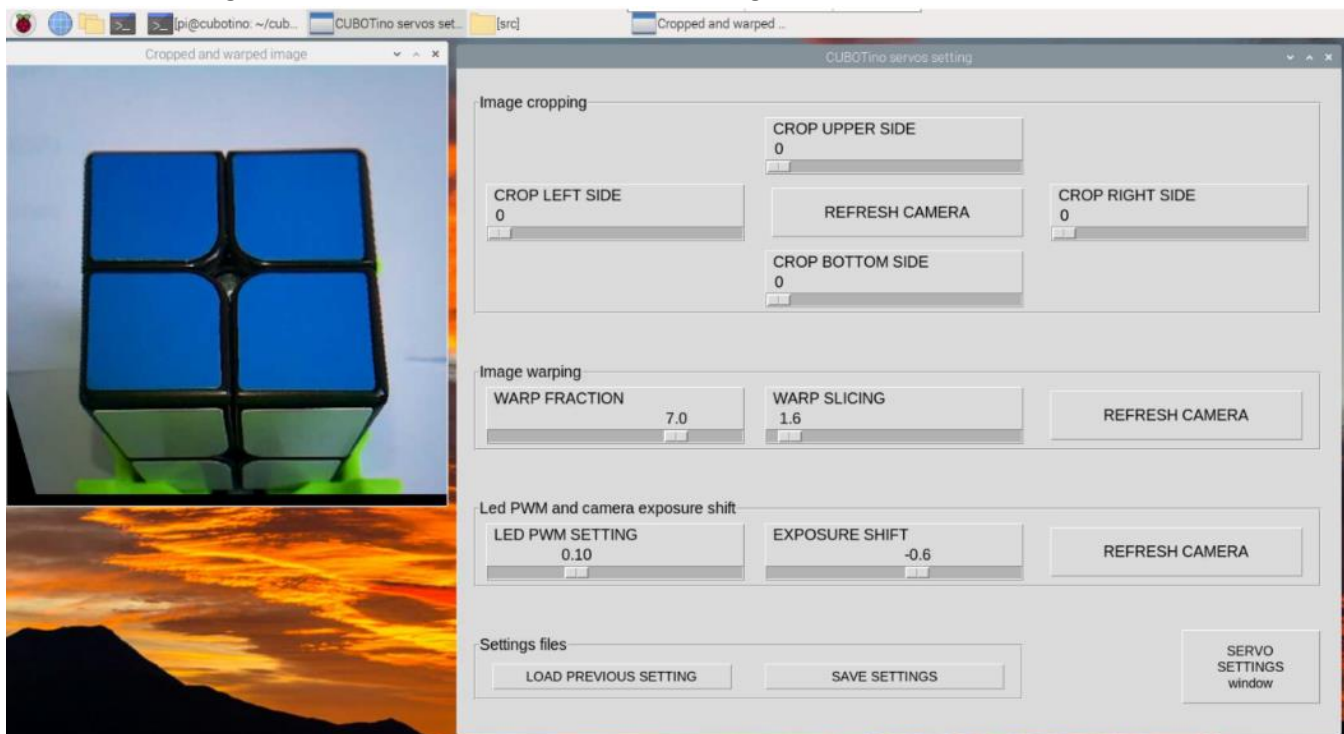
Access the tuning GUI, as done earlier for tuning the servos:

- 1.- Enter the cube folder by typing `cd cubotino_pocket/src`
- 2.-Activate the virtual environment by typing: `source .virtualenvs/bin/activate`
Do not forget the dot before virtualenvs.
- 3.- Run the GUI script by typing: `python Cubotino_P_GUI.py`

The GUI window will open:



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



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.

Remember to press the SAVE SETTINGS button; This way the settings will be correct at the next usage.

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.

Congratulations!

You have made it, the Cubotino Pocket is now ready for solving your first cube!

To preserve all your work to get here, it is advisable to make a second backup copy of the SD card of the Cubotino. Please refer to step 13 of "Setting up the Raspberry Pi" in the present document.

22. How to operate the robot

1. Before starting:

At the robot start, the Top_cover will 'suddenly' open: Do not let kids sticking their nose right on top of robot!

2. Use the robot on a uniformly colored table:

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

Keep some free space around, and use a table of a uniform color, to prevent cables or other objects from being eventually detected as facelets.

3. Power up the robot:

a) Single power supply

Connect a 5.1V power source to the microUSB type C connector.

In my case the servos work flawlessly with the original Raspberry Pi 3A power supply.

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

b) Independent power supply,

Power up the servos

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

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.

If the servos are already connected (or at least energized before the Raspberry Pi boots), *Cubotino_P.py* script takes care to position the servos according to a pre-defined order.

Power up Raspberry Pi Zero 2:

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).

4. Run the Cubotino_P.py script

- Access src folder: `cd cubotino_pocket/src`
- Activate the virtual environment: `source .virtualenvs/bin/activate`
- Run the script: `python Cubotino_P.py`
- 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 the 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 purse

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:

- Arguments don't have an order.
- Most of them can be combined (when --cycles then --timer is skipped as not compatible).

Examples:

```
python Cubotino_P.py --debug --F_deg
python Cubotino_P.py --cv_wow --F_deg
python Cubotino_P.py --slow_t 15
```

5. 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.

6. Start a scrambling cycle:

- f. Position the cube on the cube holder.
- g. Cube layers should be reasonably aligned.
- h. Double touch (within 1 second) the PCB_cover in front to the touch sensor (the circle suggests the touch sensor location).
- i. 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.
- j. The scrambling process can be interrupted at any time, by pressing the touch sensor for about 1 second.

7. 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_P.py script.

8. 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_P_bash.sh file (*halt -p*); This means the SBC will automatically shut down when the *CUBOTino_P.py* file ends.

To quit the *CUBOTino_P.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!

9. Un-power the robot:

Detach the robot from the power supply.

10. Running the robot from VNC Viewer:

When the robot script is already running, and you'd like to connect via VNC viewer to interact with the robot, it is necessary to interrupt some processes:

- It is not an option to quit the script from the robot, by keeping the 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 script first.

The easy way:

Press one of the robot buttons until the "EXITING SCRIPT" appears on the display (about 10 seconds), then you can release the button.

This action ends the Cubotino_P_bash.sh file and the Cubotino_P.py script: This means the Raspberry Pi is still fully active, and it will accept a connection via VNC.

This also means you'll be forced to use "*sudo halt -p*" when it will be time to shut the Rpi off.

The general way:

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

```
pi@raspberrypi:~$ 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/a
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 ?        Sl   16:24   0:00 /usr/lib/gvfs/g
root      674 33.2 27.3 360008 102152 ?        Rll  16:24   0:13 python Cubotino
pi        684  1.2  2.1  62392   8132 ?        S    16:24   0:00 openbox --confi
```

- 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 it easier to find it.
- Search for bash command, from root user, located above python Cubotino, and note the ID (657 on the above example)
- First** kill the bash process *sudo kill -9 ThePIDNumberForBash* (by using the above example the command will be *sudo kill -9 657*)
- After** kill the python process *sudo kill -9 ThePIDNumberForPythonCubotino* (by using the above example the command will be *sudo kill -9 674*)

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

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 😊

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

In case you’d like to run many cycles, for statistical purposes, 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_P.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 buttons.
-- 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 buttons. It does work only if the --cycles is also provided.

Examples:

- python Cubotino_P.py --cycles 10* will scramble and solve the cube 10 times.
- python Cubotino_P.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_P.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 buttons, to automate the shut-off).

Notes:

- Active cycle and the total cycles plotted on robot screen.
- The remaining time for next cycle plotted on robot screen.
- When “--pause” is provided, on the robot screen is indicated the remaining time to start the next cycle.
- When “--cycles” is provided without “--pause” argument, there won’t be pause between cycles.
- 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.
- Image on PC is completed with last performed cycle and total cycles, additionally to date and time.
- The image is not saved.
- 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
```


23. Automatic start

Once everything is set up and works properly, it is possible to make the robot start automatically after the Raspberry Pi boot.

Warning: during the test and tune phase, do not use this possibility. Do it only when the tuning is completely stable, and you've reached the desired performances.

From the root or from the virtual environment, type: *sudo crontab -e*

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

```
(.virtualenvs) pi@cubotino:~/cubotino_pocket/src $ 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 1920x10800"
```

```
#@reboot /bin/sleep 5; bash -l /home/pi/cubotino_pocket/src/Cubotino_P_bash.sh >
/home/pi/cubotino_pocket/src/Cubotino_P_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_P_bash.sh* file.

The *Cubotino_P_bash.sh* file is a file governing the automated startup of the robot, calling the necessary tasks one after another. Edit with caution.

Example: change "*python Cubotino_P.py*" to "*python Cubotino_P.py - -timer - -F_deg*"

25. Rpi shut down via Touch button

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

From the folder `/home/pi/cubotino_pocket/src`, edit the file with `sudo nano Cubotino_P_bash.sh` and uncomment the `#halt -p` command.

```

~/cubotino_pocket/src/Cubotino_P_bash.sh · Mousepad
File Edit Search View Document Help
#!/usr/bin/env bash

#####  Andrea Favero, 26 January 2024 #####
# This bash script activates the venv, and starts the Cubotino_P.py script after the Pi boots
# When the python script is terminated without errors (long button press), the Pi shuts down
# (check notes below before uncommenting the "halt -p" command).
#####

# activate the venv
source /home/pi/cubotino_pocket/src/.virtualenvs/bin/activate

# enter the folder with the main scripts
cd /home/pi/cubotino_pocket/src

# runs the robot main script, without or with arguments
python Cubotino_P.py
# python Cubotino_P.py --slow 10

# exit code from the python script
exit_status=$?

# based on the exit code there are three cases to be handled
if [ "${exit_status}" -ne 0 ];
then
    if [ "${exit_status}" -eq 2 ];
    then
        echo ""
        echo "Cubotino_P.py exited on request"
        echo ""
    else
        echo ""
        echo "Cubotino_P.py exited with error"
        echo ""
    fi
else
    echo ""
    echo "Successfully executed Cubotino_P.py"
    echo ""

    # '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

fi

```

Notes:

1. Do not uncomment it before being sure the robot code runs without errors; A little indentation error sometimes happened when changing a parameter.
2. Do not uncomment it before having made an image of the microSD.

26. Troubleshooting

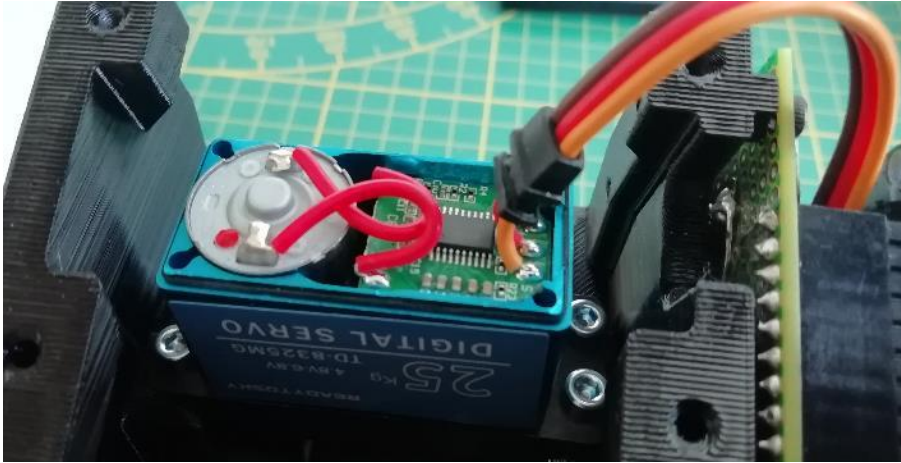
Some of the below aspects were encountered during this or previous robots' development, other were posted at Instructables for previous robots' design, other were suggested by some of the Makers, and remaining are hypothetical:

1. Servos rotate about 180 degrees, yet not more than that.
2. Servos rotation angle of about 90 degrees, 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. Low Voltage Throttle.
14. Updating the CUBOTino software.
15. Raspberry Pi freezing (memory management).
16. Raspberry Pi Wi-Fi dropping.
17. Errors during the Raspberry Pi installation process.
18. Error "Total corner twist is wrong" after scanning the cube.
19. Display and/or Top_cover_led flickering right after the boot.
20. Touch sensor issues

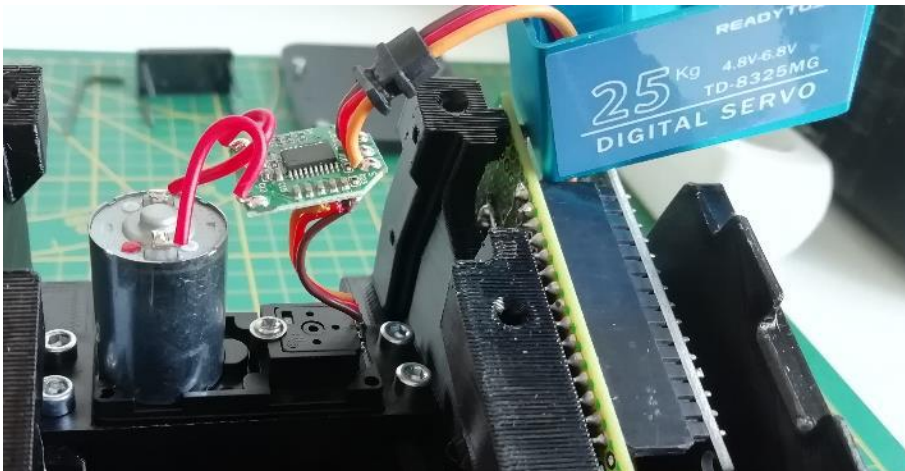
1 Increase the servo rotation range (if needed)

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)

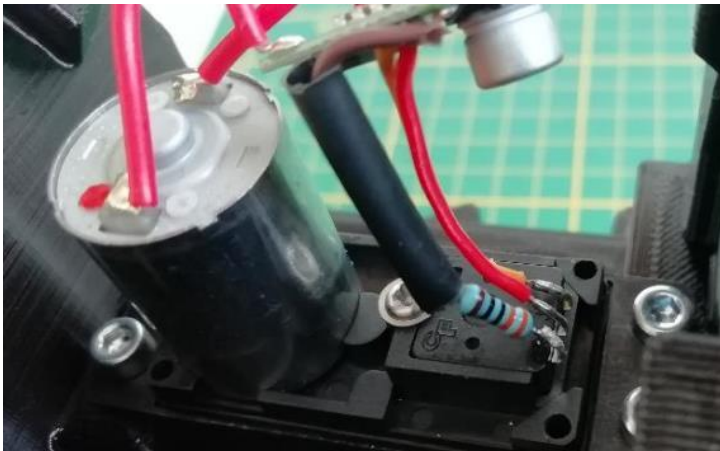


2. Slide out the pcb, and afterward slide the case out of the way.

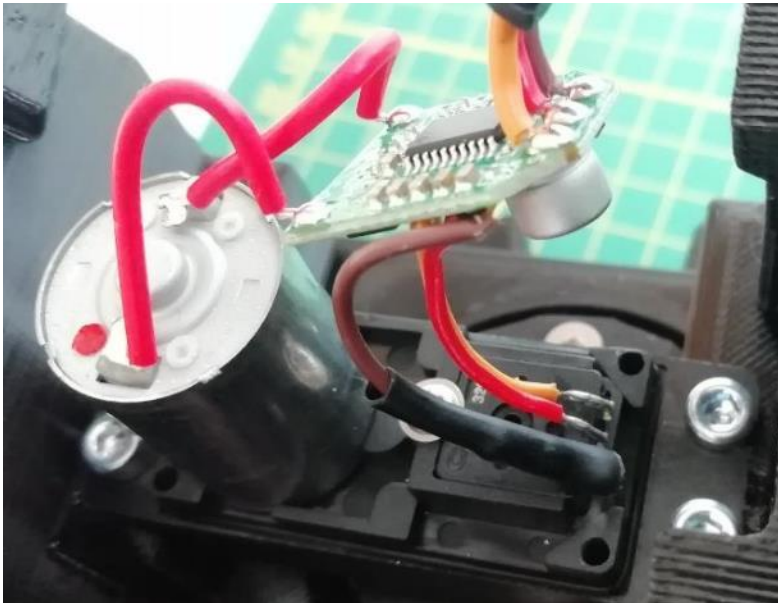


3. Solder one resistor to one external pin of the potentiometer (or one resistor per each of the two external potentiometer pins to keep the same PWM value for the mid position).

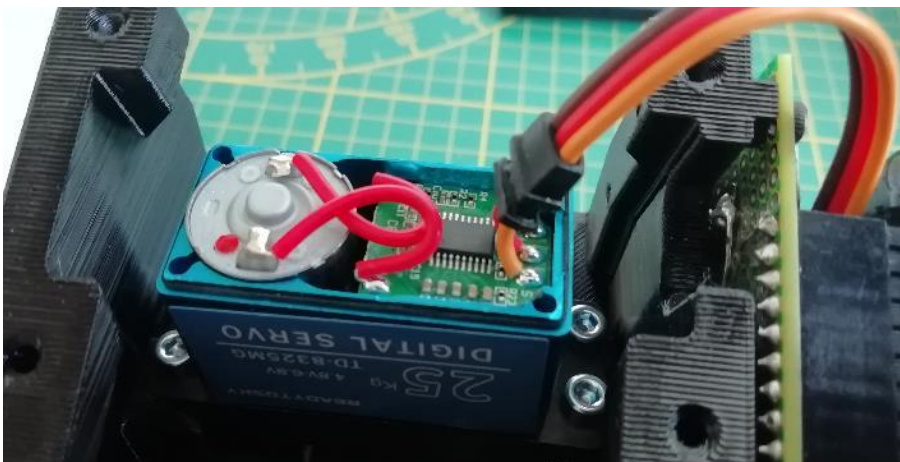
I've added 330ohm to gain about 5 degrees on a servo having pulse width from 1 to 2 ms.



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

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 500µs to 2500µs instead of from 1 to 2 ms.

This project uses as default a Pulse Width range from 1ms to 2ms, yet you can adjust some parameters and get your servos working fine; At Cubotino_P_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)

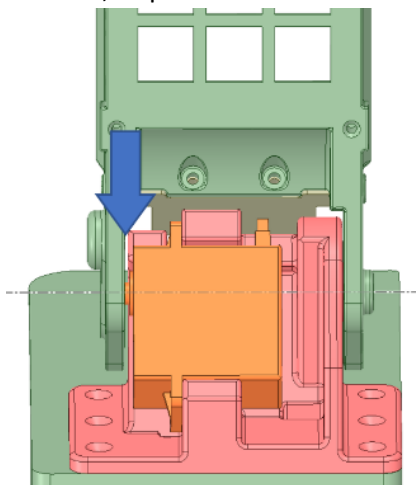
3 In case you've got servos with 270deg rotation, it shouldn't be of a problem.

If the Pulse Width is from 500µs to 2500µs, 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 102ren'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 (Ø4.1 to Ø4.3mm).
 - ii. Rub some candle wax on the screw.
 - iii. Ensure the M4 screw head is not pushing toward the Top_cover; 1mm gas is suggested
 - iv. Ensure gap presence between Top_cover inner surface and the Hinge at the servo gear outlet side, as per below arrow:



In case your robot has little or no gap:

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

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 happens:
 - i. Increase the timers, as too small time don't give sufficient time to the servo to make the stroke visible when testing the cube holder position.
 - ii. Adapt the PWM release CCW/CW value.
 - iii. Place the PWM release CCW/CW at zero, and test if the CCW and CW position have a slightly overstroke from the 90°. If this is not the case, check if the other servo has a larger rotation range. If still not the case:
 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 flippings.

