How to make

CUBOTino micro: The World's smallest Rubik's cube solver robot

https://www.instructables.com/CUBOTtino-Micro-the-Worlds-Smallest-Rubiks-Cube-So/
https://github.com/AndreaFavero71/CUBOTino_micro

Andrea Favero, Groningen (NL)

15/03/2024 Rev. 1.0 (always check if a newer version is available)

Robot demonstration at YouTube: https://www.youtube.com/watch?v=EbOHhvg2tJE

Highlights:

- As far as I know, this is the smallest Rubik's cube solver robot in the World.
- It uses a 30mm Rubik's cube.
- On average it takes less than 70 seconds for scanning and solving a scrambled cube.
- About 100 € of materials.
- This is the smallest version of CUBOTino series; It is scaled down from Cubotino Autonomous (https://youtu.be/dEOLhvVSBCg).
- 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. White LED: get or prepare it for diffuse light.
    d. Test the display and connections.
    e. Test PiCamera and set the focus for the right distance.
    f. Test the servos range and set them to their mid position.
    g. Select the Servo Offset Compensator.

3. 3D print, and assembly
    a. Print the parts.
    b. Assemble the robot.

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.

REMINDER: YOU ARE ASSEMBLING THE WORLD’S SMALLEST RUBIK’S CUBE SOLVER ROBOT!
This intrinsically implies small parts and large patience 😊

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

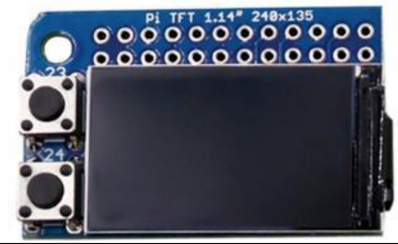
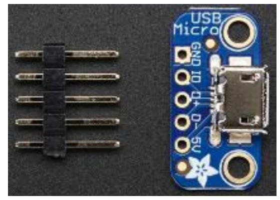
Appendix 1 - Additional information

Appendix 2 - THE CUBOTINO PROJECT

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 30mm Highly recommended GAN 330key-chain	See next page	10	
2	Servos I used: Adeept Micro Servo Motor AD002 9G Metal Geared (**180deg** 2Kg/cm, **Pulse width 0.5 to 2.5ms**)	https://www.adeept.com/ad002-servo-motorx2_p0274.html or https://www.amazon.com/Adeept-....	10 (2 servos + arms)	
1	Raspberry Pi Zero2W (H needed, yet the header can be bought at side) **Raspberry Pi ZeroW works too**	https://www.sossolutions.nl/	27.5 (Zero2W version, Dec 2023)	
1	microSD HC 16GB	https://www.dataio.nl/sandisk-ultra-micro-sdhc-16gb-uhs-i-a1-met-adapter/	8.5	
1	PiCamera V1.3 (PiCamera V2 should by changing the *s_mode* parameter. Not verified the minimum focus distance!)	https://www.amazon.nl/gp/product/B01M6UCEM5/ref=ppx_yo_dt_b_asin_title_o05_s00?ie=UTF8&th=1	7.5	
1	16 cm cable Raspberry Pi Zero/Camera **Note**: The Raspberry Pi Zero uses a specific ribbon cable! See picture	https://www.amazon.com/A1-FFCs-Cable-Raspberry-Camera/dp/B07T4SHQ28/ref=sr_1_1?crid=2JNY3R237W2DK&keywords=pi+camera+cable+16cm&qid=1675506822&sprefix=picamera+cable+16cm%2Caps%2C154&sr=8-1	5	

Q. ty	Part	link to the shop I used	Cost (€)	Notes
1	Mini PiTFT - 135x240 TFT 1.14inches display	https://www.amazon.com/Mini-PiTFT-135x240-Add-Raspberry/dp/B09Q1SRJ6H/ref=sr_1_4?crid=1BU461Z72K12M&keywords=mini+Pi+TFT+1.14&qid=1675506942&sprefix=mini+pi+tft+1.14%2Caps%2C156&sr=8-4	19 (7€ Aliexpress)	
1	USB MICRO-B BREAKOUT BOARD	https://www.adafruit.com/product/1833	2	
~120g	Filament 1.75mm		~2.5	Suggested PETG, yet other materials will do the job

Note: all components are also available from AliExpress

Electronic and electrical small parts:

Q. ty	Part	Notes
1	White LED	To illuminate the cube
1	Prototype board	To connect all the parts
1	14pin (2x7) or 12pin (2x6) GPIO female header (Plastic body height ca 8 to 8.5mm)	To connect the Connections board to Raspberry Pi Zero
1	1x8 Male Headers 90deg	To connect the servos and LED.
1	40pin (2x20) GPIO male header	In case you could not get the WH version of Raspberry Pi
1	Heat-sink for Raspberry Pi	Not really needed

Screw types:

Quantity (indicative)	Dimension	Head type
3	M3x12	Conical
10	M3x8	Conical
12	M2.5x8	Cylindrical
12	M2.5x4	Cylindrical

Power supply:

If micro-USB: 2A phone charger with micro-USB cable.
Because of the small amount of power consumed by the servos, a 5V power bank for phone works well too; See next page for a suggested version.

Off course some other common materials are needed (wires, solder and solder device, tire wraps, self-adhesive rubber feet, etc.).

5. The suggested cube

GAN 330 Keychain is the cube I have built the robot around.

In theory every 30mm Rubik's cube will work, but this GAN 330 keychain is extremely recommended.
The rational 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/4001038623950.html?pdp_npi=2%40dis%21EUR%21%E2%82%AC%2014%2C05%21%E2%82%AC%208%2C01%21%21%21%21%21%402103222716761077538436432e8bef%21120000032064721934%21btf&_t=pvid:faade393-c22d-4858-b04e-ad8bc281ef12&afTraceInfo=4001038623950__pc__pcBridgePPC__xxxxxx__1676107754&spm=a2g0o.ppclist.product.mainProduct
- https://www.amazon.com/GAN-Speed-Cube-Ring-Keychains/dp/B086V59NTF/ref=sr_1_1_sspa?crid=17YELTO4BF1KB&keywords=gan+330&qid=1676107792&sprefix=gan+330%2Caps%2C189&sr=8-1-spons&psc=1&spLa=ZW5jcnlwdGVkUXVhbGlmaWVyPUFBVzdBSDkzNko5RVYmZW5jcnlwdGVkSWQ9QTAyMTU3MTMxS1FBSjFIQ0RRNUlBJmVuY3J5cHRlZEFkSWQ9QTA3Mjc2MDUyVVhQUkIxS1JXVEpMJndpZGdldE5hbWU9c3BfYXRmJmFjdGlvbj1jbGlja1JlZGlyZWN0JmRvTm90TG9nQ2xpY2s9dHJ1ZQ==

Notes:

1. The blue logo on the white center facelet alters the color recognition: refer to the troubleshooting section for some info on how to remove that logo, to improve the cube status readability.
2. I'm neither affiliated with Amazon nor with GAN; I'm just sharing my positive experience with this cube 😊.
3. The Amazon link is 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 gain 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:
    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.

Purchasing a Raspberry Pi Zero (Info valid at the moment of writing, 15 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.
- In February 2024, a Raspberry Pi Zero2W without headers was 19.95€ (+shipping) via official shop in The Netherlands.

Power bank

Because of the small servos used and the low overall power budget, I searched for a power bank having small footprint. Other parameters considered were large capacity, fast charging, high R&R and limited cost 😊

The one I bought claims to be 10000mAh; I don't know whether it really has that capacity, yet it lasts very long.
The shape is perhaps a bit too thick and round, but I don't care much.
The black version is cheaper that the other colors, therefore I'll re-print the Baseplate in a different color to highlight the separation between robot and power bank:

Power bank Intenso Powerbank XS10000

I got his power bank for 15€, from Amazon.nl:
https://www.amazon.nl/dp/B07Z8DF4DG?ref=ppx_yo2ov_dt_b_product_details&th=1

This **USB – microUSB** cable is on the high-cost side. Reasons for choosing this model relates to the right-angle terminals, and short cable length.

The cable length description was for 10cm, while I got a cable with almost 20cm, which is good. My initial intention was to keep the charge indicator upside, yet it would stay under the bot.

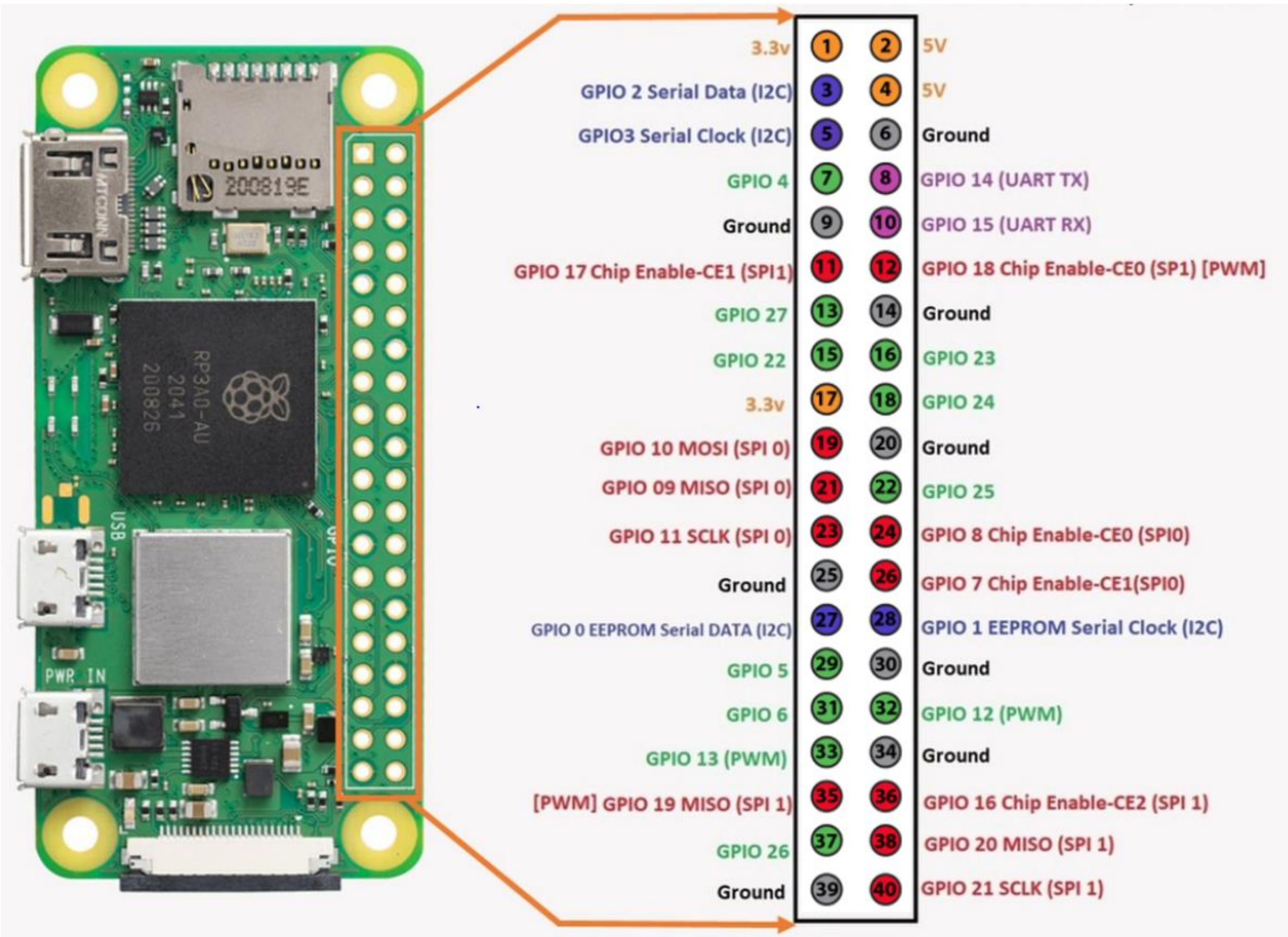
I got this cable for 9€ from Amazon.nl:
https://www.amazon.nl/dp/B01FPOXKQG?psc=1&ref=ppx_

7. Initial preparation

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

1. Make the connections board (see specific chapter).
2. Solder the power supply cables to the Raspberry Pi board and microUSB board (see specific chapter).
3. Setup the Raspberry Pi (see specific chapter).
4. Modify the LED, by making its tip flat (see specific chapter).
5. Test the Connections_board, LED, servos, display (see preliminary test chapter)
6. Adjust the PiCamera focus, and test it (see preliminary test chapter)
7. Enlarge the holes at servos flange with a Ø2.5mm (max Ø3.0mm) drill bit. This allows using bigger/better screws than those supplied with the servo, for which the 3D part is designed for.
8. Check the servos rotation range and set them to their middle position (see specific chapter).
9. Determine which servo offset compensator is needed in your robot (see specific chapter).
10. Print the parts (see specific chapter).

8. Raspberry Pi Zero 2 GPIO pinout



Used pins:

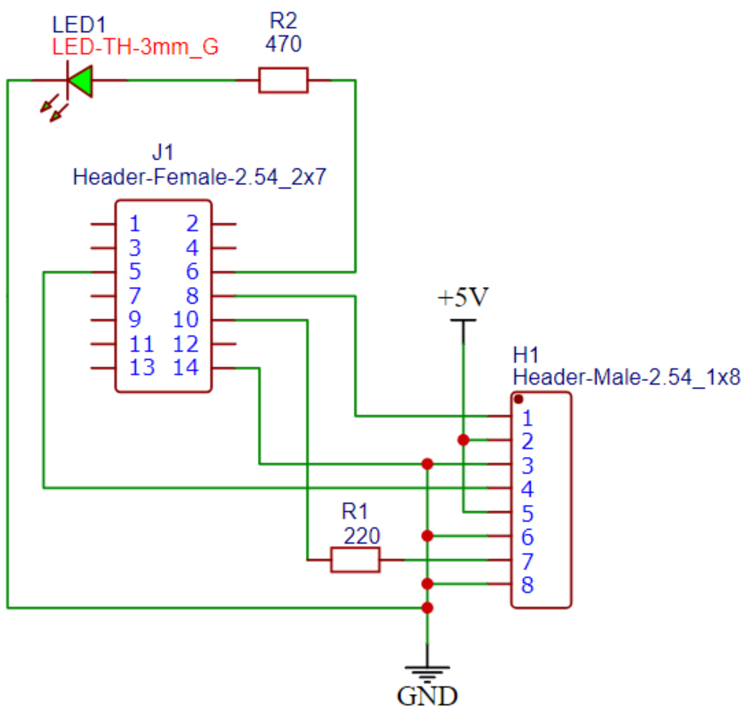
Pin	Label/GPIO	Purpose
1 to 24		Display board with buttons
31	GPIO 6	ACT LED on the Connections_board
32	GPIO 12 (PWM)	PWM bottom servo (Cube_holder)
33	GPIO 13 (PWM)	PWM top servo (Top_cover)
35	GPIO 19 (PWM)	PWM Led at Top_cover
39	GND	Common GND for servos and Top_cover LED

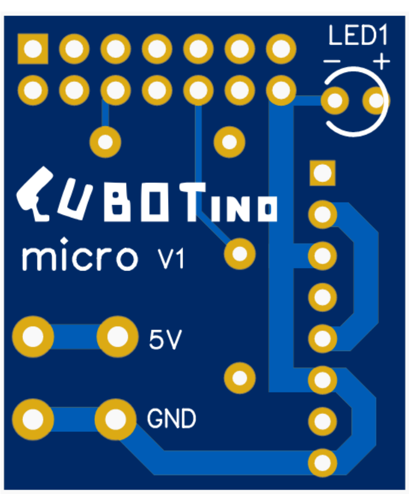
10. Connections board

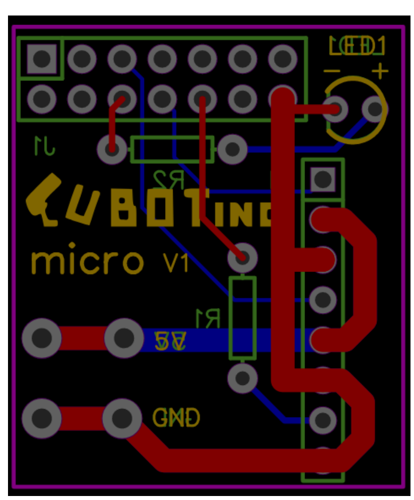
The Connections_board is a simple passive board, that serves as a hub for the Servos and the Led connections.

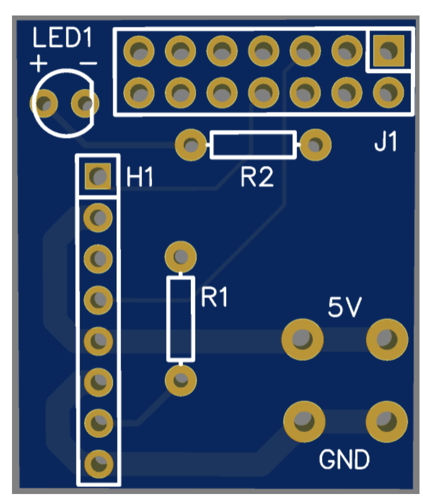
PCBWay has offered their sponsorship to this project. This means:

- It makes me happy 😊
- I got the motivation to learn how to design a (simple) board.
- It makes possible to easily add:
  - a LED for the ACT function, by considering the Raspberry Pi ACT LED is quite hidden.
  - a couple of soldering pads for the power supply cables.
- The test on the first board went ok.
- You can decide whether to make the board by yourself, order it from PCBWay or to place the order to another board manufacture (I hope you'll order to PCBWay).



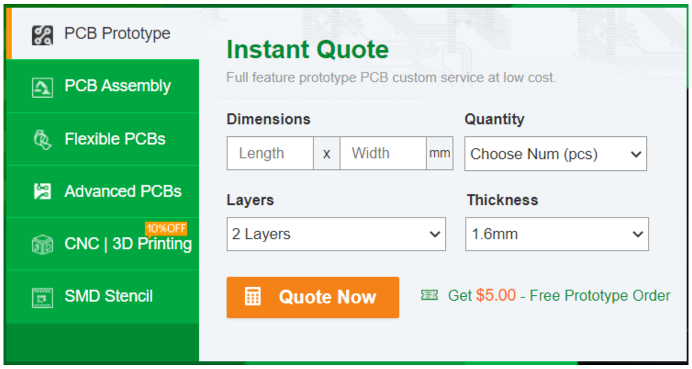




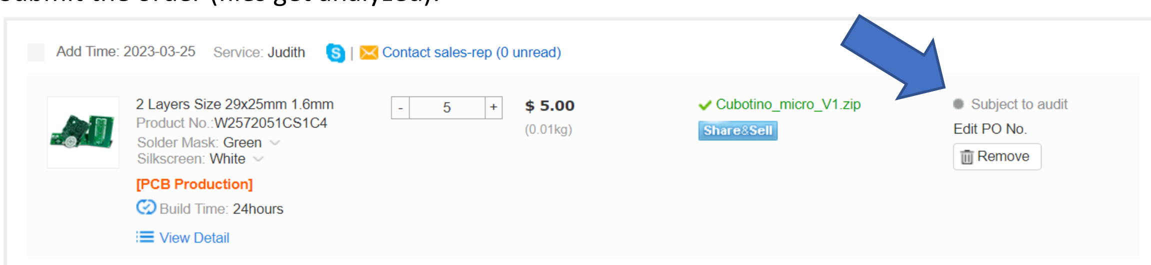


A) Order the board:

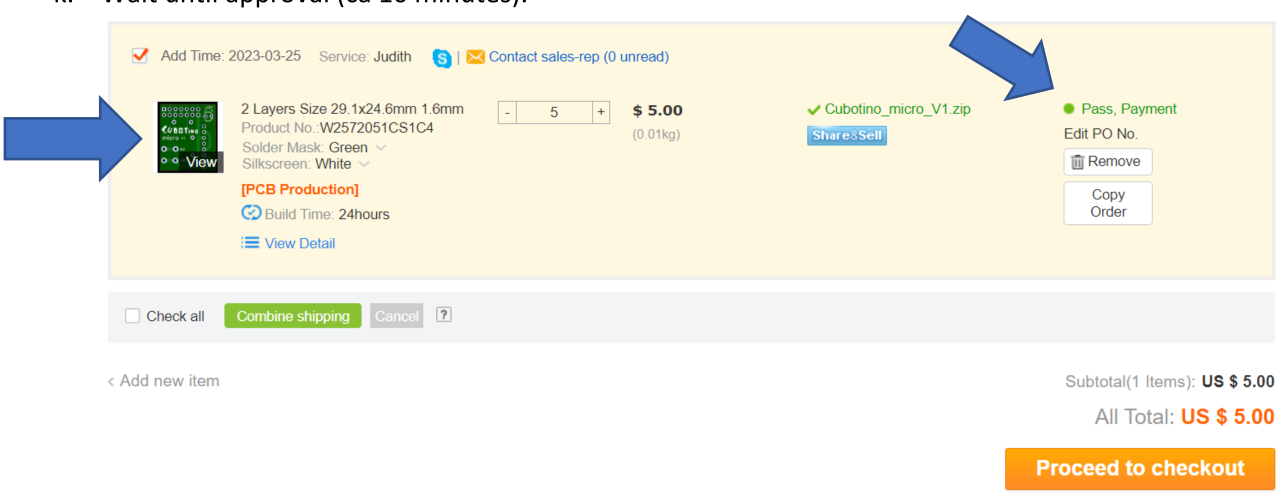
a. At PCBWay.com enter the board dimensions: Length=29, Width=29 mm, 2 layers, 1.6mm thickness



b. Check the Quote.
c. The minimum ordering quantity is 5 pieces.
d. Leave all the other parameters at default.
e. Solder mask color up to your preference (be aware some colors are more expensive).
f. Select the transportation courier (I'm testing the "Global Standard")
g. Save to chart.
h. Make or login to your account.
i. Drop the "Cubotino_micro_V1.zip" file.
j. Submit the order (files get analyzed):



k. Wait until approval (ca 10 minutes).



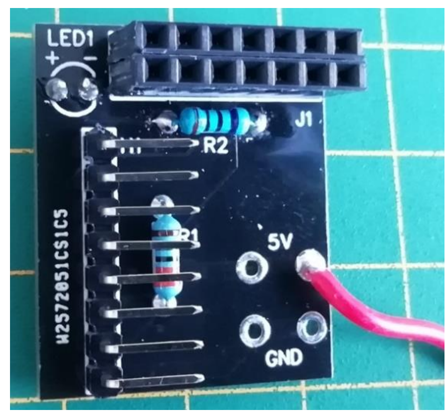
l. Proceed to checkout:
   i. Check the address.
   ii. Select the transportation.
   iii. Make the payment.

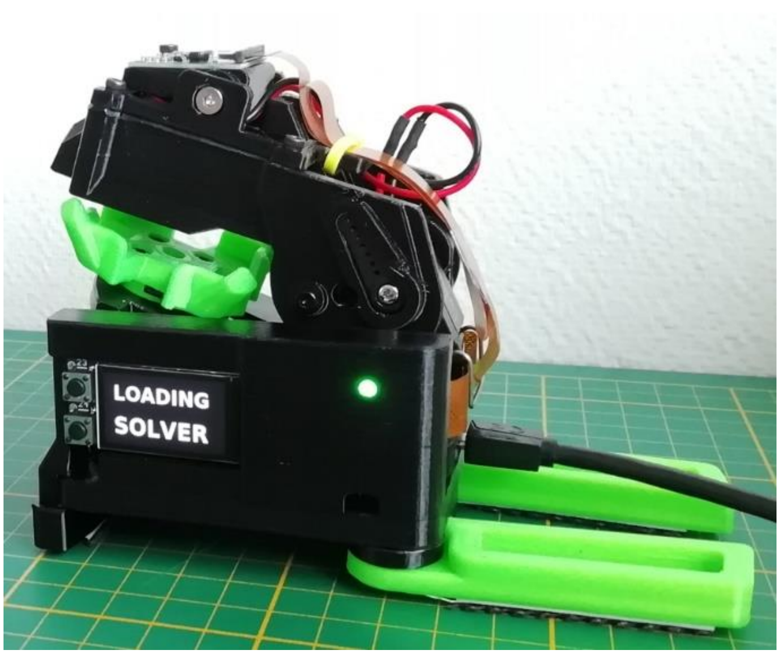
B) Assemble the ordered board:

Suggested soldering order:

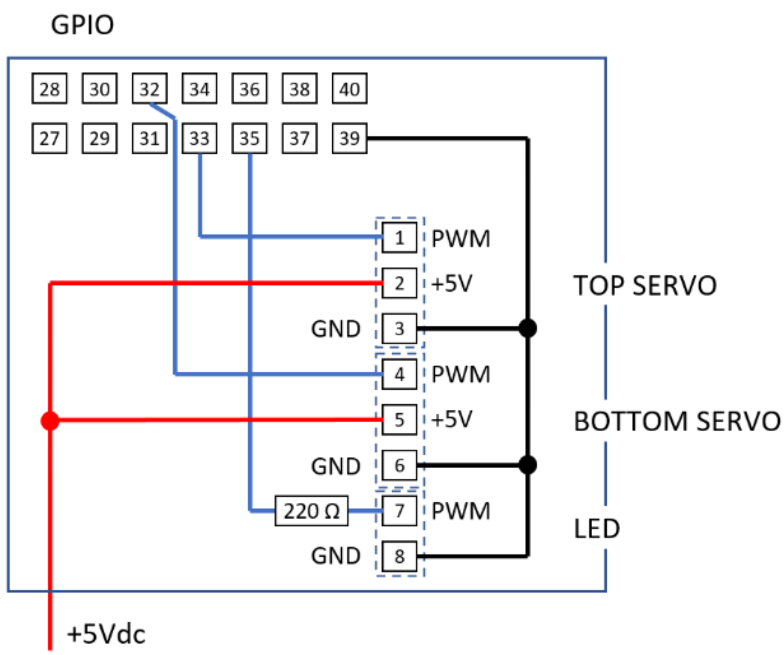
1. Solder the resistors (R1=220ohm, R2= 470ohm).
2. Solder the 1x8 male header, right angled, with terminals pointing to the board center.
3. Solder the 2x7 header:
   - A 2x7 header can be obtained by grinding some plastic from a 2x8.
   - A 2x7 header can be made by using 2 strips of 1x7 header.
4. The GND of the board is provided by the GPIO pin 39. The additional pads can be used as hubs.
5. The +5Vdc of the board is provided by the soldered wire.
6. Solder the 3mm LED; For the LED you can choose the soldering side:
   - Opposite to the connectors, to be used with the Cover version having the "led hole". This solution makes the LED well visible, at the same side of the display.
   - At the connector side. The LED will be less visible.

Note: The ACT-LED function needs to be activated; See chapter 18.

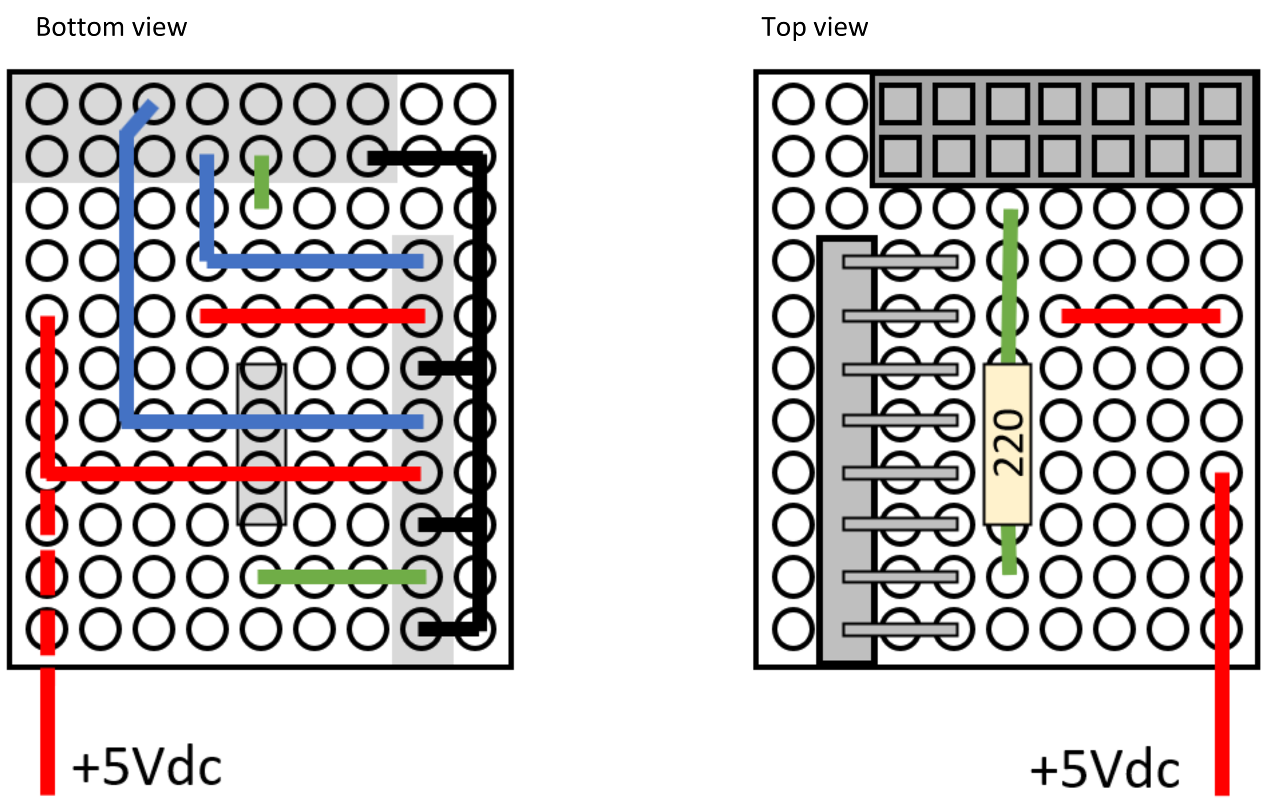




C) Make the board by yourself, from a perfboard:



You can make it on your own via a (perfboard) prototyping board.

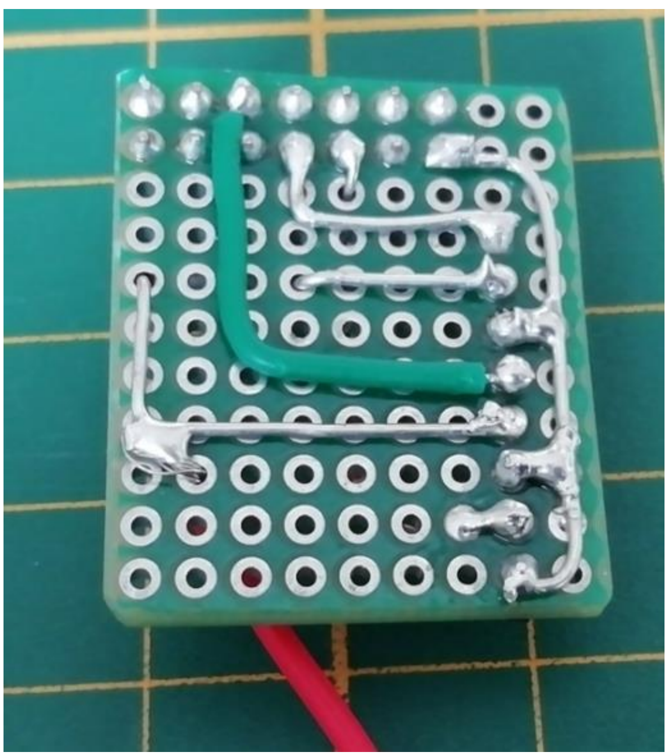


A few notes:

1. The perfboard can be of a single side type. Displayed here is the “Without ACT-Led” version.
2. Board dimension (WxH) is about 24mm x 29mm (9x11 holes).
3. Suggested to use a 2x7 header, yet a 2x6 will also work:
   - A 2x7 header can be obtained by grinding some plastic from a 2x8.
   - A 2x7 header can be made by using 2 strips of 1x7 header.
4. Position the 2x7 header at the corner hole of the board and solder it.
5. Position the 220ohm resistor before placing the 1x8 male header.
6. Position the 1x8 male header, right angled, and solder it.
7. Use an insulated wire to connect GPIO pin32 and header pin 4; This ensures proper insulation between the GPIO pins 31 and 33.
8. The GND of the board is provided by the GPIO pin 39.
9. The +5Vdc of the board is provided by the soldered wire.

Note: The red circle on the picture serves as reference for the ACT-LED, see next page.

Bottom view



Top view

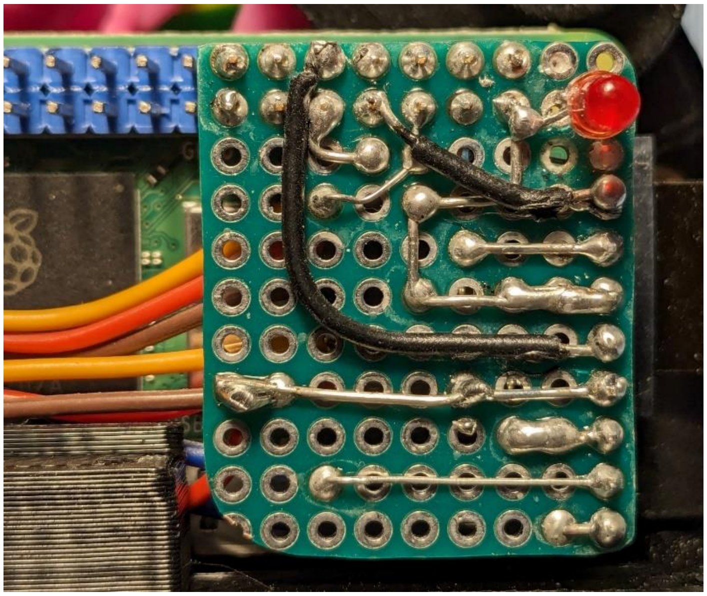
10. Denis, a helpful Maker, has proved that it is possible to add the ACT-LED on this perf board version.

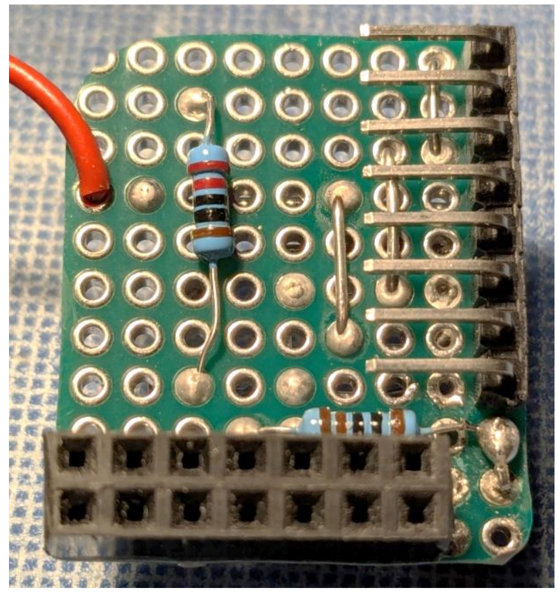
    There is room for the additional 3 mm diameter LED, the 470 Ω resistor and the insulated wire from GPIO pin6 to the positive pin of the LED.

    The red circle in the above picture shows the right location for the LED, which should be protruding from the bottom side, to be later inserted in the cover hole, during assembly.

    The 470 Ω resistor is installed on the top side.