Most of Rubik's cube can be adjusted on the friction:

- a) remove the cap at the center 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 4 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 for 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_P_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 4 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 4 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_P.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 facelets, that I've made matt with sandpaper (grit 1000)



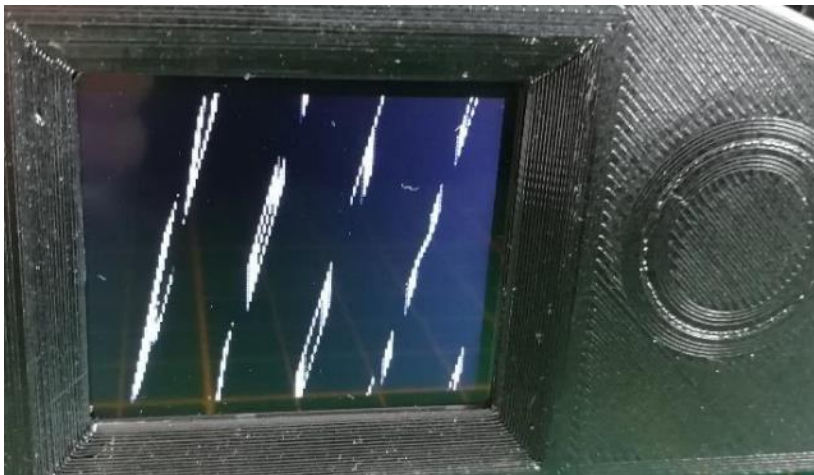
Cube with glossy stickers (after taking this picture I made matt these facelets too)



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

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

- Run Cubotino_P.py with -- debug argument: This variable is used by many functions to printout info to the terminal.
- Check the prints.
- 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:

- Edit Cubotino_P_servos.py
- 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.
- Run Cubotino_P_servos.py (activate the venv first and recall typing python in front). This code activates the servos like solving a predefined scrambled cube.
- Check the prints.
- Run Cubotino_P.py and let it call the Cubotino_P_servos.py.
- Check the prints.
- If the prints out don't suggest much to you, share it at the Instructables s chat.

12. PiCamera focus

See "Set the correct Focus for the PiCamera" chapter.

13. 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

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.

14. 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 “Updating the files” chapter in Annex 1 for the details on how to check / update your system.

15. 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 irresponsive 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.

16. 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:

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.

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

17. 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".

18. 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 😊.

19. 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_P.py script, or others, is started; When the Robot automatically starts at the boot, the flickering hasn't enough time to become apparent.

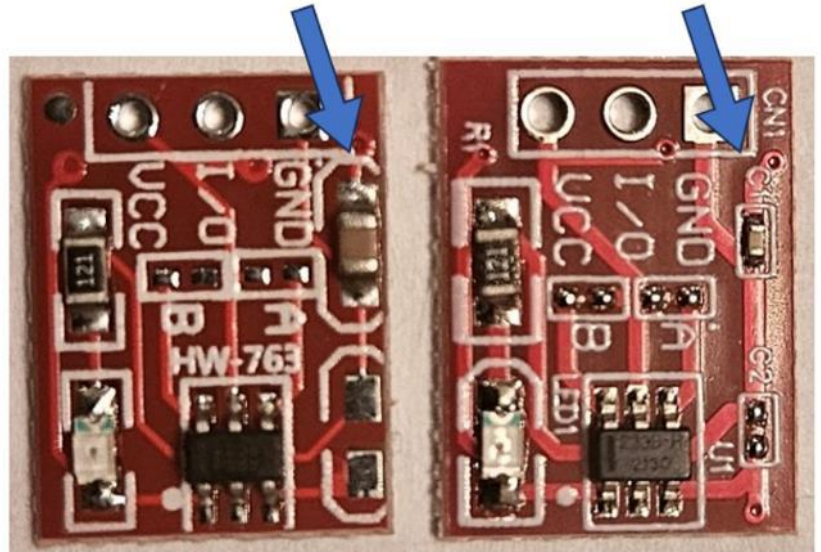
20. Touch sensor issue.

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.



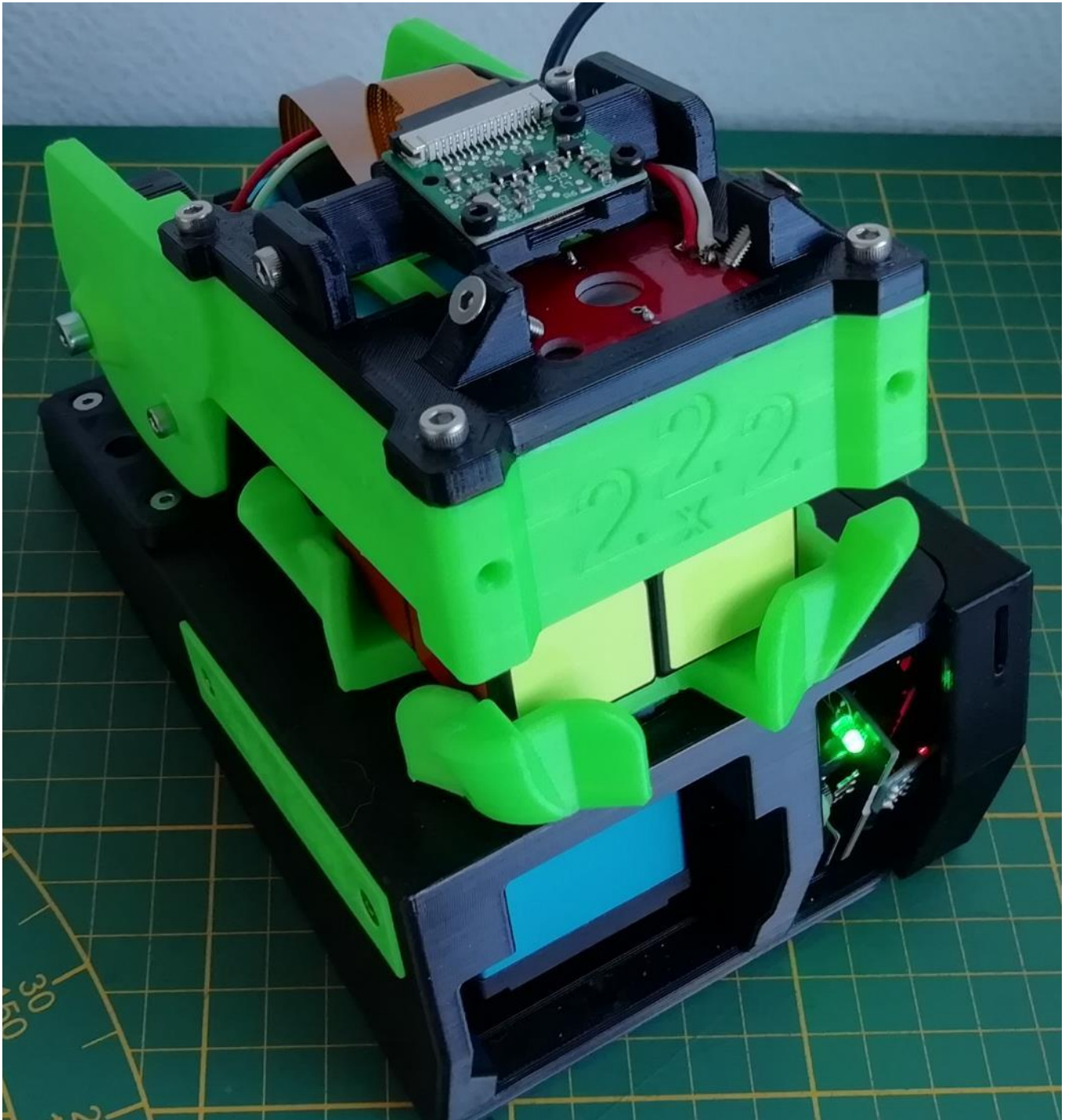
In case you experience issues with the touch sensor, please consider:

- 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.

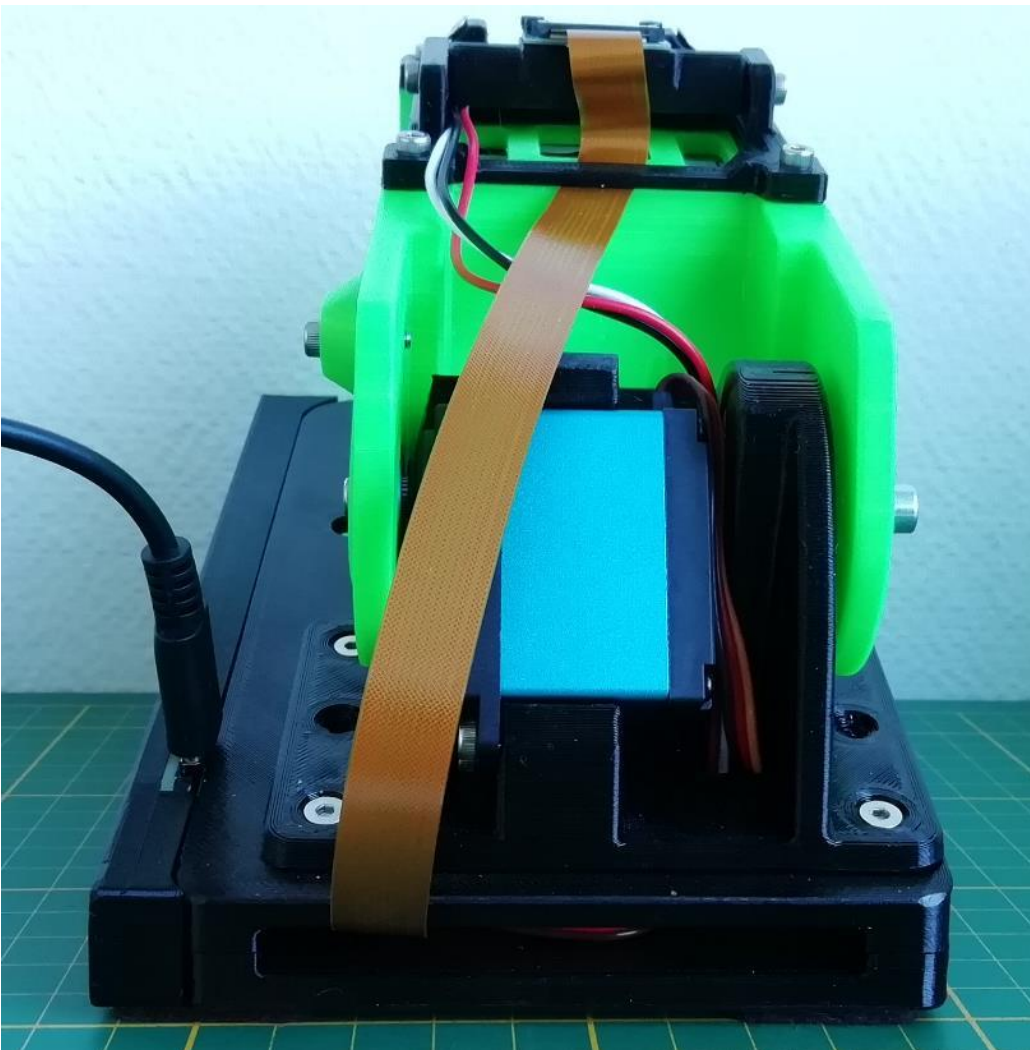
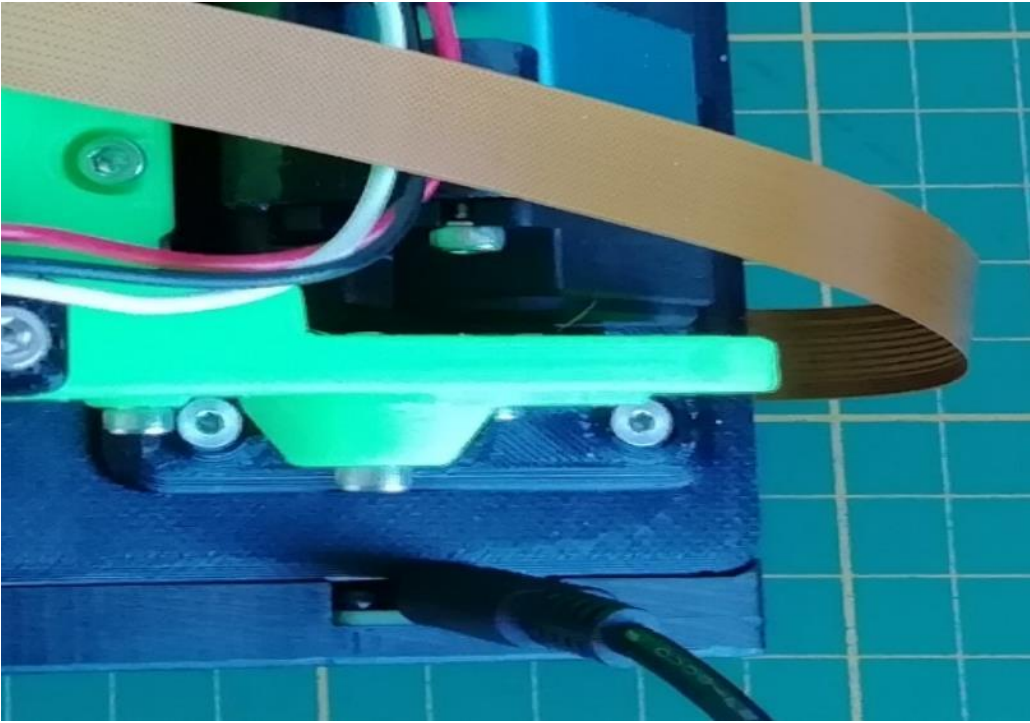
27. Collection of robot's pictures

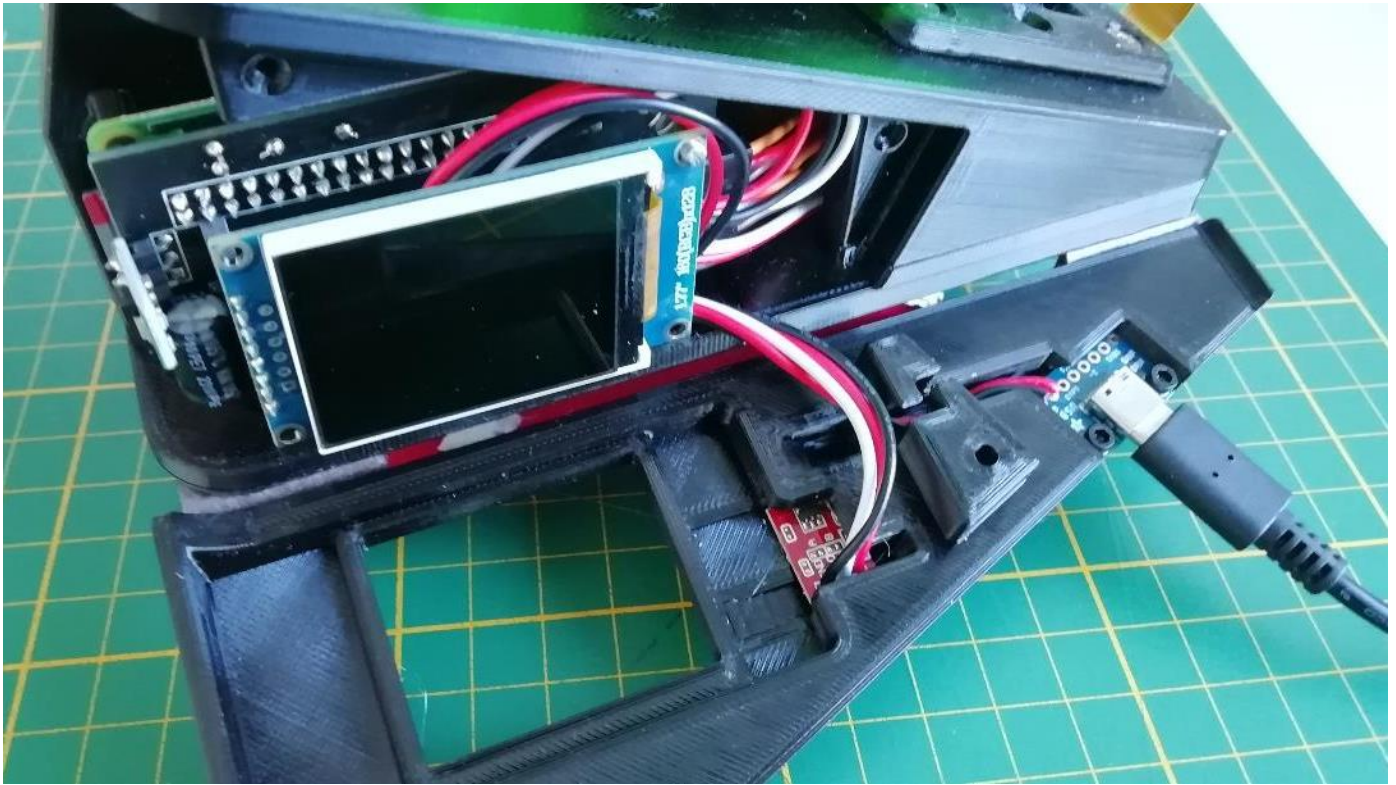












Appendix 1

ADDITIONAL INFORMATION

1. Updating the files

I'm still rather new to the vast git world; In case you experience problems by following these instructions, please let me know. Also, well accepted tips 😊.

The installation made via git command, allows you to easily update your robot, in case newer revisions will be made available at GitHub.

1. Enter cubotino folder : `cd ~/cubotino_pocket/`
2. Type `git status` to check if updates are available.
 - a. In case there are updates available, type `git pull` to receive them.

Notes:

1. Before updating your robot, you're encouraged to take notes (or save a copy) of your personal settings; If you make a file copy, it should be copied outside cubotino_pocket folder.
2. Personal settings are "Cubotino_P_settings.txt and Cubotino_P_servos.settings.txt"
3. The robot also makes a backup copy of your settings every time the Cubotino_P.py is started; This gives you one more chance to recover your previous settings (at least until Cubotino_P.py is executed after the update); In that case remember to rename the backup files:

Cubotino_P_settings_backup.txt	→	Cubotino_P_settings.txt
Cubotino_P_servo_settings_backup.txt	→	Cubotino_P_servo_settings.txt
4. Eventual new parameters that will be added after your first installation, will be automatically appended to your settings the first time Cubotino_P.py (or Cubotino_P_GUI.py) is launched.

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.

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

2. Files copied to Raspberry Pi

1) Below robot files are copied into */home/pi/cubotino_pocket/src* folder, by the installation process:

File	Purpose	Notes
Cubotino_P.py	Main robot script	
Cubotino_P_bash.sh	Bash file to start-up the robot script automatically after Raspberry Pi boots	
Cubotino_P_camera_os10.py	Manages the PiCamera when OS 10	It uses the picamera library
Cubotino_P_camera_os11.py	Manages the PiCamera when OS 11	It uses the picamera2 library
Cubotino_P_display.py	Manages the display	
Cubotino_P_GUI.py	GUI to help with servos setting	
Cubotino_P_Logo_265x212_BW.jpg	Cubotino logo, for the display	
Cubotino_P_moves.py	Translates the cube solution (Singmaster notation) in robot moves	
Cubotino_P_pigpiod	Activates and disactivates the pigpio service	
Cubotino_P_settings_manager.py	Class interacting with the setting files	From 10 th March 2024
Cubotino_P_servos.py	Manages the servos and LED.	
Cubotino_P_settings_default.txt	Json file with settings for the Camera, Display, and other functions	Default values, to start the tuning
Cubotino_P_settings_AF.txt		Optimized values for my robot
Cubotino_P_servo_settings_default.txt	Json file with settings for Cubotino_P_servos.py script	Default values, to start the tuning
Cubotino_P_servo_settings_AF.txt		Optimized values for my robot
Cubotino_P_test_random.py	Tests by generating random cube to be solved by a virtual cube handler	
Cubotino_P_test_all_permutations.py	Tests by generating all the possible cube status, by increasing depth. Each status is tested by a virtual cube handler.	
Cubotino_P_update_settings_file.py	Reads the settings from the Json settings files	
macs_AF.txt	Lists of macs addresses	Allows me (AF) to manage specific settings files for my robots while keeping generic start-up settings files for other Makers

2) Below library is copied by the installation process:

Library	Forced version	Scope and notes
Solver2x2x2		Kociemba solver for the optimal cube solution: https://github.com/hkociemba/Rubiks2x2x2-OptimalSolver This library is made by about 13 python files and 3 Lookup tables files (The python files of the solver have been modified for this project). All files copied to <i>/home/pi/cubotino_pocket/src/solver2x2x2</i>

3) Below python libraries are installed by the installation process:

Library	Forced version	Scope and notes
NumPy	1.21.4	Image array manipulation and many other functions
picamera[array]		(When OS 10) Outputs camera images in array format
st7735	0.0.4.post1	Driver for the display (purpose to have the installation compatible with Autonomous version too)
st7789	0.0.4	Driver for the display
getmac	0.8.3	Get MAC address (useful when more Cubotino bots with different settings)

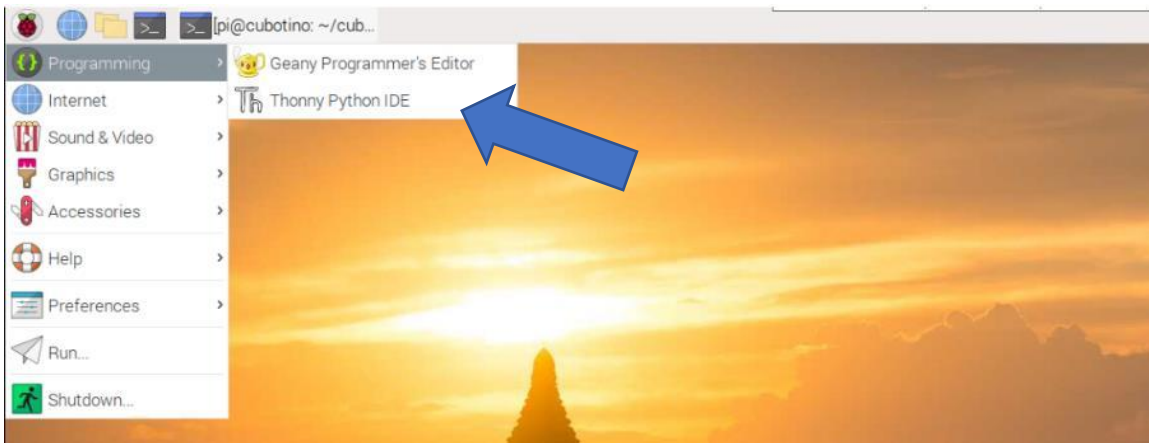
3. Set Thonny IDE interpreter

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.

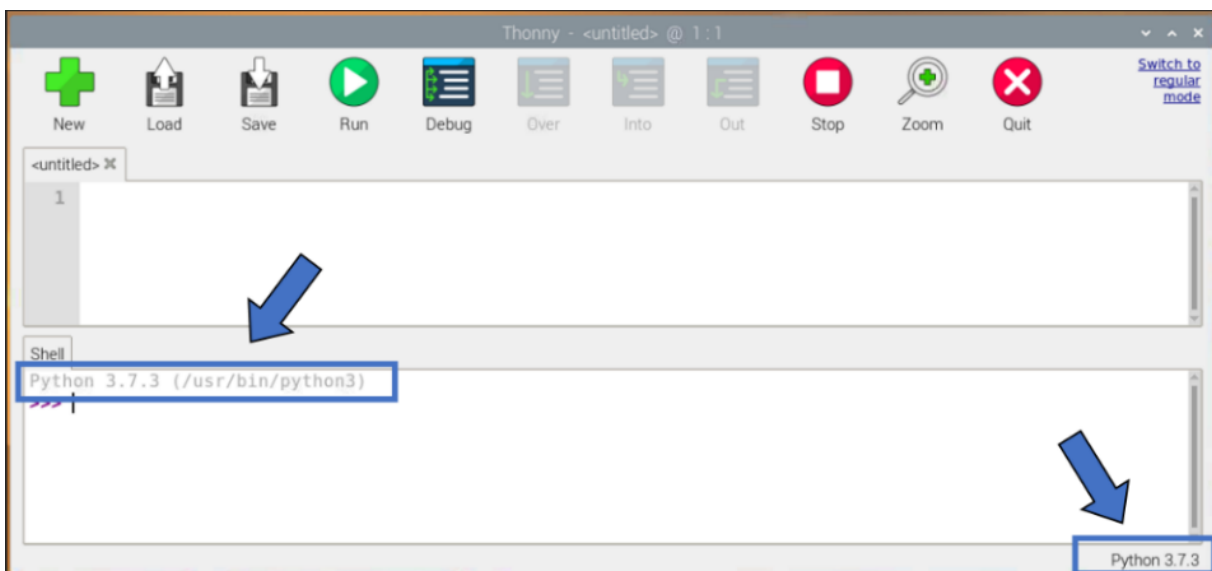
Thonny is part of the Raspberry Pi installation (according to the 'Setting up Raspberry Pi' procedure):

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.

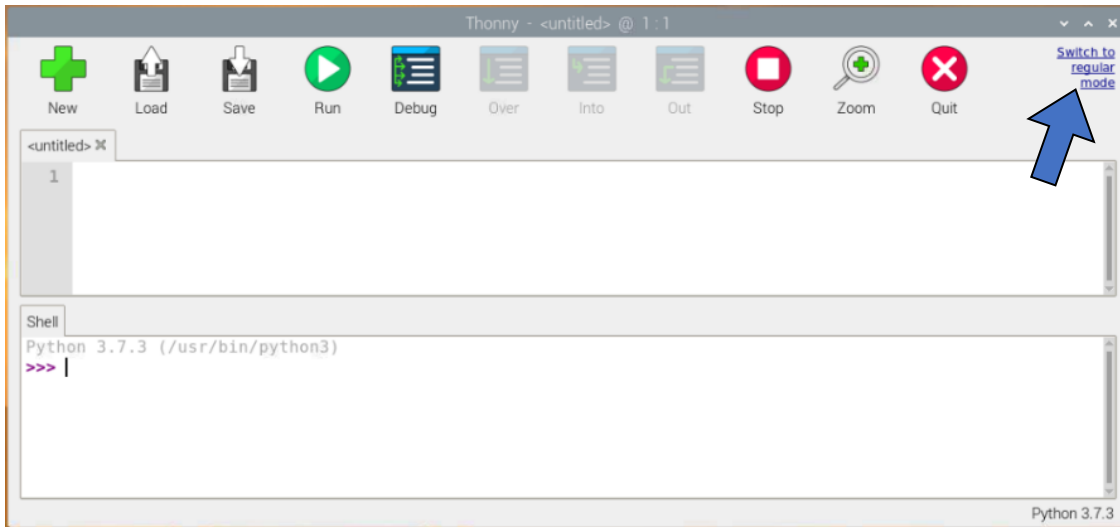


Setting up Thonny IDE interpreter, to work with the venv, it will be handy to tune the parameters hard coded in the scripts.

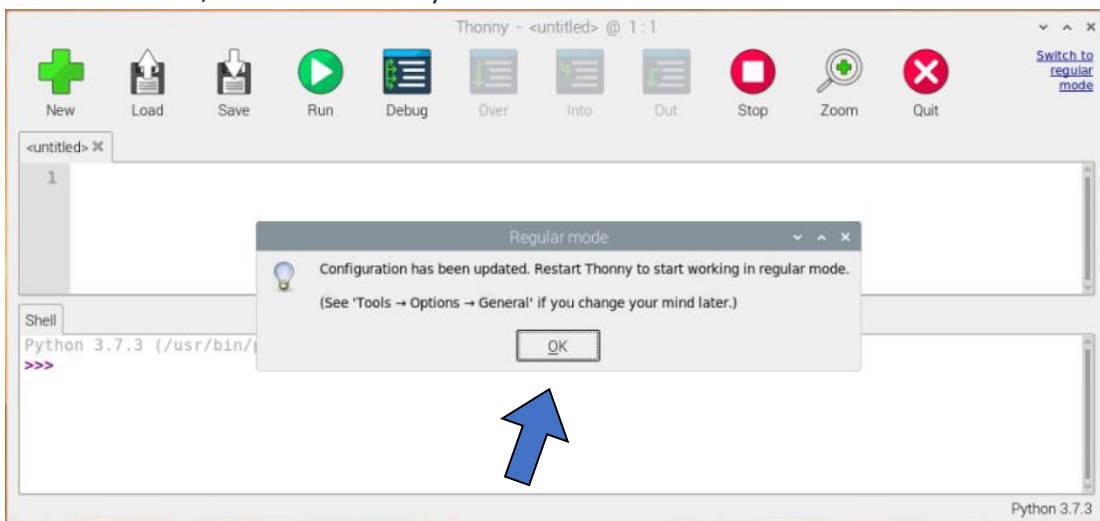
Thonny opens with the standard interpreter (`/usr/bin/python3`), and if you run `Cubotino_P.py` it won't find the libraries....



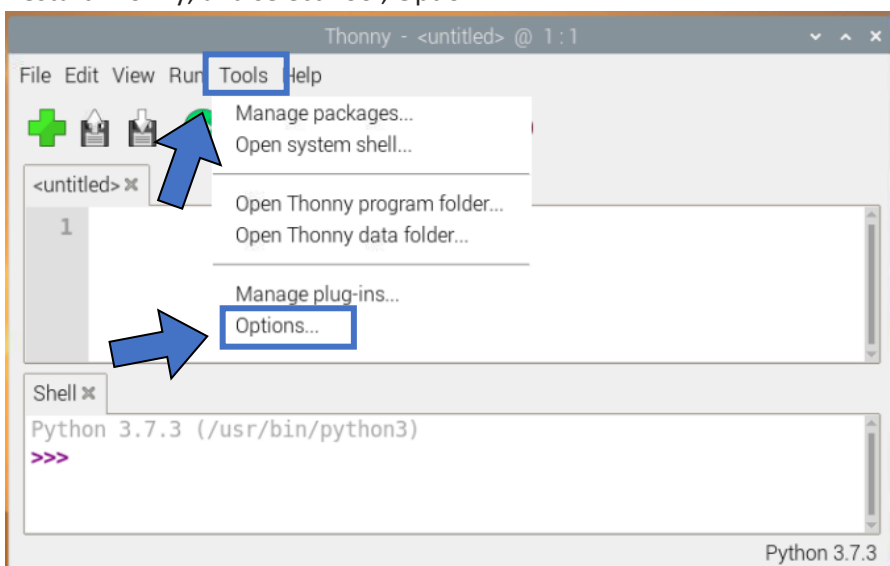
To have the Option menu, it is necessary to change the mode:



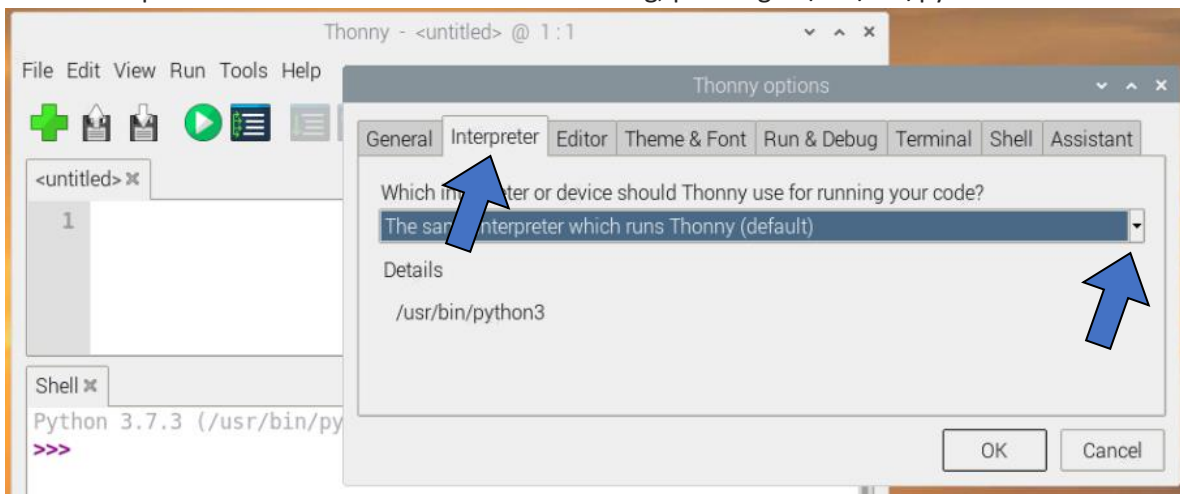
Confirm the info, and restart Thonny.



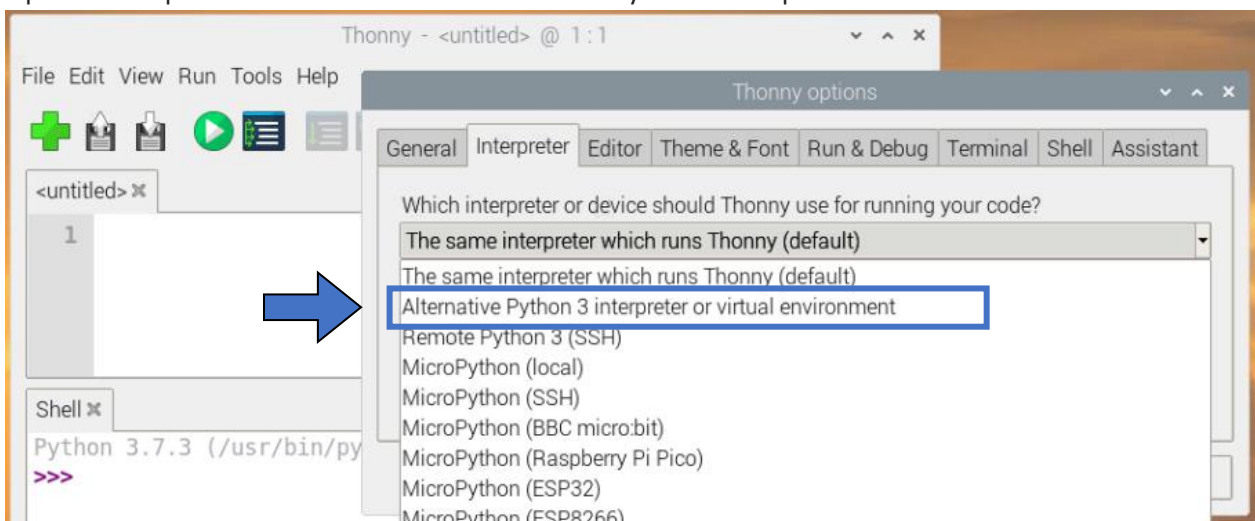
Restart Thonny, and select Tool, Option



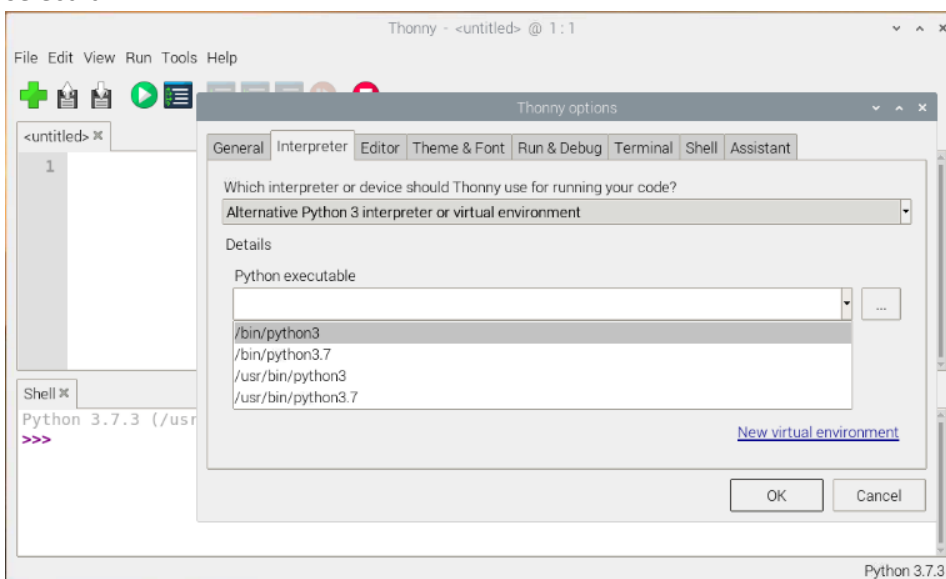
Select Interpreter where it is shown the default setting, pointing to `/usr/bin/python3`.



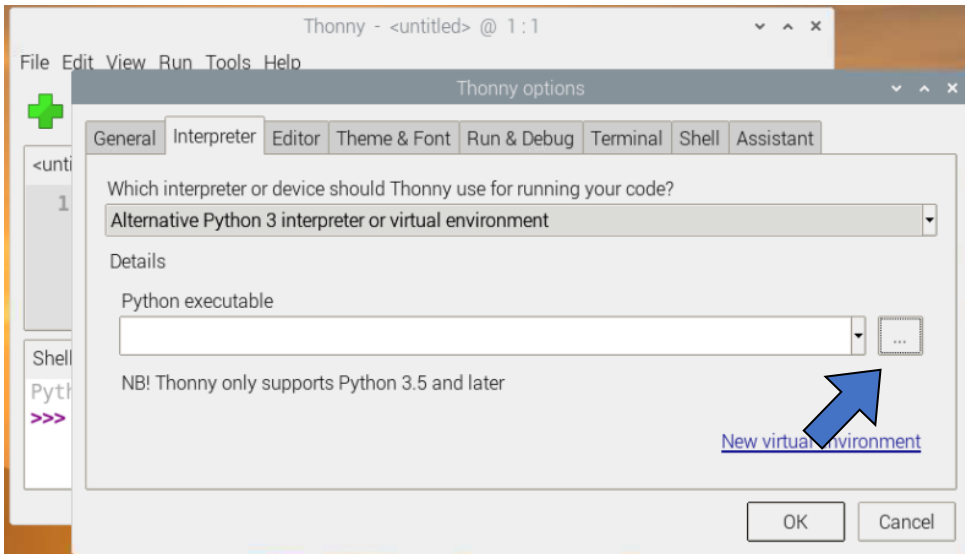
Open the drop-down menu and select 'Alternative Python 3 interpreter or virtual environment':



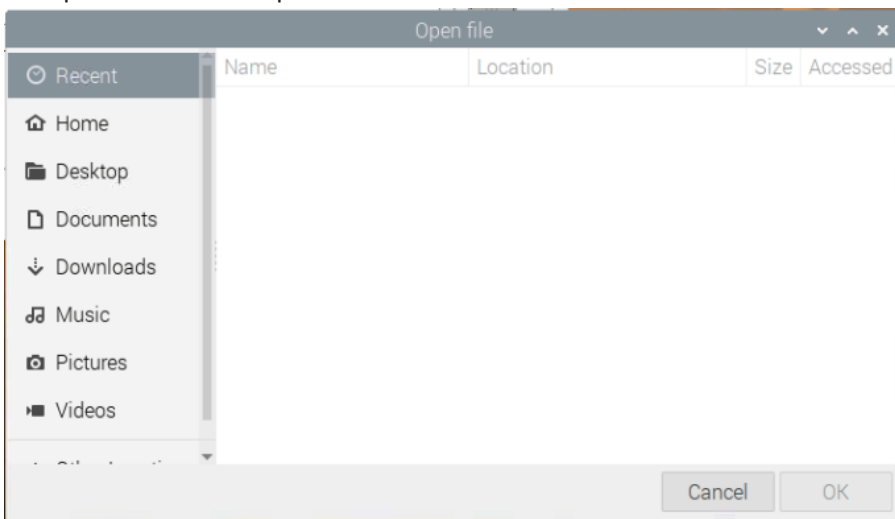
Open the drop-down menu and if `/home/pi/cubotino_pocket/src/.virtualenvs/bin/python3` is listed just select it.