    Below a couple of pictures as example:





The file to be used for the 3D printing of that specific cover is the Cover_Perfboard_hole.stl (the LED has a slightly different location than the PCB board)

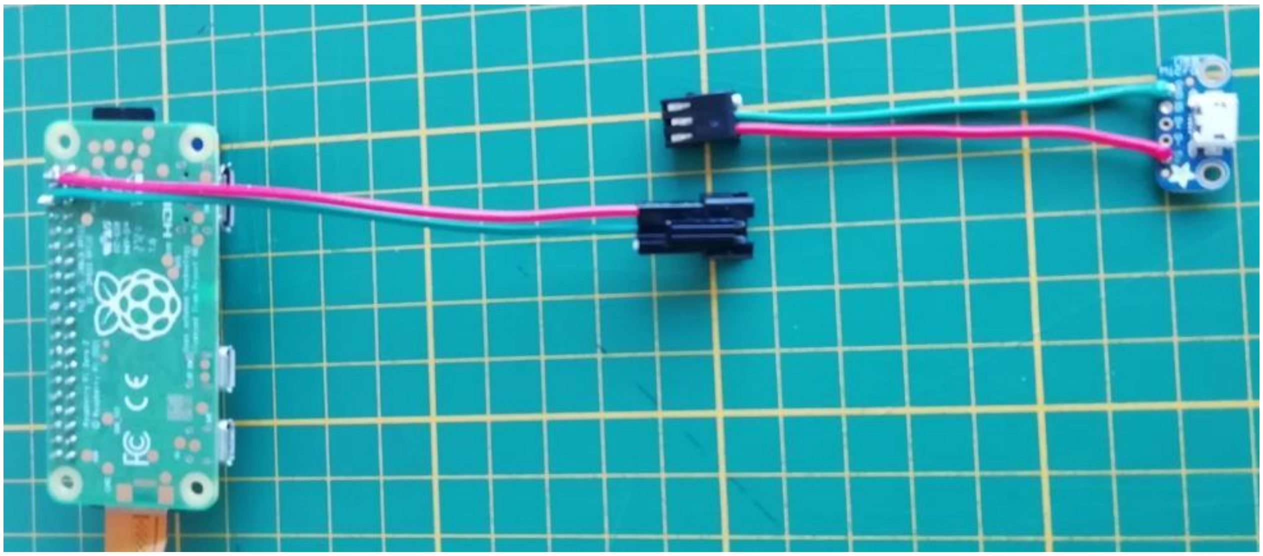
11. Power supply wiring

Because of:

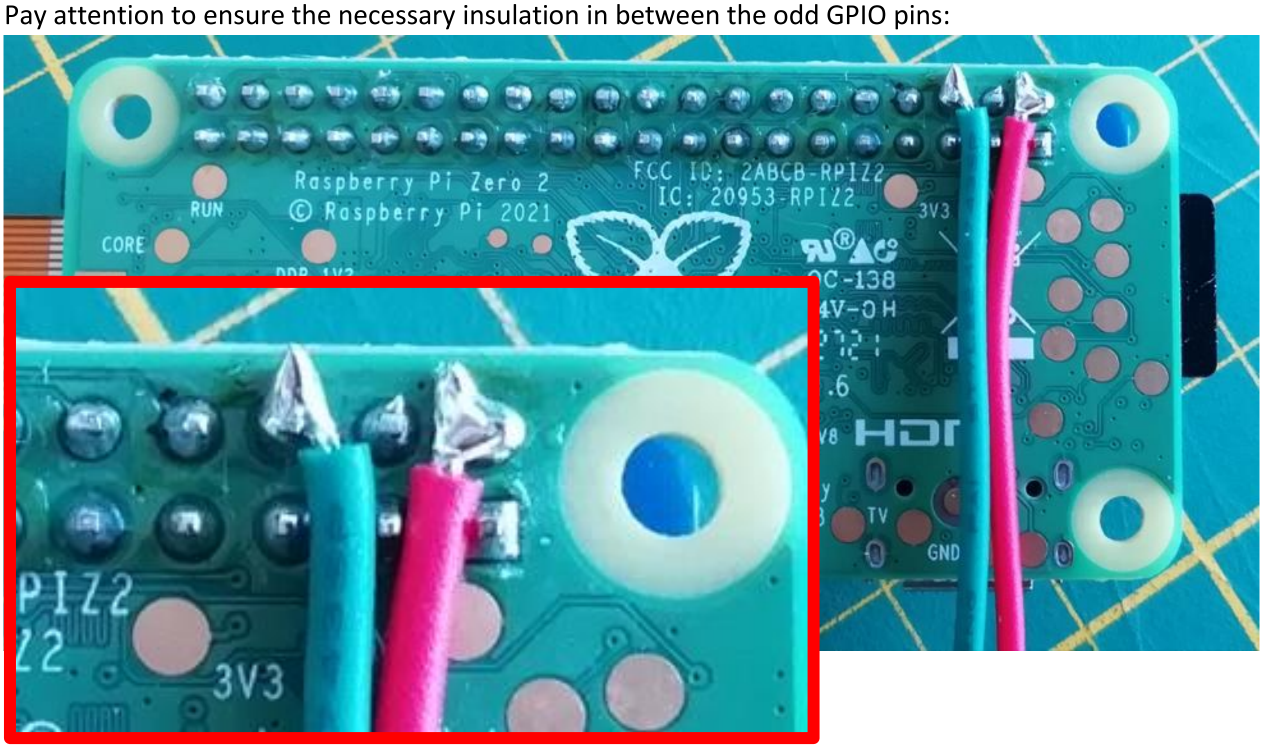
- The chosen display (small and integrating two buttons) occupying the GPIO power pins.
- The desire to constrain the robot to a small size.

The power supply cables are soldered directly on the Raspberry Pi.
This is also the case for the +5Vdc of the Connections_board.

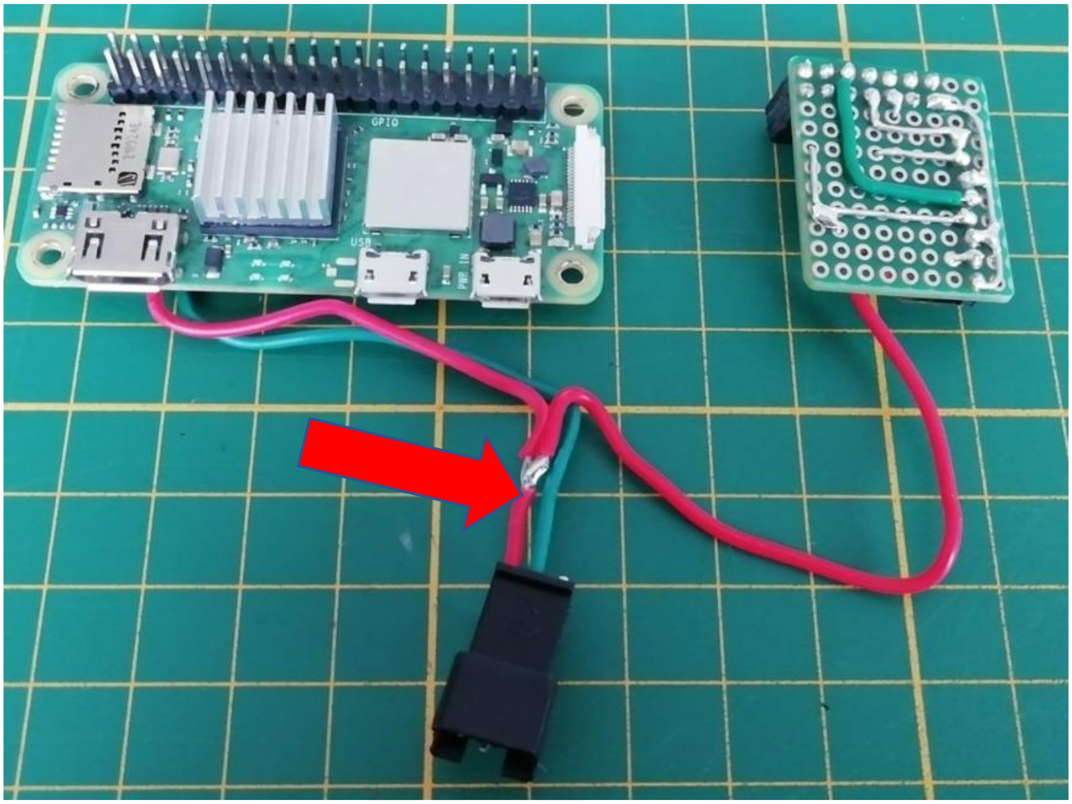
Use a couple of wire with at least 0.5mm² cross section, coupled to any small size two wire connector (JST connectors used in radio-controlled models are a perfect fit):

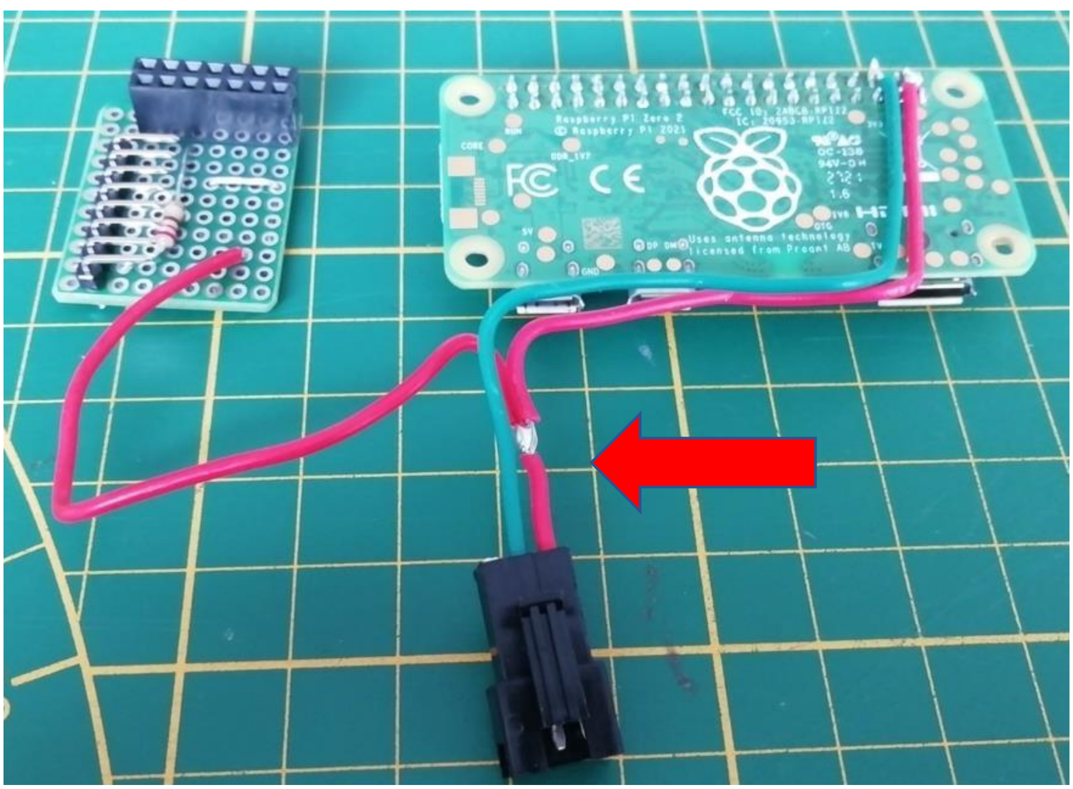


For the Raspberry Pi: Cable length of about 8 to 10cm
Positive → Pin2, or Pin4, or Pin2 & Pin4
Negative → Pin6
Pay attention to ensure the necessary insulation in between the odd GPIO pins:

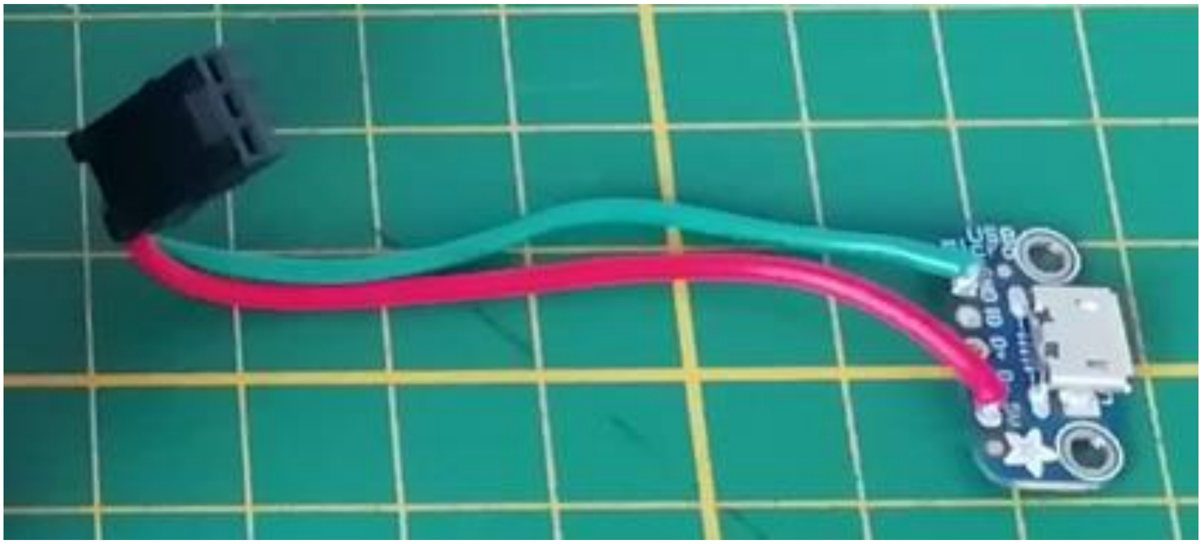


Solder the Connections_board positive power wire as shown, close to the power supply connector. Preferably, add a piece of heat shrink tube, for insulation (not necessary but always a good precaution).
For the Connections_board consider 8~9 cm of wire.





For the microUSB breakout board: Cable length of about 6cm.
Cable insertion side to the board as per below picture:



12. Setting up the Raspberry Pi

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).

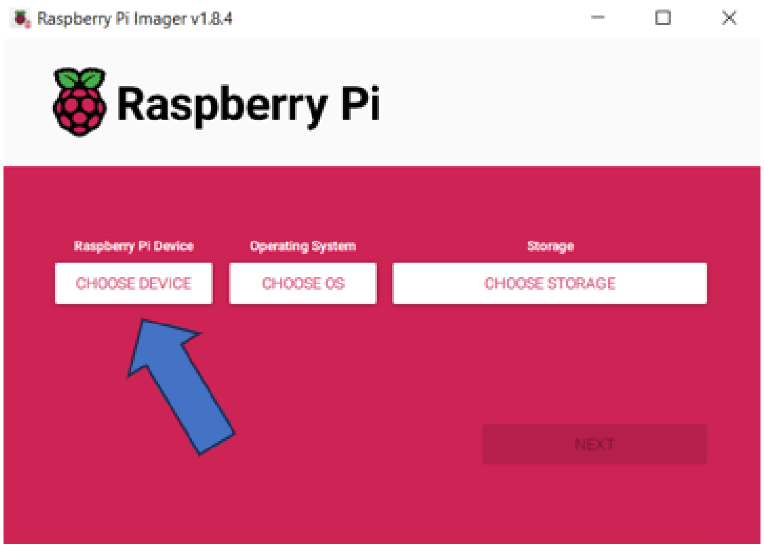
All the needed files, and Raspberry Pi settings, are stored in a GitHub repository.

Step1: Download the "Buster" OS image (*2023-05-03-raspios-buster-armhf.img.xz*) from the official site:
https://downloads.raspberrypi.org/raspios_oldstable_armhf/images/raspios_oldstable_armhf-2023-05-03/2023-05-03-raspios-buster-armhf.img.xz

Step2: Flash the OS to the microSD:

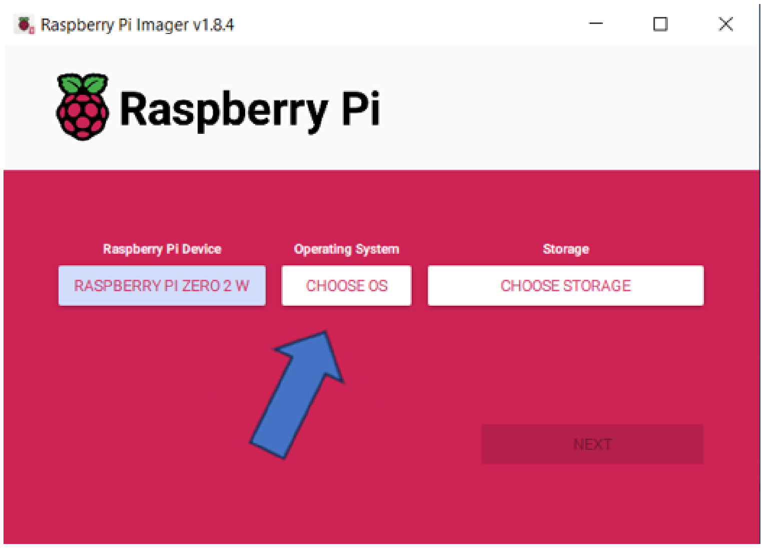
a. Download the Raspberry Pi Imager, from the official site https://www.raspberrypi.com/software/

b. Select ***CHOOSE BOARD***:

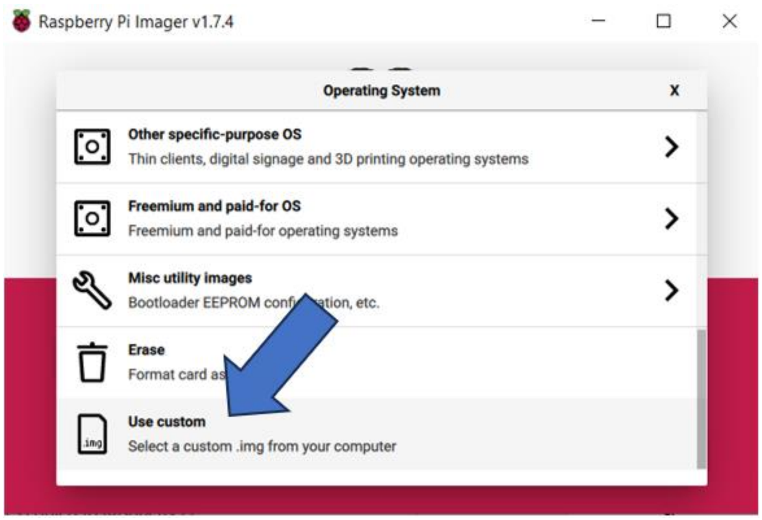


and select the board type you are using:

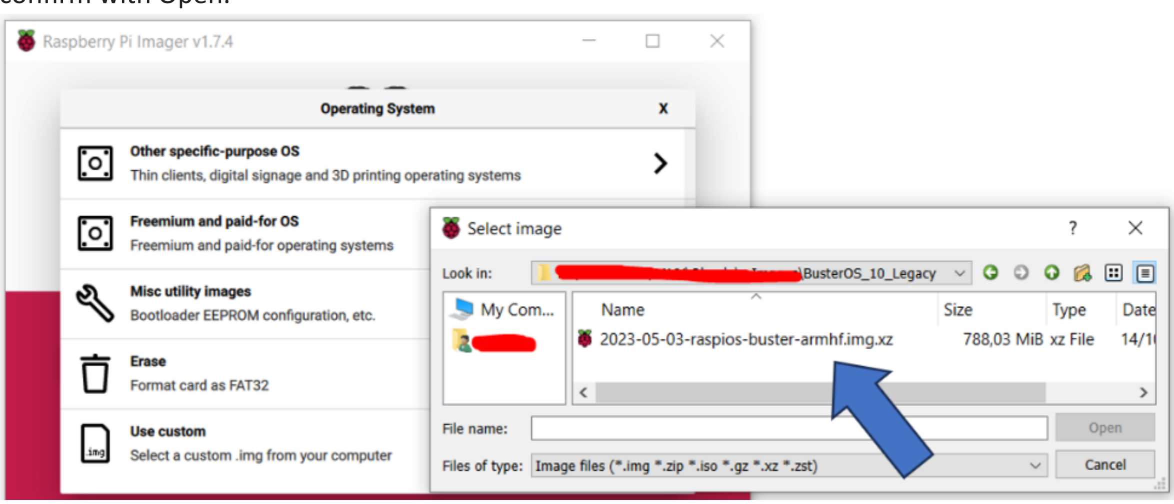
c. Select ***CHOOSE OS***:



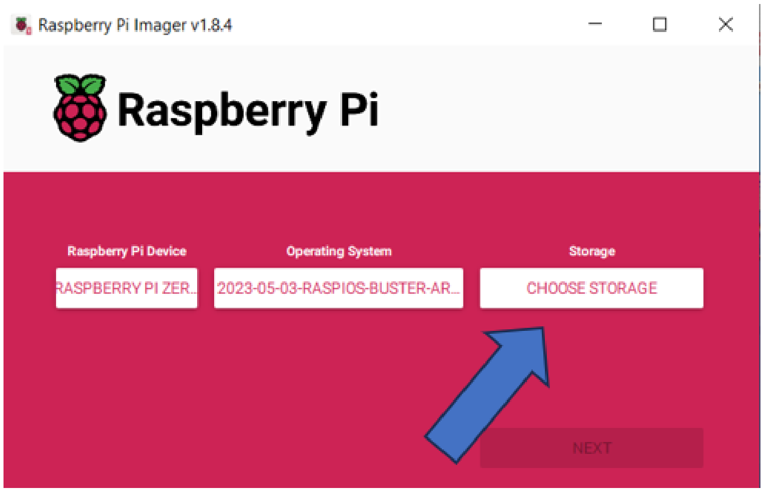
d. Scroll completely down and select ***Use custom***:



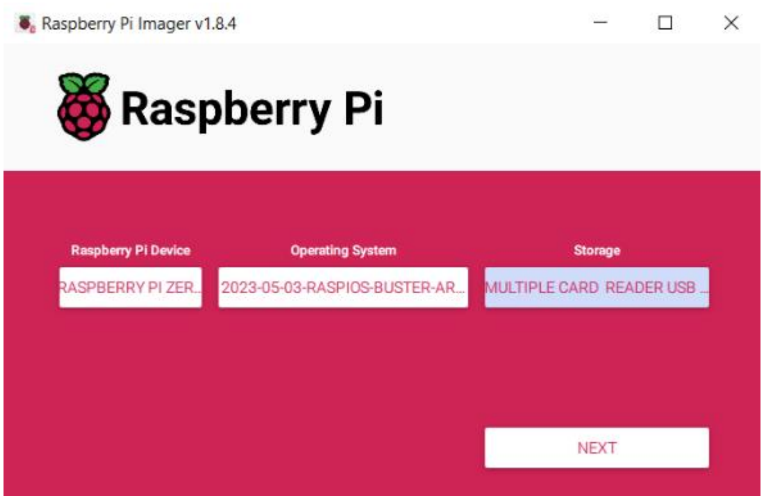
e. Navigate on your PC and select the downloaded Image (*2023-05-03-raspios-buster-armhf.img.xz*), confirm with Open.



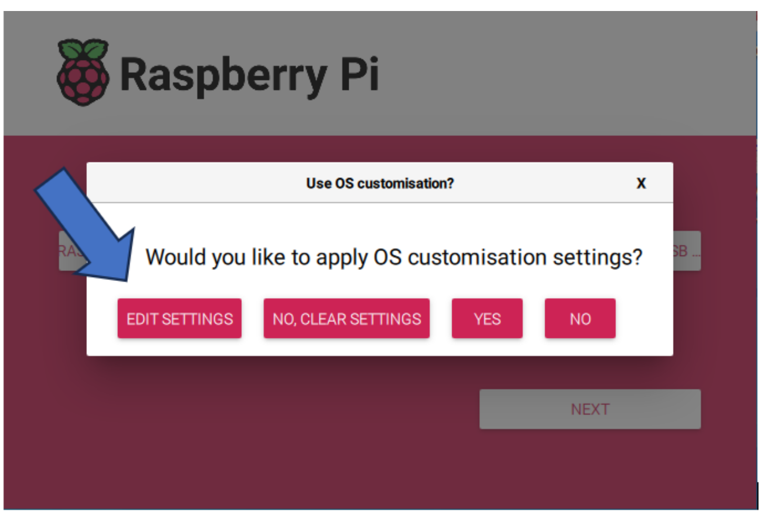
f. Select the ***Choose Storage*** disk (microSD card):



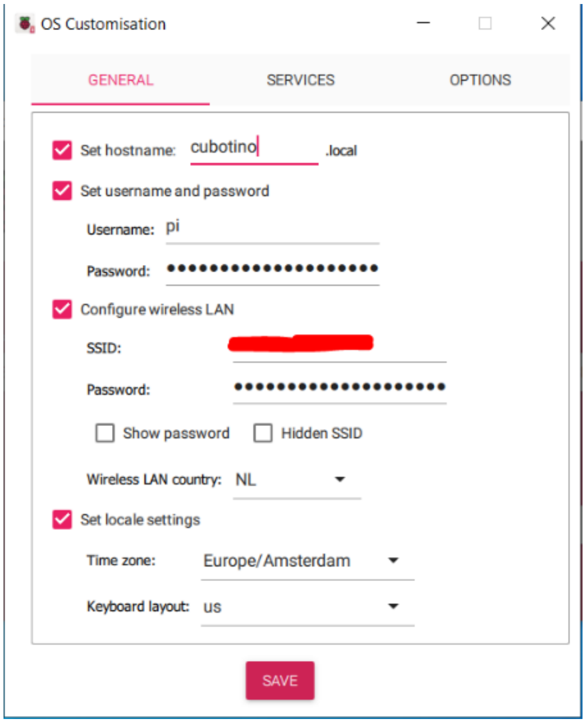
g. Select ***NEXT***:



h. Select ***EDIT SETTINGS***:

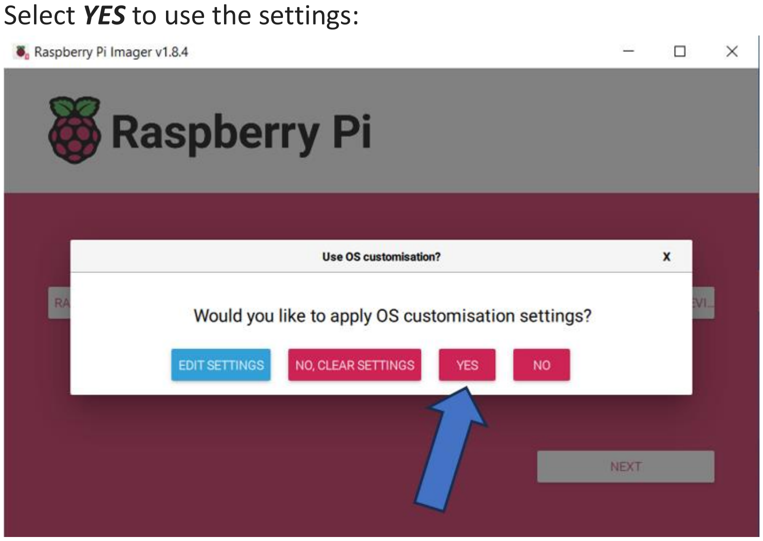


On settings, enter:

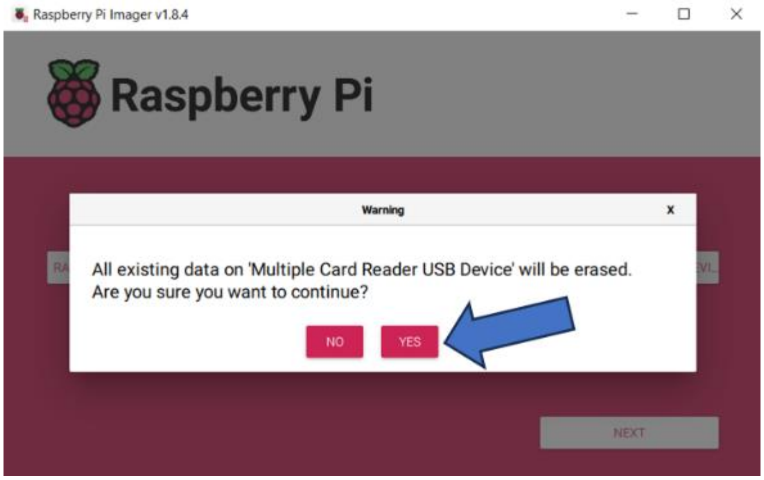


a. *Cubotino*.local
b. check SSH.
c. enter *pi* as username.
d. enter a password (my choice has been *raspberry,* as no sensitive data, but you are encouraged to use a more secure password).
e. check set Wi-Fi.
f. enter your SSID.
g. enter your network password.
h. enter your Country code.
i. set your local settings.
j. Press SAVE to store these settings (these settings will also be available the next time).

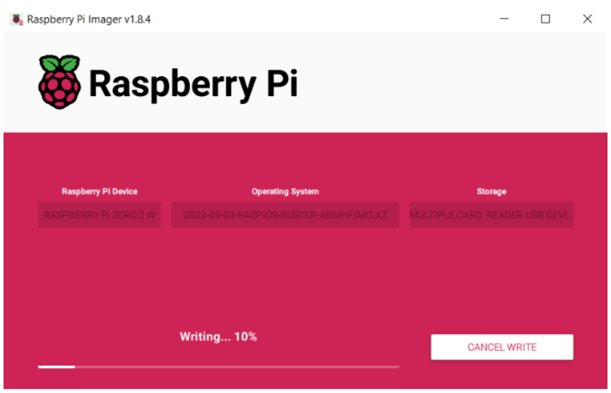
i. Select ***YES*** to use the settings:

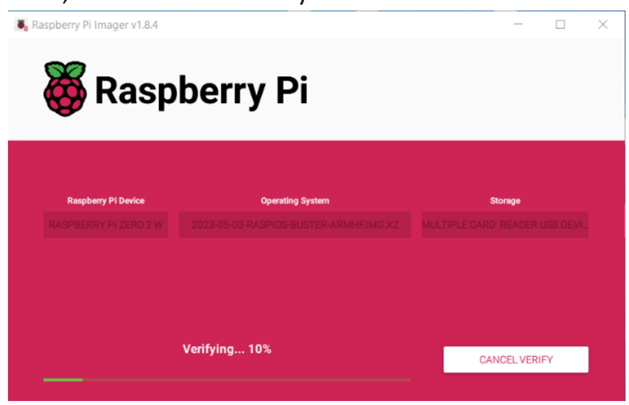


j. Select YES to start flashing the OS to the microSD:



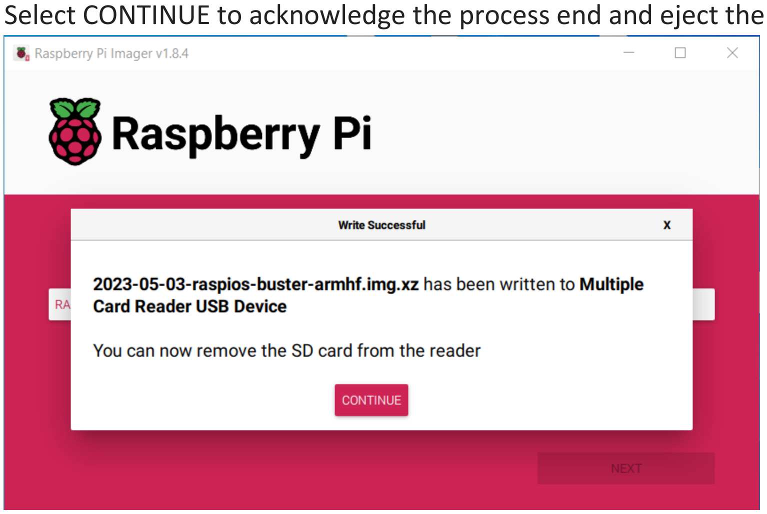
Raspberry Pi Imager starts writing the OS in the microSD, and it's followed by a verification:





Writing and verifying takes around 10 minutes.

k. Select CONTINUE to acknowledge the process end and eject the microSD:



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

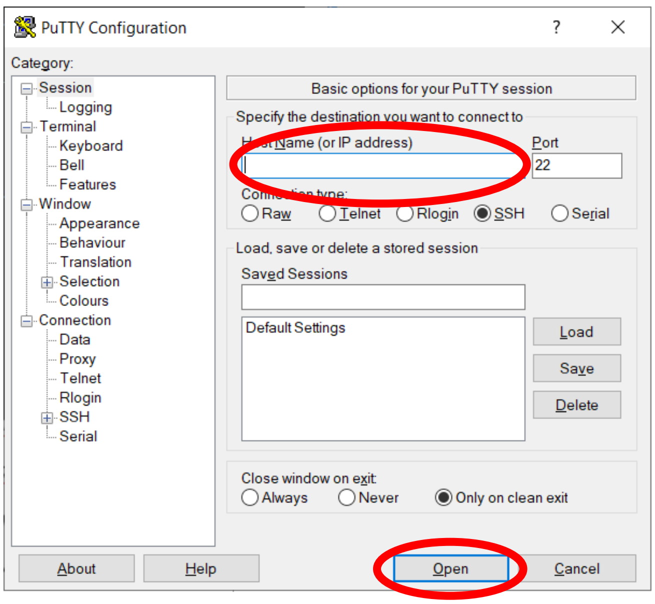
The first boot takes longer, up to two minutes on Zero2W and up to 4minutes on ZeroW.
Wait until the LED stops blinking.