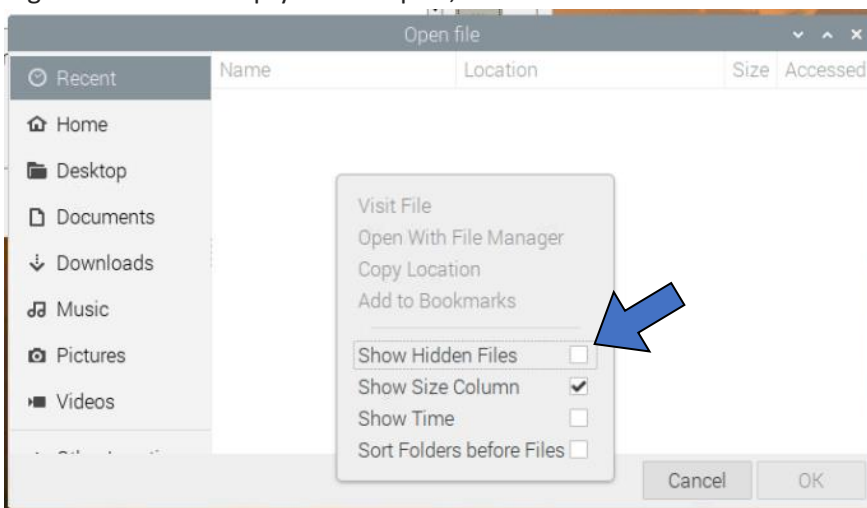
If `"/home/pi/cubotino_pocket/src/.virtualenvs/bin/python3"` is not listed, select the browse button:



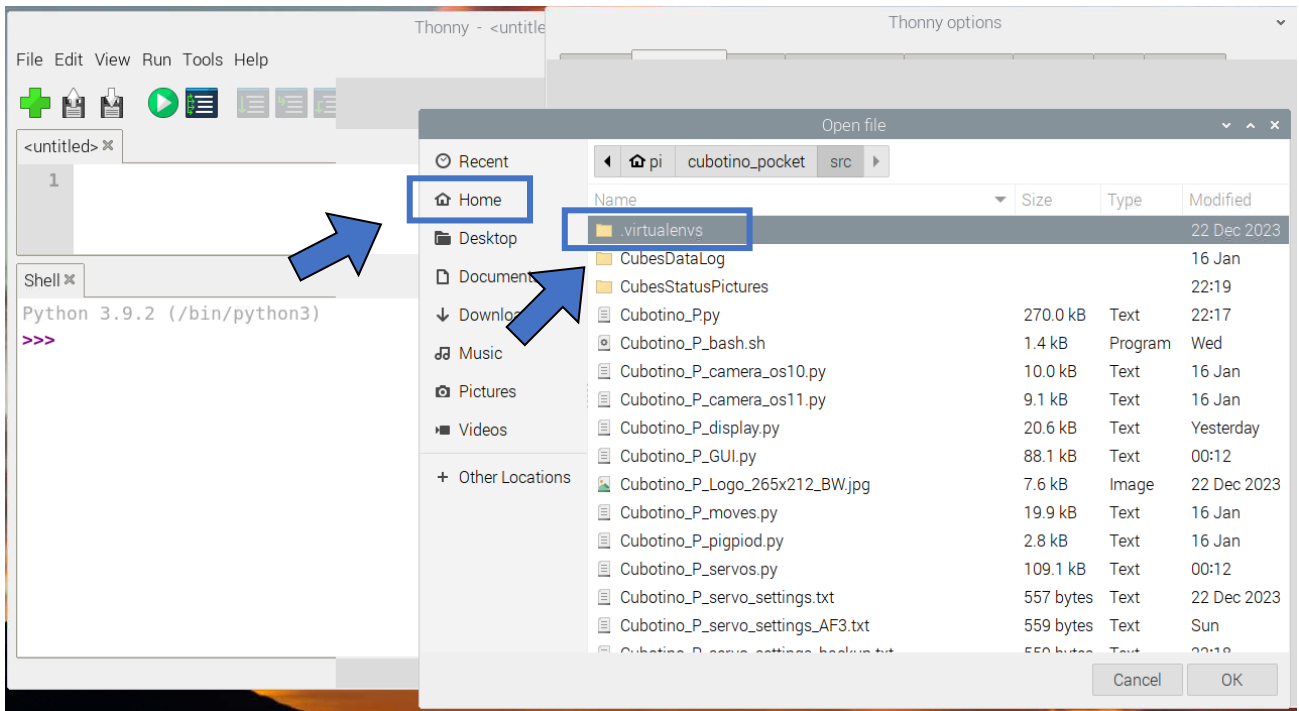
An Open file window opens:



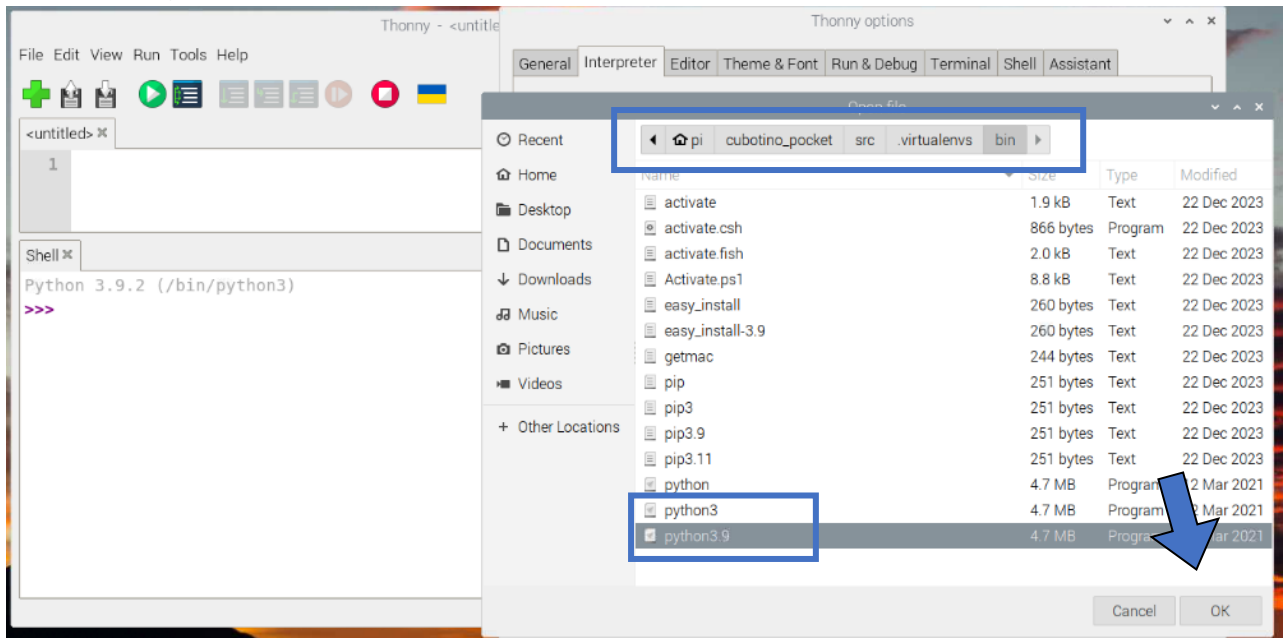
Right click on the empty window part, and check 'Shows Hidden Files':



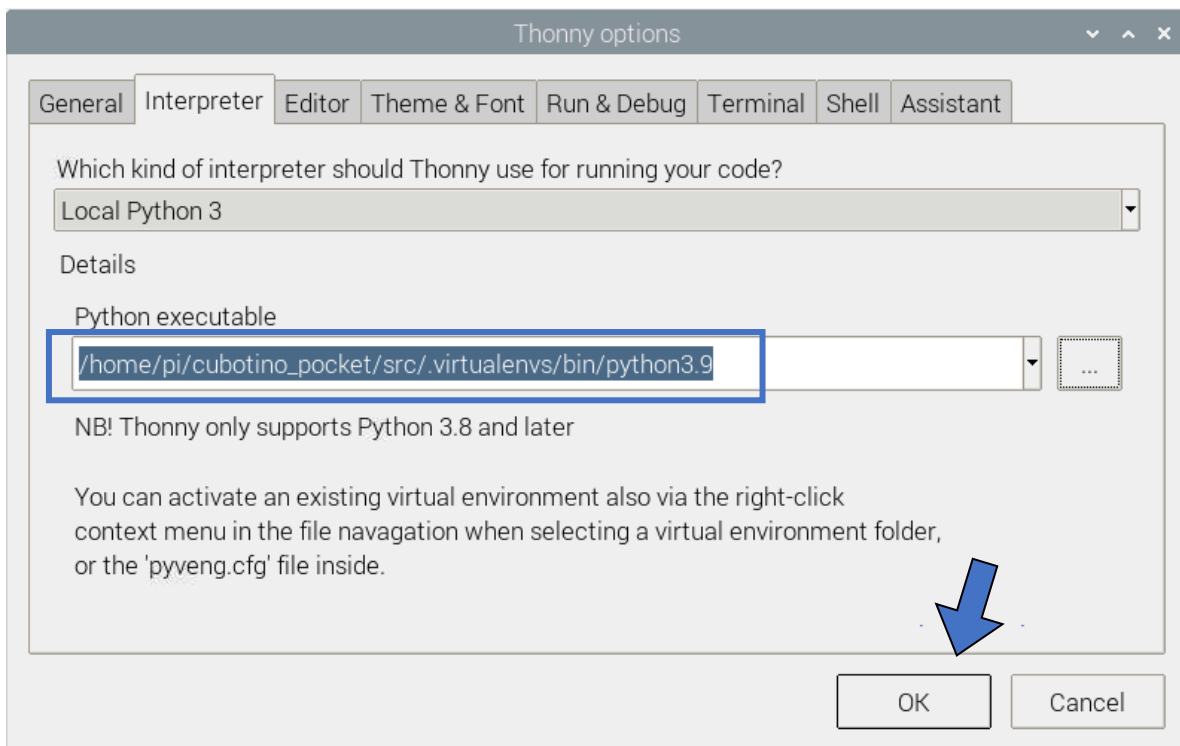
Select Home, .virtualenvs should appears (Note: all folders and files starting with a dot are hidden type)
The path should be “/home/pi/cubotino_pocket/src/.virtualenvs”



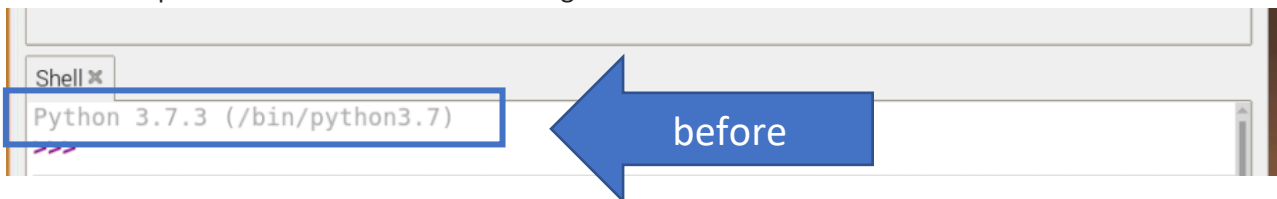
Select python3 from the path: ‘/home/pi/cubotino_pocket/src/.virtualenvs/bin/python3’ (or python 3.SomeValues) and confirm.



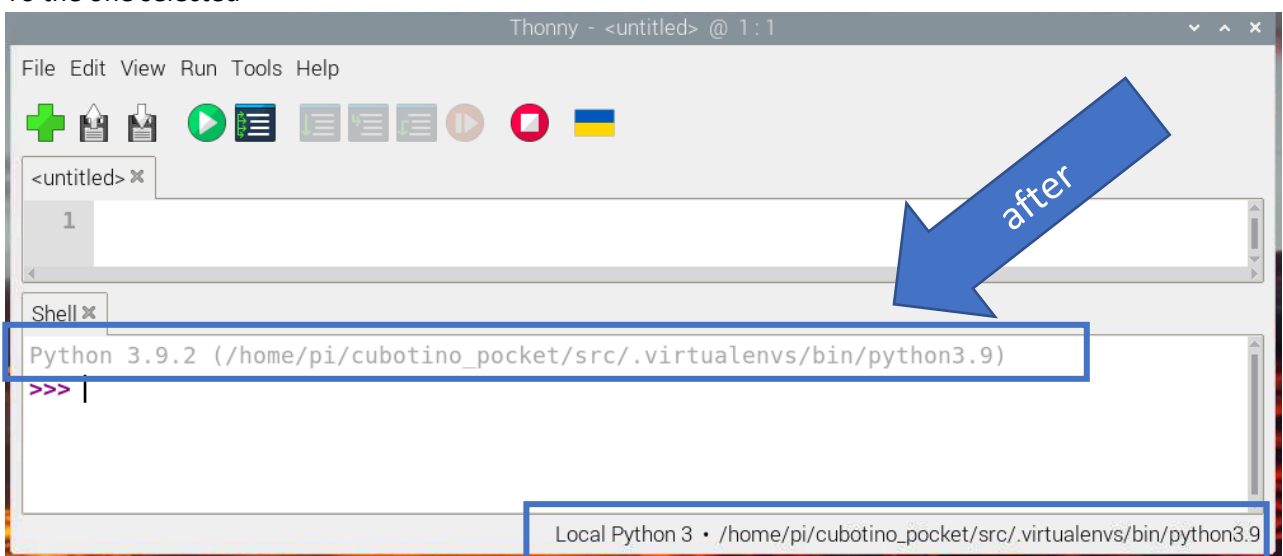
Confirm one more time.



And the interpreter indicated on the Shell changes from:



To the one selected



Notes, to get this change proposed as default:

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

4. Fine tuning servos via CLI

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.

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_pocket/src`.
4. activate the venv: `(source .virtualenvs/bin/activate)`; Attention to the dot in front of virtualenvs.
5. run the script: `python Cubotino_P_servos.py --tune`; Attention to the space in between '-'

```

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

```

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

```

File Edit Tabs Help

Code to check the individual servos positions.
It is possible to recall the values stored at the Cubotino_m_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_m_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 '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: █

```

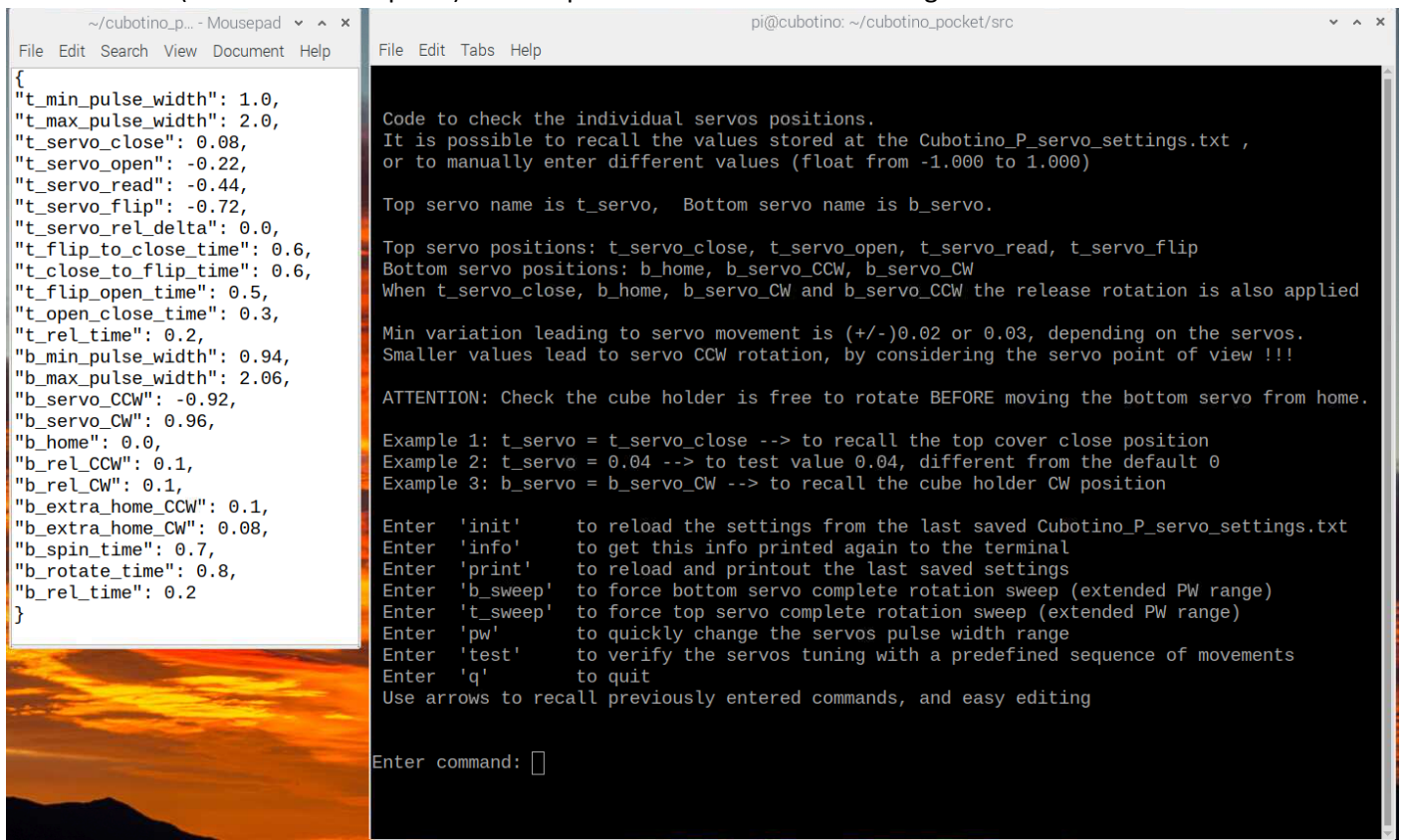
7. it is possible to recall the settings stored at Cubotino_P_servo_settings.txt as well as to enter different target values (value should be a float ranging from -1.000 to 1.000).

8. After the 'Enter command:' type the below commands to test the servos positions:
- `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)
9. 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: █
```

10. Once the position(s) are satisfactory, edit the `Cubotino_P_servo_settings.txt` and save the file to apply the new settings; It is suggested to keep open the `Cubotino_P_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.



```
~/cubotino_p... - Mousepad  x
File Edit Search View Document Help
{
  "t_min_pulse_width": 1.0,
  "t_max_pulse_width": 2.0,
  "t_servo_close": 0.08,
  "t_servo_open": -0.22,
  "t_servo_read": -0.44,
  "t_servo_flip": -0.72,
  "t_servo_rel_delta": 0.0,
  "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,
  "t_rel_time": 0.2,
  "b_min_pulse_width": 0.94,
  "b_max_pulse_width": 2.06,
  "b_servo_CCW": -0.92,
  "b_servo_CW": 0.96,
  "b_home": 0.0,
  "b_rel_CCW": 0.1,
  "b_rel_CW": 0.1,
  "b_extra_home_CCW": 0.1,
  "b_extra_home_CW": 0.08,
  "b_spin_time": 0.7,
  "b_rotate_time": 0.8,
  "b_rel_time": 0.2
}

pi@cubotino: ~/cubotino_pocket/src
File Edit Tabs Help

Code to check the individual servos positions.
It is possible to recall the values stored at the Cubotino_P_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_P_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: █
```

To verify if everything goes well:

11. If you type *info* , instead of a command after the 'Enter command:', guidance is printed again on the screen.
12. If you type *init* , instead of a command after the 'Enter command:', the new settings from Cubotino_P_servo_settings.txt are re-applied to the robot.
13. If you type *print* , instead of a command after the 'Enter command:' the latest values saved at Cubotino_P_servo_settings.txt are printed to the screen.
14. 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.
15. run the script *python Cubotino_P_servos.py* , without any argument, to test the robot maneuvering 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.
16. 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 maneuvers a cube and watch it back to see in which position the misalignment is generated.
17. Re-adjust the setting for the position that leads to the cube layers misalignment.

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 a decrease in 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_P_servo_settings.txt file, use your defined values to better cope with your robot characteristics.

5. Parameters and settings

Parameters that are more likely to differ on each system, are into two json files: *Cubotino_P_settings_default.txt* and *Cubotino_P_servo_settings_default.txt*

Note: These two files will be automatically duplicated on your local repository, and renamed as: *Cubotino_P_settings.txt* and *Cubotino_P_servo_settings.txt*, to prevent overwriting from “git pull”.

On below tables are listed these parameters, with the proposed value to start the tuning (Default value), the values that work on my Cubotino as reference (AF value), with some basic information on the parameter meaning.

Highlighted in yellow the default settings that differ from mine.

Cubotino_P_settings_default.txt (becoming *Cubotino_P_settings.txt* in your local repository) and *Cubotino_P_settings_AF.txt* as reference)

Parameter (dict key)	Default value	AF value	Data type	Info
frameless_cube	auto	auto	string	Set the facelets edge detection according to the cube. Options are: 'false', 'true' and 'auto'. Cubotino micro works best with the “auto” mode, because of the hole presence in a couple of facelets.
disp_width	130	130	Int	Display width (in pixels)
disp_heigh	160	160	Int	Display height (in pixels). The variable name has a typo, yet this has become its name....
disp_offsetL	0	-2	Int	Display offset on width Left (in pixels)
disp_offsetT	0	0	int	Display offset on height Top (in pixels)
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	(only used when OS10) PiCamera sensor mode: 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 $\text{abs}(1-\text{kl})$ 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


Cubotino_P_settings_default.txt (becoming **Cubotino_P_settings.txt** in your local repository) and **Cubotino_P_settings_AF.txt** as reference)

expo_shift	-0.6	-1.0	float	Changes the camera exposure time target (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.
x_l	0	50	Int	Image cropping at the left, before warping (in pixels). This is meant to remove 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	84	Int	Image cropping at the right, before warping (in pixels). This is meant to remove 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	44	Int	Image cropping at the top, before warping (in pixels). This is meant to remove 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	86	int	Image cropping at the bottom, before warping (in pixels). This is meant to remove 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	2	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	Facelet 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	Facelet 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	Facelet 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).

Cubotino_P_settings_default.txt (becoming **Cubotino_P_settings.txt** in your local repository) and **Cubotino_P_settings_AF.txt** as reference)

Parameter (dict key)	Default value	AF value	Data type	Info
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	0.05	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 LED at Top_cover. Range from 0 (no PWM) to 1 (PWM=100%).
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 pressing a 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 pressing a 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	true	true	string	Top_cover auto closing at shut down. Options are 'false' and 'true'.
vnc_delay	0.5	0.5	float	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_P_settings.txt belongs to an older version (absence of this parameter), 0.5secs will be used as default.

Cubotino_P_settings_default.txt (becoming **Cubotino_P_settings.txt** in your local repository) and **Cubotino_P_settings_AF.txt** as reference)

Parameter (dict key)	Default value	AF value	Data type	Info
built_by	"" (empty string)	"" (empty string)	string	<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_P_display.py</i> to check the result.</p>
built_by_x	25	25	int	From rev 0.4: X coordinate for the maker's name, for good cantering.
built_by_fs	22	22	int	From rev 0.4: Font size for the maker's name.
fcs_delay	3	3	float	<p>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.</p> <p>The fix coordinates are automatically generated, on cycles when the fcs are not used. The coordinates are saved on a text file.</p>

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_P_servo_settings_default.txt (becoming **Cubotino_P_servo_settings.txt** in your local repository) and **Cubotino_P_servo_settings_AF.txt** as reference)

Parameter (dict key)	Default value	AF value	Data type	Info
t_min_pulse_width	0.50	0.50	float	Min pulse width, in ms of the used top servo. Most of the servos accepts a slightly extended value (<0.5)
t_max_pulse_width	2.50	2.50	float	Max pulse width, in ms, of the used top servo. Most of the servo accepts a slightly extended value (>2.5)
t_servo_close	0.00	0.12	float	"Angle" for Top_cover to constrain the top and mid cube layers
t_servo_open	0.00	-0.22	float	"Angle" for Top_cover not constraining the cube and Cube_holder
t_servo_read	0.00	-0.48	float	"Angle" for Top_cover for PiCamera reading. Lifter almost touching the bottom cube face
t_servo_flip	0.00	-0.82	float	"Angle" for Top_cover to flip the cube (~2 cube layers)
t_servo_rel_delta	0.00	0.00	float	Delta "angle" for Top_cover to retract after closing
t_flip_to_close_time	1.0	0.14	float	Time for t_servo to move from Flip to Close position
t_close_to_flip_time	1.0	0.22	float	Time for t_servo to move from Flip to Close position
t_flip_open_time	1.0	0.30	float	Time for t_servo to move from Flip to Open position and vice versa
t_open_close_time	1.0	0.10	float	Time for t_servo to move from Close to Open position and vice versa
t_rel_time	0.00	0.00	float	Time for t_servo to move to release the tension from Close position (t_servo_rel_delta movement)

Cubotino_P_servo_settings_default.txt (becoming **Cubotino_P_servo_settings.txt** in your local repository) and **Cubotino_P_servo_settings_AF.txt** as reference)

b_min_pulse_width	0.50	0.42	float	Min pulse width, in ms, of the used bottom servo. Most of the servos accepts a slightly extended value (<0.5)
b_max_pulse_width	2.50	2.60	float	Max pulse width, in ms, of the used bottom servo. Most of the servo accepts a slightly extended value (>2.5)
b_servo_CCW	-1.00	-0.98	float	"Angle" for the Cube_holder at ~90° CCW from Home. CCW is from motor point of view
b_servo_CW	1.00	1.00	float	"Angle" for the Cube_holder at ~90° CW from Home. CW is from motor point of view
b_home	0.00	0.00	float	"Angle" for the Cube_holder in between CW and CCW positions
b_rel_CCW	0.06	0.08	float	"Delta angle" for the Cube_holder to retract from CCW
b_rel_CW	0.06	0.06	float	"Delta angle" for the Cube_holder to retract from CW
b_extra_home_CCW	0.06	0.08	float	"Extra angle" Cube_holder does before stopping Home, when rotating from CCW. This does not apply when spinning to Home
b_extra_home_CW	0.06	0.08	float	"Extra angle" Cube_holder does before stopping Home, when rotating from CW. This does not apply when spinning to Home
b_spin_time	1.0	0.14	float	Time for the Cube_holder to spin ~90° (cube not constrained)
b_rotate_time	1.0	0.24	float	Time for the Cube_holder to rotate ~90° (cube constrained)
b_rel_time	0.20	0.00	float	Time for the Cube_holder to rotate back, at CCW, CW and Home

Appendix 2

THE CUBOTINO PROJECT

1. Introduction

To explain why I've started the CUBOTino 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.

2. Project scope

The CUBOTino robot series wants to be affordable, to attract more people and especially students into robotics and programming.

The overall CUBOTino idea is to build a scalable robot, based on a minimalist base version; This clearly doesn't apply for the micro version.

CUBOTino Pocket is a derived model from the CUBOTino Autonomous.

This version wasn't on my thoughts, until I got triggered by a teacher: He aims to capture children's interest in Rubik's cube and robotics by starting with a simpler version of the cube, with the hope of maintaining their enthusiasm for these subjects. Isn't it a great trigger?

Project targets for CUBOTino robot series:

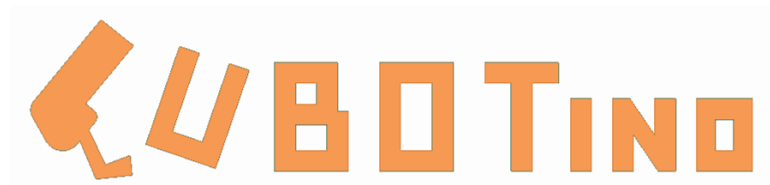
- The Base version must be as cheap as possible.
- The mechanical part should be the same for all the versions; Cubotino Pocked has 7 parts differing from the Autonomous version.
- The robot should be scalable (in automation, and consequently in complexity/materials cost).
- The robot should not require changes to the cube for gripping.
- Compact design.
- Fully 3D printable.
- How to make it instructions and files.
- Learning & Fun 😊

3. Robot name

I've started the Cubotino project s 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 **CU**be, **roBOT** and **ino** (INO 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:



4. 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 maneuver 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, for a 3x3x3 Rubik's cube of 30mm size. 	Finalized (March 2023) Link
Pocket	Autonomous	<ul style="list-style-type: none"> This essentially is the Top version adapted for a 2x2x2 (Rubik's cube of ~50mm size). 	Finalized (January 2024) Link

Notes for the Medium version:

The cube status detection method I've tried for the Medium version, is via 9 colors 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.

5. High level info

1. The robot is based on a Raspberry Pi Zero2WH with a 16Gb microSD.
2. A PiCamera (ver. 1.3) is used to detect the cube status; Camera is placed at an angle with respect to the cube.
3. Python CV2 library is used for the computer vision part.
4. A led is used to reduce the influence from the ambient light conditions.
5. A small display provides feedback on the robot task/progress.
6. All coded in Python.
7. This robot works with Rubik's cube 2x2x2 with size of ca 50mm.
8. Nice information about the Pocket Cube at: https://en.wikipedia.org/wiki/Pocket_Cube
9. Cube notations are from David Singmaster, limited to the uppercase: https://en.wikipedia.org/wiki/Rubik%27s_Cube#Move_notation

10. Cube's orientation considers the Western color scheme (<https://ruwix.com/the-rubiks-cube/japanese-western-color-schemes/>):

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

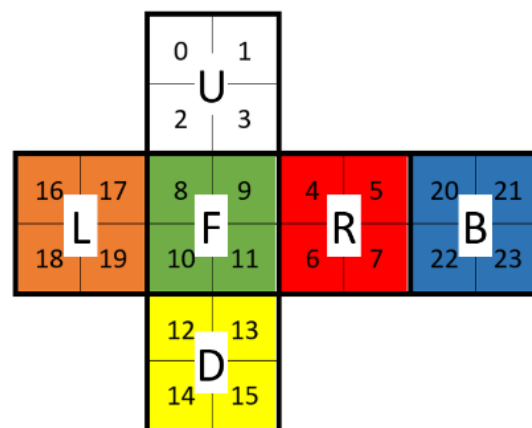
Cubers who use this color scheme usually start solving the Rubik's Cube with the white face and finish with the yellow.

This color scheme is also called **Minus Yellow** because if you add or extract yellow from any side you get its opposite.

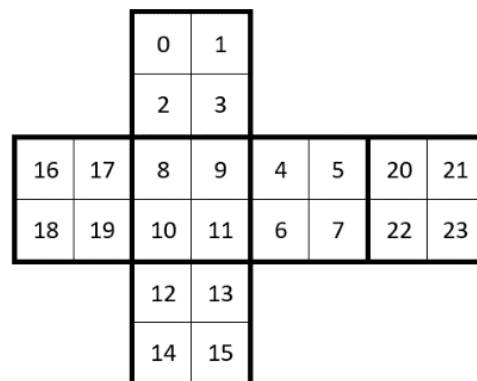
White + yellow = yellow

red + yellow = orange

blue + yellow = green



11. Cube solver used is the Hegbert Kociemba "Rubiks2x2x2-OptimalSolver". All optimal (=shortest) solving maneuvers are computed. The solver only uses moves of the U, R and F. <https://github.com/hkociemba/Rubiks2x2x2-OptimalSolver>
12. The robot also considers the move of D, B and L faces to reduce the number of robot movements.
13. When rotations are expressed 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.
14. The robot detects the cube status on cubes with and without the black frame around the facelets.
15. In case the facelets' edges are difficult to get detected, after a predefined time a Fix Coordinates System (FCS) is used as location reference to pick the facelets' color info. The FCS is automatically generated, and updated, each time a cube is successfully scanned via edges detection technic.
16. Cube's sides follow the URFDLB order, and facelets are progressively numbered according to that order (sketch at side). Facelets numbers are largely used as key of the dictionaries.



6. Construction

The Pocket version has a few parts scaled down from the CUBOTino Autonomous, due to the smaller cube size.

Overall targets remain those for Cubotino Autonomous:

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

Construction principles:

1. The inclined cube-holder is inspired to Hans construction [Tilted Twister 2.0 – LEGO Mindstorms Rubik’s Cube solver – YouTube](#);
This is a clever concept, as it allows to flip the cube around one of the horizontal axes by forcing a relatively small angle change (about 30 degrees, over the 20 degrees of the starting cube holder angle); Once the cube center of gravity is moved beyond the foothold, the cube falls on the following face thanks to the gravity force.
Overall, it allows to flip the cube via a relatively small and inexpensive movement.
2. The Top-cover, combined with the cube Lifter, is the logical simplification step from my previous robot:
 - The Top-cover provides a constrainer for cube layer rotation, further than suspending 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 it possible to pursue the needed geometries, also complex shapes.
 - The biggest parts can still be printed on a relatively small plate (min plate 100x100 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.

7. 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 Zero2 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)

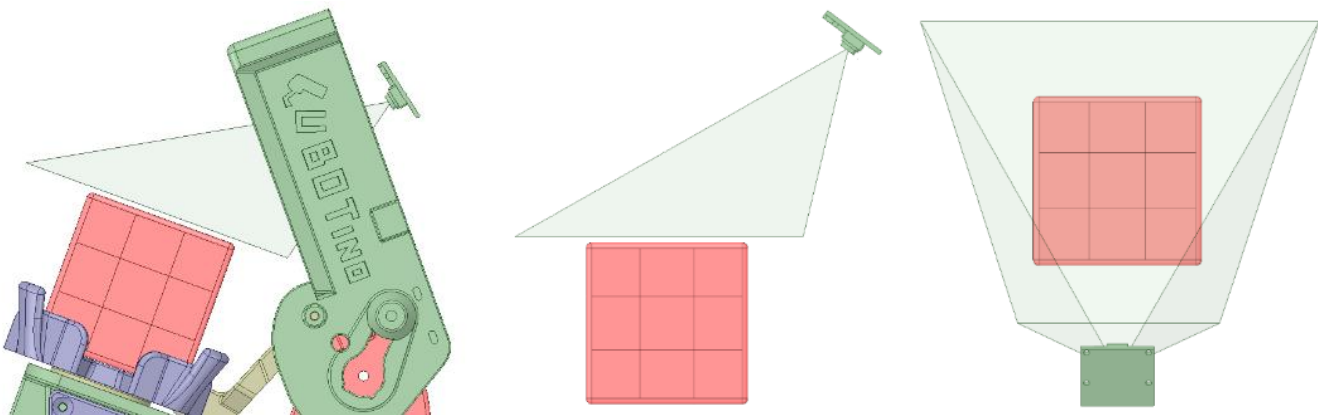
In which the python script '**Cubotino_P.py**' is responsible for the interaction with these elements.

Below listed aspects, are presented on the next pages:

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

Colors detection strategy is described in 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 color uniformity, therefore the robustness to always read (and assign) the correct colors.
2. The cube facelets have a relevant perspective; This makes it more difficult to evaluate if a detected contour is really a facelets, by evaluating if fitting a square shape.

To solve the above problems:

1. A controllable LED has been placed close to the camera; This mitigates 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:

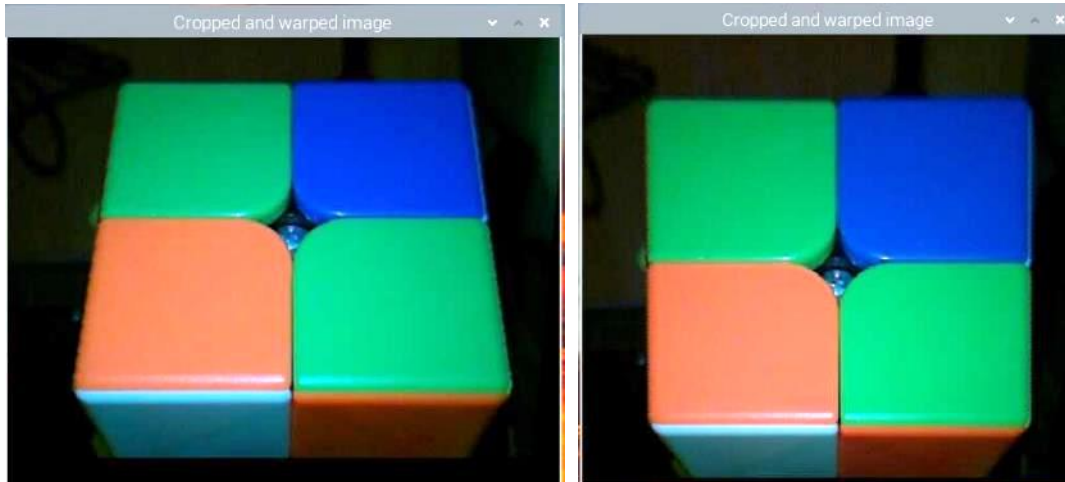
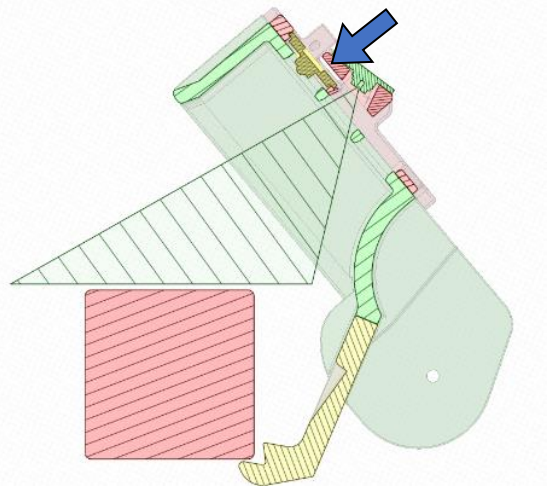


Image warping requires some computation power, but it does not affect the overall timing.

B. Taking consistent images

This is a crucial aspect for proper color analysis.

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

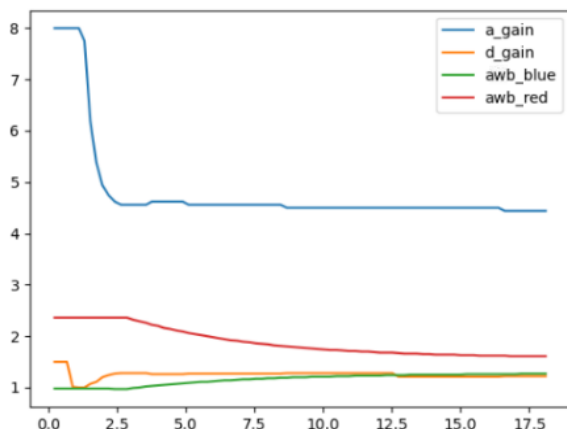


the LED power is controllable by the MCU.
The power level can be set with a parameter
at the Cubotino_m_settings.txt.
Parameter ranges from 0 (=0%) to 1(=100%)

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

To cover these situations, a so called 'warm-up' function is implemented in Cubotino_P.py script: Once all these parameters are within 2% from the average value, 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 the 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 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-551cceb80f0>

1. The warped image is converted to grey scale: `grey = 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(grey, ...`
 - on frameless cube type (frameless_cube set 'true'): `blurred = cv2.bilateralFilter(grey, ...,`
3. The de-noised grayscale image is analyzed with a Canny filter; This function transforms the image to binary, assigning 1 (white) the pixels detected as edges: `canny = cv2.Canny(blurred....`