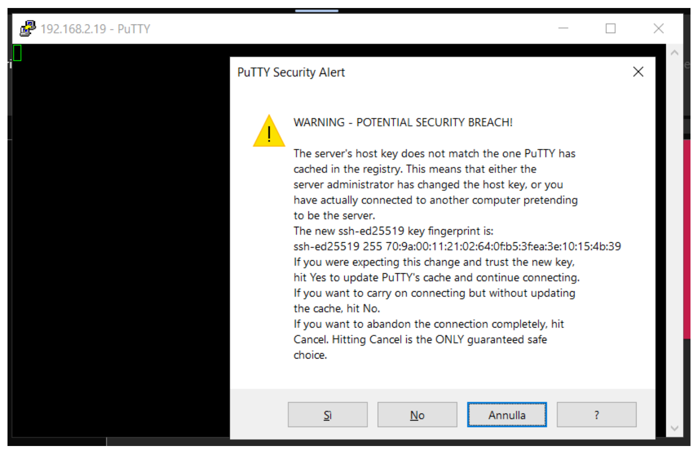
Step 4: from a command prompt try to connect to the raspberry Pi via *ssh pi@cubotino.local*; insert the password and jump to Step 6

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* (*raspberry* in my case)

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

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

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

```
pi@cubotino:~ $ git clone https://github.com/AndreaFavero71/cubotino_micro.git
Cloning into 'cubotino_micro'...
remote: Enumerating objects: 155, done.
remote: Counting objects: 100% (114/114), done.
remote: Compressing objects: 100% (92/92), done.
remote: Total 155 (delta 30), reused 77 (delta 15), pack-reused 41
Receiving objects: 100% (155/155), 116.54 MiB | 1.67 MiB/s, done.
Resolving deltas: 100% (33/33), done.
Checking out files: 100% (65/65), done.
pi@cubotino:~ $
```

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

Step 7: Start the installation:

- Enter cubotino/src folder from the root type: *cd cubotino_micro/src*
- 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. For a Raspberry Pi ZeroW consider about 20 minutes.
- 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_micro/*, the main robot folder, below folders will be made at Raspberry Pi:

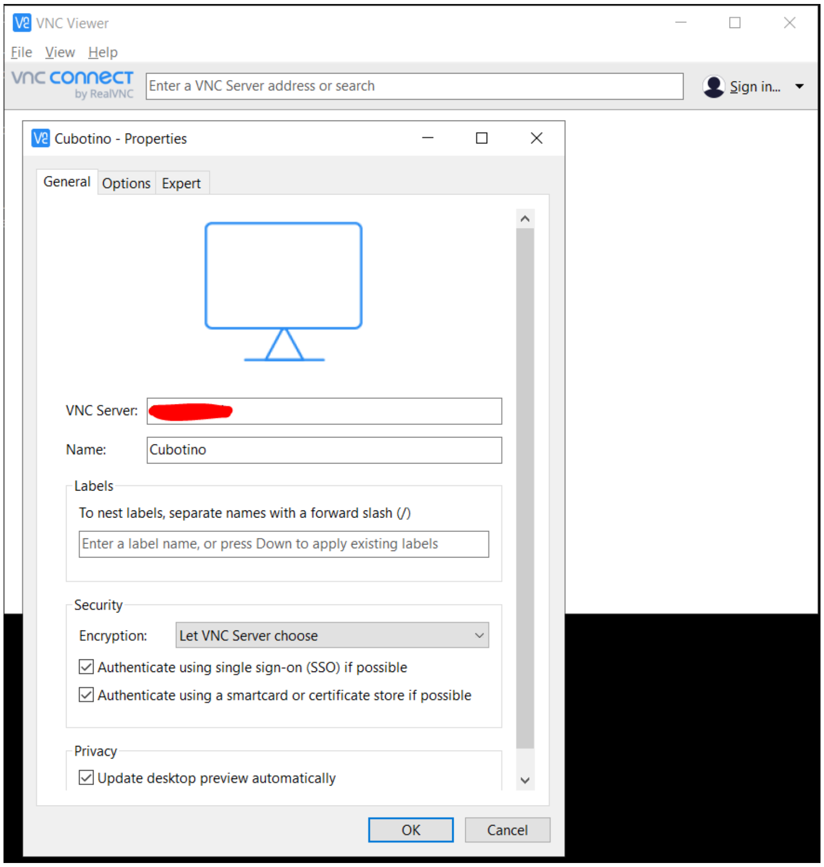
connections_board	info for the Connections_board
doc	How_to_build… .pdf file
extra	link to the Instructables page of this robot
images	Cubotino logo image for the display
stl	all the robot stl files
stp	all the robot stp files (not locally downloaded)
src	all the robot specific code files
src/twophase	lookup tables for the Kociemba solver
src/install	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/

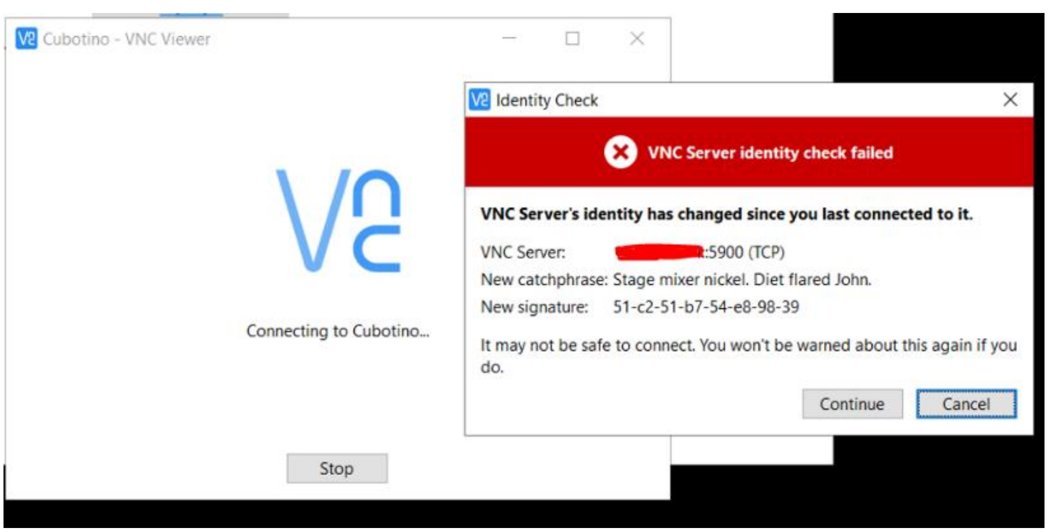
You must then register (for free) with RealVNC and create an account, to be able to set up a new connection with the Raspberry Pi board.

Step 8: Create a new connection in VNC Viewer

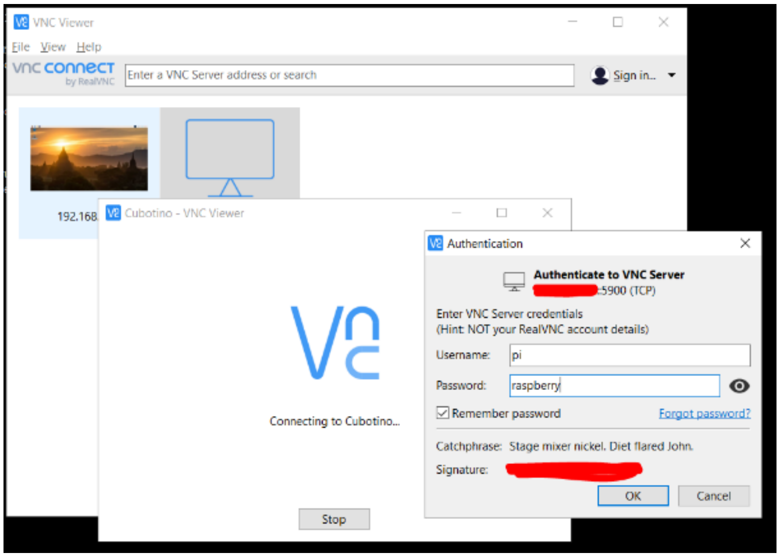
Create a new connection: File, New connections …., insert the IP address of your Raspberry Pi into the VNC Server frame, and a name, Cubotino, as an example, in the Name frame.



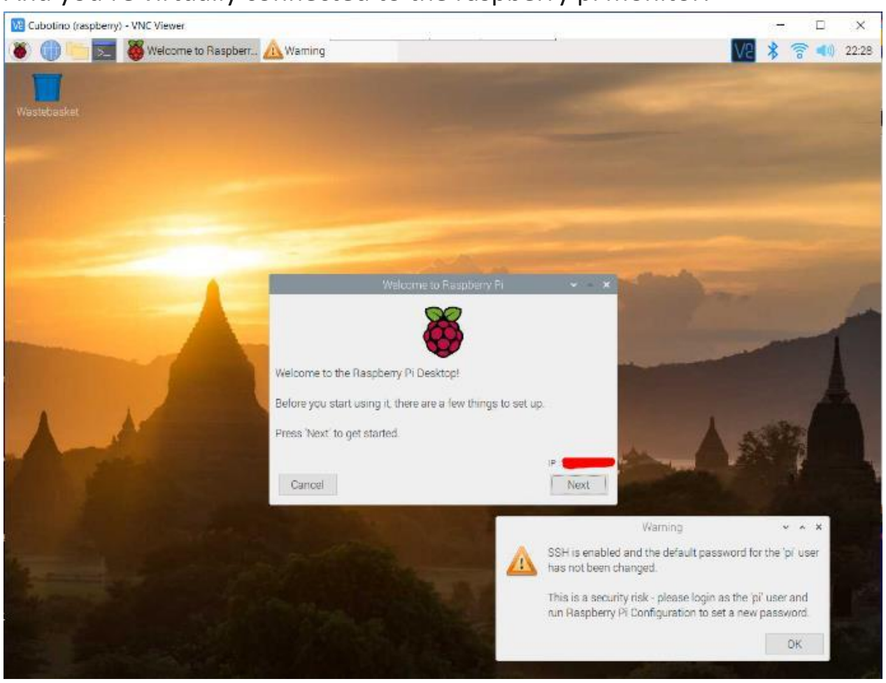
Accept the warning by clicking on “Continue”:



Insert username, *pi* in my case, and the password, *raspberry* in my case.
Check the option Remember password:



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



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 *raspberry*).
- Ignore the Setup screen and click on "continue" (see below).
- SKIP Select Wi-Fi Network.
- SKIP Update Software.

Step 9: Change the monitor size to use servos_settings_GUI.py, or in case you don't feel comfortable with the initial proposed resolution of 1280x720.

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

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

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

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

Choose 1-3 [1]:
```

The Crontab content will look like:

```
pi@cubotino: ~
File  Edit  Tabs  Help
 GNU nano 3.2                                  /tmp/crontab.Kkjq94/crontab

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

To tune the servos via the Graphics interface (GUI), it will be necessary to use a larger screen.

Comment the row with the 1280x720 pixel setting and uncomment the row with the 1920x1080 pixel setting.

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

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

Step10: 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

Step11: 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
}
```

Step12: 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 (51Mb), and very slow swap_memory writing (microSD) it is suggested to make more use of the cache memory than swap_memory, to maintain certain responsivity from the system.

Default swappiness = 60 → suggested value 20 (lower values reduces chances for microSD access).
Default cache_pressure = 100 → suggested value 200 (higher value increases the RAM cache).

Steps to modify swappiness and cache_pressure, from the terminal:

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

Step 13: Make a backup image of the microSD:

This is the perfect moment to secure the time spent to get here…..

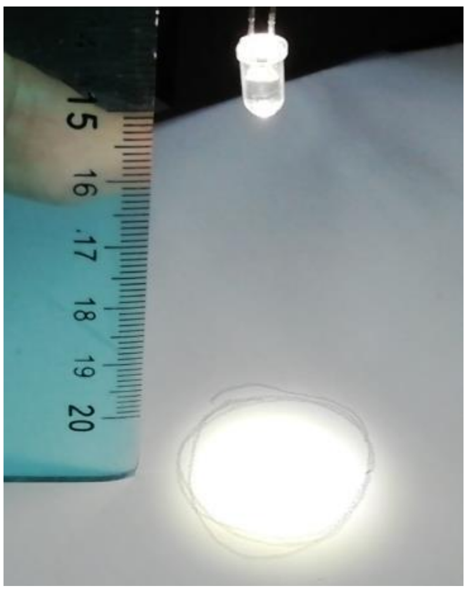
There are many tutorials available for this task, as reference: https://howchoo.com/g/nmexndnlmdb/how-to-back-up-a-raspberry-pi-on-windows

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.

13. White illumination Led: purchase or make a diffused light version



The hemispheric tip of a “common” 5mm LED, when pointing a surface at about 6cm distance, concentrates the light on a small spot: This clearly prevents a uniform illumination of the cube’s face.

At home I hadn’t any white Led with wide angle / diffused light.
Online shops offer this type of led but are sold on large quantity and/or at hight price.

No worry, there is often a DIY solution 😊

I had filed off the round tip and polished it by rubbing the LED over a piece of normal paper (one single sheet of paper for printers, laying over a hard and smooth surface, like a desk). Great result, isn’t it?

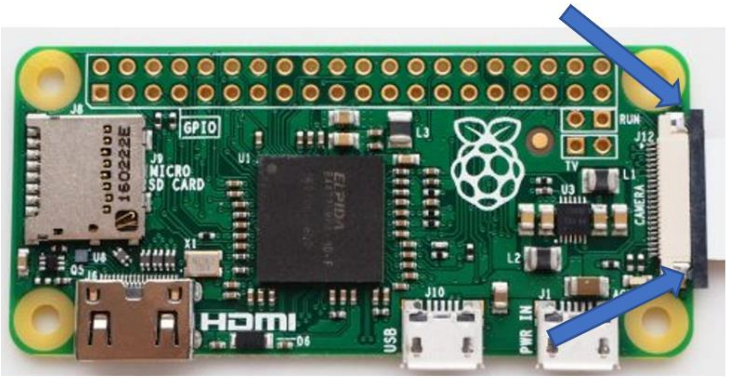
Afterward, I realized that a **non-polished finishing is less efficient, yet it better diffuses the light** 😊
Final finishing, for what a picture can show, is a LED with a smooth flat tip yet with limited translucency:

Filing and unpolishing the tip of the LED is also easy to do in a single operation with a DREMEL, using a rotary sanding tool with fine grit. Proceed slowly as the material of the tip disappears really fast!

14. Display and connections tests

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



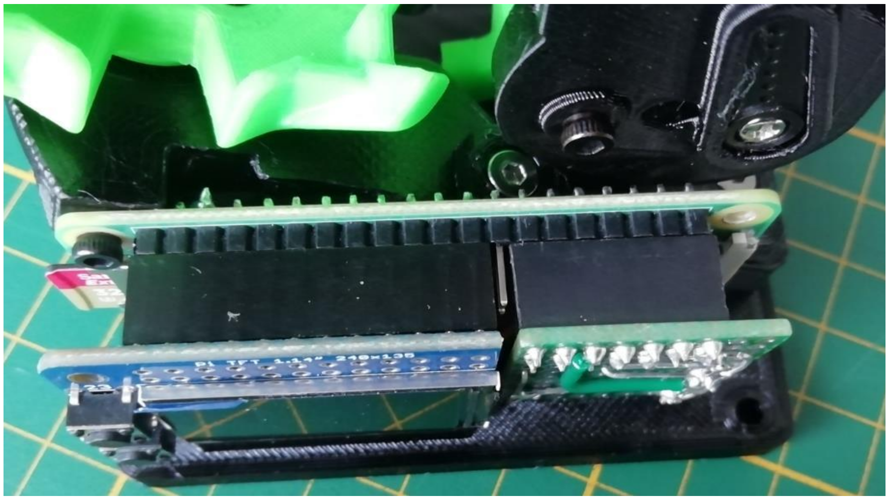
To unlock the ribbon cable connector of the Raspberry Pi Zero, push very gently the black part of the connector forward, in the direction of the arrows. Insert the narrow end of the ribbon in the Pi, making sure the connections side of the ribbon is facing down, then push very gently the black part in its original place, locking the ribbon. Make sure that the ribbon cable is fully seated in its connector before and after locking it. The ribbon should enter ~3mm into the connector.

Do the same on the camera side, with the wide end of the ribbon. The ribbon should enter ~4mm into the connector.

2. The servos to the Connections_board.

3. The connection board to the Raspberry Pi.

4. The display to the Raspberry Pi
   Connections_board and display are connected as follows:



   The display connector and the Connections_board are aligned to the opposite extremes of the GPIO connector: These leave one GPIO pin row unused.

Energize the circuit and wait until the Raspberry Pi green light stops blinking.
Connect to the Raspberry Pi via SSH (i.e., putty). If the installation went well, the connection should be possible.

Display test:

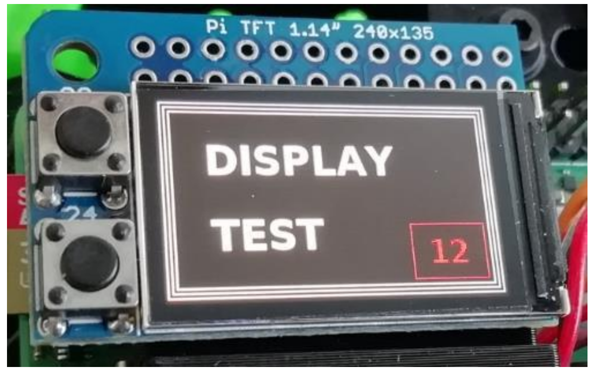
The display, and the buttons on it, can be tested:

a. enter the cube folder. From root, type *cd cubotino_micro/src*).
b. activate the virtual environment: type *source .virtualenvs/bin/activate* Pay attention to the dot before virtualenvs.
c. run the script *python Cubotino_m_display.py* .

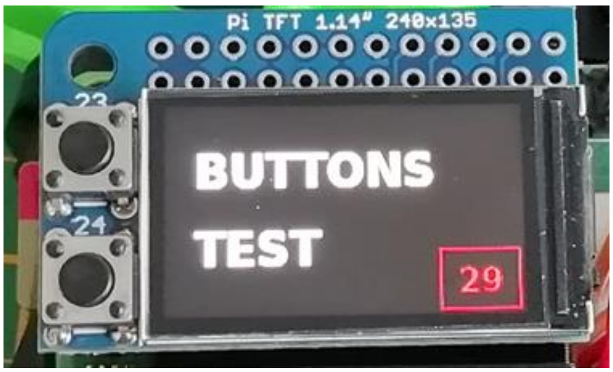
```
pi@cubotino:~ $ cd cubotino_micro/src
pi@cubotino:~/cubotino_micro/src $ source .virtualenvs/bin/activate
(.virtualenvs) pi@cubotino:~/cubotino_micro/src $ python Cubotino_m_display.py

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


Buttons test for 30 seconds
Text color changes when buttons are pressed
Buttons test finished

(.virtualenvs) pi@cubotino:~/cubotino_micro/src $
```

This test is split in two parts, the screen and the buttons:
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_offsetT) offset at Cubotino_m_settings.txt. See "Parameters" in "Appendix 1, Additional Information".

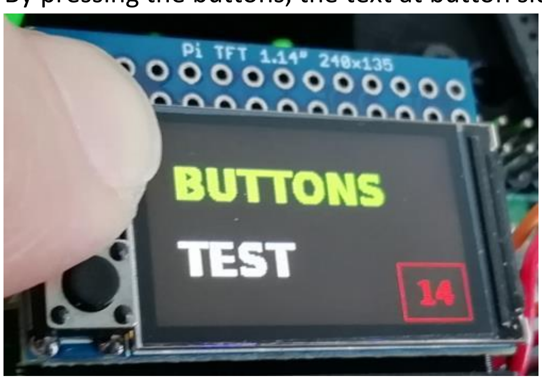
The time count-down is plotted in the little red square.
The screen test part can be interrupted by pressing any button.

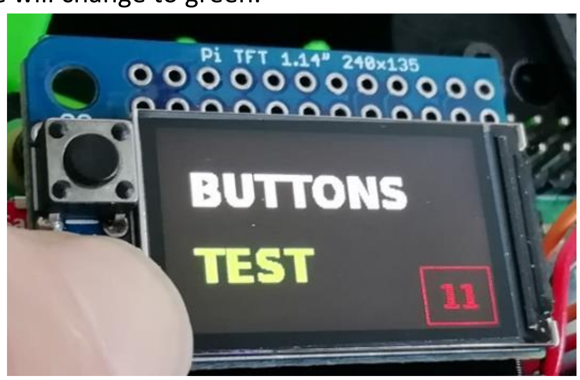


The last 30 seconds are meant to test the buttons:
Initially the screen shows “BUTTONS TEST” in white characters, and a time count-down into the small red square:

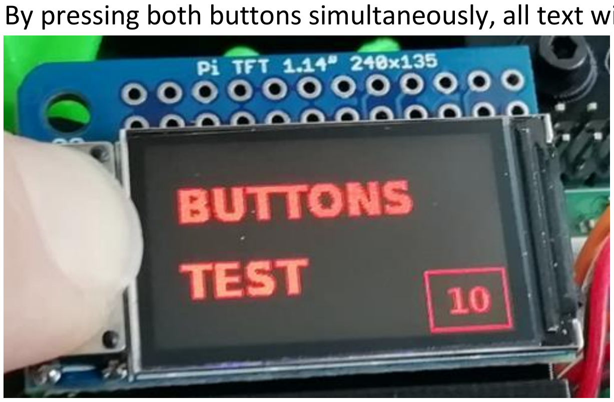


By pressing the buttons, the text at button side will change to green:





By pressing both buttons simultaneously, all text will change to red:



At the count-down end, the test ends.

Type *sudo halt -p* to shut it off and wait 10 s before removing the power supply from the Raspberry Pi.

15. 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 on 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_micro/src*
3. Activate the virtual environment: *source .virtualenvs/bin/activate*
4. Set the servos to the mid position: *python Cubotino_m_servos.py --set 0* (pay attention to the double '-' without space in between, and the space before the zero).

```
pi@cubotino:~ $ cd cubotino_micro/src
pi@cubotino:~/cubotino_micro/src $ source .virtualenvs/bin/activate
(.virtualenvs) pi@cubotino:~/cubotino_micro/src $ python Cubotino_m_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.

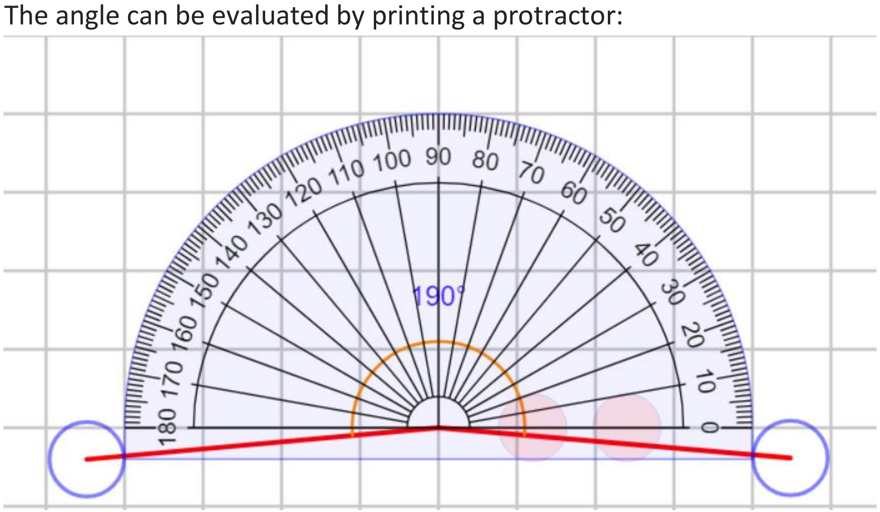
   Type any letter to quit:

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

Quitting Cubotino_m_servos.py

(.virtualenvs) pi@cubotino:~/cubotino_micro/src $
```

An alternative and easy 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).

On the servo which will drive the Top_cover, position the servo arm as per below picture.
Use the M2,5mm screw provided with the servo.

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!

16. Test the PiCamera

1. Connect via VNC
2. Enter the cube folder: *cd cubotino_micro/src*
3. Activate the virtual environment: *source .virtualenvs/bin/activate*
4. Run the camera test: *python Cubotino_m.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 6cm distance, if not proceed with the following "Set the correct focus" instructions.

17. Set the correct focus for the PiCamera

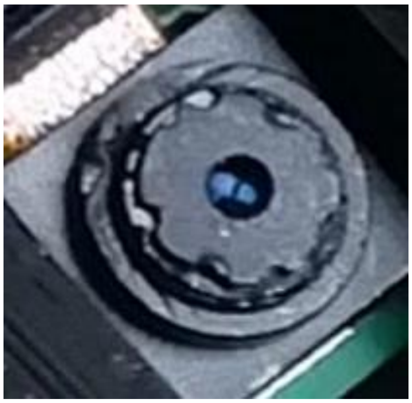
The V1.3 PiCamera has fixed focus.
Because of the very short distance between the PiCamera and the cube face, it's necessary to adjust the focus.

To verify the camera focus, activate the PiCamera as indicated above "Test the PiCamera
The camera will be activated, and its image sent via the VNC:
After 20 seconds the test ends, and the command can be repeated as long as needed.
Check if the focus is roughly ok at 6cm distance, if not proceed with the following instructions until the focus is satisfactory.

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

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

- Remove **all** the glue, all around, **before** trying to rotate the lens.
- Use a cutter with a new blade, to have a very sharp 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 clue. I did not have to glue the lens, as there was still quite a bit of friction, preventing the lens from getting loose.

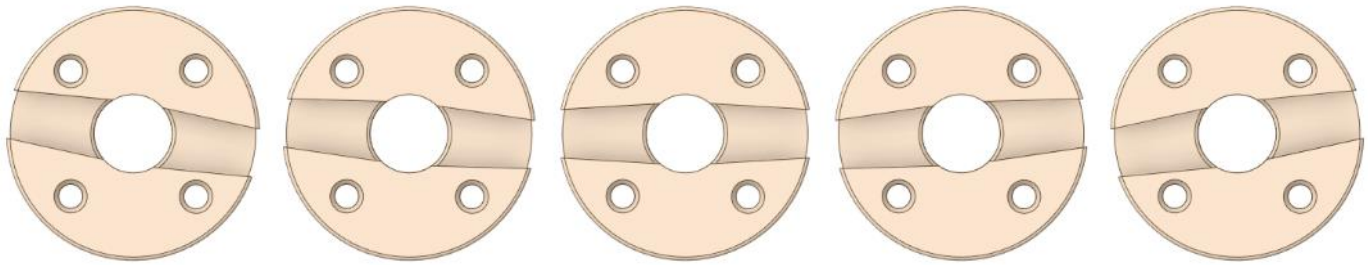


18. Servo offset compensator selection

5 different "Servo offset compensator" (SOC) geometries are provided, to accommodate differences between servos. The **stl** files are available at: https://github.com/AndreaFavero71/cubotino_micro/tree/main/stl STL files get downloaded to the Raspberry Pi when cloning the repository.

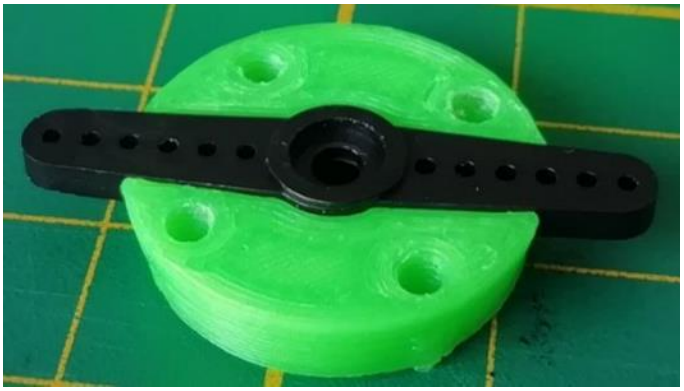
These SOCs versions are differentiated by the angle between the screw's holes and the recess for the servo arm:

SOC_-9	SOC_-5	SOC_0	SOC_+5	SOC_+9



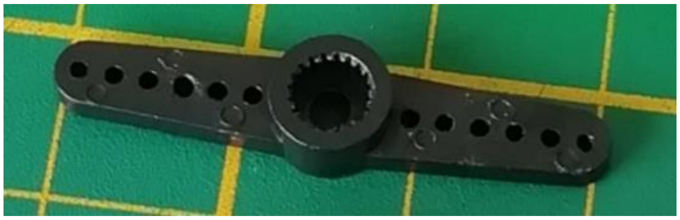
The overall target is to have the 4 little holes along the main robot axes, when the servo is set to its neutral position, as the Cube_holder is attached to those holes. Small corrections will be made by the servos settings (tuning chapter).

Below image shows the servo arm into a SOC_+9
(this is my specific robot case)



Below image:

how the servo arm to be used looks like:

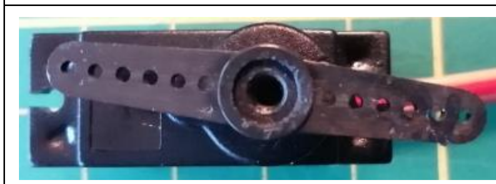


Only one SOC is needed, and it must be selected according to the bottom servo output gear. Some notes:

1. Servo and servo arm have 20, 21 or 25 teeth coupling geometry, meaning a step of 18, 17 or14 degrees per tooth, depending upon the brand and type of servos you are using.
2. The SOC versions with (+/-) 9 degrees corrections, can roughly recover for half of the offset angle corresponding to one tooth shift.
3. The SOC versions with (+/-) 5 degrees corrections, can roughly recover for one quarter of the offset angle corresponding to one tooth shift.
4. Use SOC_+5 or SOC_+9 when the servo arm has a positive angle (CCW) from the servo body main axis when the servo is set to its middle position.
5. Use SOC_-5 or SOC_-9 when the servo arm has a negative angle (CW) from the servo body main axis when the servo is set to its middle position.

Procedure to select the proper SOC for your case:

1. Keep the servo energized, after sending the "-set 0" argument (see previous chapter), to make use of the motor torque to maintain the output gear position.
2. When the servo is set to its neutral position, gently insert the arm, by searching the best fitting to minimize the angle between the servo arm and the servo main axis.
3. Very likely the servo arm won't be perfectly aligned to the servo body, but this isn't the target.
4. The target is to have the Cube_holder aligned with the servo main axis, without being forced to deviate much from mid PWM value (1500 µs). This allows to have a roughly symmetrical rotation from the neutral position to the two extreme positions.

"Best arm position" with servo in its neutral position	Notes	SOC to use
	This is the ideal situation, having the arm perfectly aligned with the main servo body axis	SOC_**0**
	The servo arm is rotated 18deg CCW: it will be sufficient to shift the arm insertion by one tooth CW.	SOC_**0**
	The servo arm is rotated ~9 deg CCW	SOC_**+9**
	The servo arm is rotated ~5 deg CCW	SOC_**+5**
	The servo arm is rotated ~5deg CW	SOC_**-5**
	The servo arm is rotated ~9deg CW	SOC_**-9**

In my opinion the SOC +/- 5 shouldn't be necessary, but it was little effort to add those geometries and to include them in the instructions.

19. ACT-LED Activation (optional)

If installed in your Cubotino, the ACT-LED at Connections-board needs to be activated.
The Connections_board, made as per gerber files or modified perfboard, has a LED (LED1 and R2) meant to ACT as per Raspberry Pi LED.
To get that functionality it is necessary to uncomment one command at the config file:
From a CLI (sudo rights are needed):

- Type *sudo nano /boot/config.txt*
- The file ends with:

```
dtparam=spi=on
start_x=1
gpu_mem=128
enable_uart=1
#dtoverlay=act-led,gpio=6
```

- Uncomment the last row (`dtoverlay=act-led,gpio=6`)
- Save and confirm the file name: Ctrl+X followed by Y
- Reboot: *sudo reboot*

Now the ACT LED is the one at Connections_board (gpio 6)

Note: The ACT LED is displayed either on the Raspberry Pi board or through the specified GPIO pin, not both.

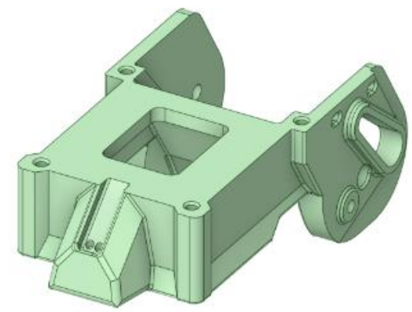
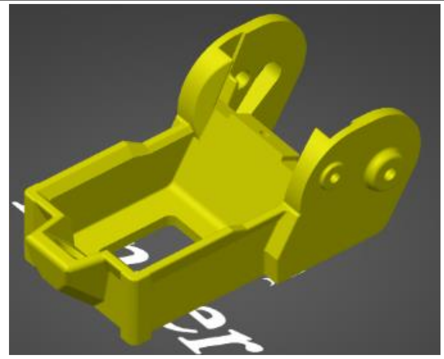
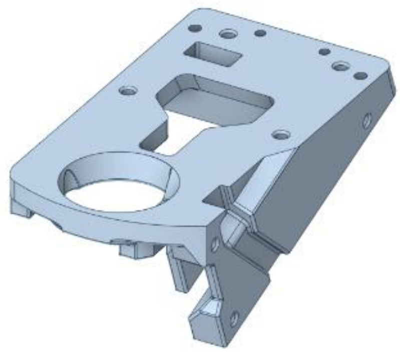
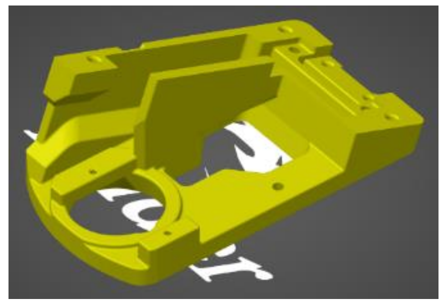
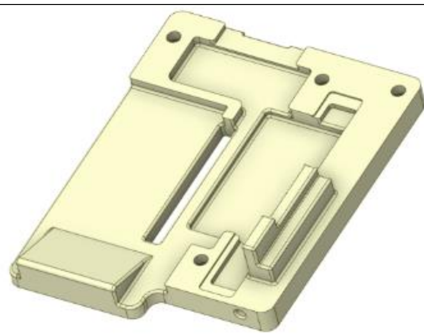
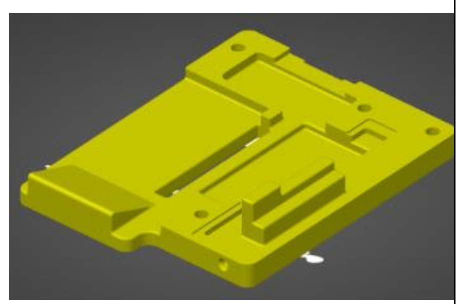
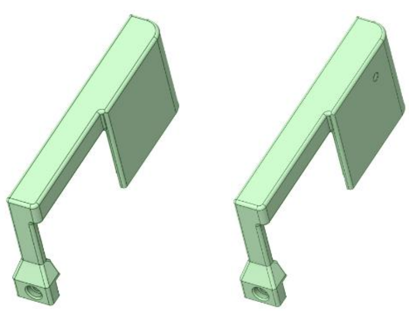
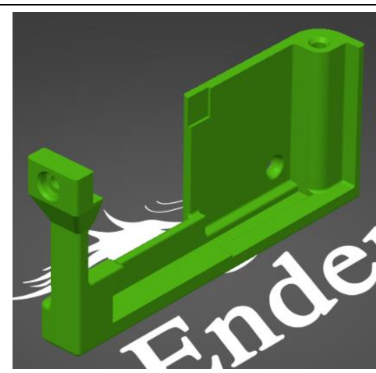
20. 3D printed parts

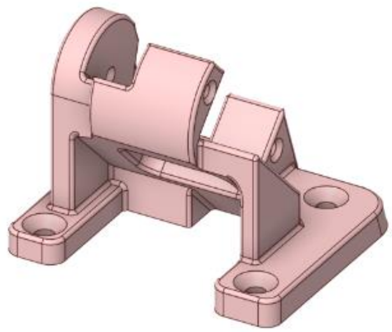
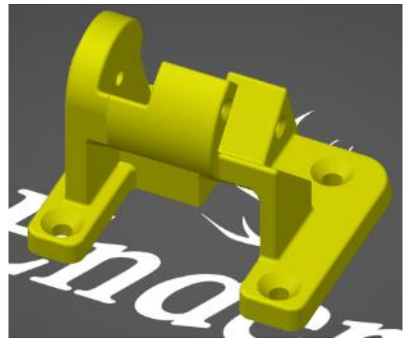
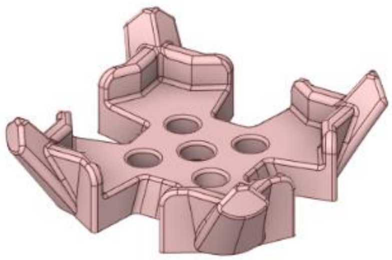
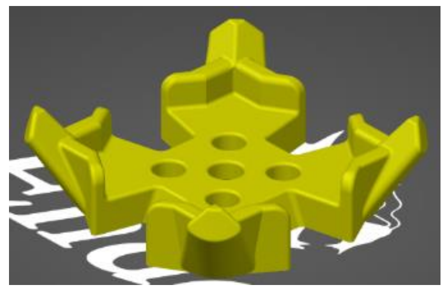
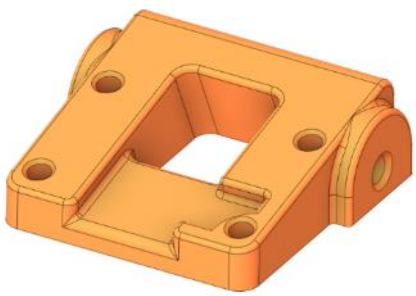
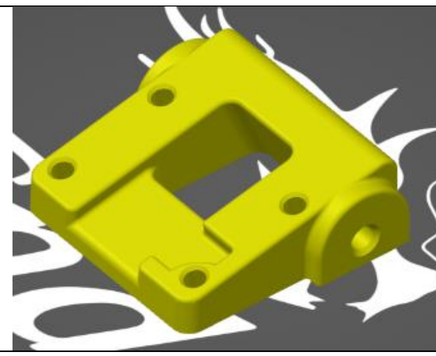
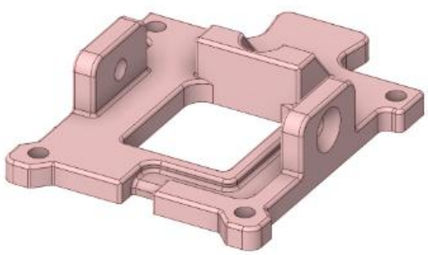
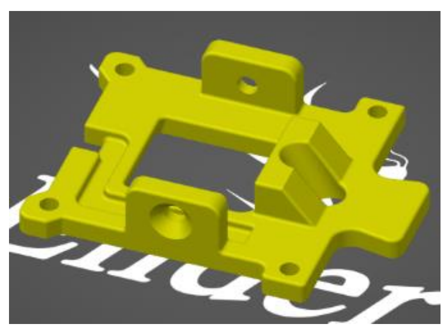
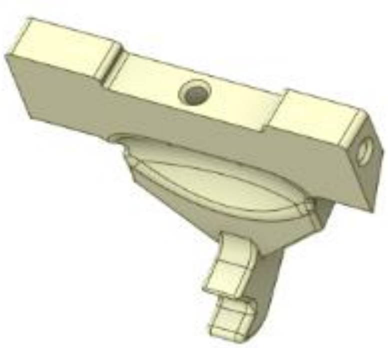
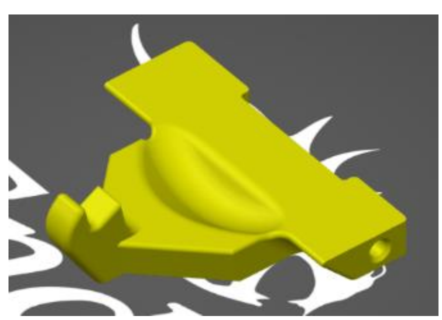
See notes below for filament quantity, and printing time …

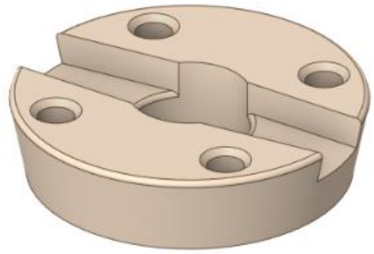
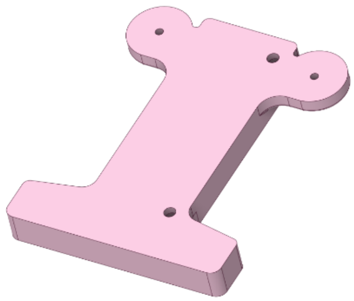
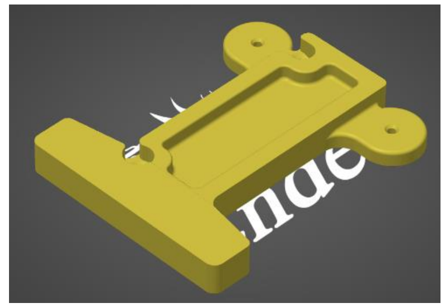
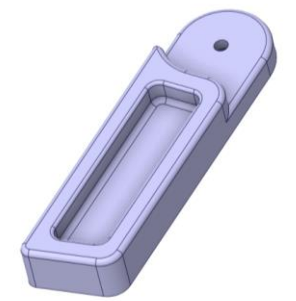
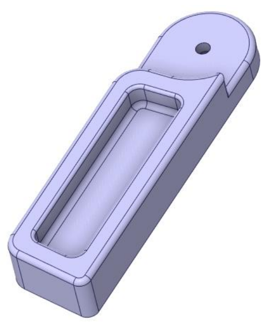
Ref	Part	Filament Grams	Printing time	Notes
1	Structure	27	3h12m	
2	Top_cover	26	3h25m	
3	Baseplate	25	2h38m	
4	Cover or alternative Cover_led_hole	11	1h29m	
5	Hinge	13	1h07m	
6	Holder	8	1h03m	
7	Picamera_frame	8	1h02m	
8	Picamera_holder	5	0h38m	
9	Lifter	4	0h31m	
10	Servo_offset_compensator (SOC), formal Servo_axis)	2	0h15m	It might be convenient to print all 5 versions at once, and later check which one better fir your servo.
11	Baseplate_addition	22	2h09m	Optional, to increase stability
12	Baseplate_L_leg	8	47m	Optional, to increase stability
13	Baseplate_R_leg	8	47m	Optional, to increase stability
	TOTAL	**~ 130g** **~ 40g**	**~ 16h** **~ 4h**	**Necessary part** **For the “extensible” base**

Notes:

1. Filament length is based on Ø1.75mm.
2. 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:
    1. 0.2mm layers
    2. Low speed (between 25 to 40mm/s for the external parts and 1st layer)
    3. 4 layers on vertical walls
    4. 5 layers on horizontal walls
    5. 30% filling
    6. 8mm brim
    7. 5 mm retract length
3. All parts have been designed to be printed without supporting the overhangs (**no support needed**).
4. Some parts have been split, for easier and better 3D printing.
5. The suggested part orientation for the 3D print is shown on below Table.
6. The **stl** files are available at: https://github.com/AndreaFavero71/cubotino_micro/tree/main/stl STL files get downloaded to the Raspberry Pi when cloning the repository.
7. The **step** files are only available at https://github.com/AndreaFavero71/cubotino_micro/tree/main/stp Step files do not get downloaded to the Raspberry Pi when cloning the repository.

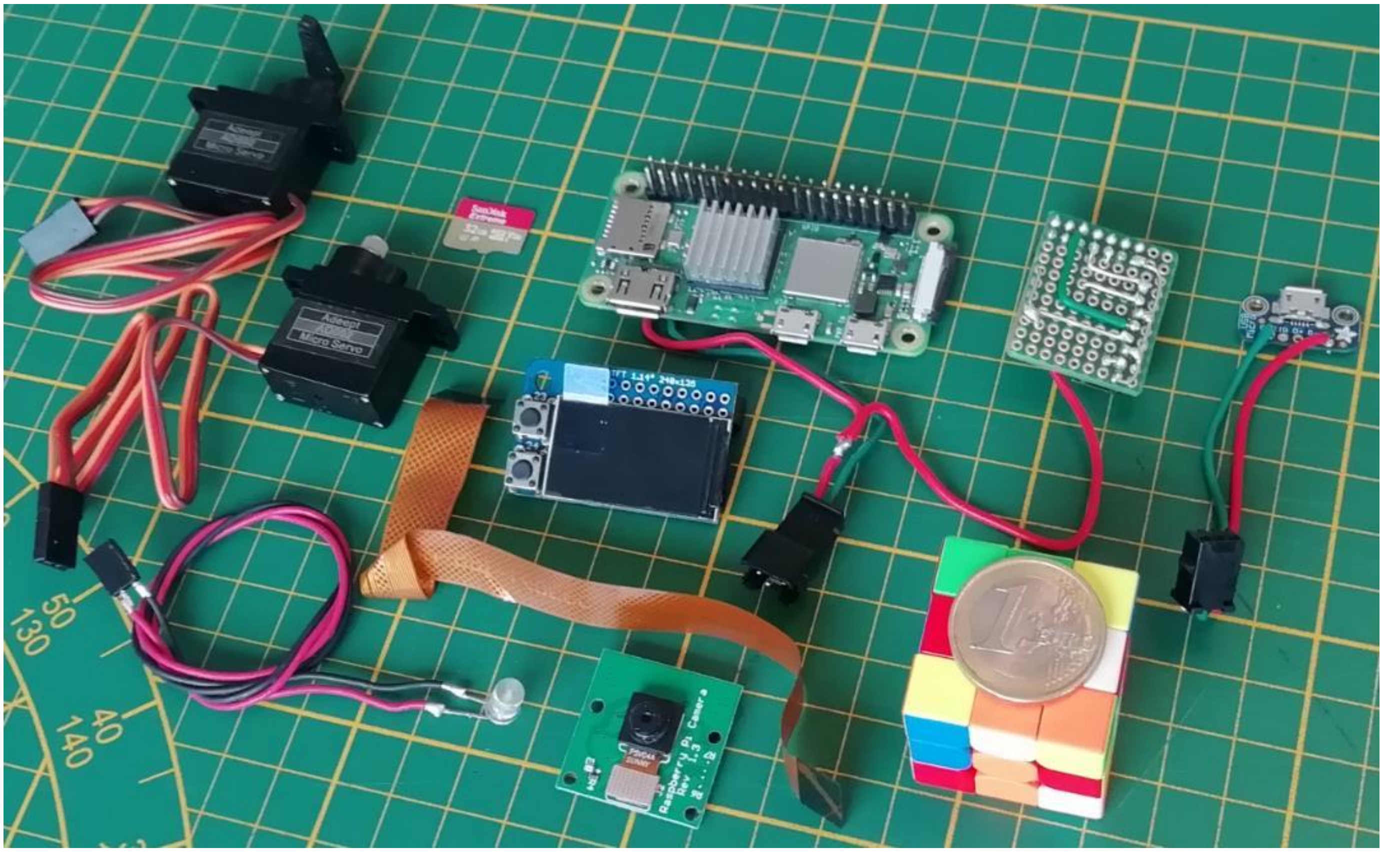
Part name	image	3D print orientation
Structure		
Top_cover		
Baseplate		
Cover or alternative Cover_led_hole		

Hinge		
Holder		
PiCamera_holder		
PiCamera_frame		
Lifter		

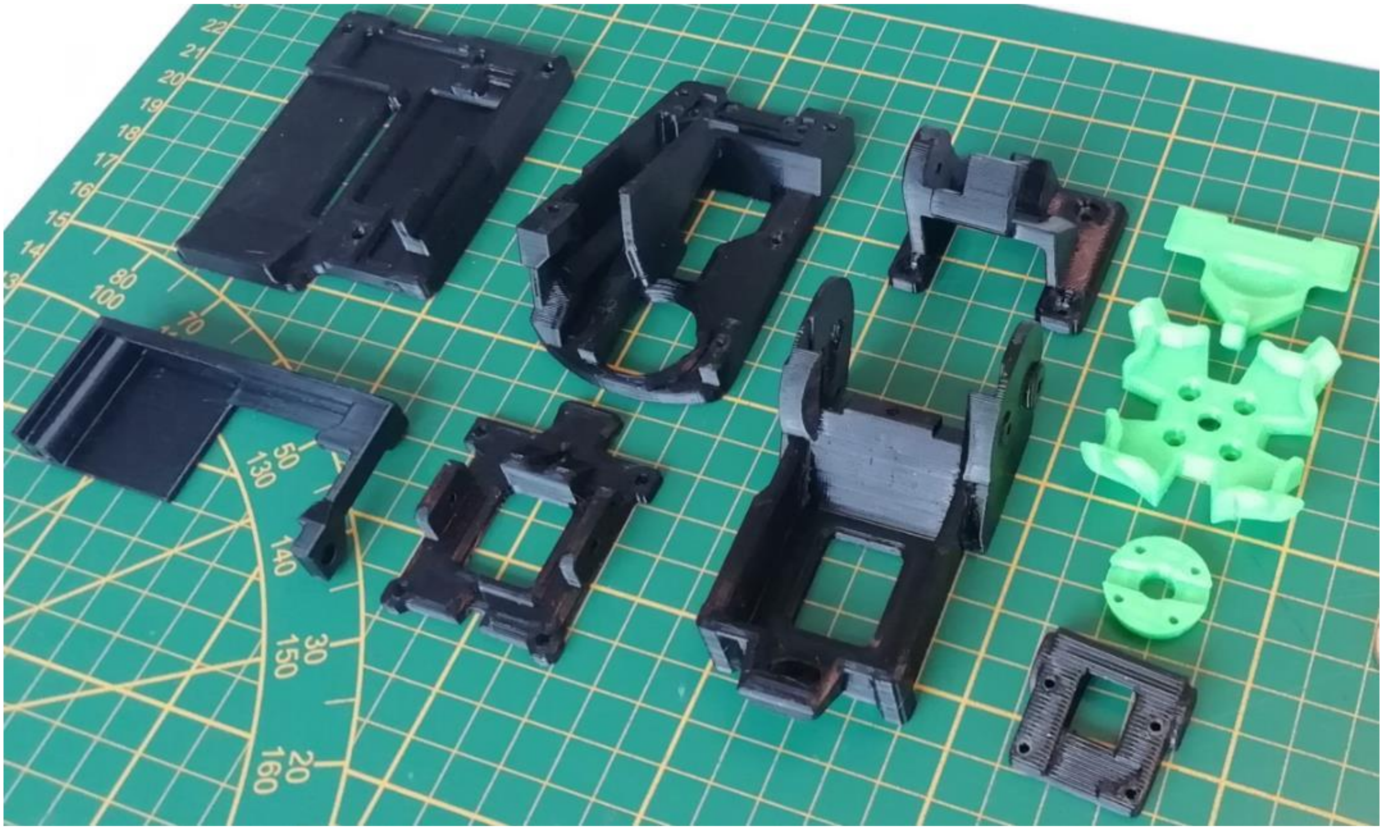
Servo offset compensators		
Baseplate_addition		
Baseplate_L_leg		
Baseplate_R_leg		

21. Parts overview

Electrical parts:



3D printed parts:



22. Assembly

Notes:

- Some of the assembly steps are not very friendly and might require to remain calm…
- On the printed parts you might form the thread, by using a screw, before starting the assembling; If the torque is high, rub the screw with candle wax.
- As a reminder: YOU ARE ASSEMBLING THE WORLD'S SMALLEST RUBIK'S CUBE SOLVER ROBOT !
- Finally, this robot has been developed by a single person and made available for your fun and enjoyment.

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

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

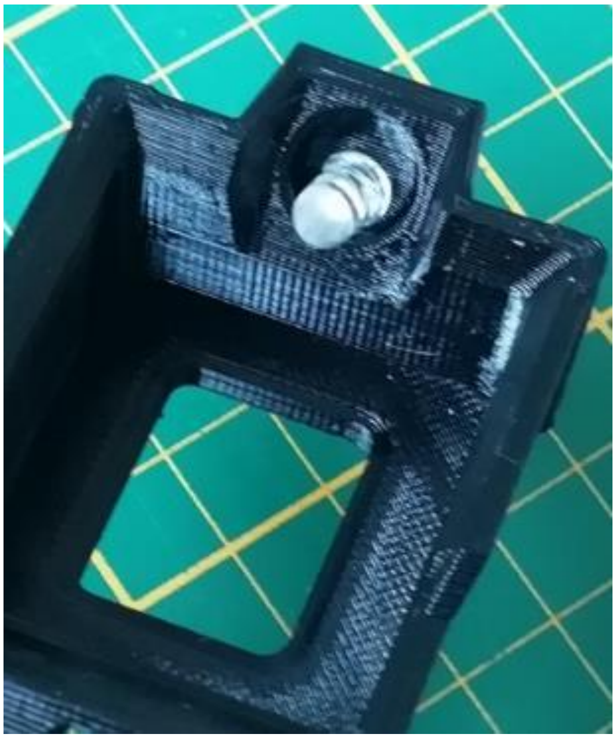
1. Assemble the LED to the Top_cover.
2. Screw the lifter to the Top_cover.
3. Screw the PiCamera_frame to the Top_cover.
4. Screw the PiCamera to the PiCamera_holder.
5. Screw the PiCamera assembly to the PiCamera_frame.
6. Assemble the hinge to the structure.
7. Screw the bottom servo to the structure.
8. Assemble the top servo with the Top_cover and the hinge.
9. Dress the cables.
10. Mark the servo connectors.
11. Compact the excess cables.
12. Screw the Raspberry Pi assembly onto the structure.
13. Assemble the microUSB breakout board on the structure.
14. Connect the power supply.
15. Connect the servos and the LED to the Connections_board.
16. Attach the Connections_board to the rightmost pins of the Raspberry GPIO.
17. Pack the cables into the structure voids.
18. Assemble the Base_plate.
19. Assemble the Cube_holder to the bottom servo offset compensator.
20. Assemble the Cube_holder assembly onto the bottom servo axis.
21. Attach the display.

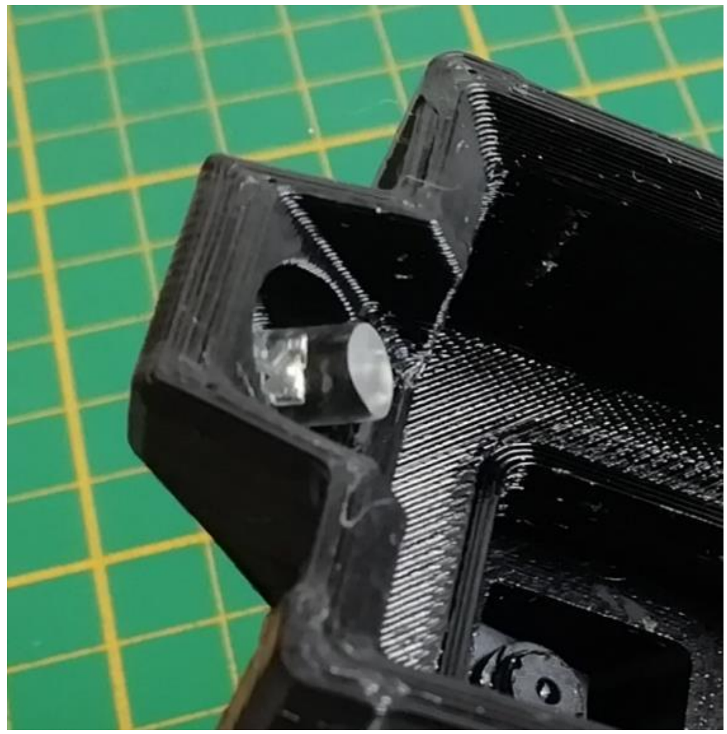
22. Connect the flexible cable to the PiCamera module.
23. Connect the flexible cable to the Raspberry Pi.
24. Assemble the cover.
25. Tire wrap the flexible cable to the PiCamera_frame.
26. Optionally, assemble the additional Baseplate.

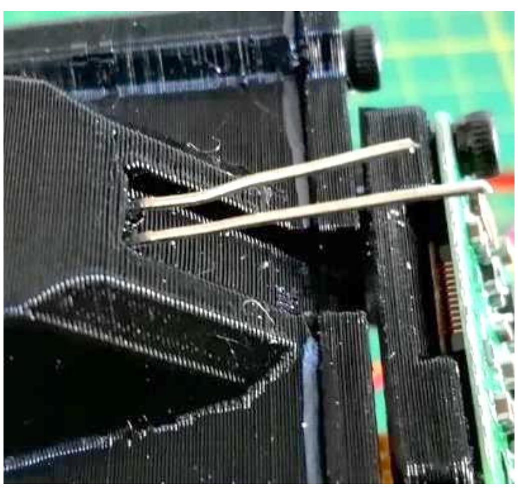
Each of these steps are described in detail in the following pages.

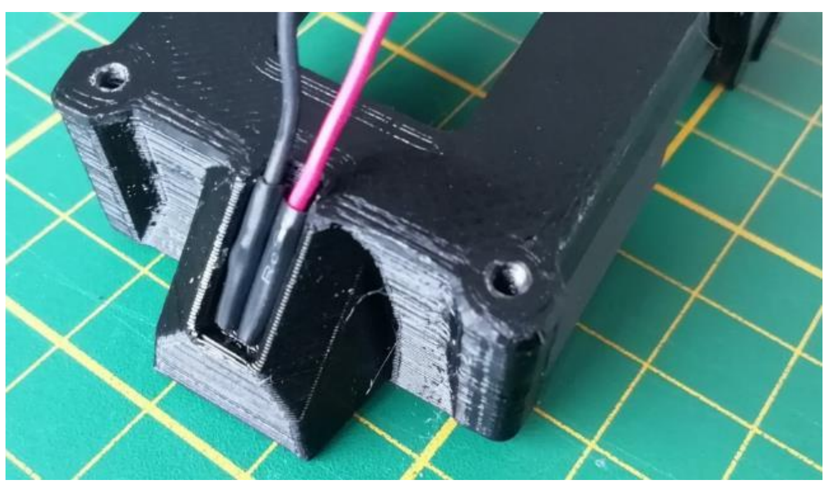
Step 1: Assemble the LED to the Top_cover.

- Use a LED with flat tip (see chapter 13)
- Insert the Led terminals through Top_cover holes.
- Shorten the terminals to about 8~10mm. Keep the positive one slightly longer, to prevent later mistakes.
- Solder ca 30 cm of wires.
- Insert 10/15 mm of heat shrinking tube.
- Keep the Led pressed in position while bending its terminal (press the terminal at their lowest part).
- Solder/connect the free wires terminals to a 2 pins female header.



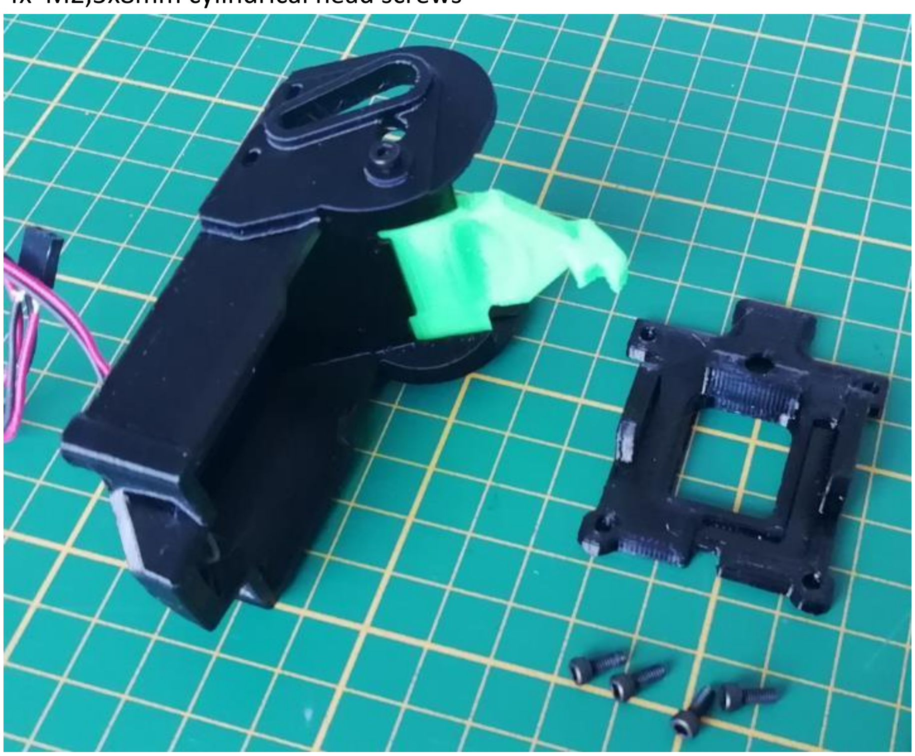






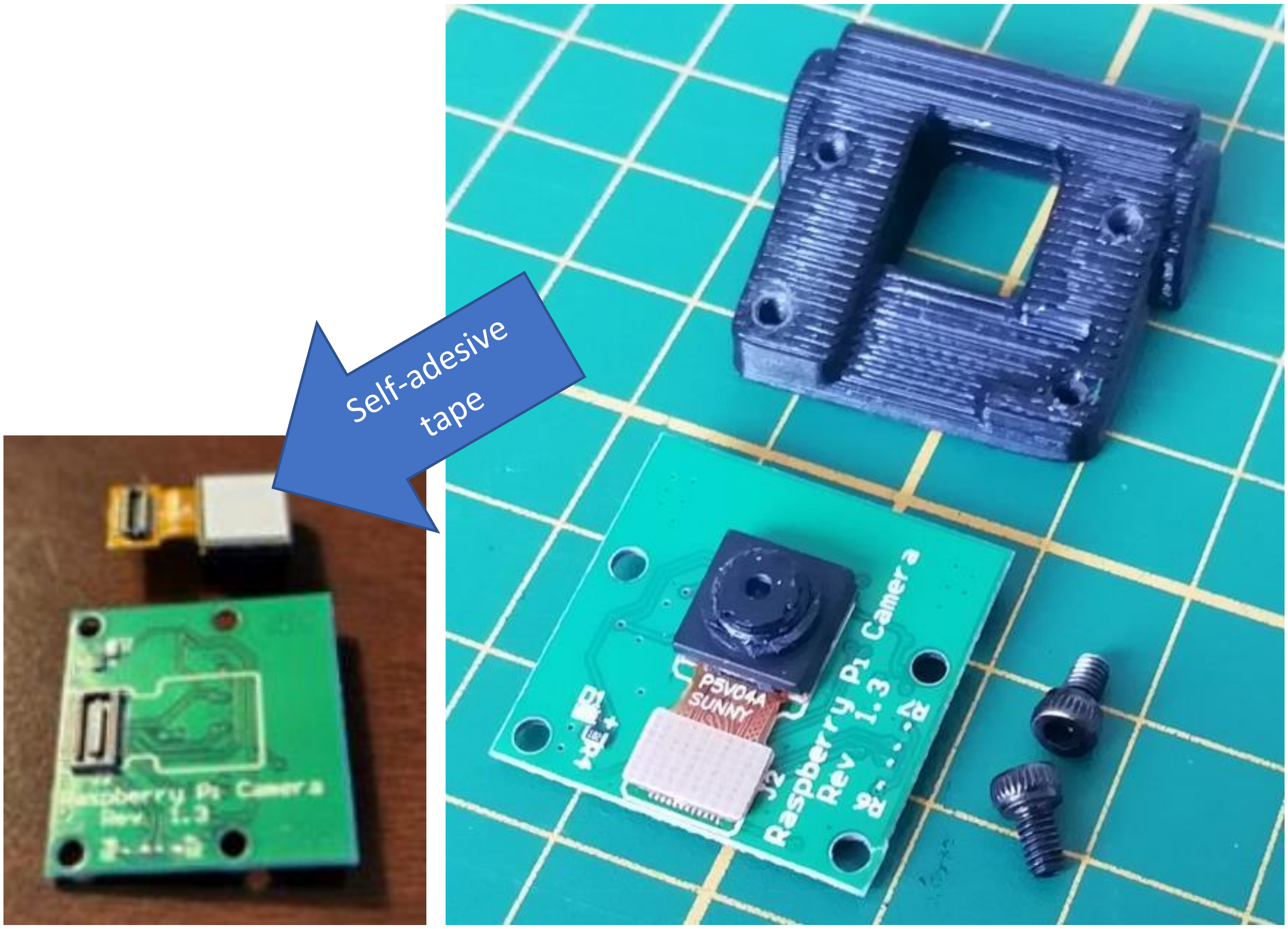
Step2: Screw the Lifter to the Top_cover.
3x M3x8mm conical head screw

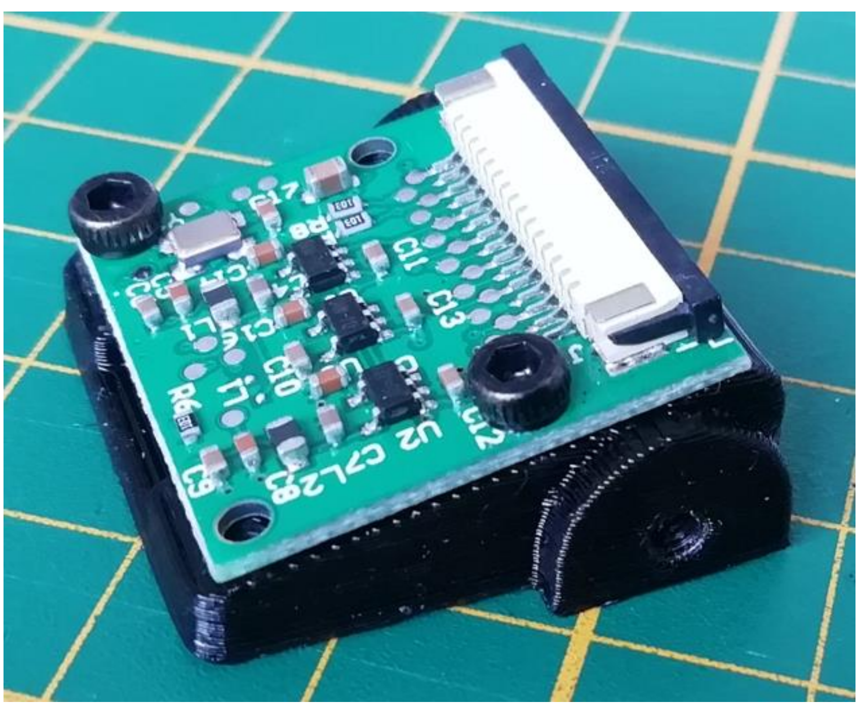
Step3: Screw the PiCamera_frame to the Top_cover.
4x M2,5x8mm cylindrical head screws



Step4: Screw the PiCamera to the PiCamera_holder.

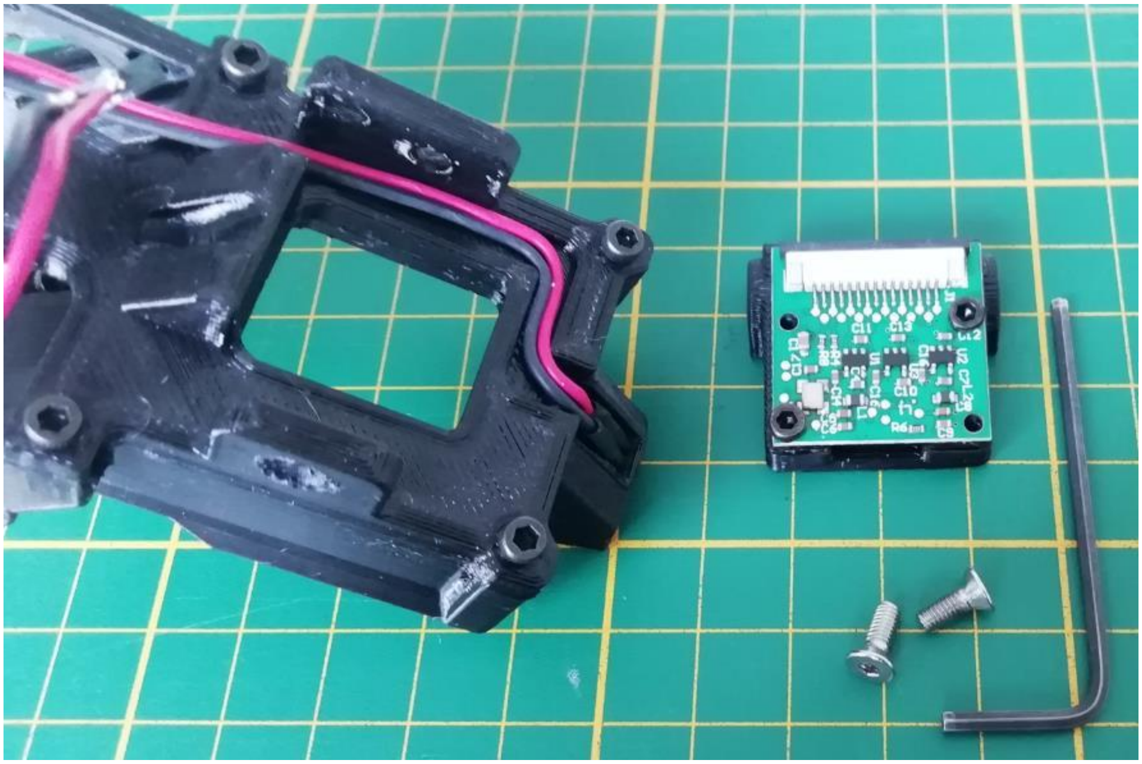
- Slightly enlarge the holes of the PiCamera holes, as very tight for a 2,5mm screw
- Stick the camera sensor to the PCB, by using its self-adhesive tape.
- 2x M2,5x4mm cylindrical head screw are sufficient (of course 4 screws will have a better appearance)

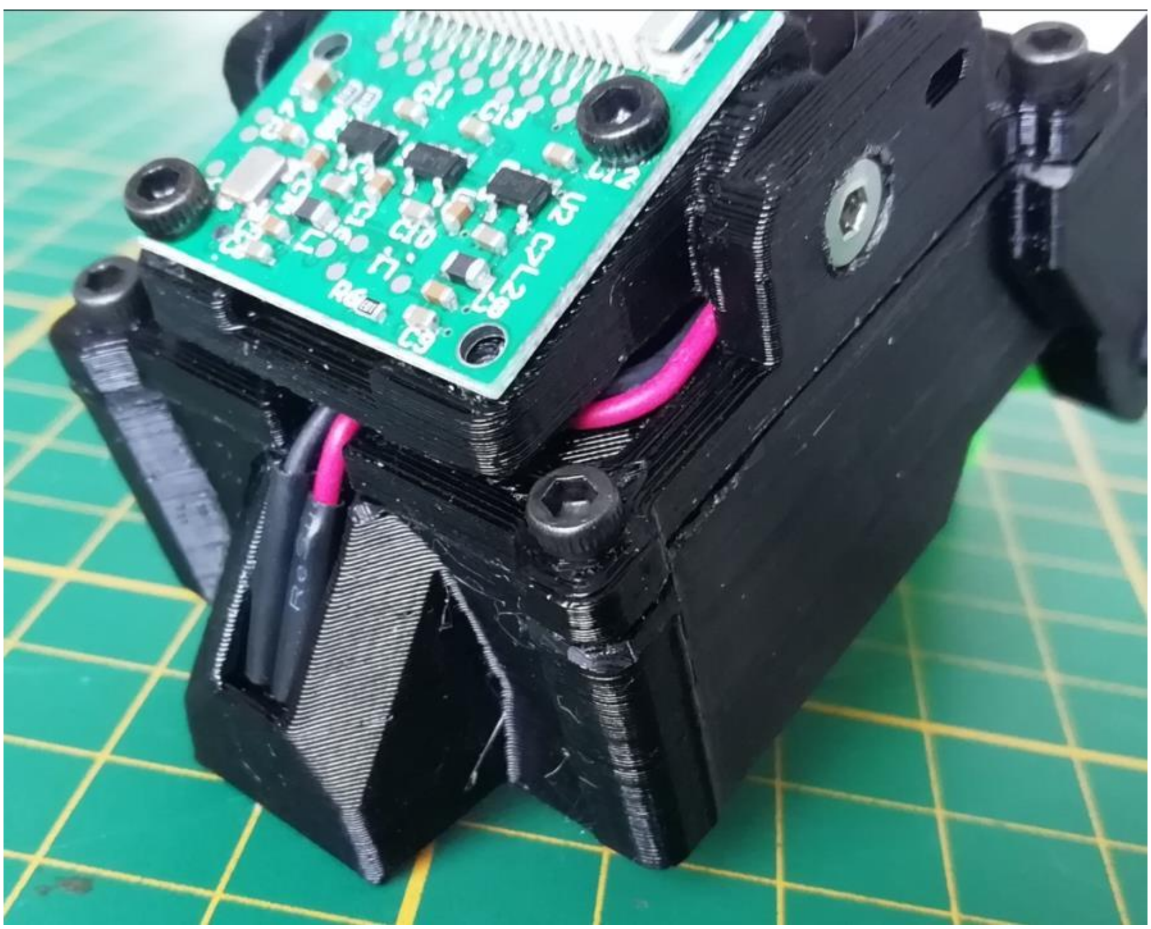




Step5: Screw the PiCamera assy to the PiCamera_frame.

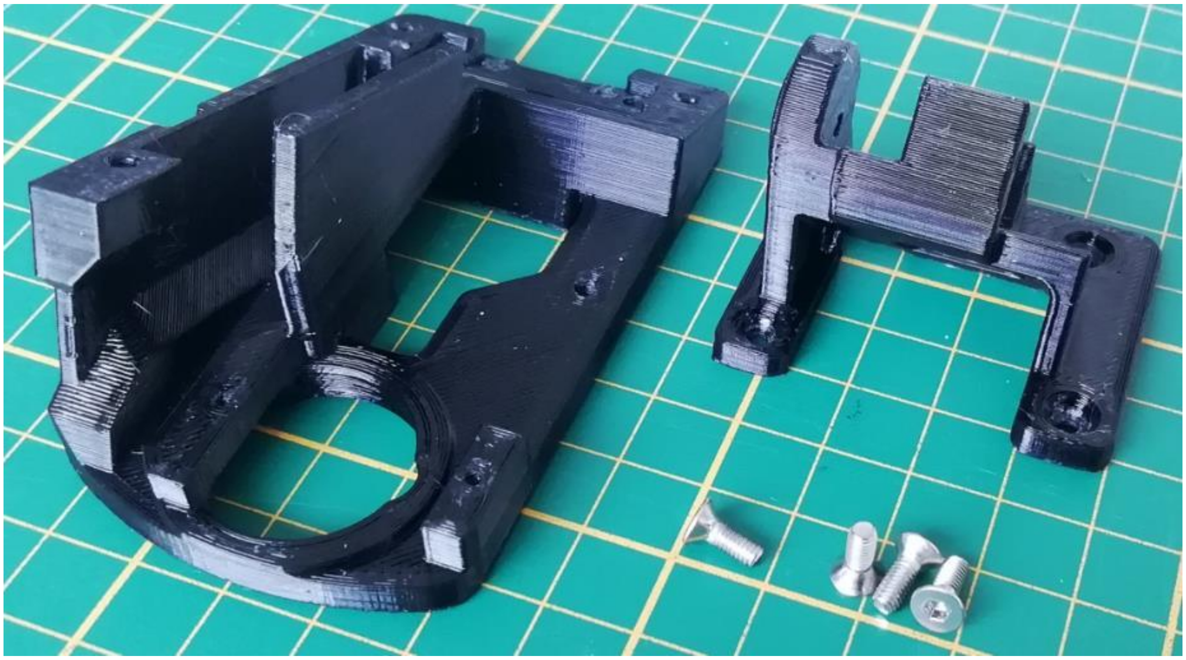
- Dress the cables into the groove.