 - when frameless_cube is set 'auto' the canny filter is applied on both burred 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 analyzed 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 analyzed 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 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 4 contours, according to their center coordinates.
 - f) Quantity of contours left, after discharging those not ok.
5. The first 4 contours, passing through this process, are then used as masks; These masks are applied on the colored warped image, as guidance for the facelets position.
In case the `frameless_cube` is set 'true' or 'auto', as soon 2 (or 3) contours are detected the remaining 2 (or 1) are estimated for their position.
6. The 'accepted' contours are plot over the colored warped image, as visual feedback.

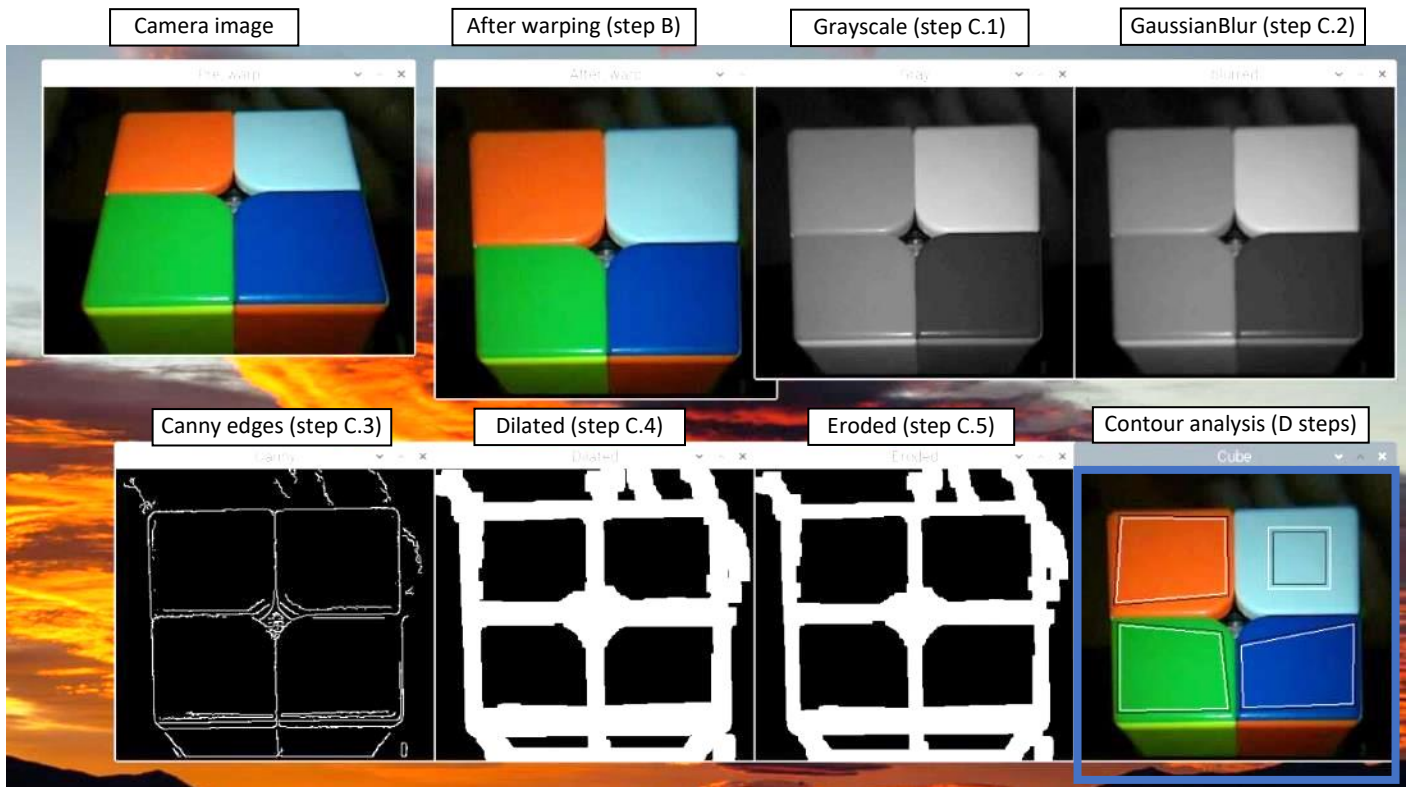
E. Colors retrieved:

On each facelets, are retrieved 2 main info:

1. Average BGR, for a portion of the facelets around the detected contour center.
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_P.py --cv_wow` , a FHD screen/setting needed)

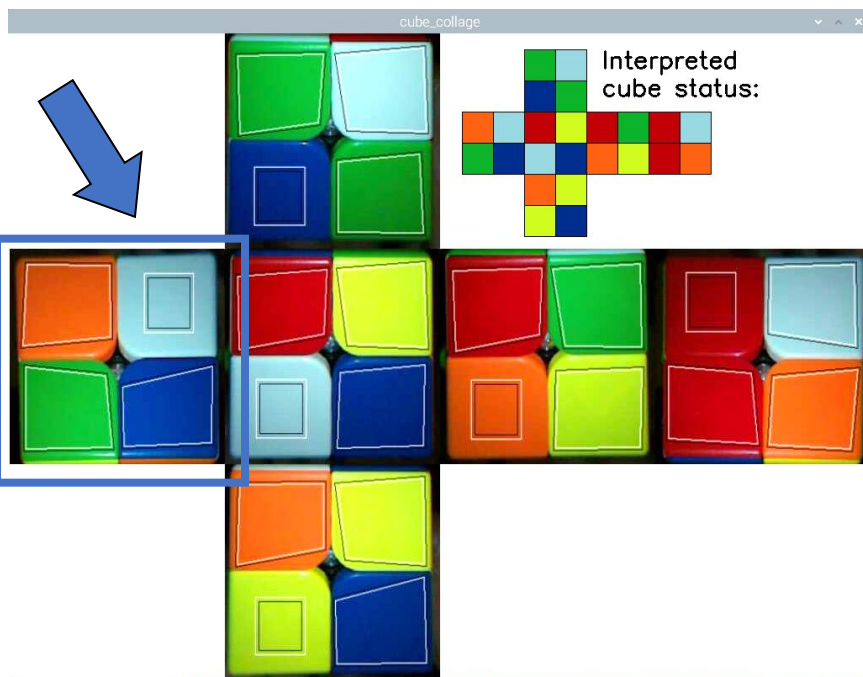


Not all images are oriented as per user point of view: Sides UBDP are 180deg rotated.

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, is 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 is added, and the whole saved to the microSD.



When individual cube faces are presented on screen, are always oriented as per camera point of view. The L side highlighted on the cube sketch at side, is the cube face image that was under analysis on the above example.

F. Fix Coordinates System (FCS):

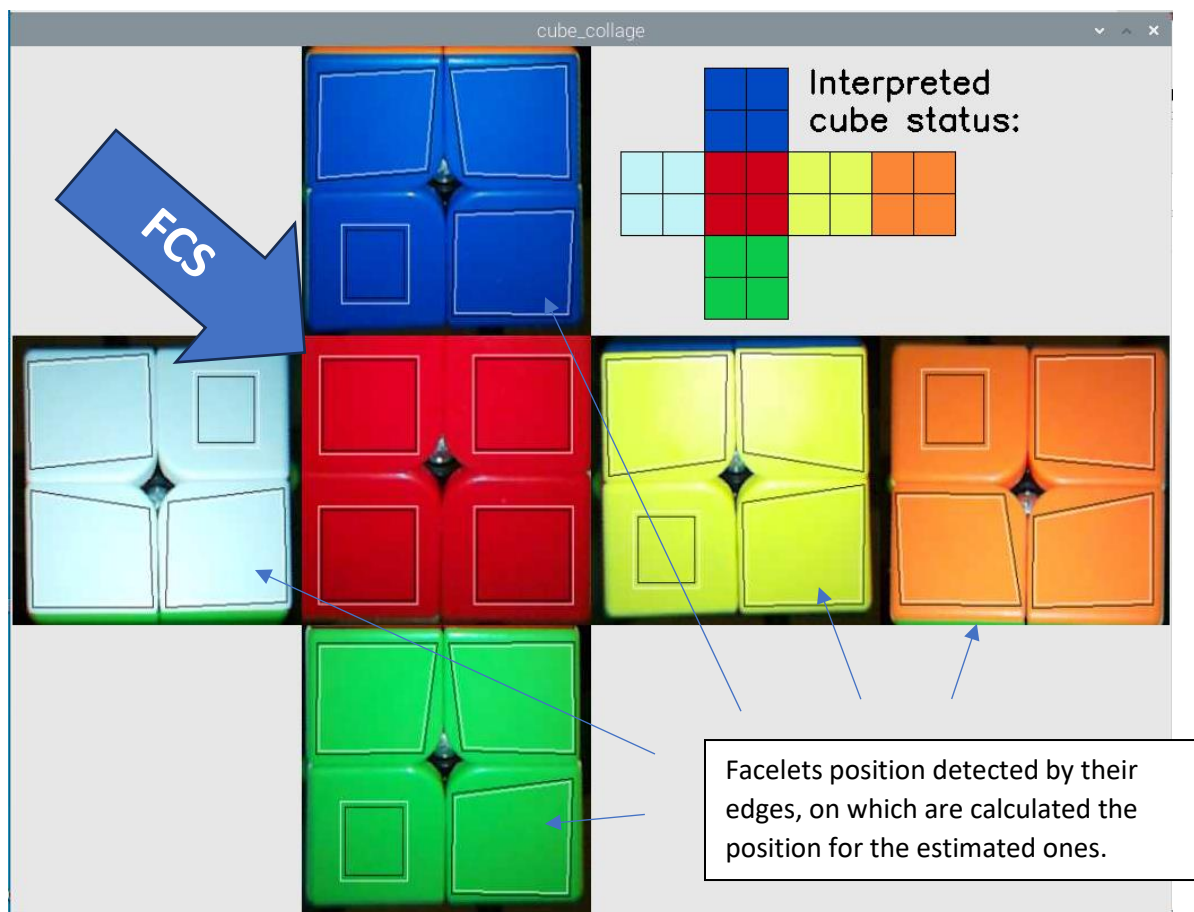
The facelets position are also estimated via an additional method (FCS), below the main aspects:

- The FCS algorithm is started:
 - After the `fcs_delay` time elapses (priority is given to the edge detection method).
 - On any cube's face.
- As the wording suggests, 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_P_coordinates.txt` file.
- The coordinates are loaded from the file once the `Cubotino_P.py` is launched.
- Coordinates are averaged from the last "n" values.
- When the `Cubotino_P_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.

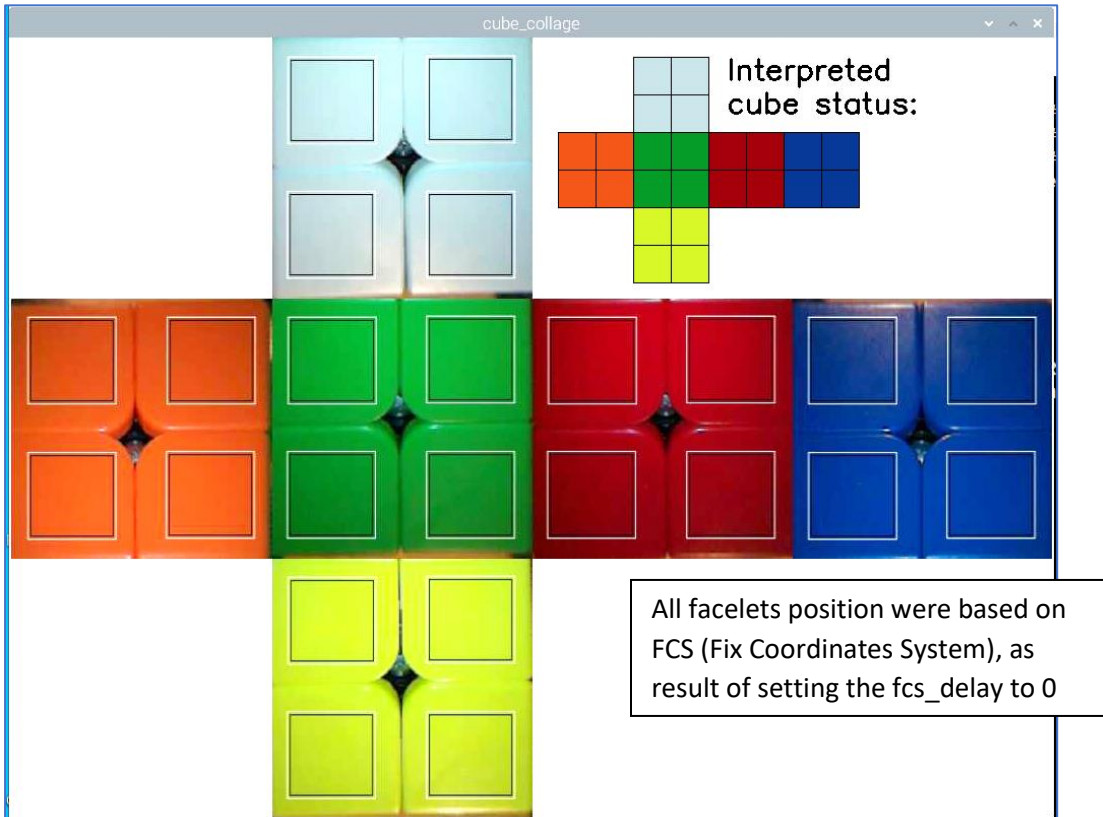
On the below example, the facelets detection of the F (front) face were done via the FCS.

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

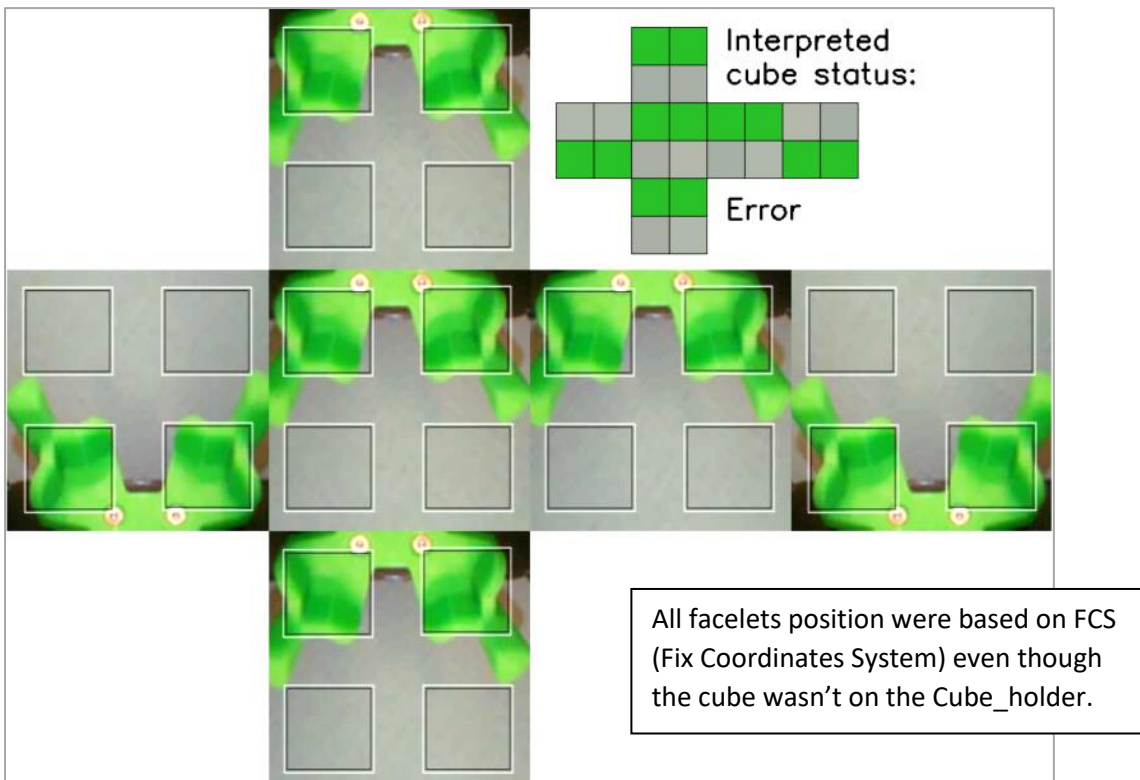
On all the other faces most of facelets are detected via their edges, and the remaining have estimated position.



If the `fcs_delay` is set to 0 (zero), it gives priority 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.



To be mentioned, the FCS prevents the code from stopping in case of cube absence, and an error will be returned:



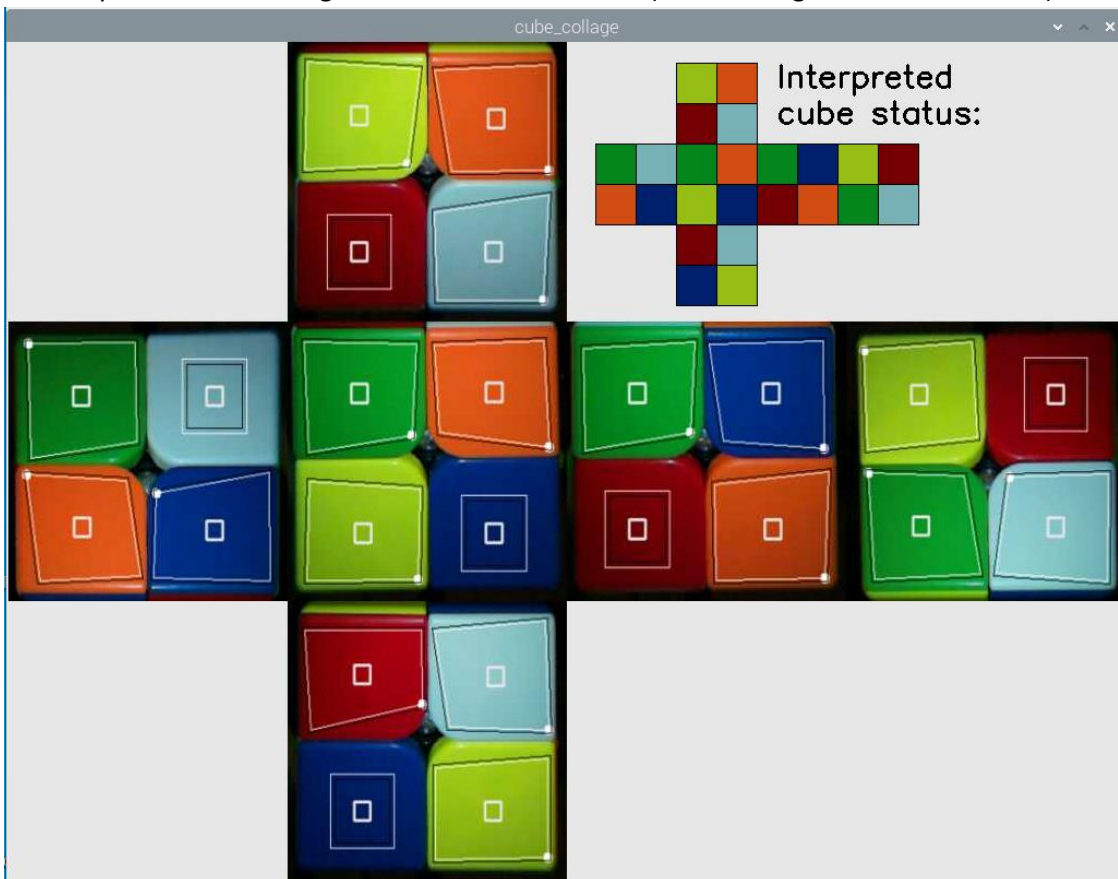
8. Color's detection strategy

Info: Color's detection strategy differs from the 3x3x3 Rubik's cube as there aren't the fix centers facelets, more about this later this chapter.

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

Based on the identified contours:

- a. The outer one, in black on below picture, shows the simplified contour retrieved by the edge analysis; This analysis is used to find the 4 facelets per each cube, and to know the contour center coordinates.
- b. The inner one, in white on below picture, depicts a smaller square area centered on the outer contour; This smaller area is used to measure the BGR average value (also converted to HSV), used for the color interpretation according to the BGR color distance (or according to the Hue distance).



2. Once all the 24 facelets colors are retrieved, a clustering function is used to get the 6 dominant colors. These colors are the ones with the least color distance from the 4 facelets of that specific color's group. These colors are then used as reference.
3. Facelets color interpretation is made by comparing the average BGR color of each facelet, in comparison with each of the 6 dominant colors, and the color decision is based on the smallest color distance. The Euclidian distance of BGR per each facelet is calculated toward the 6 dominant colors.
4. In case the BGR color analysis delivers an incoherent cube status, a similar approach is used with the HSV color space. In this case the white facelets are initially identified by their large Saturation – Value difference, while the colored facelets are analyzed via the Hue distance from the 5 dominant Hue.

5. Facelets position estimation

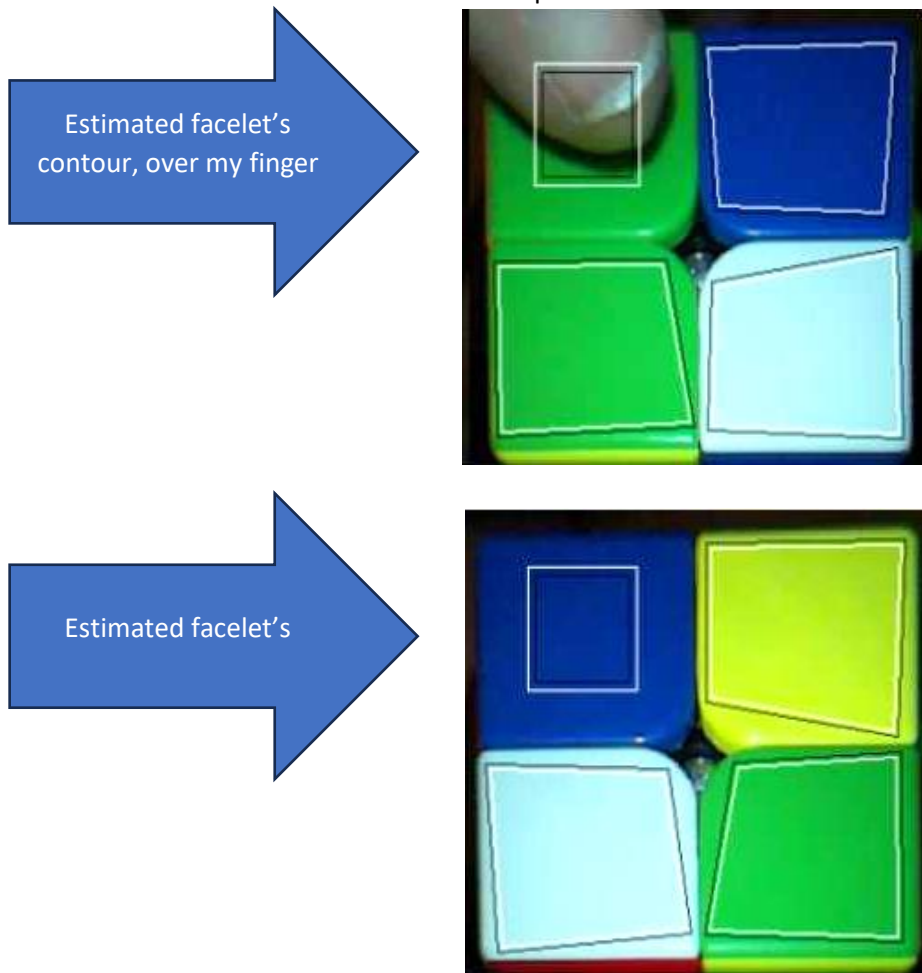
The “frameless” parameters defines whether the algorithm searches the contour for all the 4 facelets or for at least 2 of them.

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.

When frameless_cube is set ‘true’ or ‘auto’, as soon as there are at least 2 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” facelets, and later leading to wrong colour detection.

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



9. Optimal solution

This robot uses the Kociemba solver to get the solutions (cube maneuvers), based on the detected cube status (permutation).

The Kociemba solver, for the 2x2x2 Rubik's cube, delivers the optimal solution for the given cube status.

“Optimal solution” means it has the smallest quantity of movements to reach the solved cube; this could be:

- One solution.
- A list of solutions, all having the same quantity of movements (depth) from the solved cube. For some cube statuses, there are up to 694 alternative solutions all having the same quantity of movements.

The optimal solutions, from Kociemba solver, have rotations of U, R and F faces as the DBL corner is considered fix.

This is a very logical choice for humans, not for CUBOTino:

- CUBOTino is slower, in 80% of the cases, by only using U, R and F faces rotations.
- CUBOTino is always faster at the first rotation, by having a U, R or F face rotation.

Strategy used in CUBOTino Pocket:

A 2x2x2 Rubik's cube has a fundamental property: Rotating a face or the opposite one delivers the exact result.

To be a truly “optimal solver”, DBL faces rotation are considered from the 2nd rotation onward.

The ‘URF’ optimal solution, or solutions, received by the Kociemba solver are enriched with the equivalent DBL, and the mix between URL and DBL.

All these generated solutions are sent to the robot solver, that returns the needed robot's movements and the estimated time for each solution: The solution with the fastest predicted time is then chosen.

To estimate the solving time, the robot uses the timers set for all the different types of movements the robot does; The prediction does not only find the fastest solution, but it gives a reasonable estimation of the solving time.

On Raspberry Pi Zero2, for a random cube status, this process takes an average of about 125 milliseconds with solver set as Quarter-Turn Metric (it reduces to ca 50ms in Full-Turn Metric, due to much lower quantity of alternative solutions.)

Investing process time, to find a faster solution, makes a lot of sense on a rather slow robot.

On about 80% of the cases, the fastest solution for CUBOTino Pocket requires D, B and L faces rotations: This

Worst computational cases:

There are 24 cases (permutations) in depth 14, for which the Kociemba delivers 694 alternative solutions (QTM).

The robot solver analyzes a total of 4858 solutions (694 having URF faces rotations + 4164 also having DBL faces rotations), to select the fastest solution.

A Raspberry Pi Zero2 takes about 9.4 seconds to analyze the 4858 solutions, to select the fastest one.

Despite such a long analysis time, it is still worth proceeding in this way.

For these 24 “longest cases”:

- The fastest solution found (predicted servos time) is 6 to 17 seconds faster than the first solution delivered by the Kociemba solver.
- In 70% of these cases, the fastest solution brings more benefit (time reduction) compared to the time spent to analyze all the solutions.

The benefit will increase for robots not well tuned for speed (longer servos timers).

10. 2x2x2 Rubik's cube and robot efficiency

In this chapter are described the main characteristics of the 2x2x2 Rubik's cube and the robot (in)efficiency 😊

More info for the cube at https://en.wikipedia.org/wiki/Pocket_Cube

The 2x2x2 version is a much easier to solve puzzle than the 3x3x3, as it has way lower quantity of possible permutations:

Cube type	Formula	Permutations
2x2x2	$\frac{8! \times 3^7}{24} = 7! \times 3^6$	3'674'160
3x3x3	$8! \times 3^7 \times \frac{12!}{2} \times 2^{11}$	43'252'003'274'489'856'000

The **depth** of a cube indicates how many movements are necessary to reach the solved status, by using the shortest possible path. The higher the depth, the more a cube is scrambled.

To describe the minimum movements (depth) to solve a 2x2x2 Rubik's cube, we must specify the metric:

- **Quarter-turn** means that each 90degrees rotation of a face counts as one movement; When the same face must be rotated twice (i.e., U2), it is considered as two movements.
- **Half-turn** means that a 90degrees or 180degrees rotation of a face is considered as one movement.

Below table shows how many permutations per depth level, according to the used metrics.

Permutations % indicates the size per depth level compares to the total.

2x2x2 Rubik's cube Depth	Quarter-turn metric		Half-turn metric	
	Permutations	Permutations (%)	Permutations	Permutations (%)
0	1	0.000027%	1	0.000027%
1	6	0.00016%	9	0.00024%
2	27	0.00073%	54	0.0015%
3	120	0.0033%	321	0.0087%
4	534	0.015%	1847	0.050%
5	2256	0.061%	9992	0.27%
6	8969	0.24%	50136	1.36%
7	33058	0.90%	227536	6.19%
8	114149	3.11%	870072	23.68%
9	360508	9.81%	1887748	51.38%
10	930588	25.33%	623800	16.98%
11	1350852	36.77%	2644	0.072%
12	782536	21.3%	0	0%
13	90280	2.46%	0	0%
14	276	0.0075%	0	0%

Takeout:

Permutation % indicates, therefore, the likelihood of a random scrambled cube falling in the different depth levels. For more than 93% of the cases, the 2x2x2 Rubik's cube will be between 9 to 12 quarter-turn rotations away from the solved status.

Whatever scrambled 2x2x2 Rubik's cube, it could be solved with 14 (quarter-turns) rotations or less.

Cubotino Pocket has a couple of virtual solvers, and one uses random generated permutations. This is a very good way to gain knowledge and a grip on what is happening under the hood.

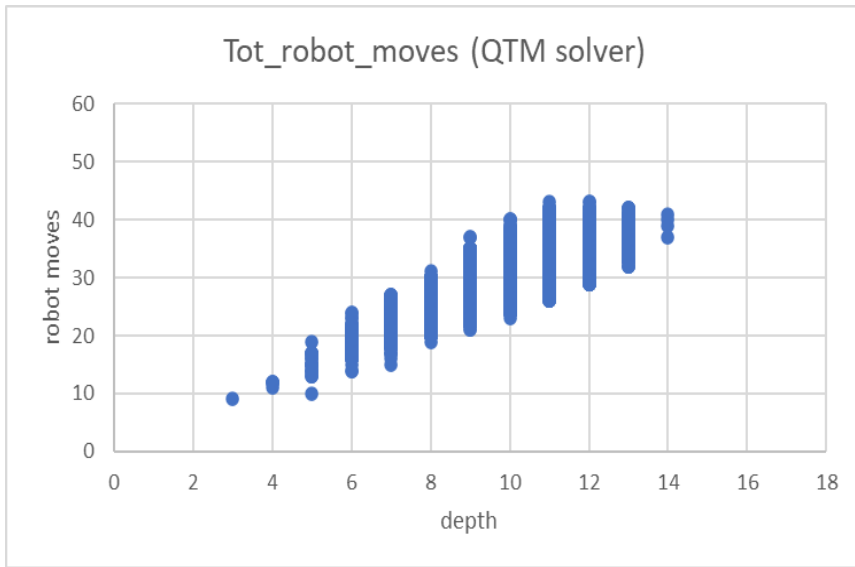
This virtual solver has been instrumental to realize:

1. The Kociemba solver delivers the optimal solution in FTM (Full Turn Metric).
2. The optimal FTM solution can have more than 14 moves if executed by 90 degrees rotations. This is important as CUBOTino rotations are limited to 90 degrees.

Based on these learnings, and thanks to the invaluable help from Mr. Kociemba:

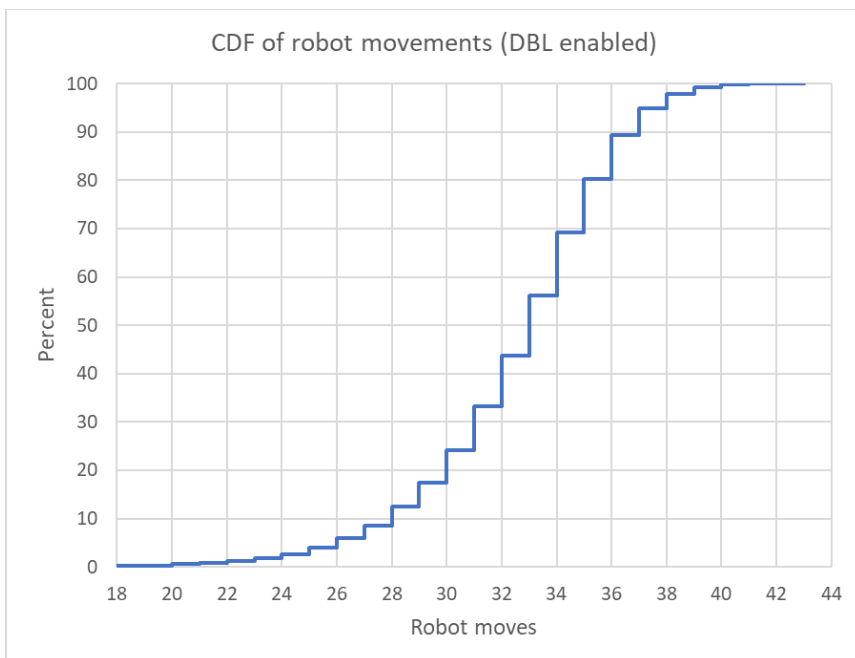
1. The Kociemba solver is now modified to deliver the best solution in QTM, a perfect match for CUBOTino.
2. The solutions from Kociemba's solver have ≤ 14 moves

Via the Cubotino Pocket virtual solver version, and 50'000 random generated permutations, the robot movements can be plot as function of the cube depth (how much a cube is scrambled):



From the above chart, we can appreciate that the robot movements are between 2 to 4 times those a human would require: This clearly indicates how much inefficient the robot is 😊

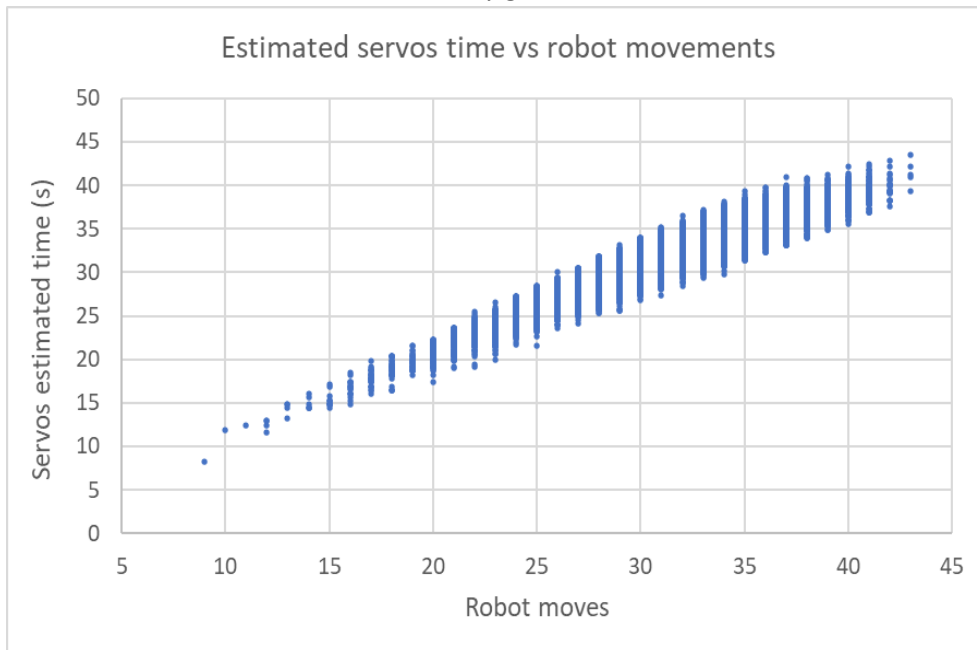
And again, out of 50'000 random generated permutations, we can have an idea on how the robot movements are distributed:



10% of the cases up to 28 movements
 50% of the cases up to 33 movements
 80% of the cases up to 35 movements

Depending on the robot's servos settings (timers) the estimated servos time has quite a linear relationship with the robot movements.

Below chart is based on 50'000 randomly generated cube statuses.



My robot tuning takes roughly 1 second per movement.

No doubt this robot being rather slow, and highly inefficient, but it still is rather easy to make 😊.

11. Robot solver moves

On this chapter it's explain the approach used to convert the cube solution maneuvers into robot moves; This part is embedded in the `Cubotino_P_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 $+90^\circ$), farther than flipping the cube around the L-R horizontal axis; This obviously requires an algorithm that prevents additional cube movements to those (many) that are strictly necessary.

The Kociemba solver provides a string with the rotations to be applied on the 6 faces, like `U2 F1 R3` etc. (I will refer to these three moves as examples 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. The above points are considered when the cube solution string is parsed, to generate a string with all the servo movements.

12. Python main scripts, high level info

1. *Cubotino_P.py* is the main python script on the robot, this script imports other custom files.
2. *Cubotino_P_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_P.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 imported from the 'solver2x2x2' sub-folder, under active folder.
5. The script uses a "tentative" approach, on some of the analysis:
 - a. (See Color detection strategy chapter for more info) When the image is analyzed, 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 facelet's requisites. This process ends when 4 facelets, all matching the filters criteria, are retrieved from a single image.
 - b. In case the facelets edges detection takes more than a predefined time, then the facelets positions (for that cube's face) will be based on fix coordinates. The reference coordinates are generated, and saved to an external text file, every time a coherent cube status is determined by finding the facelets via the detection of their edge detection.
 - c. In case the BGR color analysis delivers an incoherent cube status, the HSV color analysis takes place. These two methods are therefore complementary and covering different light situations.
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 returns the optimal solution, or a series of optimal solutions (all having the same quantity of face's rotations). The solver returns an error if the cube status is not coherent.
7. From cube solution to robot movements:

(see "Optimal solution" and "Robot solver moves" chapters for more info) Once the optima solution is chosen it is sent (in Singmaster notation) to *Cubotino_P_moves.py* that returns a (long) string with the sequence robot movements. Movements are Spin, Rotate, Flip. In the case of multiple solutions, from the Kociemba solver, all the solutions will be analyzed and the fastest one for the robot will be used. The time estimation function is in the *Cubotino_P_servos.py* (it uses the servos timers, and solutions, used to operate the servos).
8. From cube robot solution to robot movements:

Robot solution string, in Cubotino notation, is sent to *Cubotino_P_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., 20230311_141910
Screen	Indicates if a graphical desktop (i.e., VNC viewer) was connected. Possible strings are "screen" or "no screen" (when graphical data sharing, the robot takes longer time, especially during the cube status detection).
Flip2close	It is a robot setting that can be passed as parameter. Indicates if the Top_cover moves to close in one or two steps; With one step is faster, yet it requires a good cube layers alignment (good servo tuning).
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
SlowTime(s)	Time in seconds as additional delay after each servo movement
FCS	0 means that facelets on all faces were detected via their edges. Value > 0 indicates the times (faces) the Fix Coordinates System has been used.
URF/DBL	'URF' when the solutions uses only URF faces, otherwise 'DBL'
OS_ver	Operative System version (10 or 11).
CubeStatus	Cube status, in URF notation: i.e., RLRBLUDLURDUBFFDLFRFDUBB
CubeSolution	Cube solution string, in Singmaster notations: i.e., U1R2F1R1F1U3F3R1
CubeColors(BGR or HSV or BGR,HSV)	Dictionary with the average colors per each facelet, according to the color space of the winner detecting method; In case both the detecting methods have failed, then the average color returned by both the color spaces are reported

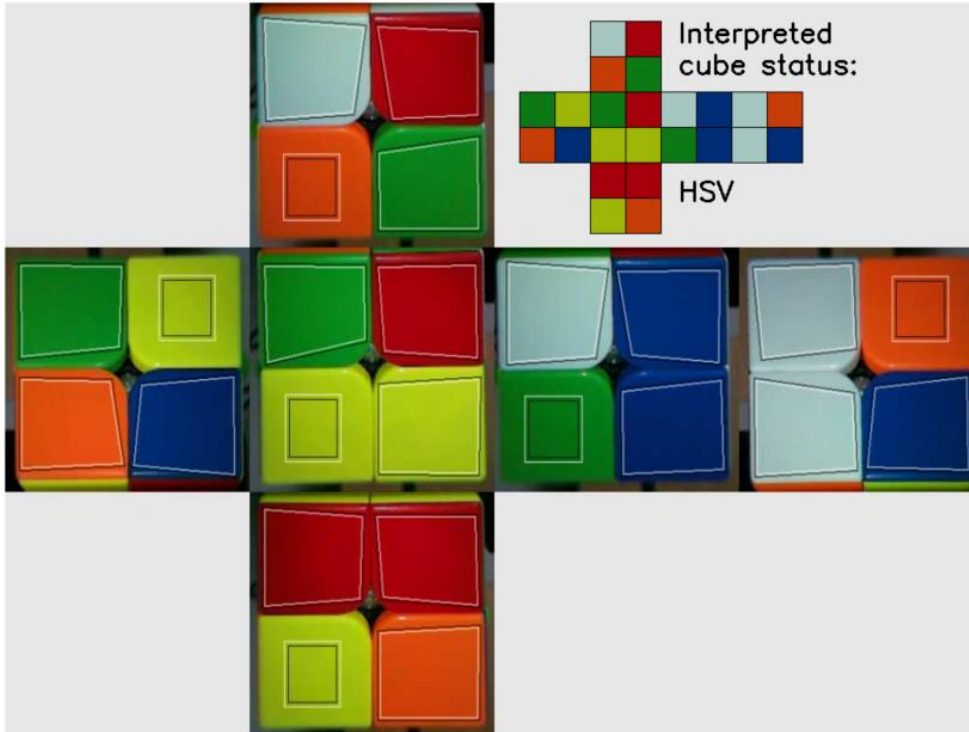
Notes:

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

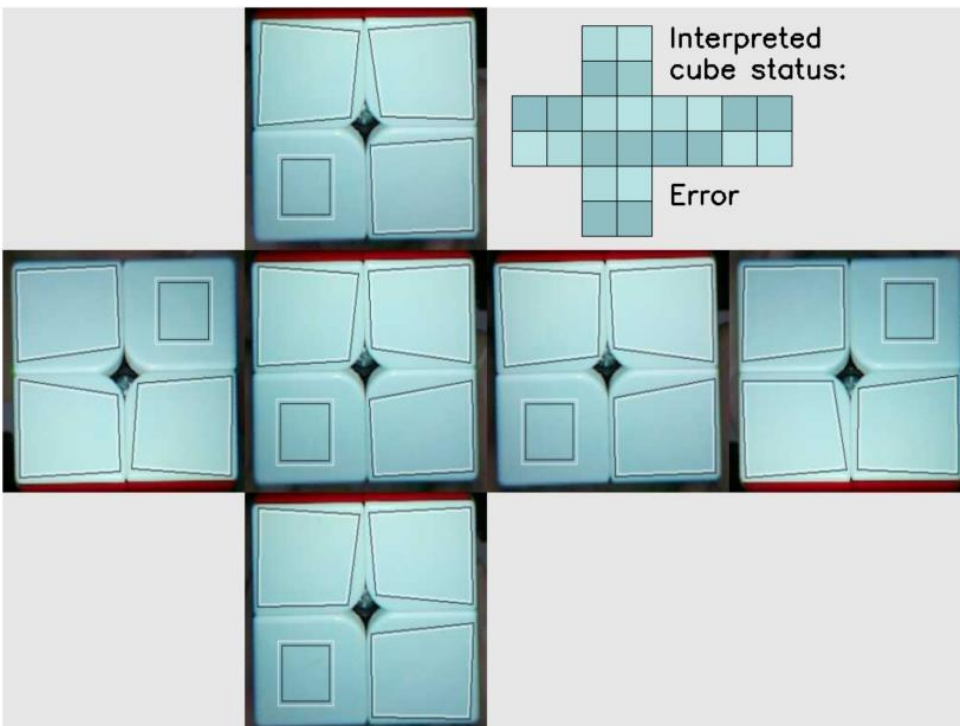
Further than saving data in the text file, a picture of the unfolded cube status is also saved.

- Folder: CubesStatusPictures
- Images: cube_collage_date_time.png

The image is a collage of the cube faces images taken during the cube status detection. A sketch of the interpreted colors is added, by using the dominant colors.



In case of an error (top servo not energized) the **Error** indication is reported:



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.

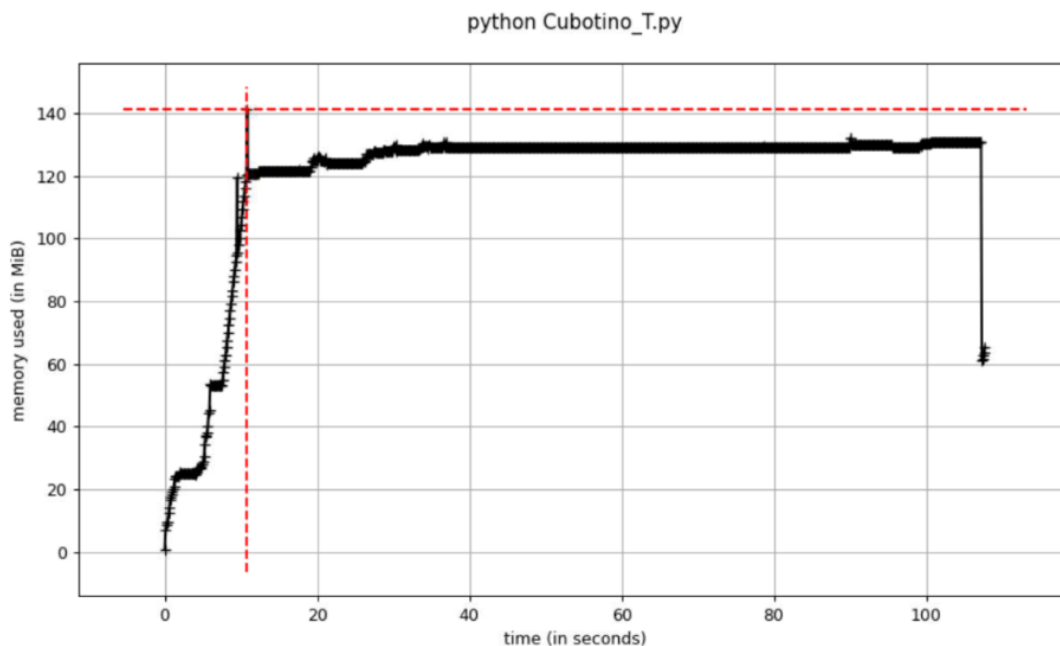
12. Memory profiling

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