- 2x M3x8mm conical head screws

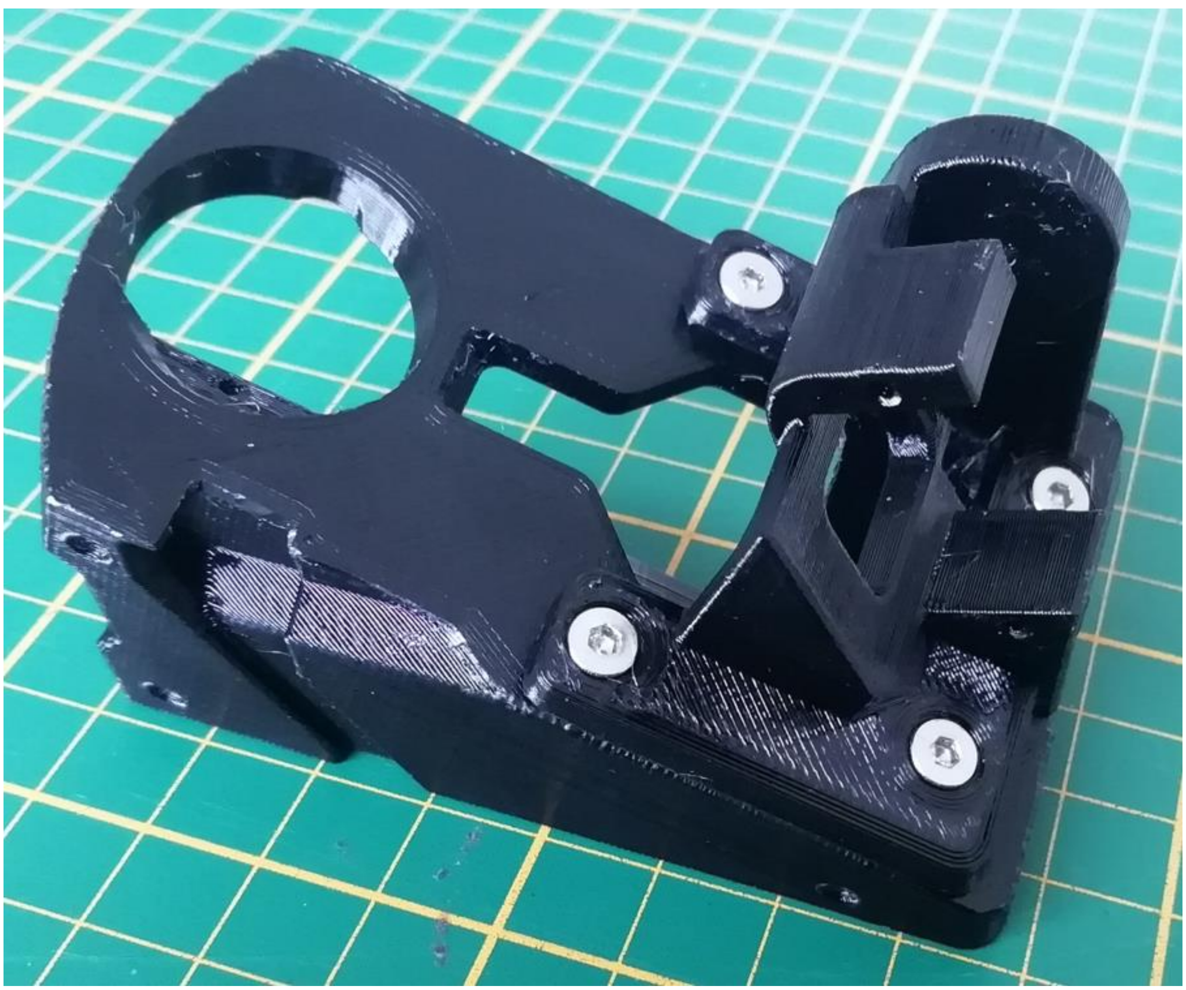




Step6: Assemble the Hinge to the Structure.

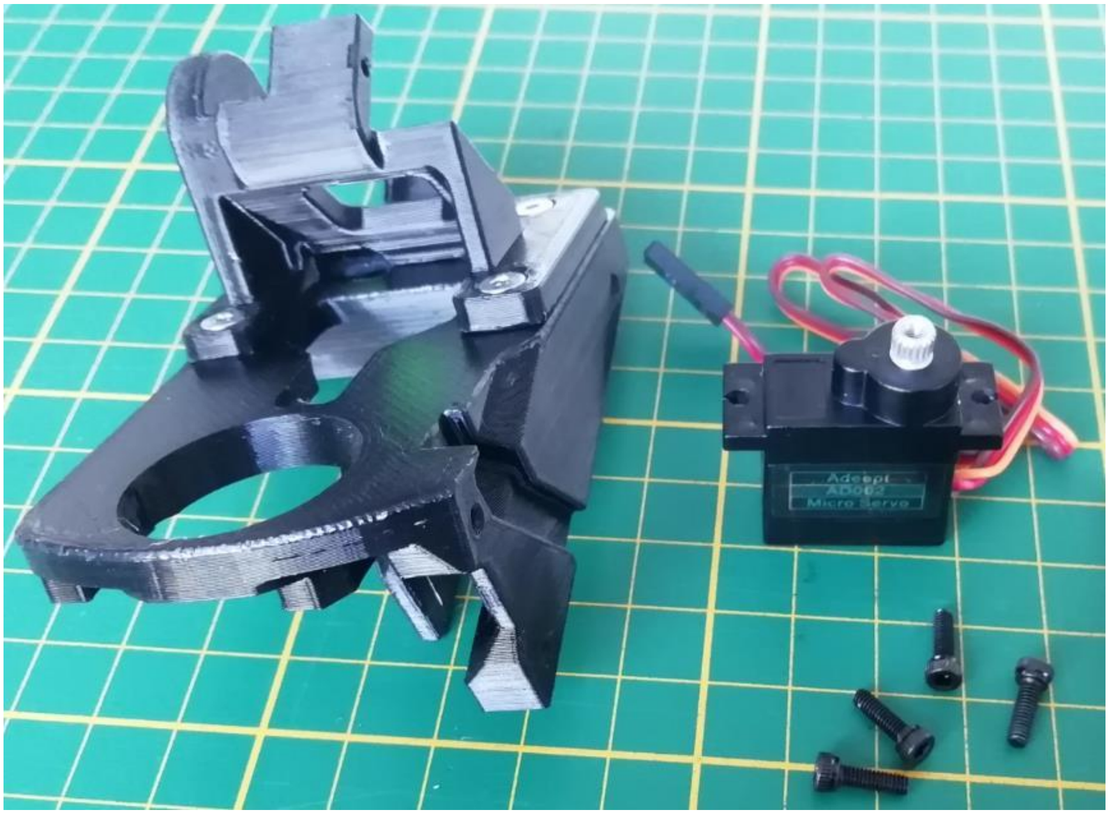
- 4x M3x8mm conical head screw

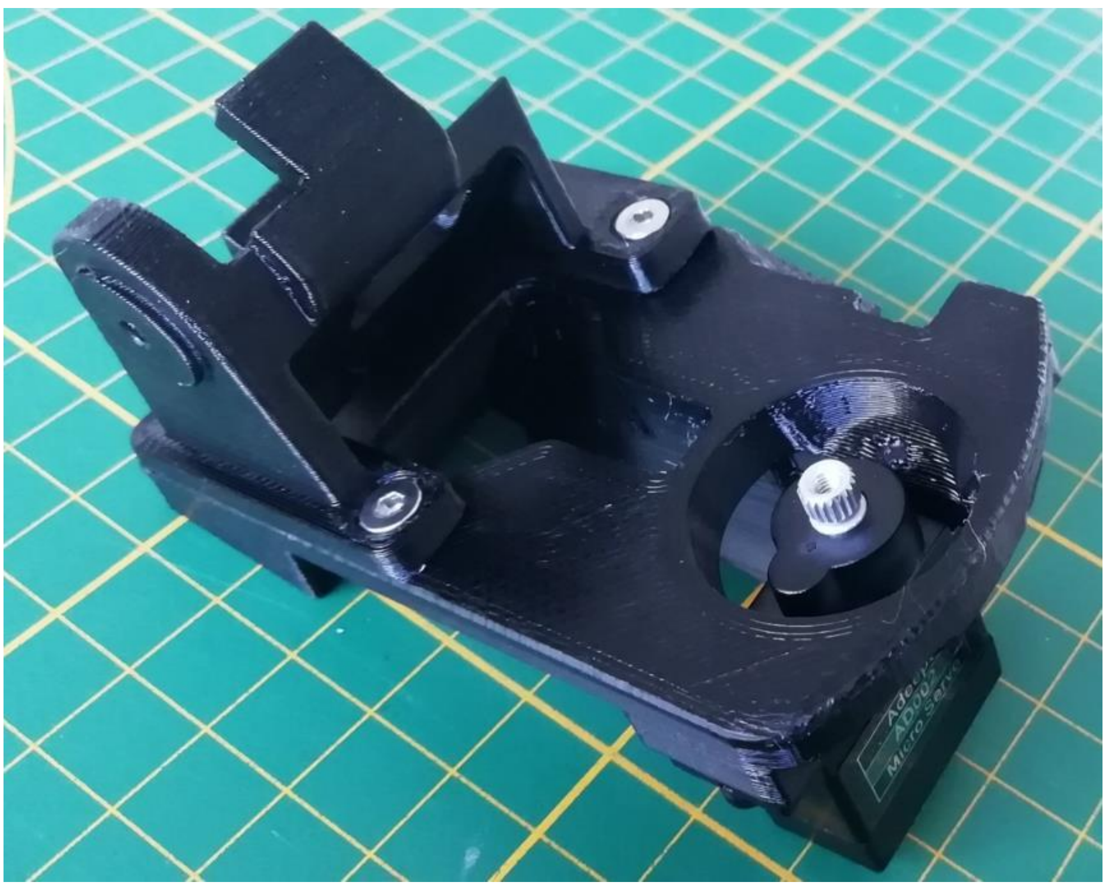




Step7: Screw the bottom servo the Structure.

- 2x M2,5x8mm cylindrical head screws





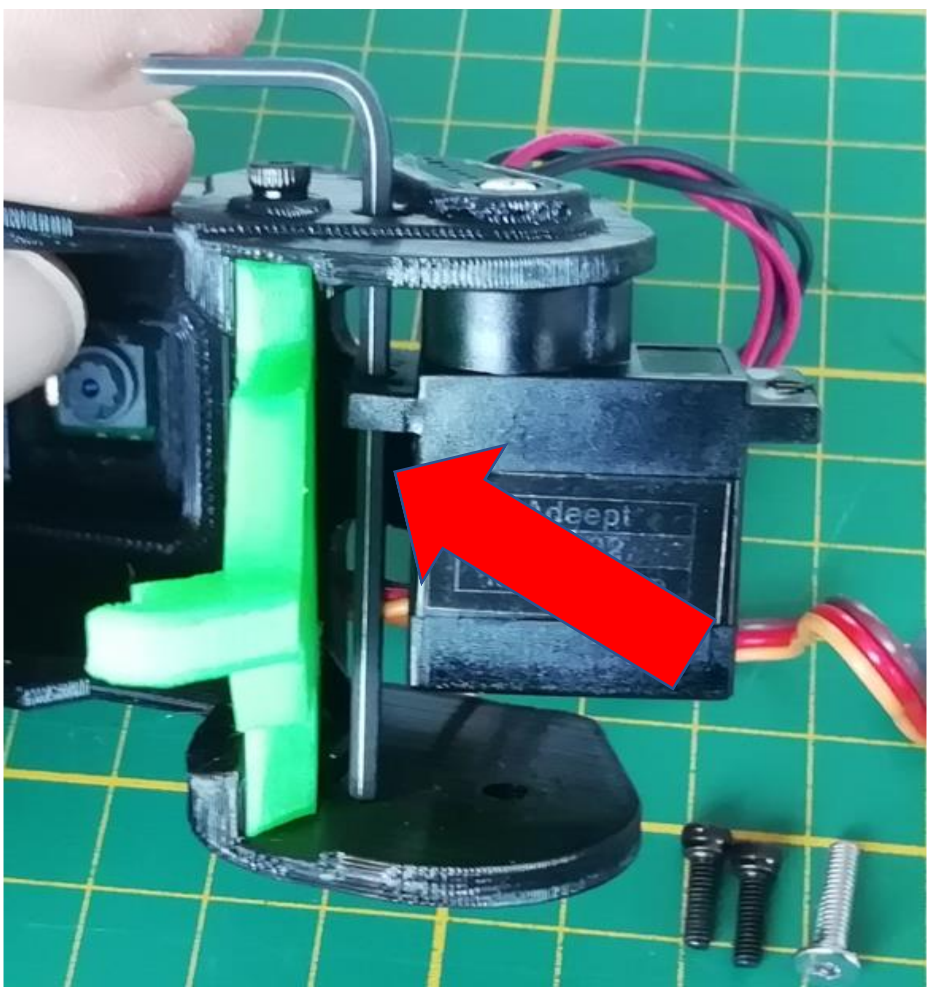
Step8: Assemble the top servo to the Top_cover and to the Hinge.

- 2x M2,5x8mm cylindrical head screws
- 1x M3x12mm conical head screw

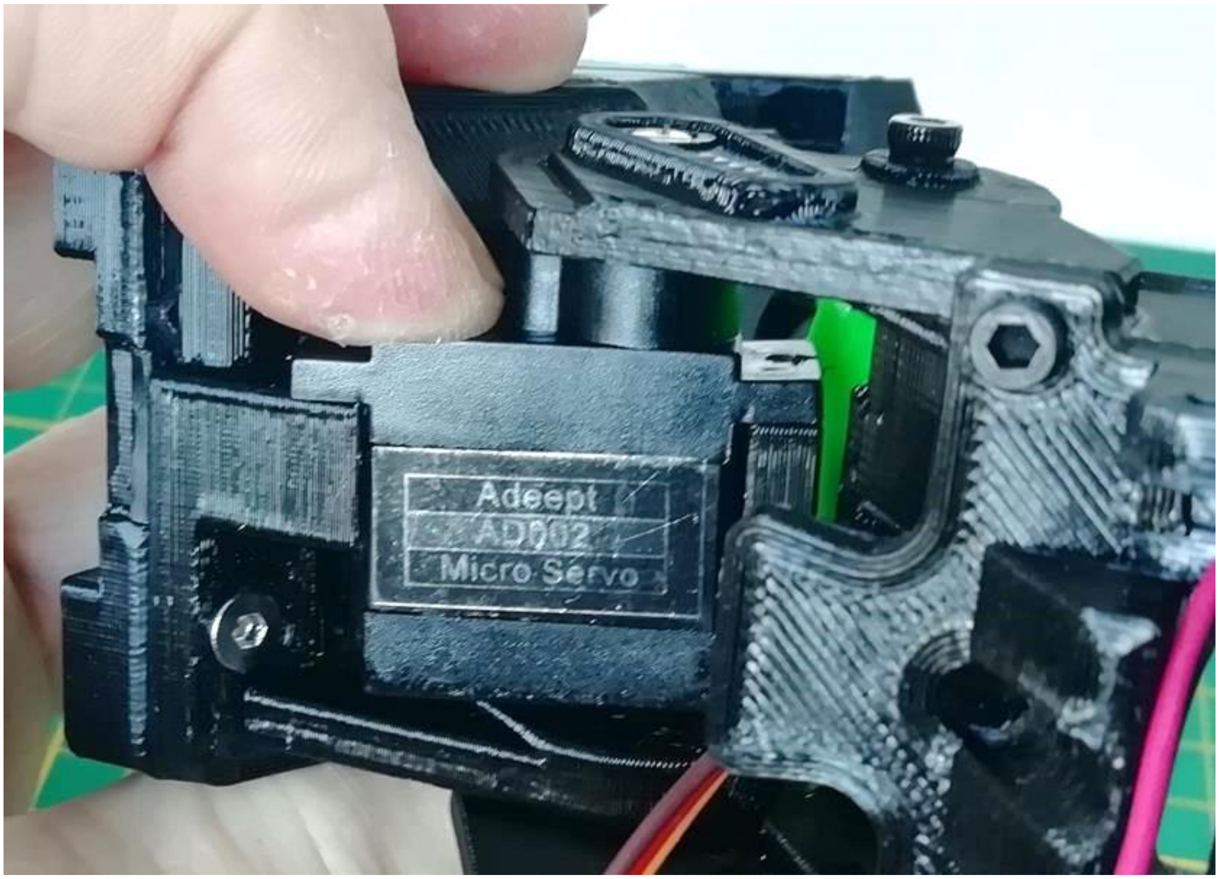
- The servo should be in its mid angle position, with the arm as per picture:

- Insert the arm through the Top_cover opening, and rotate the servo body until its flange “left” hole will be on the projection of the hole at the Top_cover:

- Rotate the servo body until its flange “left” hole will be on the projection of the hole at the Top_cover:



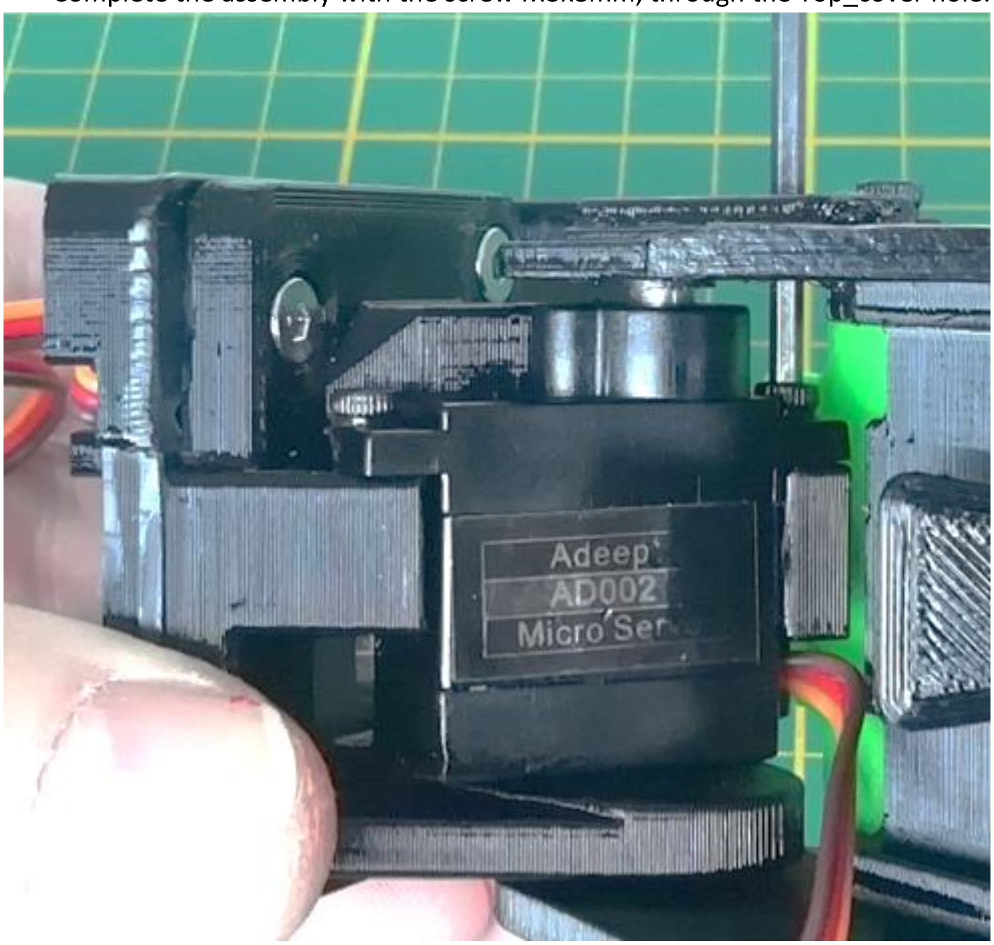
- Position the servo on the Hinge seat:



- Screw the M3x12mm conical head screw till the end, and turn back half turn to prevent excess of friction:

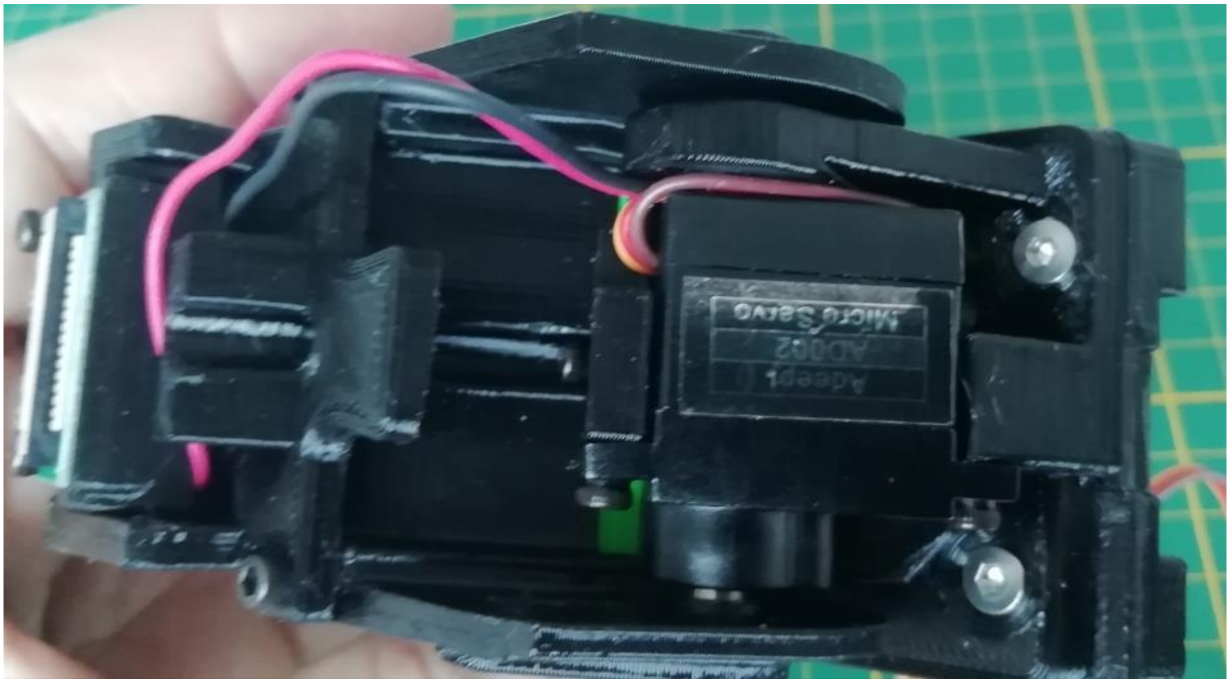
- Fix the servo body; Screw the M3x8mm on the servo's flange hole that is more accessible:

- Complete the assembly with the screw M3x8mm, through the Top_cover hole:



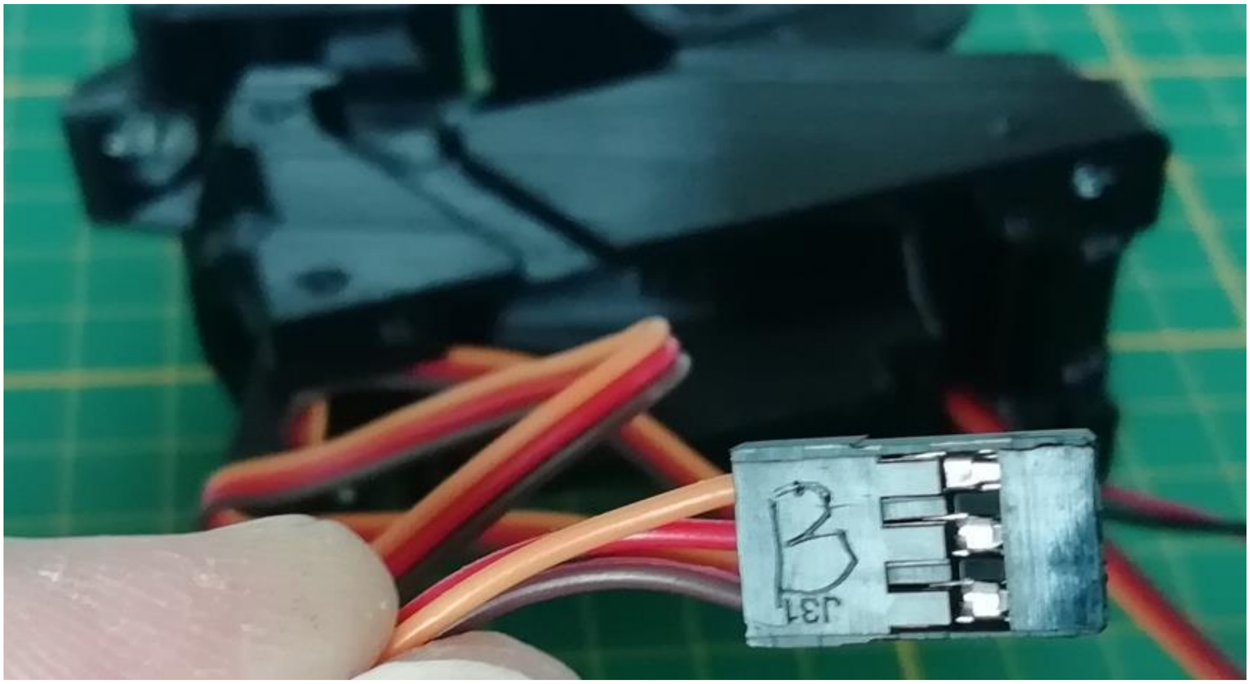
Step9: Dress the cables.

- Insert the Led cable first, and the cable from the servo afterward, through the Structure hole.
- Keep a loop of ca 2cm in diameter for the LED cable, to prevent tension at Top_cover rotation.

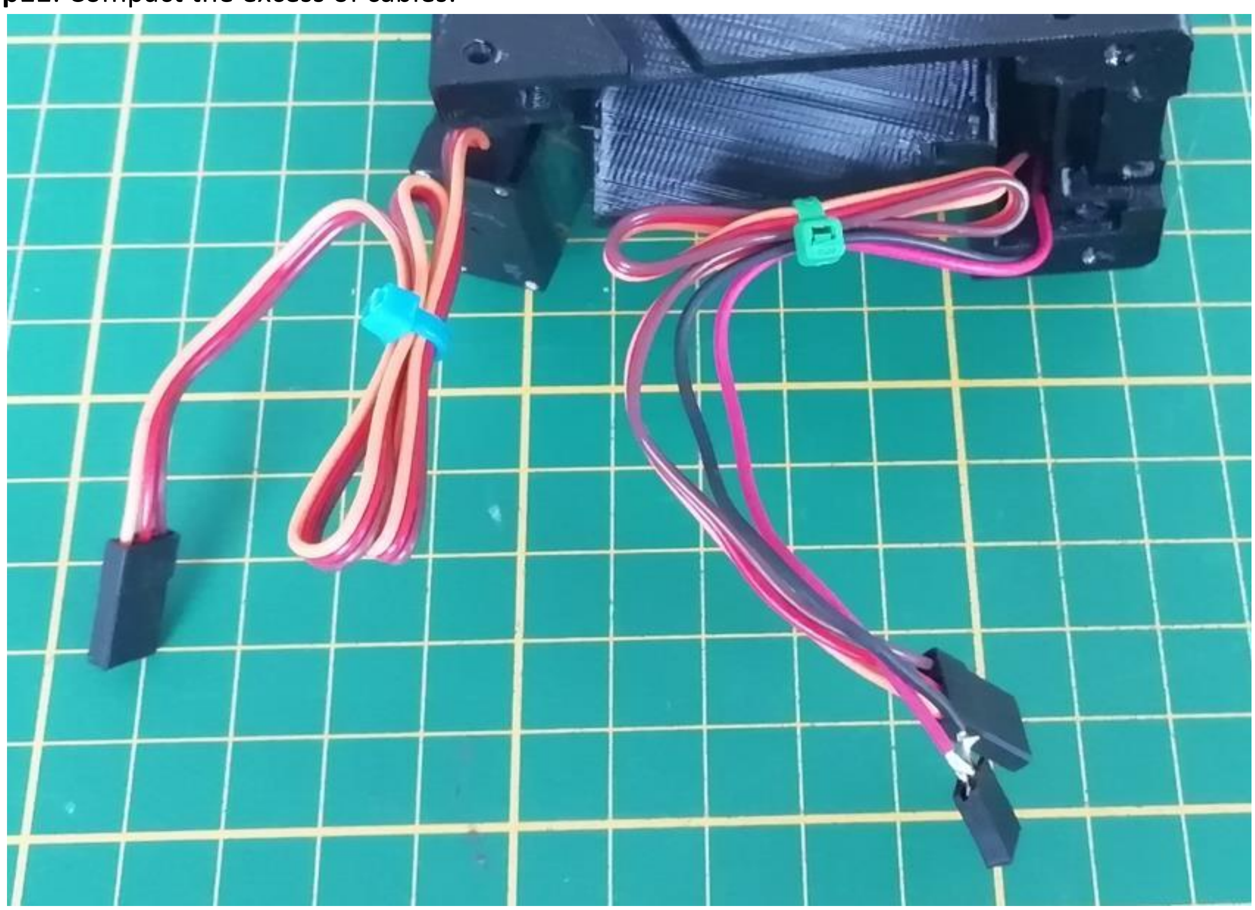


Step10: Mark the servos connectors.

- At this moment, it is still possible to distinguish the connectors belonging to the two servos, therefore it is suggested to mark at least one of those (in my case I used a B as "bottom")

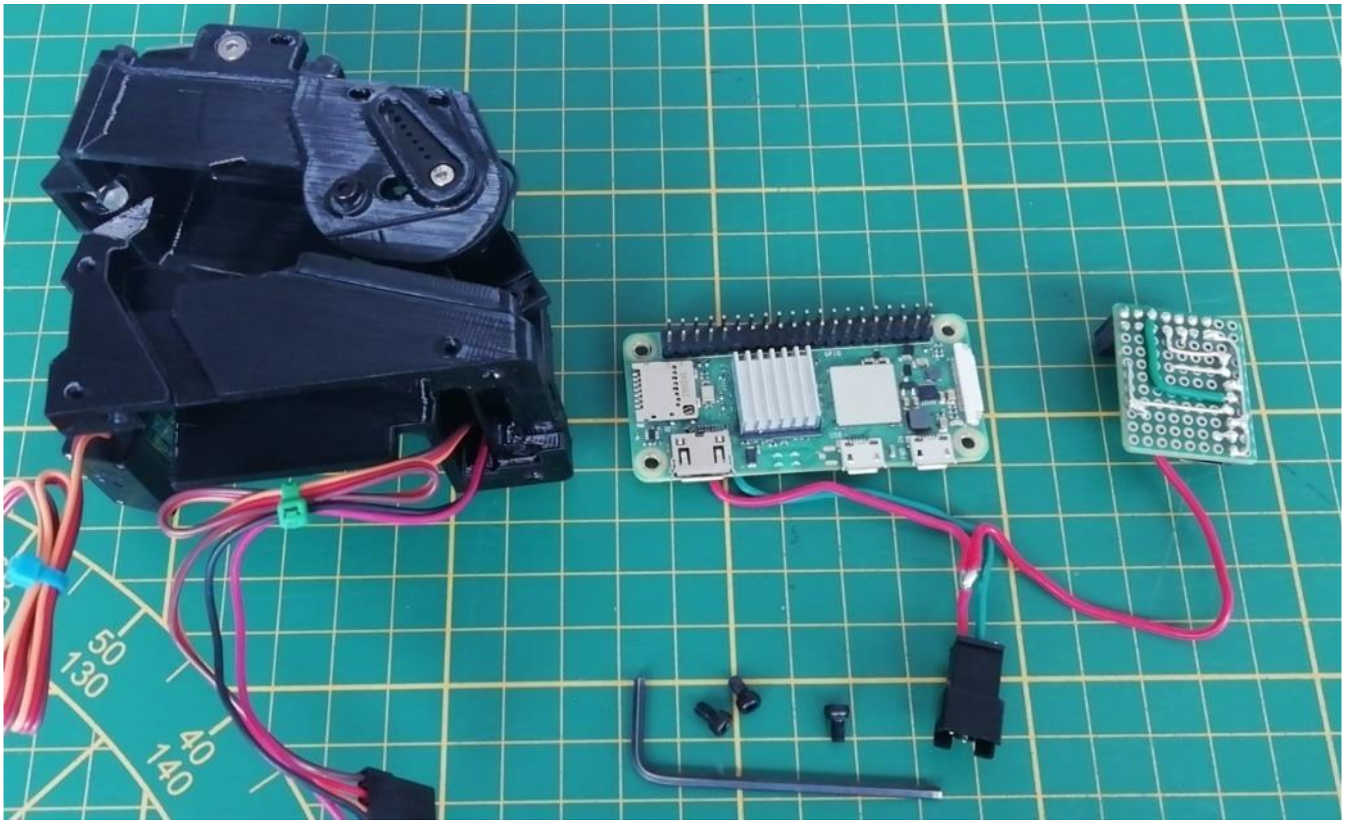


Step11: Compact the excess of cables.

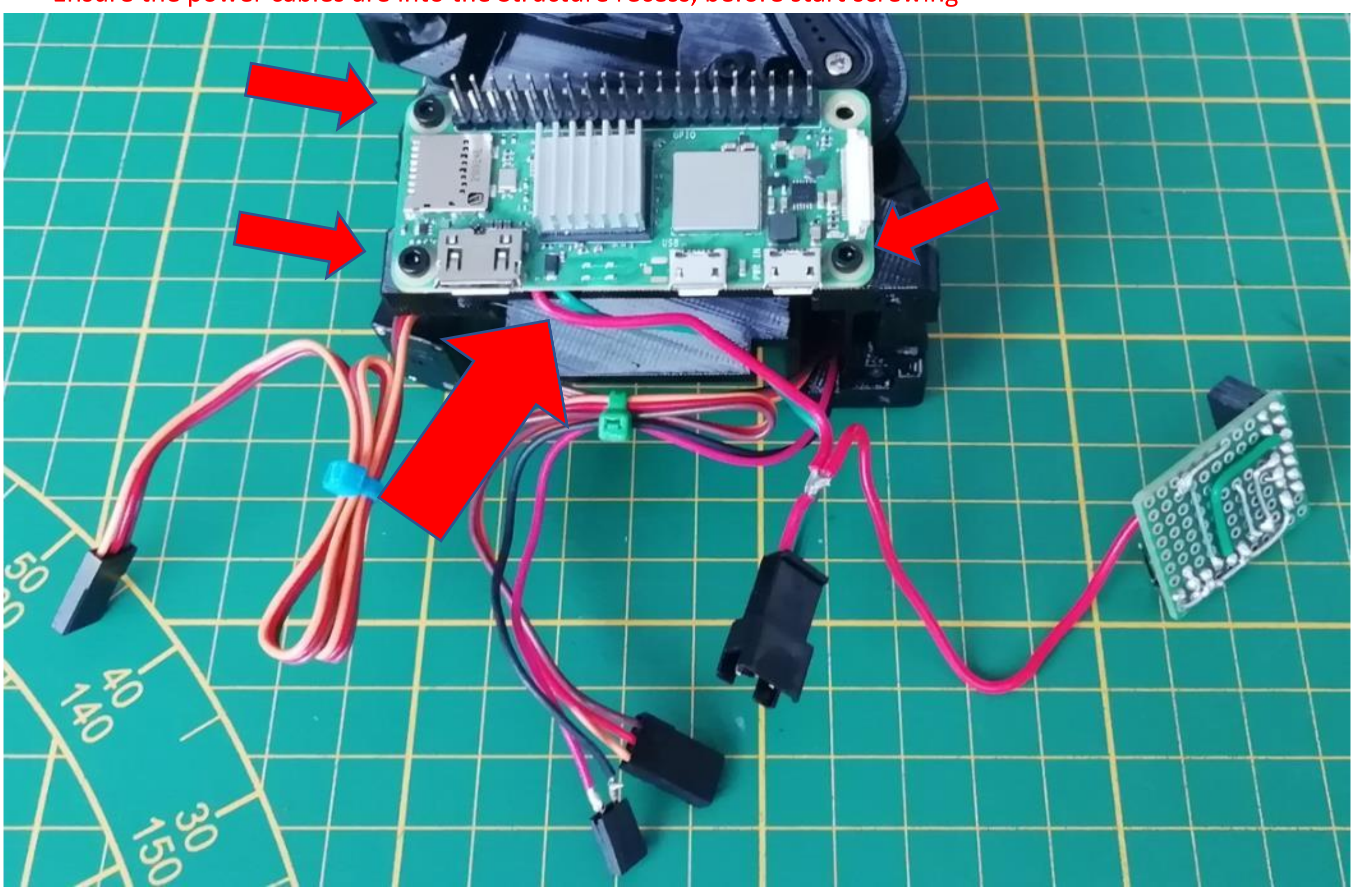


Step12: Screw the Raspberry Pi assembly to the Structure.

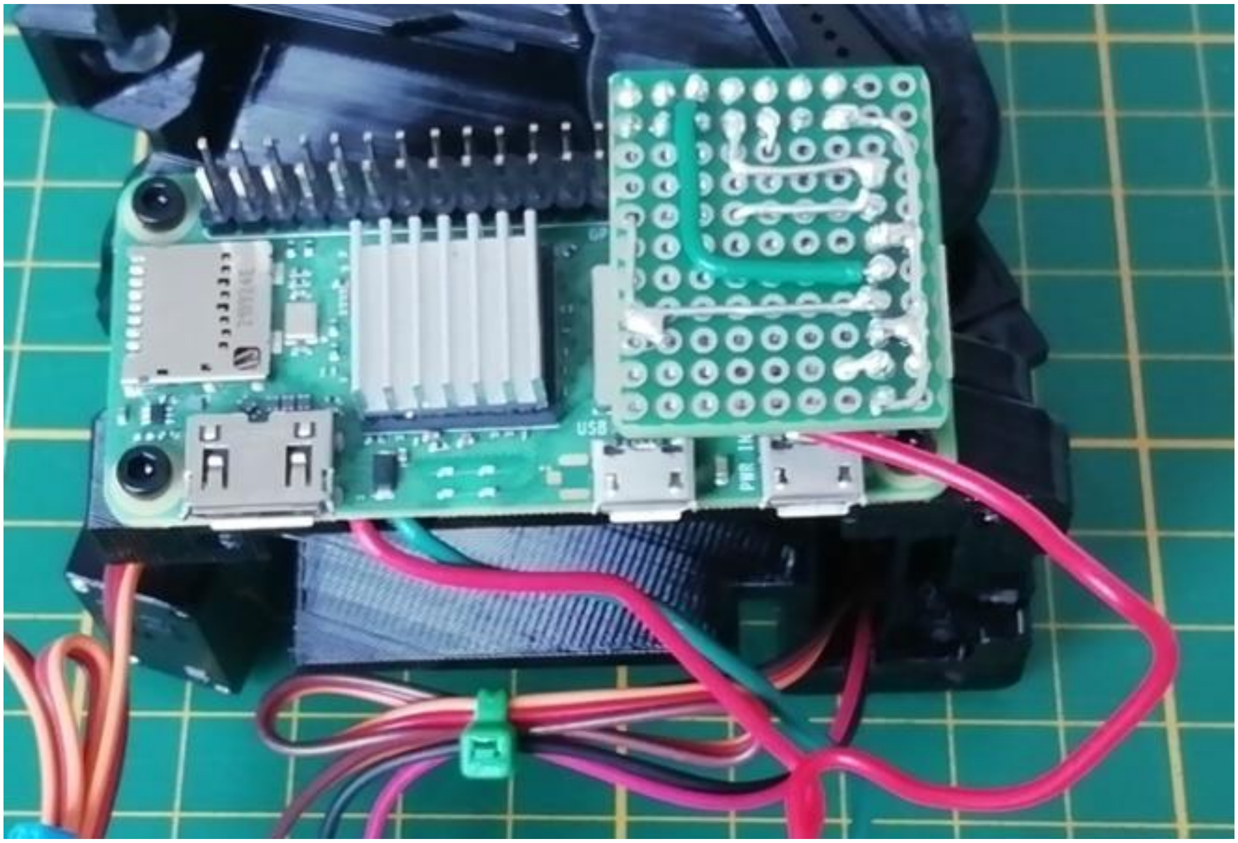
- 3x M2,5x4mm



- Ensure the power cables are into the Structure recess, before start screwing

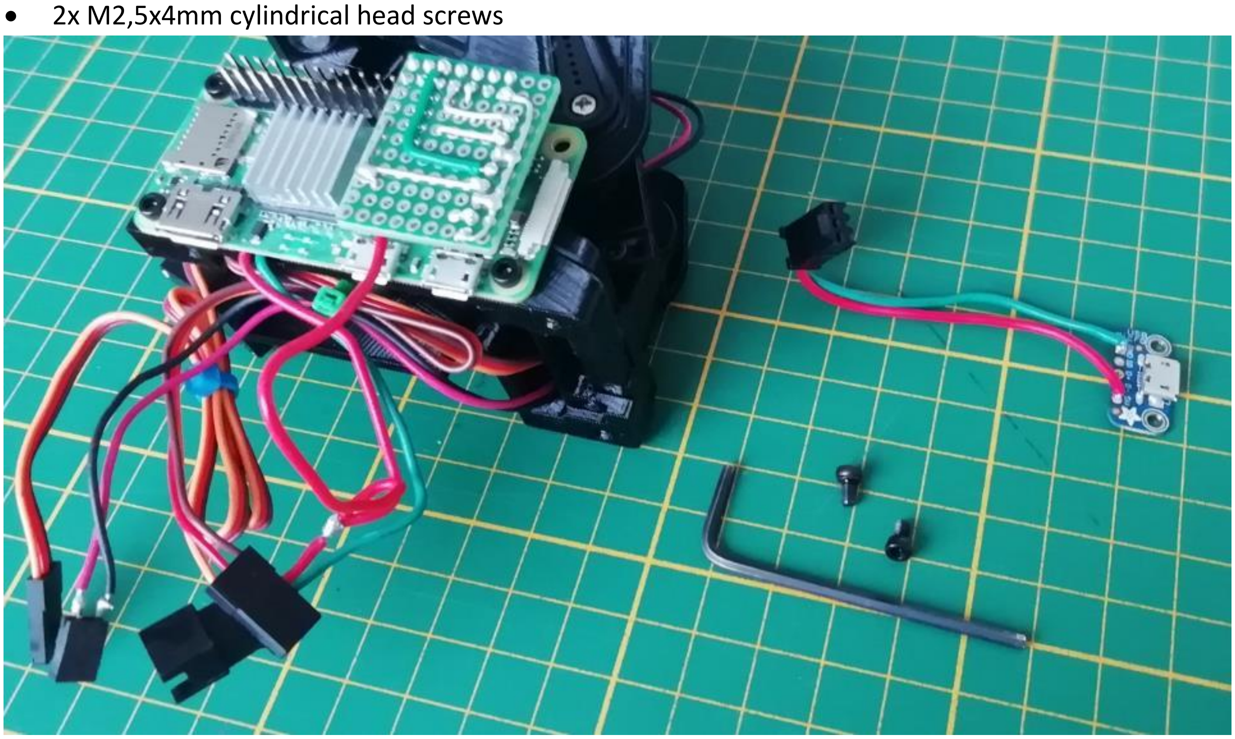


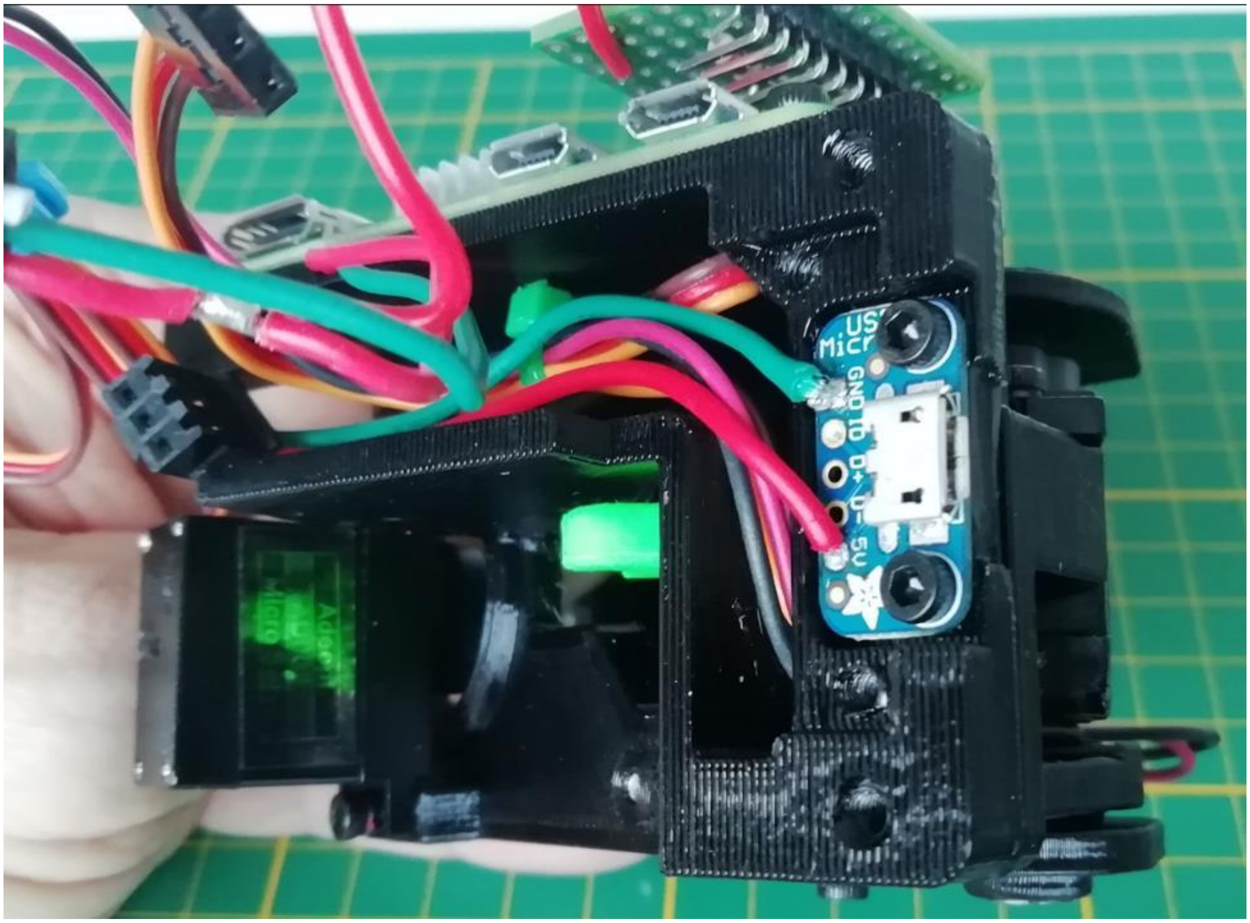
Temporary: Connect the Connections_board to the Raspberry Pi GPIO pins, to avoid running around



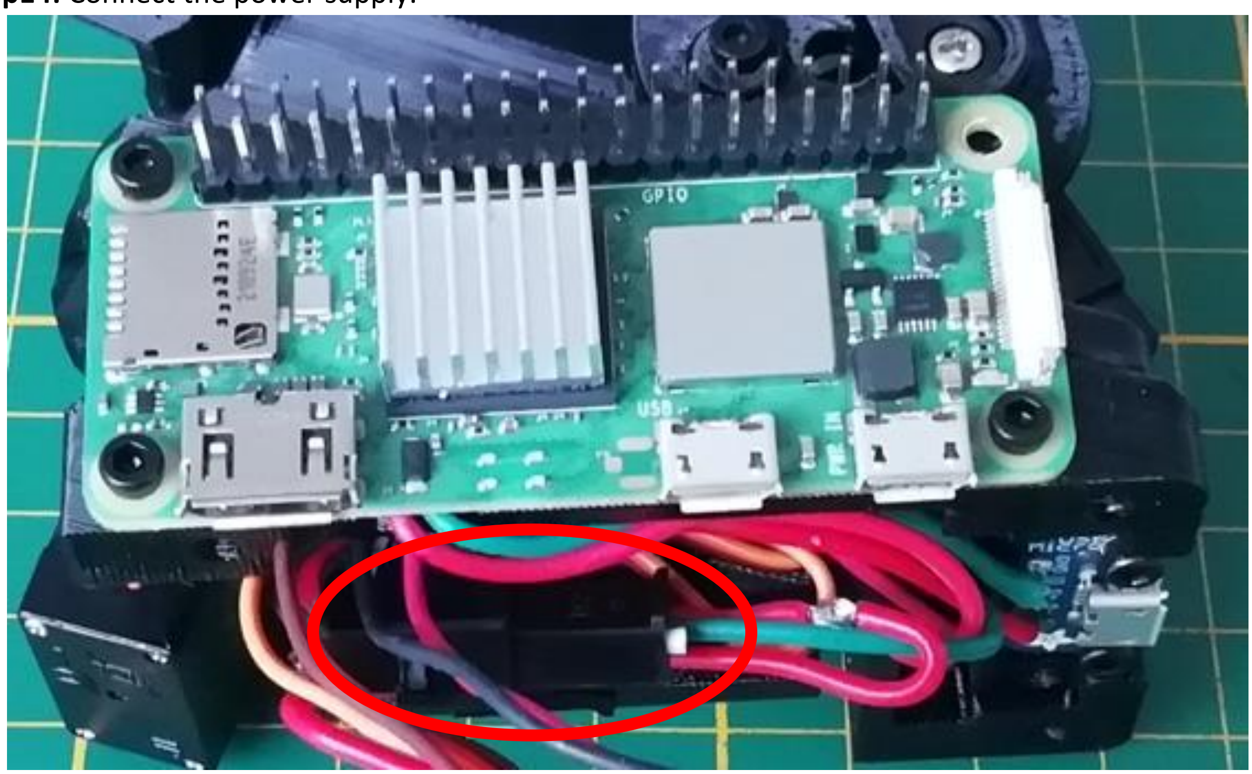
Step13: Assemble the microUSB breakout board to the Structure.

- 2x M2,5x4mm cylindrical head screws



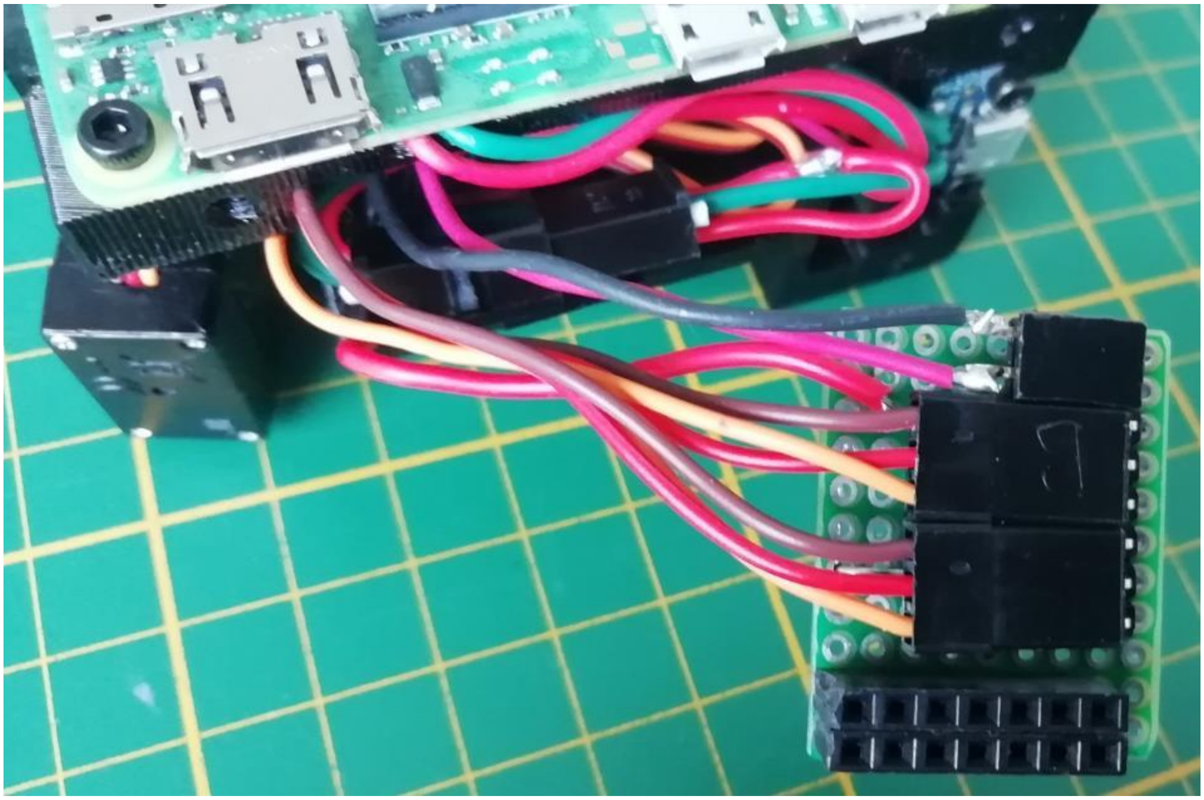


Step14: Connect the power supply.

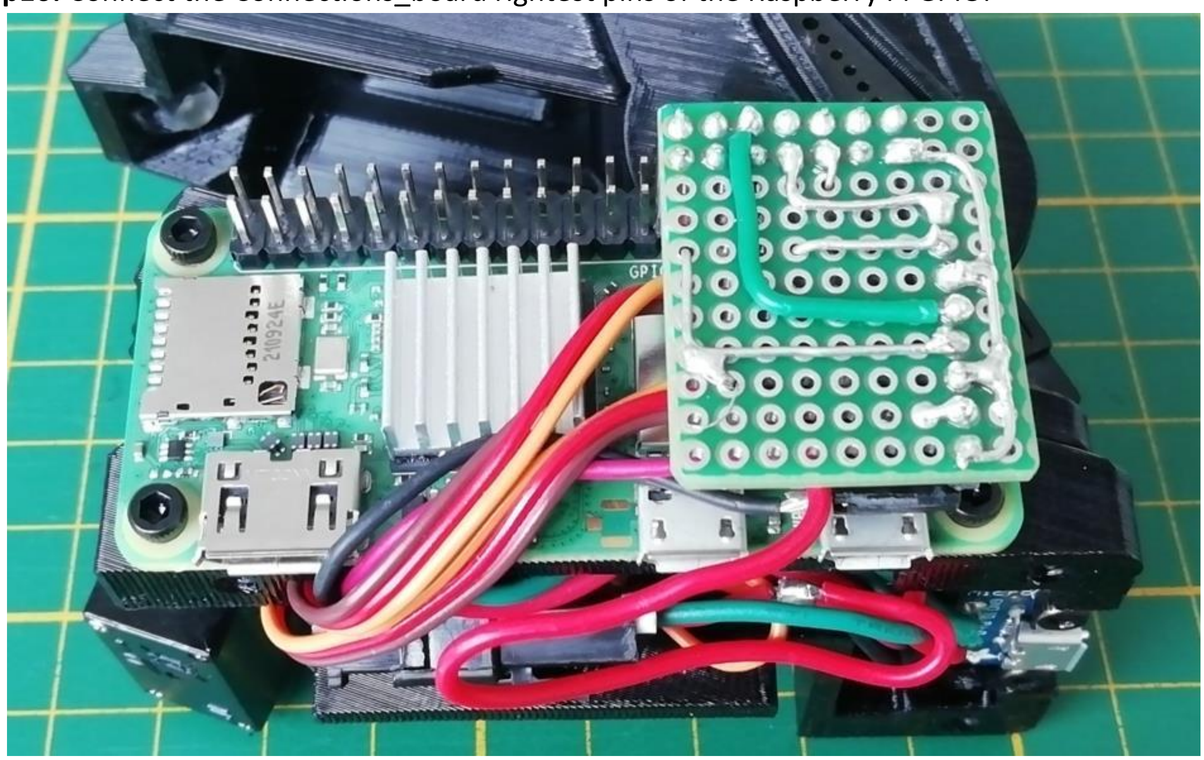


Step15: Connect the servos and the LED to the connection board.

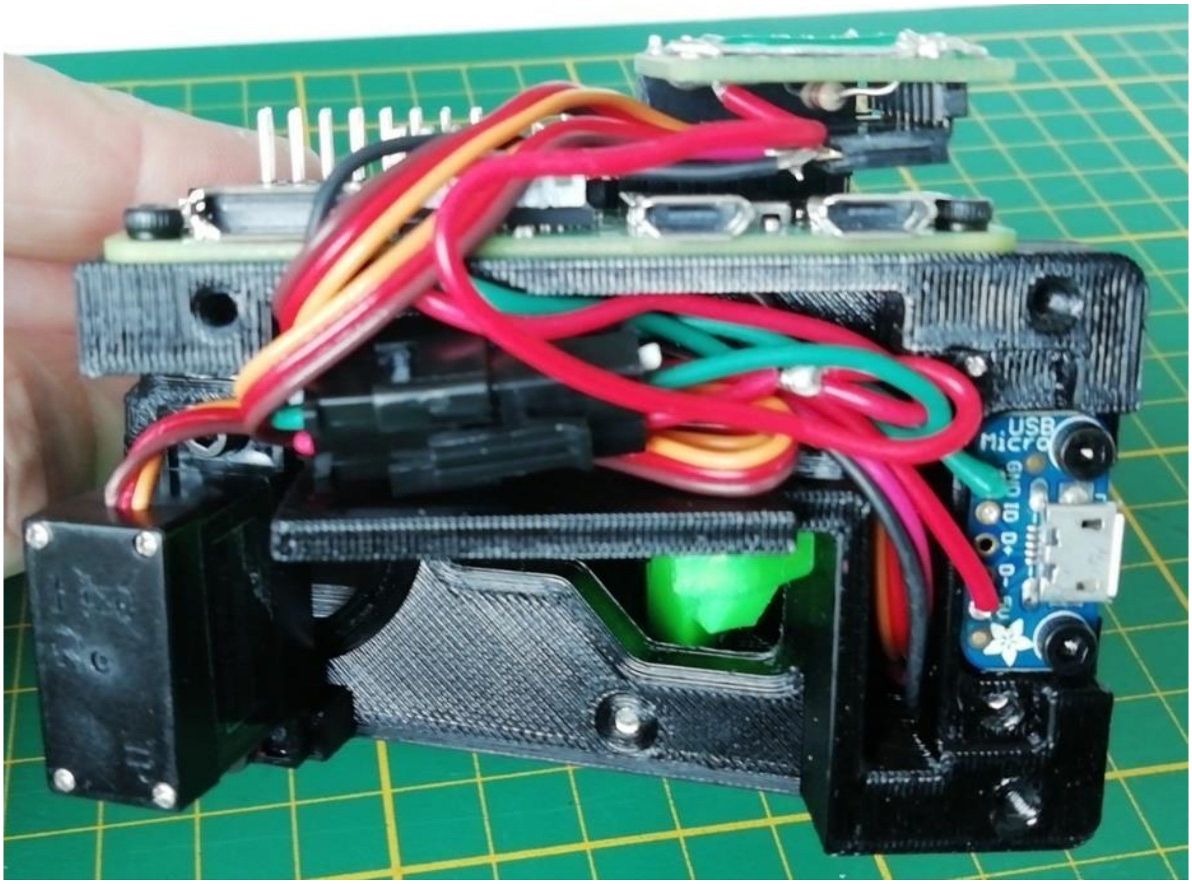
- Once the board is flipped in position:
    - Top servo top connector, bottom int the middle, Led on the bottom.
    - Negative wires toward the bottom board part (servo brown wire is the negative, LED negative wire is black in my case)



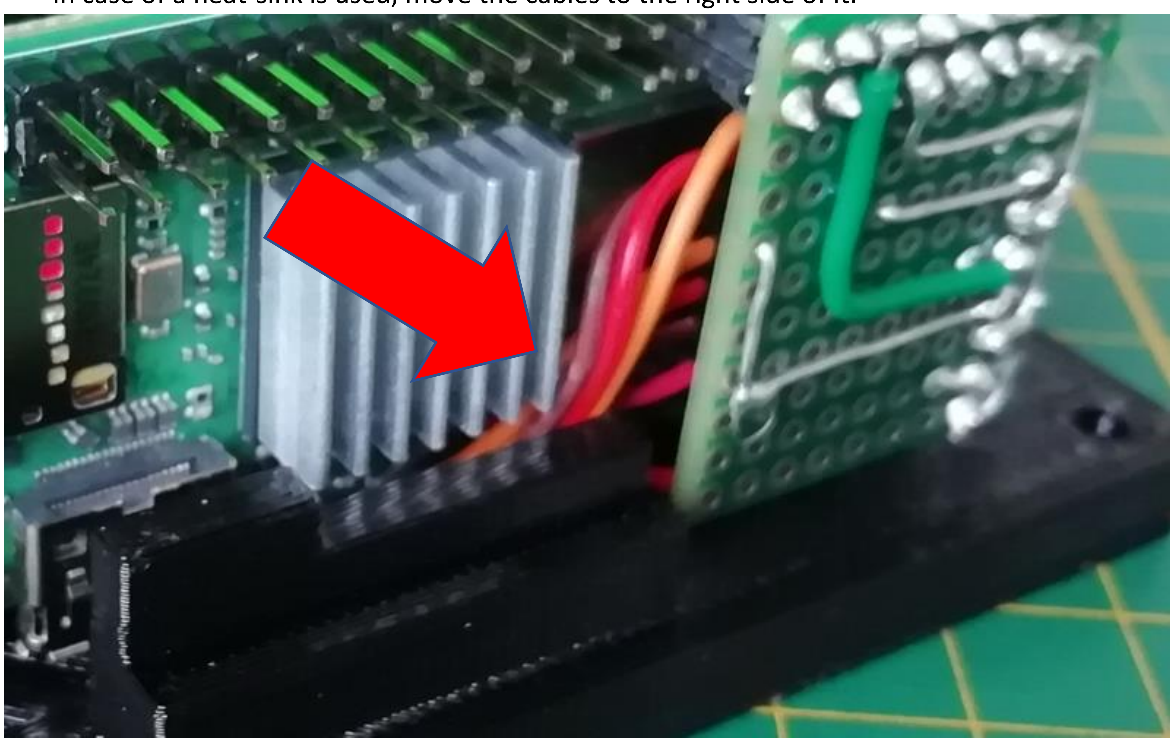
Step16: Connect the Connections_board rightest pins of the Raspberry Pi GPIO.



Step17: Pack the cables into the Structure voids.

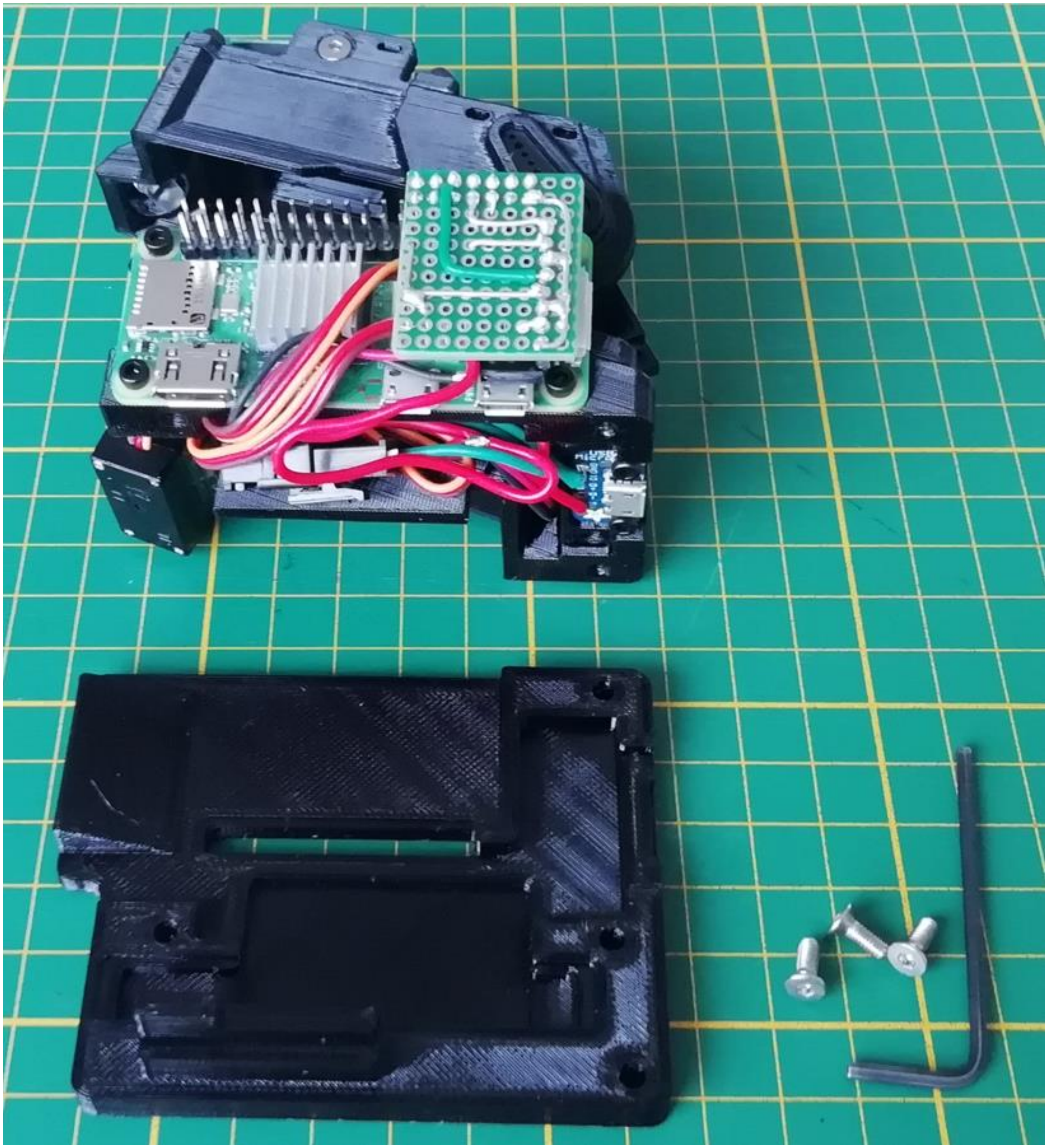


- In case of a heat-sink is used, move the cables to the right side of it:



Step18: Assemble the Base_plate.

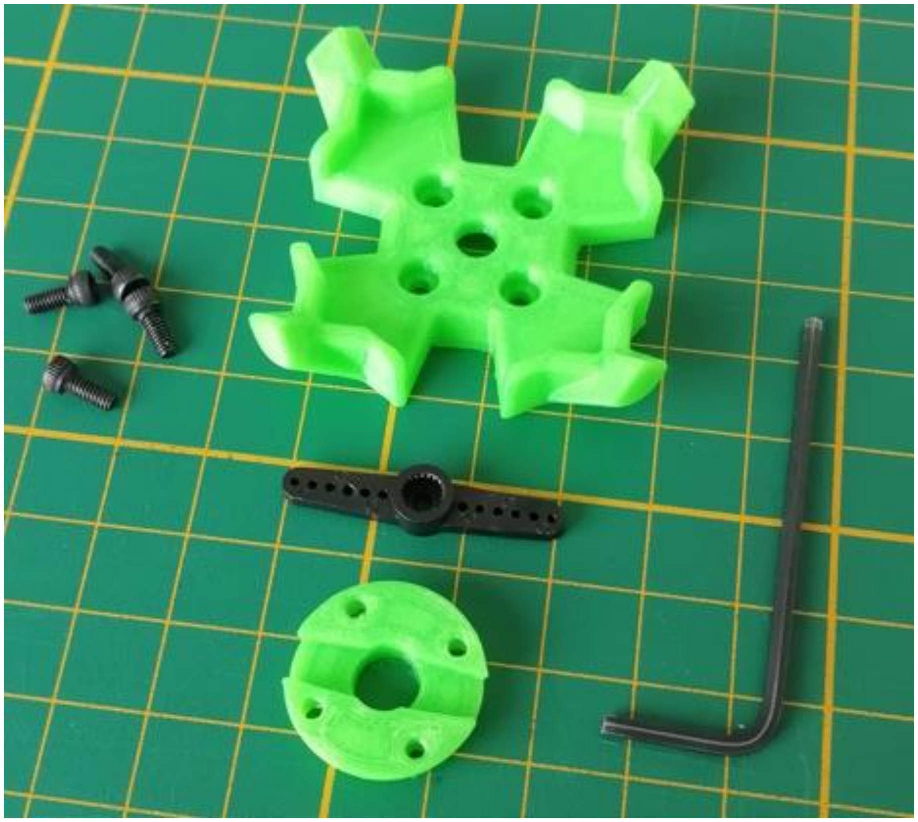
- 3x M3x8mm conical head screws



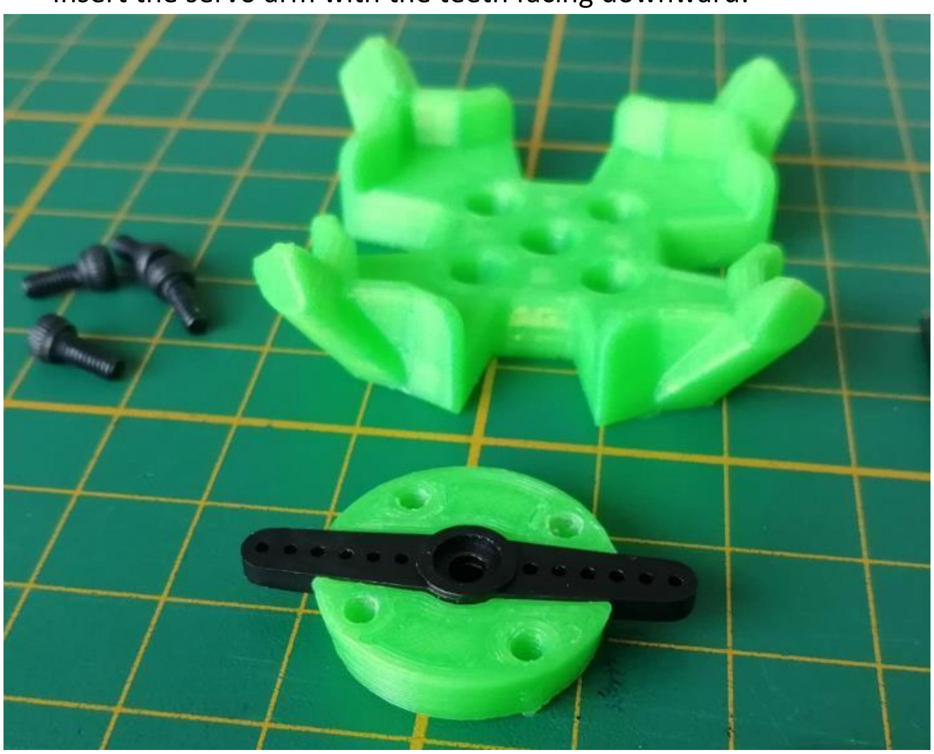
- Ensure the Base_plate touches the Structure without pinching any cable, and without force.

Step19: Assemble the cube Holder to the servo offset compensator:

- Take the servo offset compensator previously chosen.
- 4x M2,5x8mm cylindrical head screws



- Insert the servo arm with the teeth facing downward:

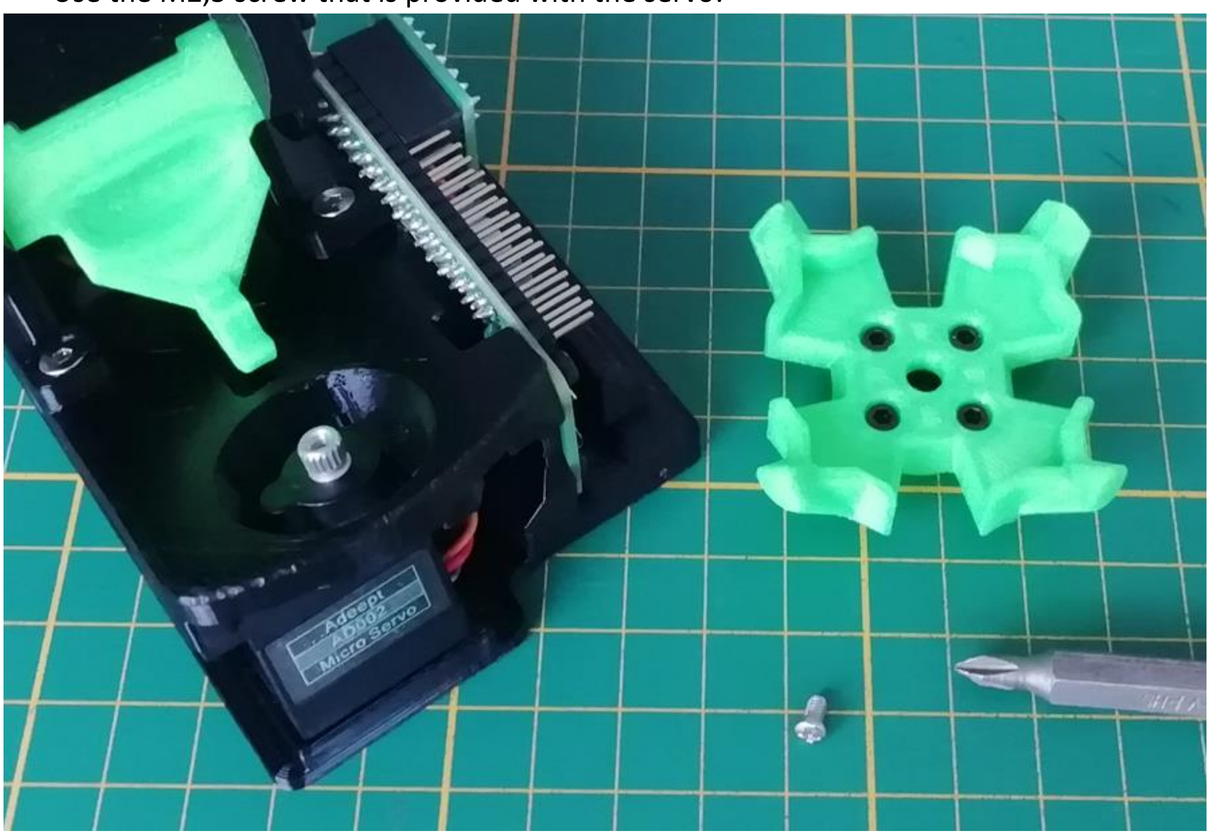


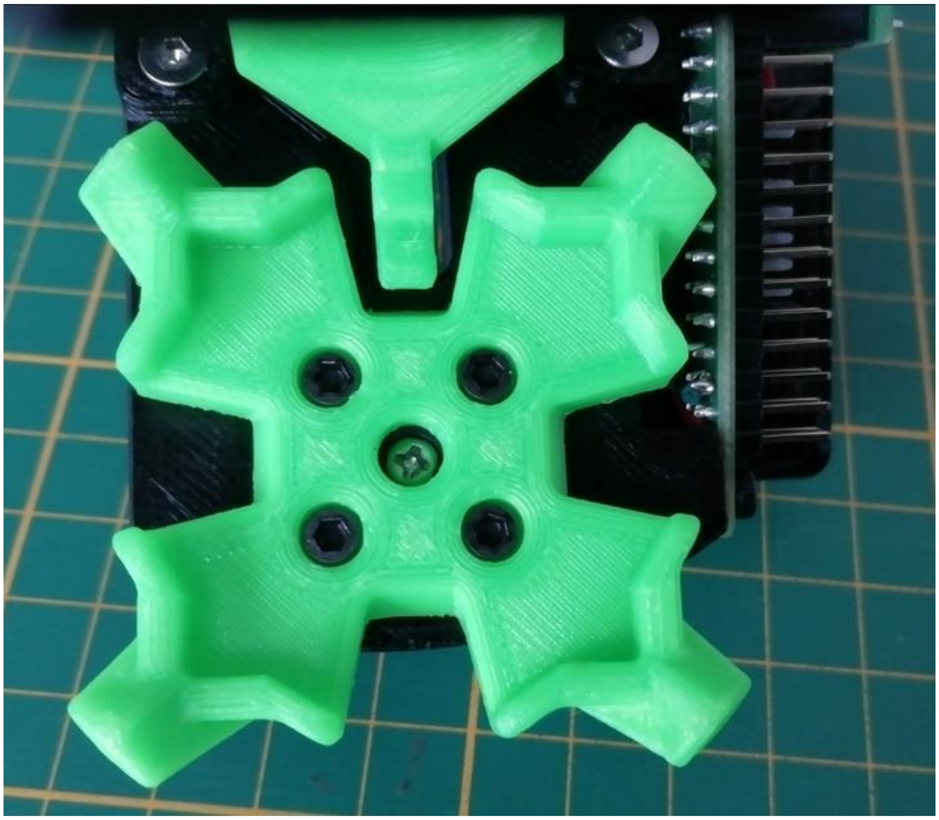
- Cut the servo arm protruding part:

- Assemble the cube Holder to the Servo offset compensator:

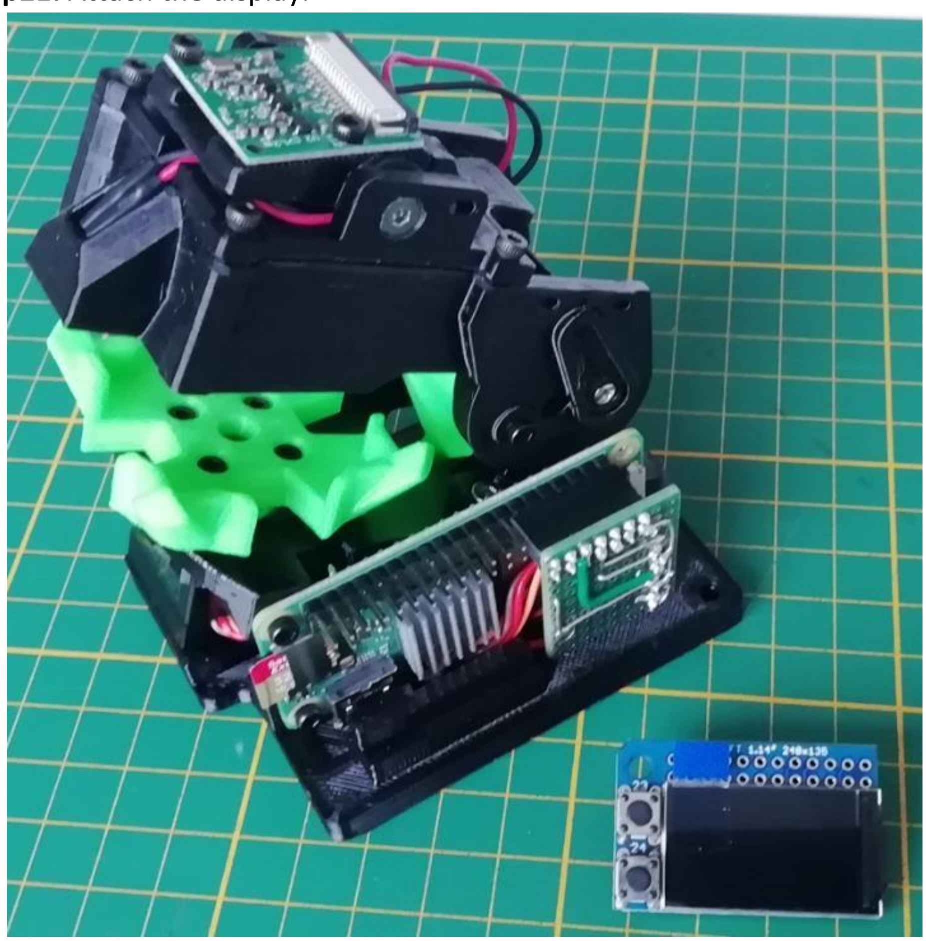
Step20: Assemble the cube Holder assembly to the bottom servo:

- Use the M2,5 screw that is provided with the servo.

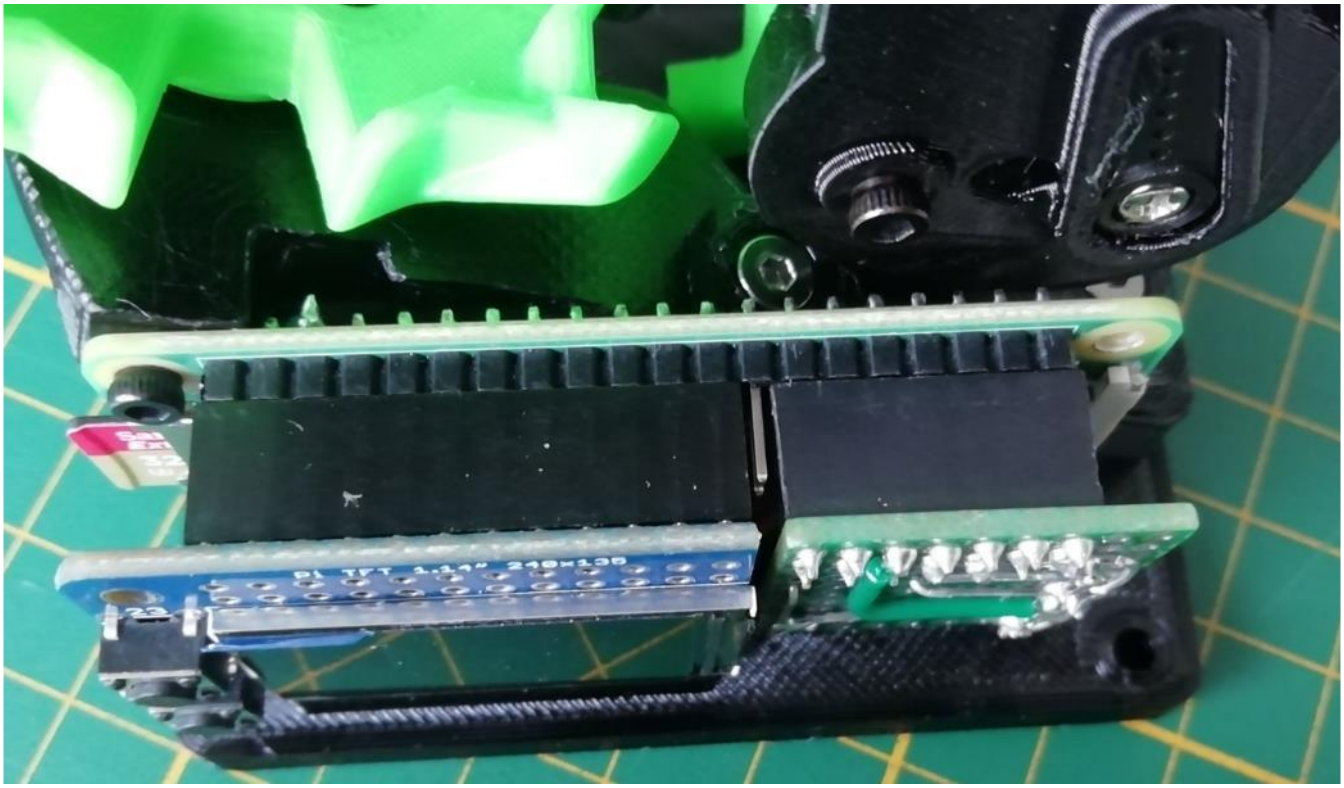




Step21: Attach the display.



- Check the display is connected to the first 24 GPIO pins.
- Check the Connections_board is connected to the last GPIO pins:

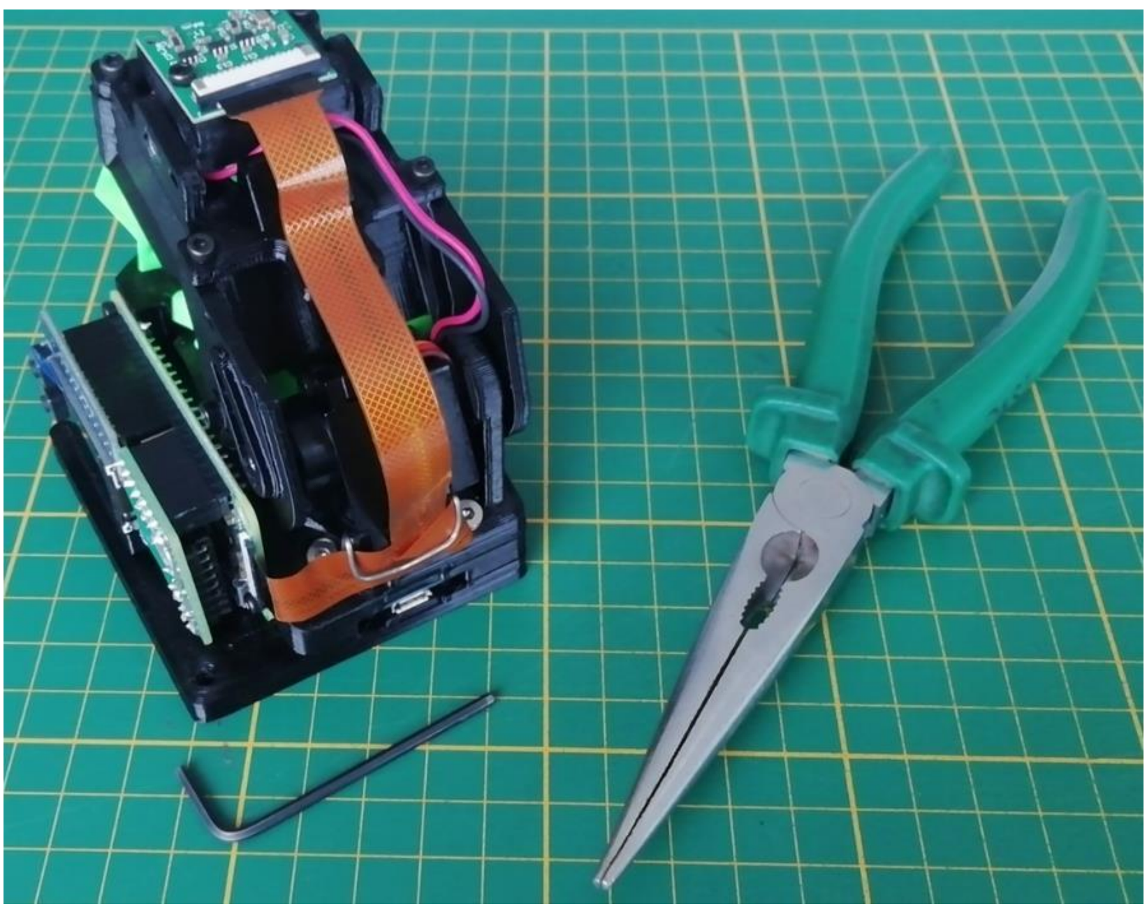


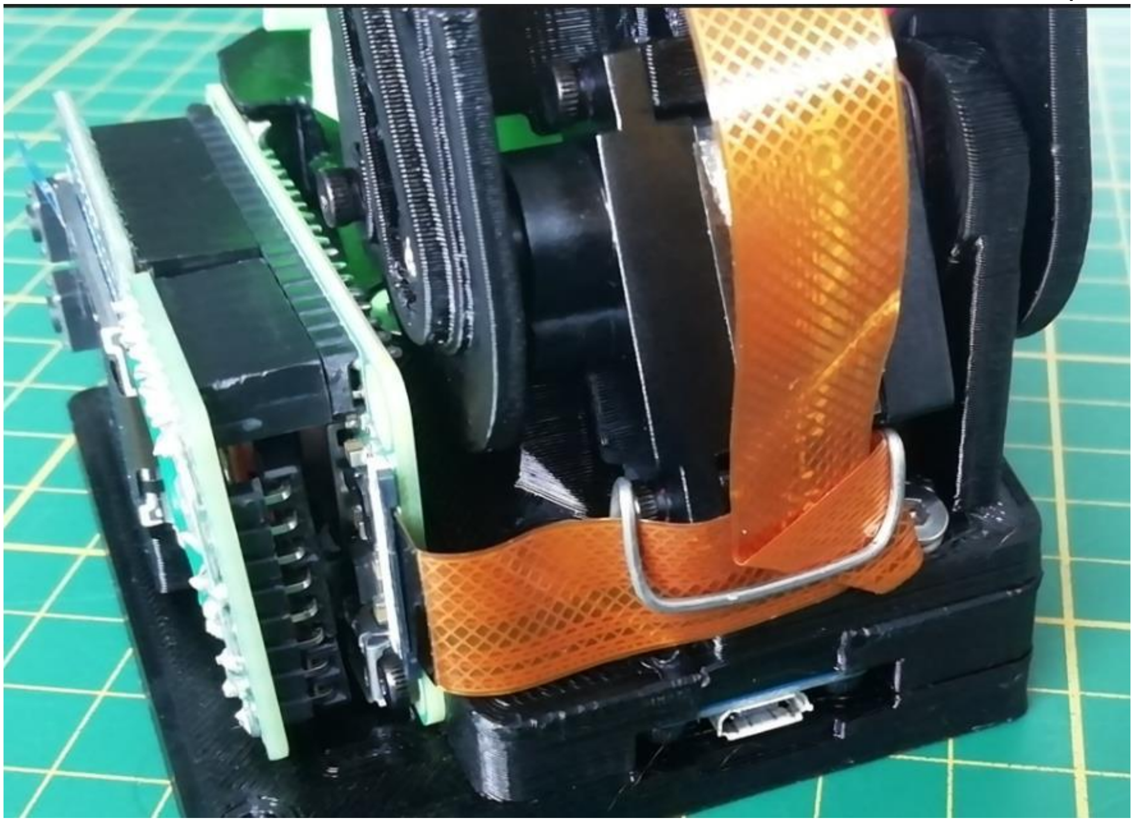
Step22: Connect the flexible cable to the PiCamera module.

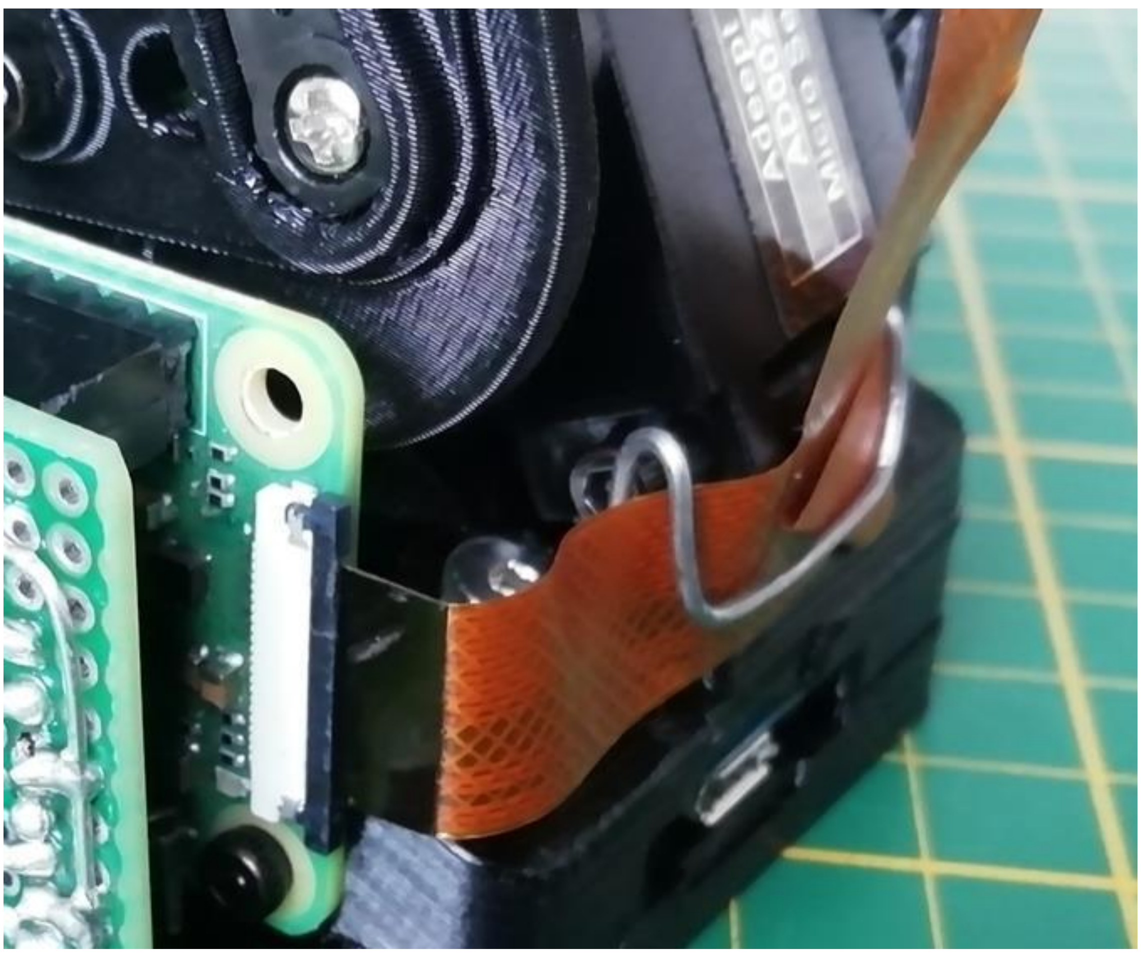
- Cable contacts should be facing downward:

Step23: Connect the flexible cable to the Raspberry Pi.

- Remove the most backward two screws from the Hinge.
- Bend the flexible cable as per the picture below.
- Cable contacts should be facing the PCB.
- Bend a piece of metal wire, like below pictures.
- Pinch the metal wire under the Hinge screws.

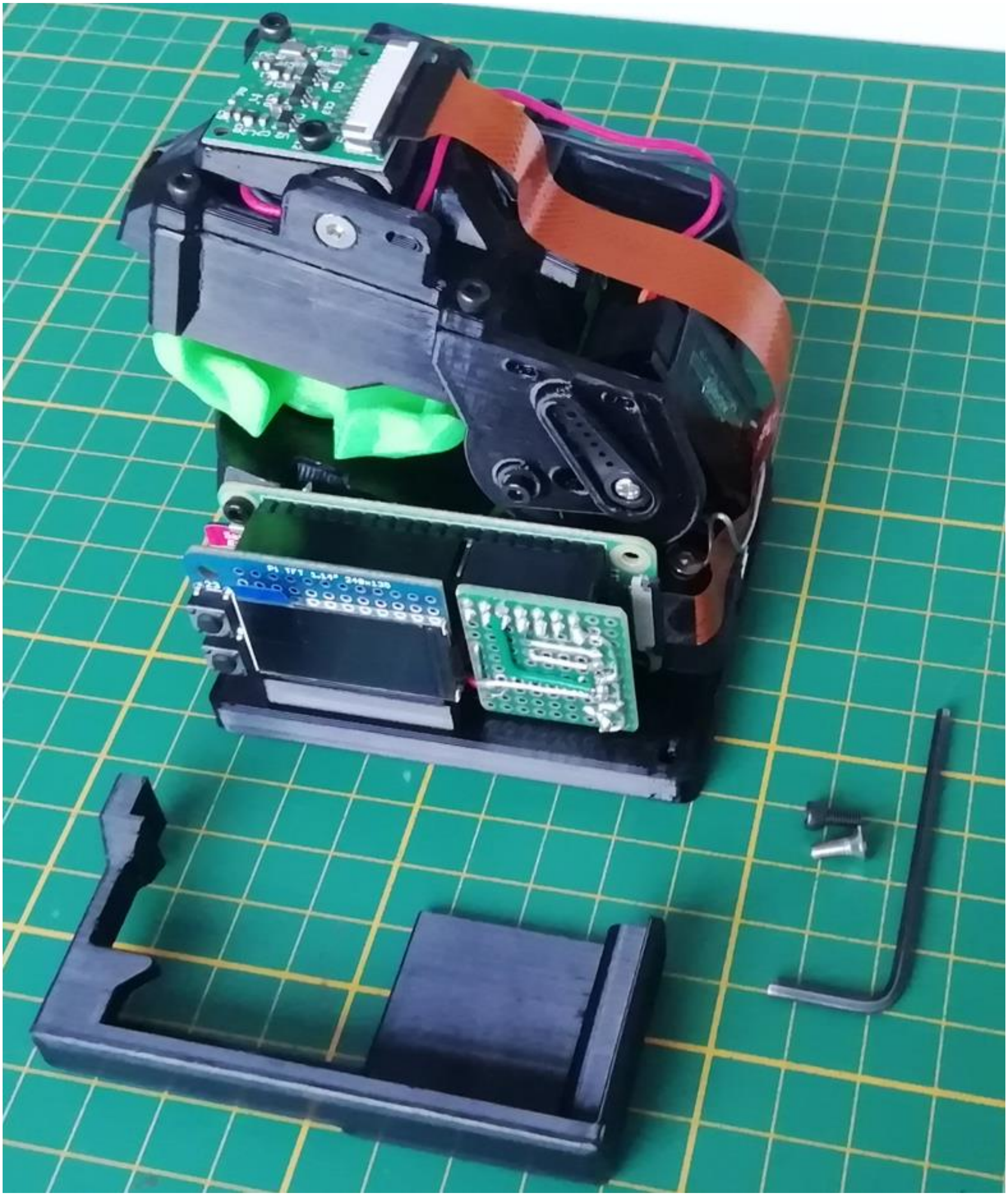


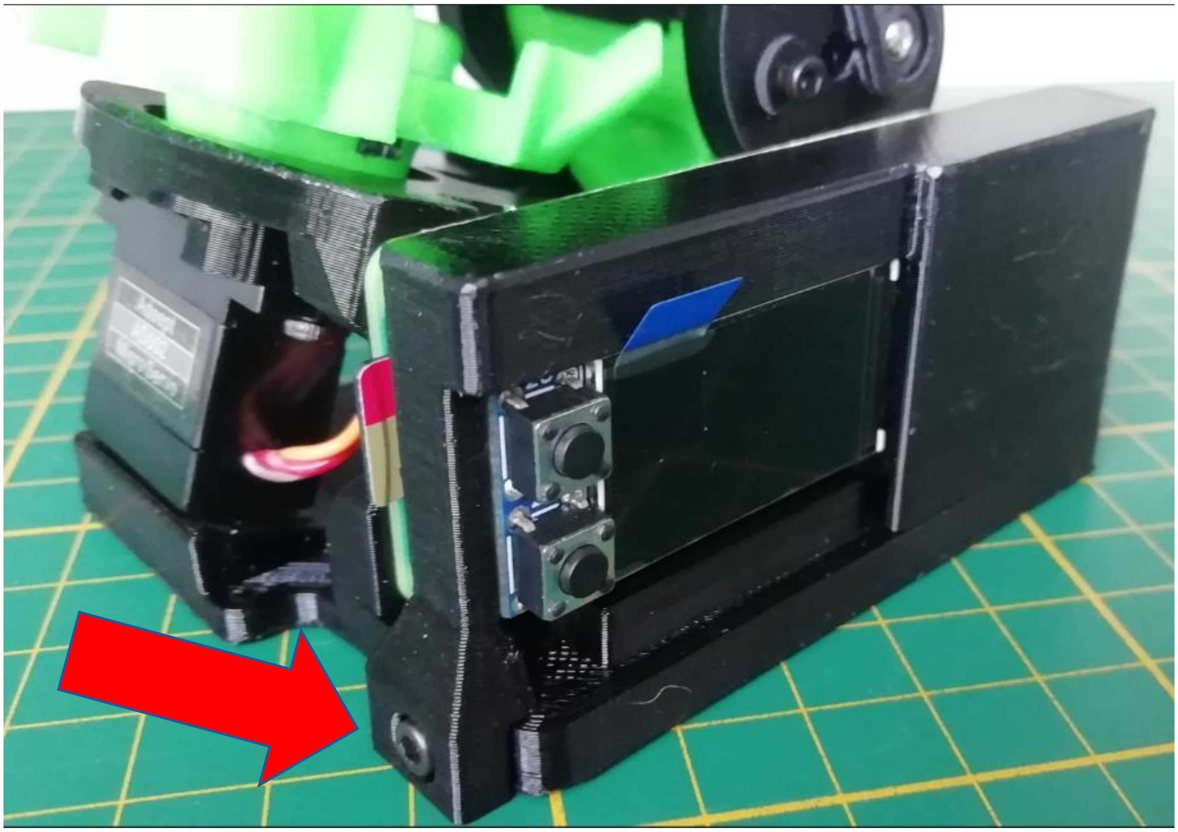




Step24: Assemble the Cover.

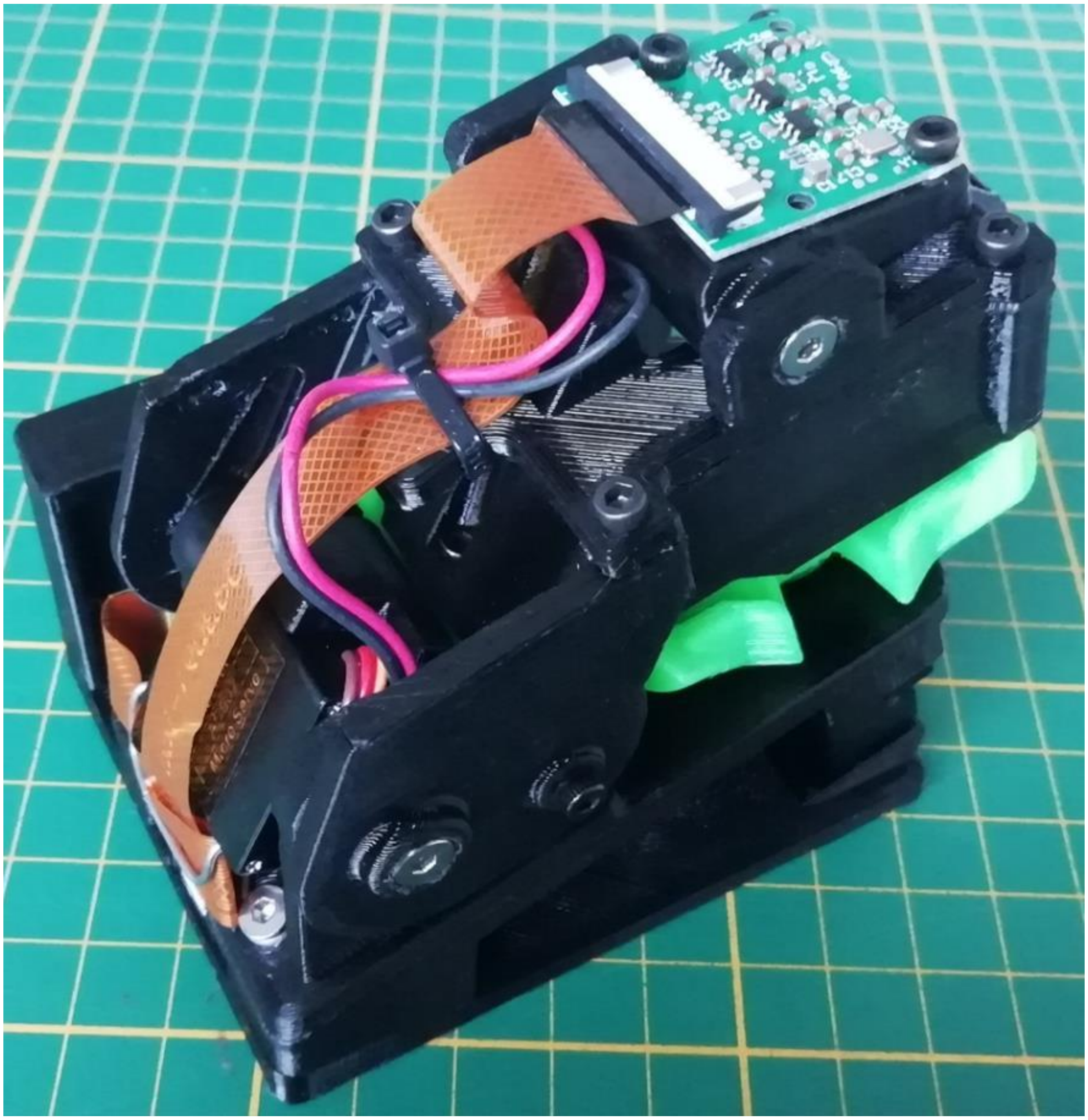
- 1x M3x8mm conical head screw.
- 1x M2,5x8mm cylindrical head screw.
- First slide in the left hook under the GPIO connector, and later rotate the right cover backward.





Step25: Tire wrap the flexible cable to the PiCamera_frame.

1. Ensure the flexible cable does not pull on the PiCamera connector.
2. Ensure the Led cables don't harm the camera flexible cable.

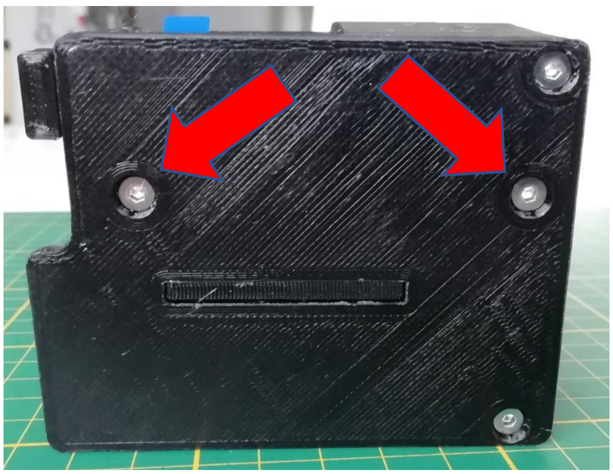


Step26 (optional): Additional and extendible baseplate.

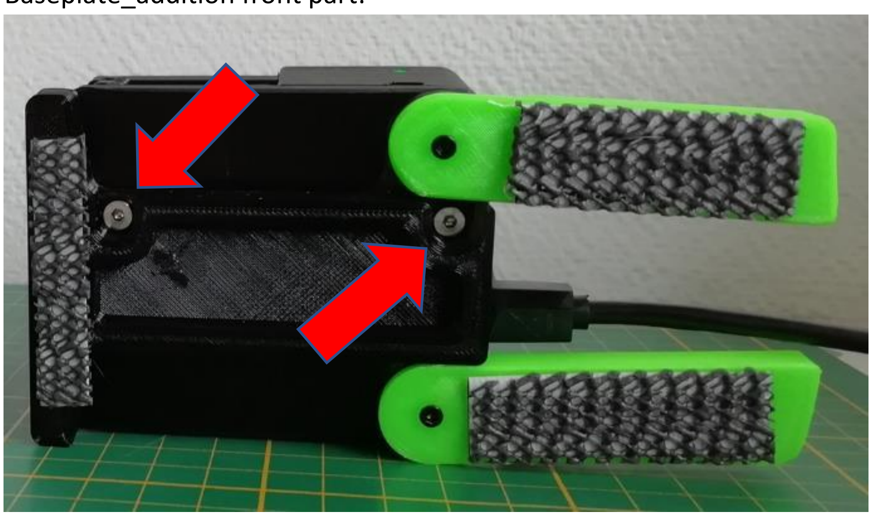
- 2x M2.5x8mm cylindrical head screw.
- 2x M3x12mm conical head screw.

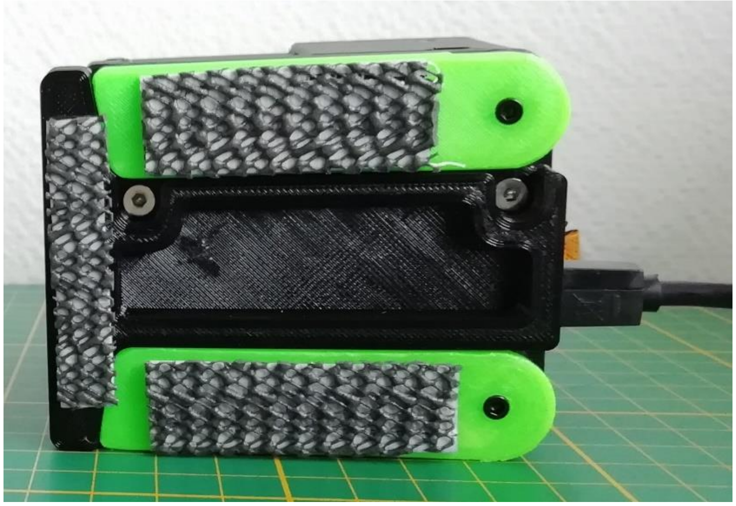
1. Connect the two “legs” to the Baseplate_addition, via the 2x M2.5x8mm:

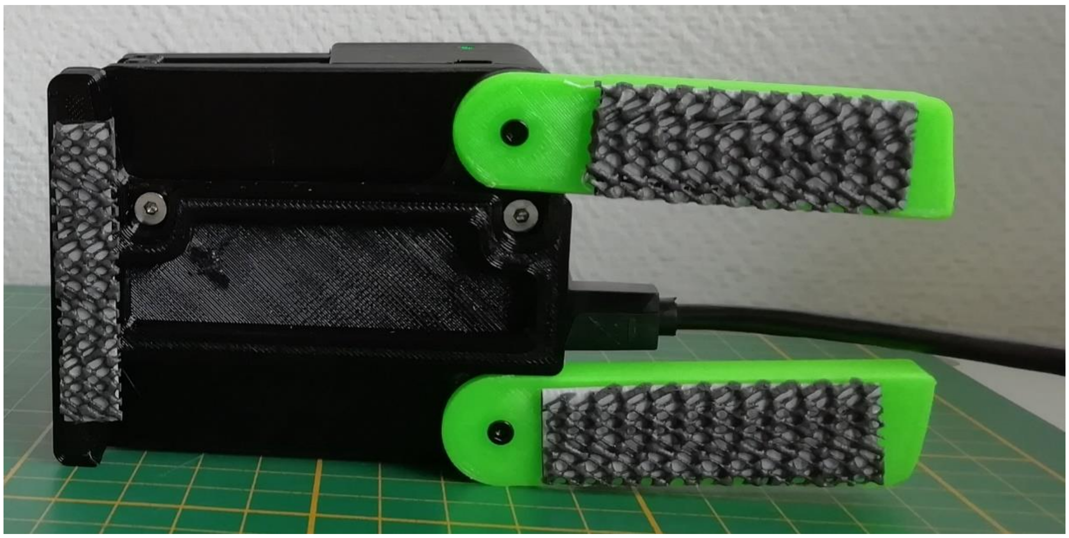
2. Remove the two screws (M3x8mm) indicated by the arrows below:

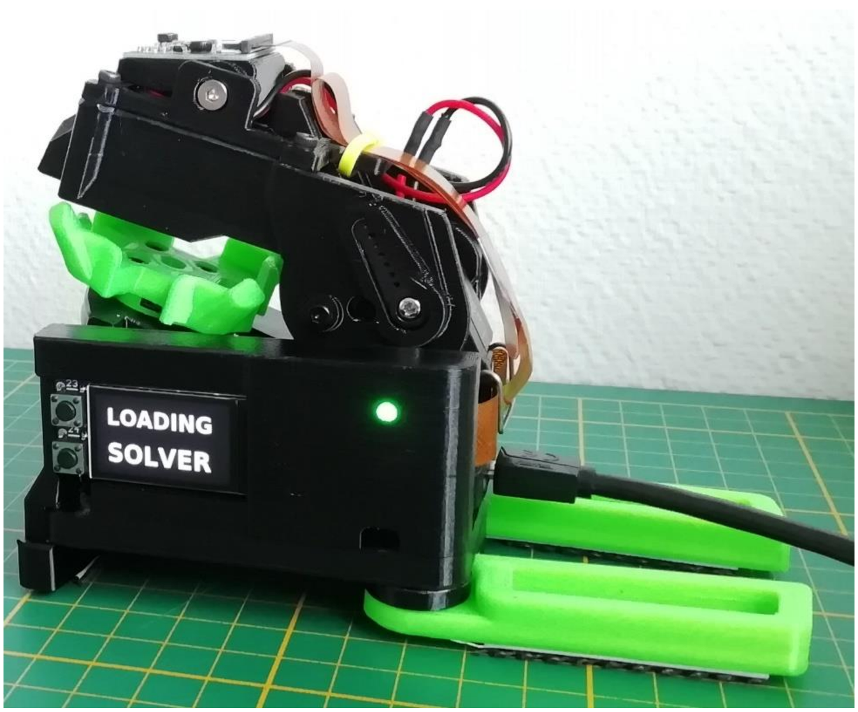


3. Connect the Baseplate_addition via the M3x12mm screws; Add some rubber pad on the legs and on the Baseplate_addition front part:








LOADING
SOLVER

23. 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_m_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_m_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:

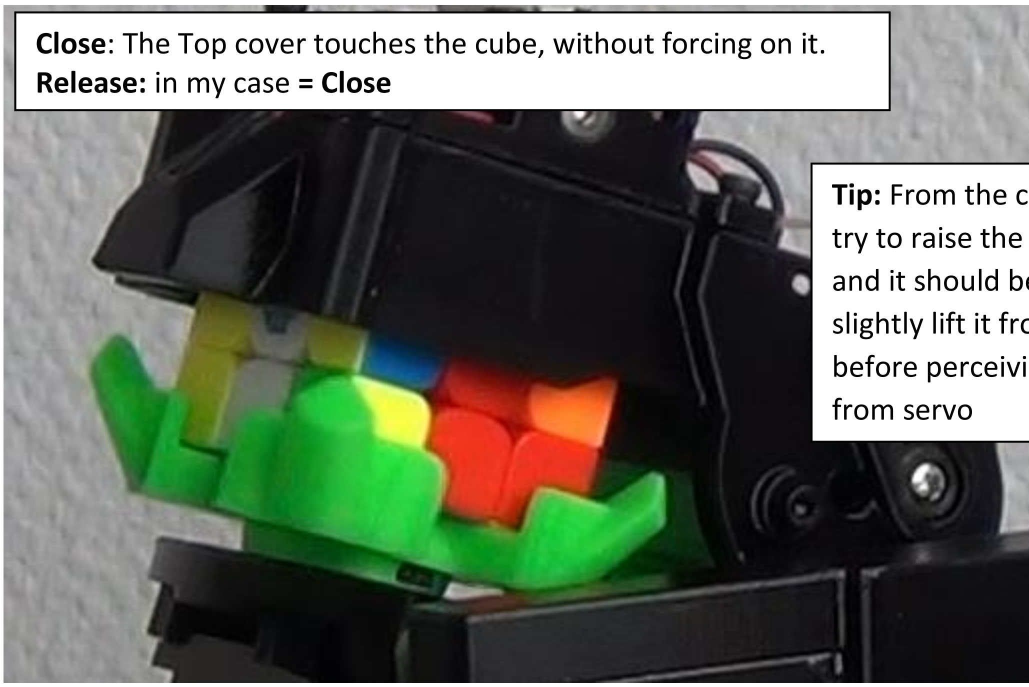
1. On Parameters and settings chapter are listed the involved variables and the default values.
2. The gpiozero library is used to control the servos, with a (float) parameter ranging from -1 to 1.
3. The float value entered on the '--set' argument (see Servos test and set to mid position chapter), represents 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).

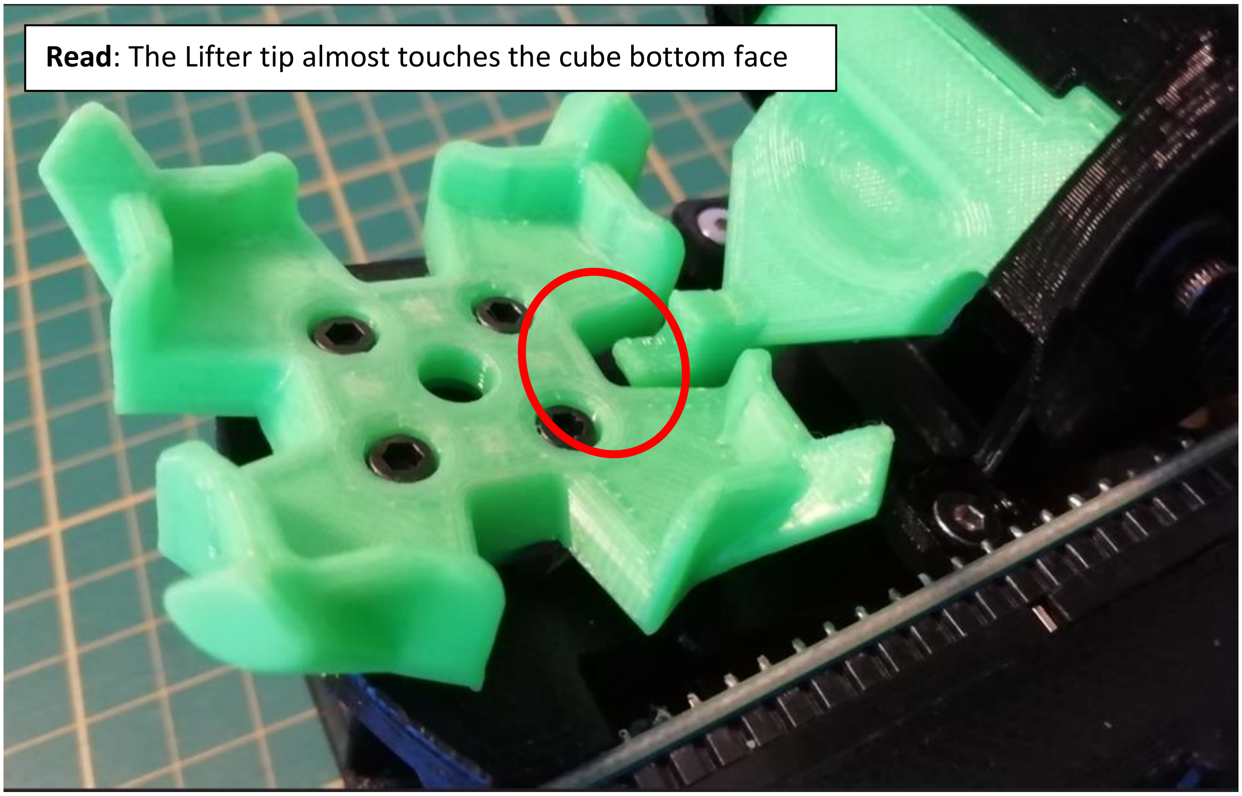
Top_cover (t_servo) angles:

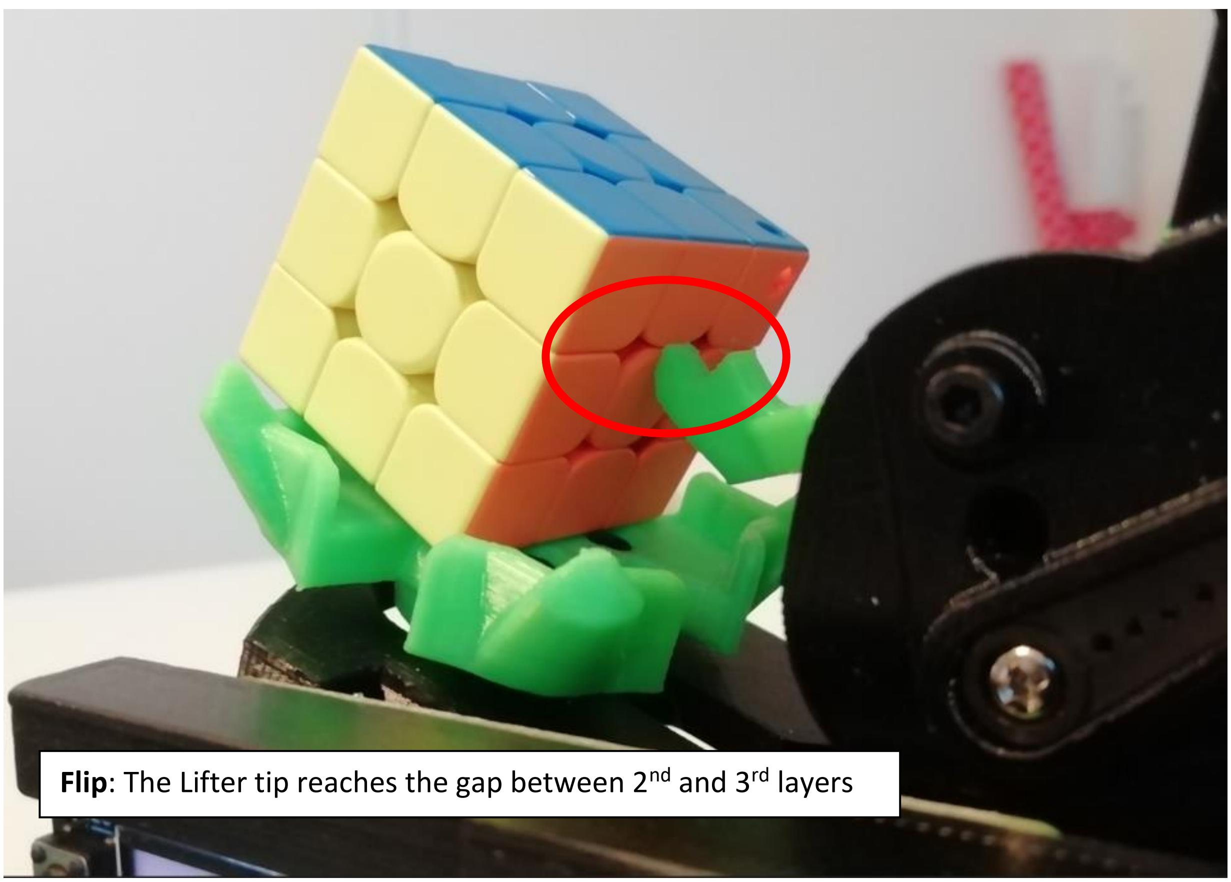
Working angles for the servos, are stored in a (json) text file: Cubotino_m_servo_settings.txt

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