```
pi@cubotino:~/cubotino_pocket/src $ free -h
              total        used        free      shared  buff/cache   available
Mem:           363Mi       148Mi       94Mi         10Mi       120Mi       153Mi
Swap:          511Mi        29Mi       482Mi
pi@cubotino:~/cubotino_pocket/src $
```

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.

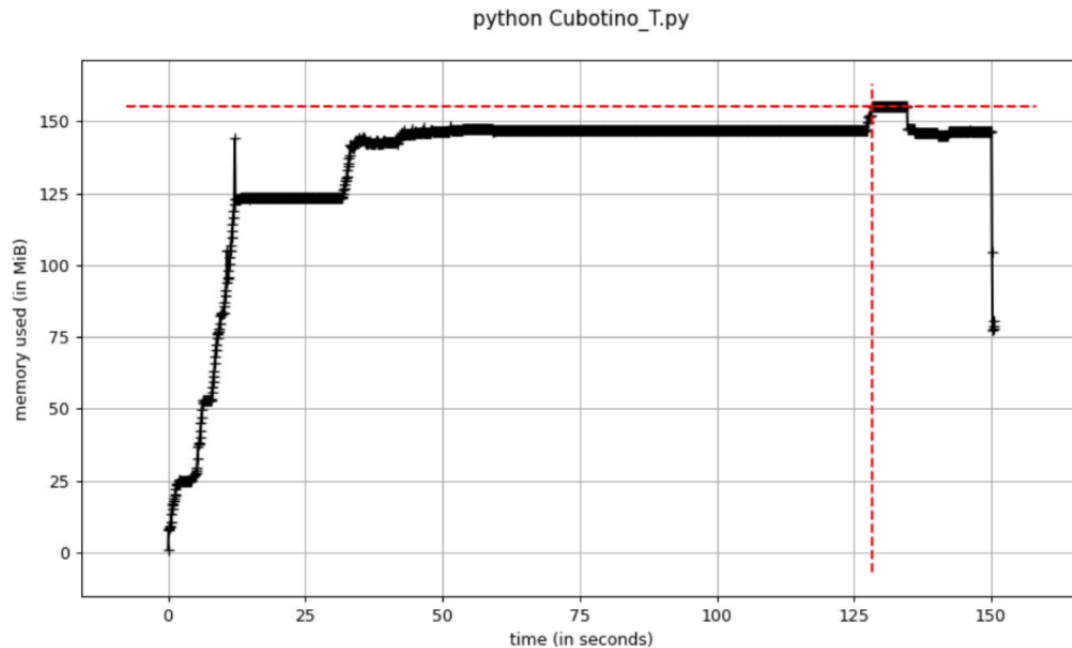


Results:

- 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 graphical information on PC screen, via VNC.
- read and solve a scrambled cube.
- quit the script.



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@raspberrypi:~$ 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 swappiness is also left to its default value of 60.

13. Virtual solvers

CUBOTino Pocket has two virtual solvers:

- `Cubotino_P_test_random.py`
- `Cubotino_P_test_all_permutations.py`

These solvers, especially the first one, have been instrumental for extensive testing and moves optimization.

`Cubotino_P_test_random.py` generates random cube status to be solved by a virtual solver.

The quantity of tests to perform can be passed with `'-- runs'` argument (i.e., `'-- runs 1000'`), otherwise it stops after 100 tests.

There are other arguments available, like preventing the DBL faces rotation, useful during the development phase.

As per the robot, the virtual solver selects the convenient robot solution, and virtually applies all the robot movements. After each movement the cube status is updated.

After the virtual manipulation of the cube, the final cube status is achieved: This cube status is verified if conforming to a solved cube.

Below how the outcome looks like, after testing 10'000 random cube status:

```
#####
Tested 10000 random cube's status with 0 failures
The test took 475.62 seconds
Average time to analyze each cube_status: 47.6 ms

Average quantity of robot movements: 33.2
Average estimated solving time (servos): 33.9
std on estimated solving time (servos): 3.87
Min estimated solving time (servos): 11.6
Max estimated solving time (servos): 47.0

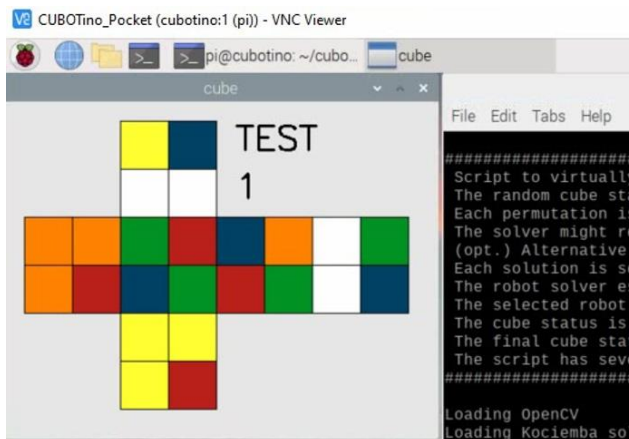
Found faster (DBL) alternative solution: 8578 out of 10000 cubes: 85.8%
Total solutions analyzed: 278795
Optimization type 1 being used: 4266
Optimization type 3 being used: 5019

Saved the cube depth and estimated servo time, of the 10000 runs
File saved at: /home/pi/cubotino_pocket/src/TestRandom_DBL_10000cubes_20240212_222542.txt
```

Be noted for 10'000 cube statuses were generated and analyzed ca 279'000 different solutions (average of 28 solutions per cube status).

`Cubotino_P_test_random.py` can be launched with `'-- plot'` argument, to have a graphical animation.

The graphical animation slows down dramatically the process, it is mainly meant for fun; An idea of how it looks like on YouTube: <https://youtu.be/ZorUd1qCANw>



Cubotino_P_test_all_permutations.py is a more interesting virtual solver developed for CUBOTino Pocket.

It generates and tests all the 3'674'160 permutations of a 2x2x2 Rubik's cube (be aware, a Raspberry Pi Zero2W takes more than two days to test all these permutations 😊).

The permutations are generated from lower to higher depth, meaning from requiring less moves to get solved to a higher quantity: Max depth is 14, meaning the optimal solution would require 14 face rotations.

In case you'd like to try it, the argument '-- stop' (i.e., -- stop 10000) can be used to limit the test to the passed number of permutations (cube statuses).

This solver, like the ***Cubotino_P_test_random.py*** generates and tests one cube status at a time, by making use of the same functions used by the robot.

This involves:

1. Generating a new cube status (permutation).
2. Passing it to the Kociemba solver, that returns one or (more likely) several optimal solutions.
3. Generating all the alternative solutions with DBL faces rotations.
4. Sending one solution at a time to the robot solver, that returns a series of information and the estimated solving time.
5. Selecting the fastest solution for the robot.
6. Virtually manipulate the cube as the robot would do (Spin, Flip, Rotate).
7. Update the cube status after each manipulation (Spin, Flip, Rotate).
8. After the last manipulation the final cube status is achieved: This is then tested if it resembles a solved cube, to track for eventual errors.

Below hot the outcome looks like, after testing the first 10'000 permutations (cube status), by passing the parameter '-- stop 10000':

```
#####
1) Permutations tested      2) Solver & DBL add. solutions      3) Robot solver errors
10000 55370 0
#####
Permutations tested: 10,000
Additional solution tested (from the solver and robot DBL): 55,370
The virtual movements applied by CUBOTino_P have correctly solved 10,000 different cubes: 100.0%
The analysis took 85.8 seconds
(average of 8.58 millisecs per each cube status)
(.virtualenvs) pi@cubotino:~/cubotino_pocket/src $
```

Credit: For the generation of all the permutation I largely used the code from ImportMathRepositories (https://github.com/ImportMathRepositories/2x2_Depth) to which goes my credits

14. ChatGPT

On January 2023, I made an account to [chatGPT](#) (for info <https://en.wikipedia.org/wiki/ChatGPT>), and decided to test it with some specific questions (searching for help on my next project...)

I must admit I got really impressed by those answers, and decided to ask a more general question...

Question: *“how to make a 3d printed rubik's cube robot solver” (...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 modelling 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-tuning 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's a complex project and requires a good understanding of mechanical engineering, robotics, and programming.

The generic question got a generic answer, yet it's 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.

I do have the feeling I will use chatGPT when searching for some help 😊

15. 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/wB4MwINGon0>), 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-Pocket-Optimal-Solver-Robot-for-Rubiks-Cu/>).

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!

16. Credits

- to Mr. Kociemba, that has provided unvaluable help to modify the solver from FTM to QTM, a perfect match for the CUBOTino working principle.
- Hans Andersson, with his Tilted Twister ([Tilted Twister 2.0](#)) Lego robot, so inspiring: Very simple yet effective mechanic concept.
- ImportMathRepositories, for the generator of all the 3'674'160 cube status (https://github.com/ImportMathRepositories/2x2_Depth)

Credits also to all the people who have provided feedback on my Rubik's cube robot projects journey.

Thanks, in particular to the people I've had contact with along CUBOTino journey: Jacques, Richard, Scott, Yannick, Chad, Kevin, Martin, Andreas, John, Derek, Denis, and many others.

For sure I'm missing many names, please accept my apology.

17. 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, 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 was 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 to think on how to make an easier version, with an eye to costs: CUBOTino project started in January 2022

Other CUBOTino versions followed in 2023 and 2024 (Micro, Pocket).

Contacts:

- I am not into social media.
- I can be reached via email: andrea.favero71@gmail.com

18. Revisions

Rev	Date	Notes
0	28/01/2024	First release
1.1	12/02/2024	<p>Quite a large update</p> <p>Modified several files, main topics:</p> <ul style="list-style-type: none"> Solved a few bugs Added the facelets color animation on the display, after a successful run. Changed the order of data to log in the Cubotino_solver_log.txt, and added the URF/DBL info. <p>Instructions, added more info about:</p> <ul style="list-style-type: none"> Optimal solution from the Kociemba solver and the one for CUBOTino. Virtual solvers.
1.2	15/02/2024	Instructions: Added command “chmod +x install/setup.sh”
1.3	22/02/2024	<p>Modified:</p> <ul style="list-style-type: none"> Cubotino_P_display.py (bug on progress bar for newer Pillow versions) <p>Instructions:</p> <p>Corrected many filename typos. Updated the “crontab” instructions to reflect the file</p>
1.4	03/03/2024	<p>Modified:</p> <ul style="list-style-type: none"> solver.py → Selectable full-turn or quarter-turn metric (QTM as default). pruning.py → Selectable full-turn or quarter-turn metric (QTM as default). defs.py → Selectable full-turn or quarter-turn metric (QTM as default). Cubotino_P.py: <ul style="list-style-type: none"> Animation showed either on display or screen. Animation can be interrupted via the Touch button. Loading OpenCV info to display. Cubotino_P_display.py → more efficient management of the Cubotino logo. <p>Instructions:</p> <ul style="list-style-type: none"> Corrected from ‘sudo Cubotino_P_bash.sh’ to ‘sudo nano Cubotino_P_bash.sh’ Added chapter ‘2x2x2 rubik’s cube and robot efficiency’
2.0	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_P_settings_manager.py</i></p> <p>Modified files (scripts using settings from the *settings.txt files): <i>Cubotino_P.py, Cubotino_P_servos.py, Cubotino_P_servos_GUI.py, Cubotino_P_display.py, Cubotino_P_camera_os10.py, Cubotino_P_camera_os11.py</i></p> <p>Renamed files:</p> <ul style="list-style-type: none"> From: Cubotino_P_settings.txt →to: Cubotino_P_settings_default.txt From: Cubotino_P_servo_settings.txt →to: Cubotino_P_servo_settings_default.txt <p>Instructions:</p> <p>Explained how the settings files are managed and corrected their name at the Parameters and settings.</p>

Rev	Date	Notes
2.1	29/03/2024	<p>Modified files:</p> <ul style="list-style-type: none"> • Cubotino_P.py: <ul style="list-style-type: none"> ○ Added HSV color space analysis (when BGR color analysis fails). ○ Added gap between faces at screen animation. ○ Some little cleaning. • Cubotino_P_display.py → the colour detection winner is plot • Cubotino_P_settings_manager.py → added the get_macs_AF function • Cubotino_P_camera_os10.py → removed unused libraries • Cubotino_P_camera_os11.py → removed unused libraries <p>Removed file:</p> <ul style="list-style-type: none"> • get_macs_AF.py → Function moved in Cubotino_P_settings_manager.py <p>Instructions:</p> <ul style="list-style-type: none"> • Briefly explained at Colour's detection strategy chapter. • Replaced some of the images that were distorted.

Notes:

- Instructions and files will be updated whenever needed/possible.
- Please feedback in case of mistakes or parts requiring a better explanation.