Position	Description
Close	position to constrain the top and mid cube layers
Release	position to release tension from cube, at Close position (not really needed…)
Open	position without interferences with the cube and Cube_holder
Read	position for camera reading, with the Lifter almost touching the cube (and unfortunately constraining the Cube_holder)
Flip	position for the Lifter to flip the cube (about 2 cube layers height)




Read: The Lifter tip almost touches the cube bottom face

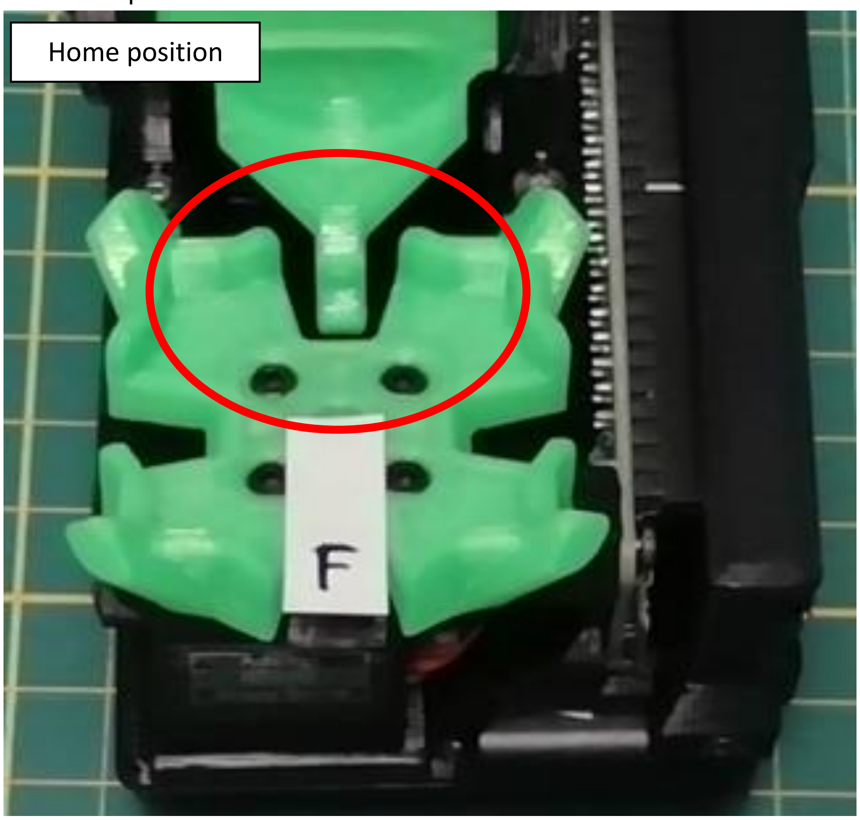

Flip: The Lifter tip reaches the gap between 2nd and 3rd layers

Cube_holder (b_servo) angles:

There are 7 defined angles (most of the 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
Release_CCW	release cube tension at CCW
Extra_home_CCW	release cube tension at home, when rotating from CCW
Extra_home_CW	release cube tension at home, when rotating from CW

The Home position must be well centered.

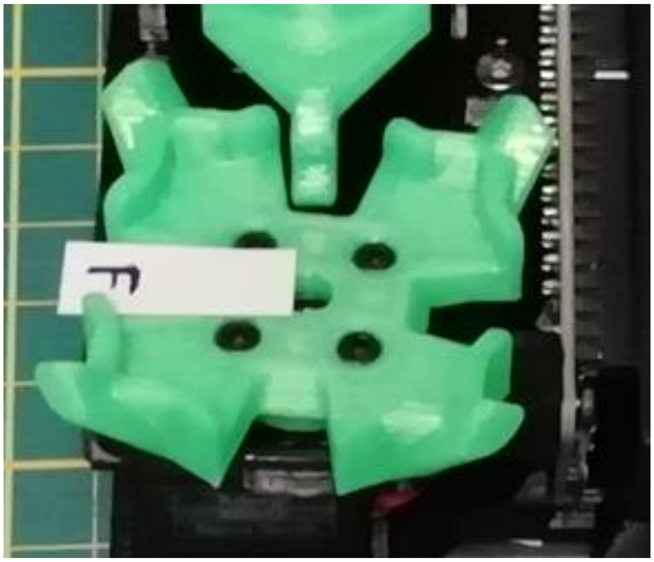


To center the Home position, it might be necessary to adjust the below parameters:

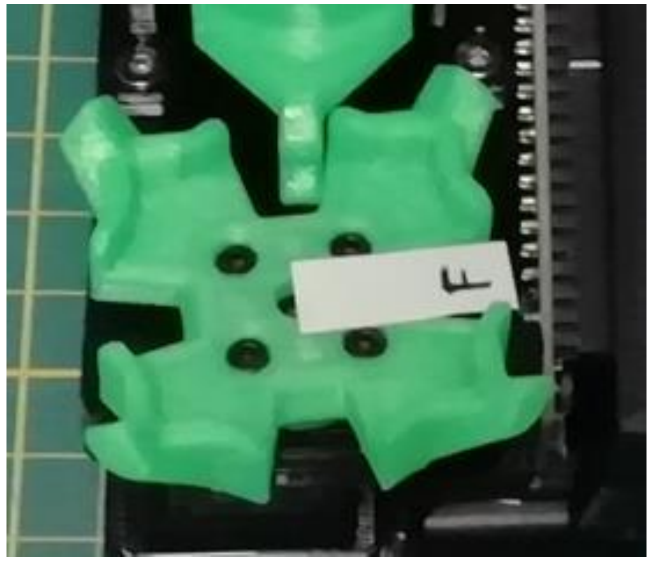
- b_min_pulse_width
- b_max_pulse_width
- b_home

Please note that the CCW and CW angles are slightly more than 90° apart from the Home position; These are the positions used by the robot when the Top_cover is closed and the Cube_holder rotates to CCW or CW.
This is needed to turn ~90° to the cube, after recovering the radial plays: There is play in between the cube and the Cube_holder, and again there is play between the cube and the Top_cover.

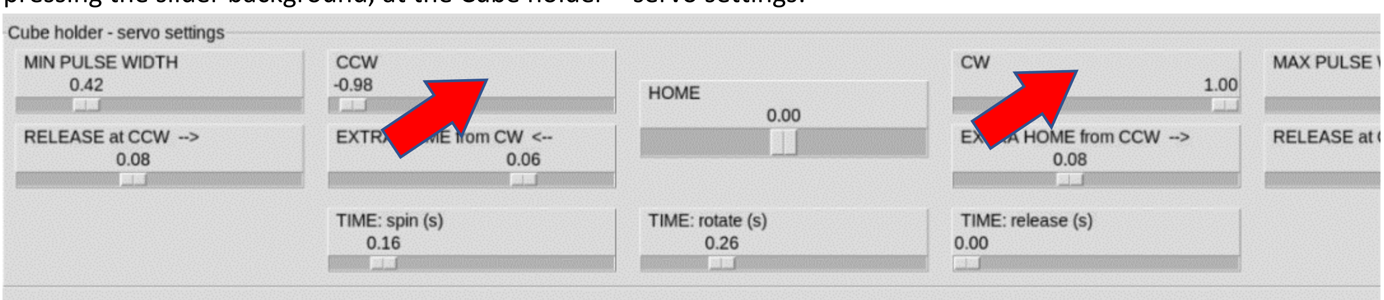
CCW position, before Release_CCW:



CW position, before Release_CW:



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:



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

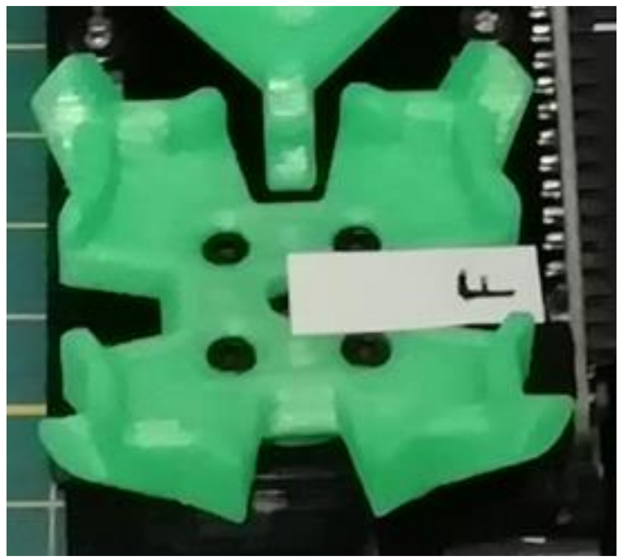
- This releases tension between the cube and the Cube_holder, and between the cube and Top_cover.
- It aligns the Cube_holder for cube flipping at CCW and CW.

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:

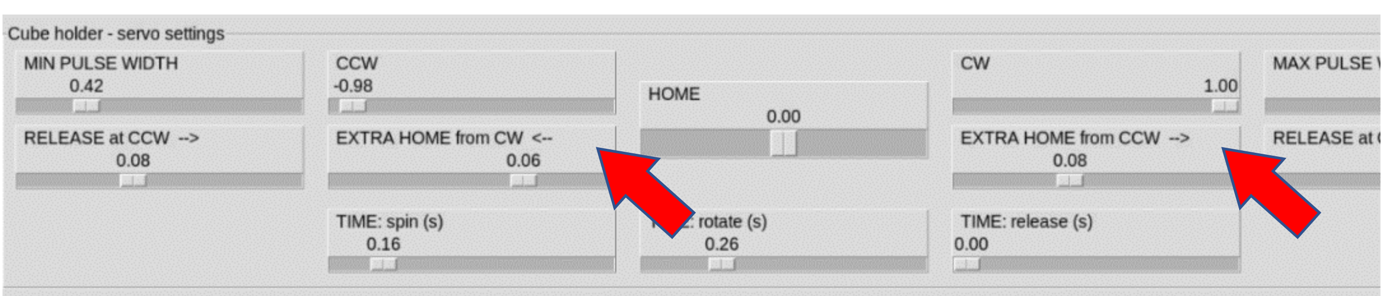
CW position, after Release_CW:



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

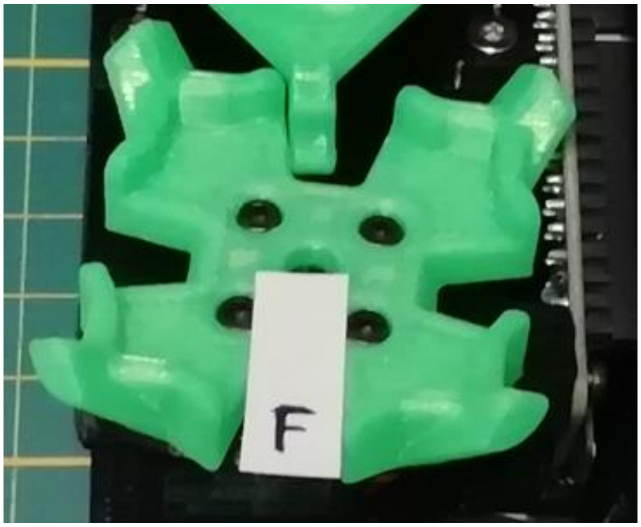
- This releases tension between the cube and the Cube_holder, and cube toward Top_cover.
- This aligns the Cube_holder for cube flipping at Home.

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



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

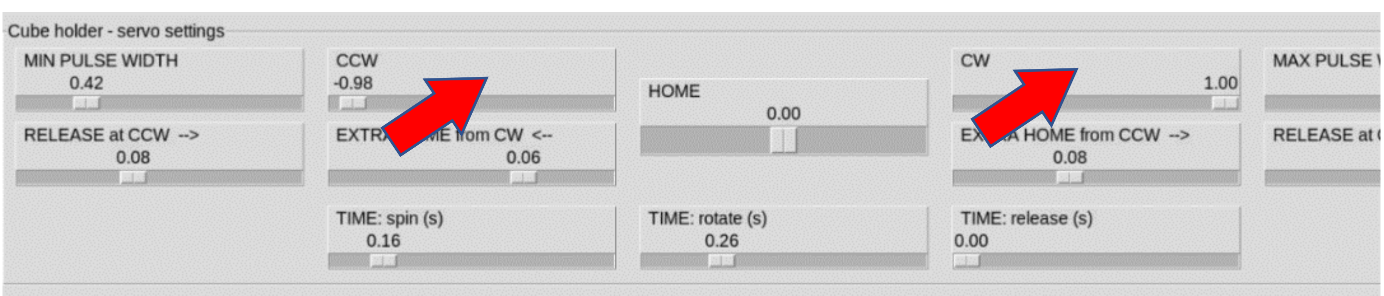
Extra Home CW position:



Extra Home CCW position:

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_m_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**.

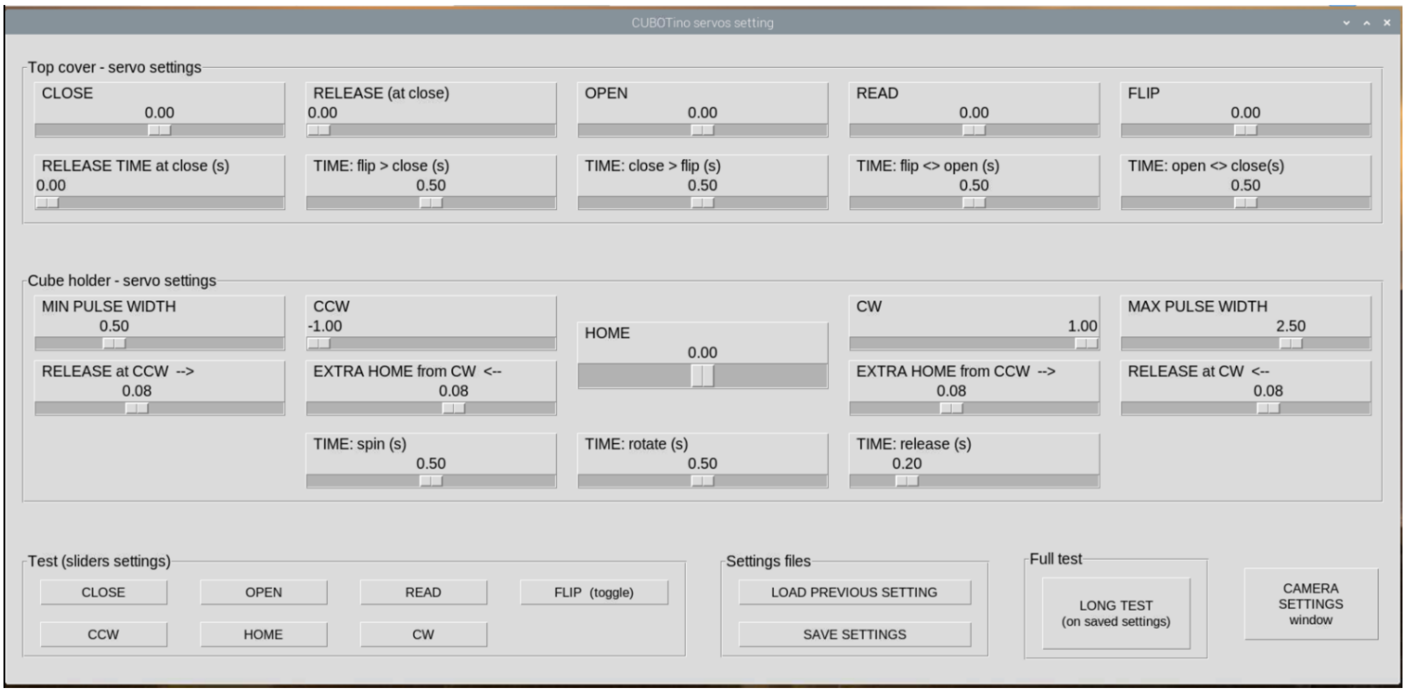
Rotate refers to the movement of the robot rotating the bottom layer of the cube only, i.e. a rotation of the Cube_holder with the top cover **closed**.

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).

24. 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_micro/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 servos_setting_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 😊

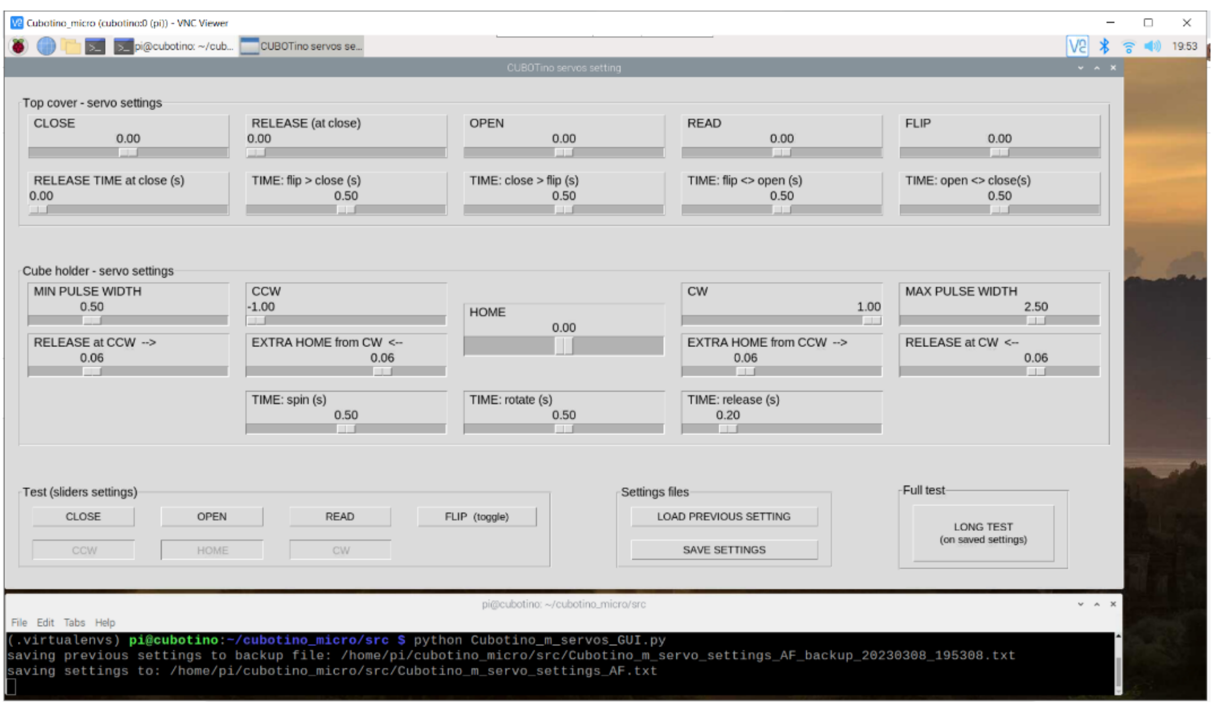
Suggested setting sequence for the Top_cover:

1. A correct “Servo_offset_compensator” (SOC) 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_offset_compensator 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:

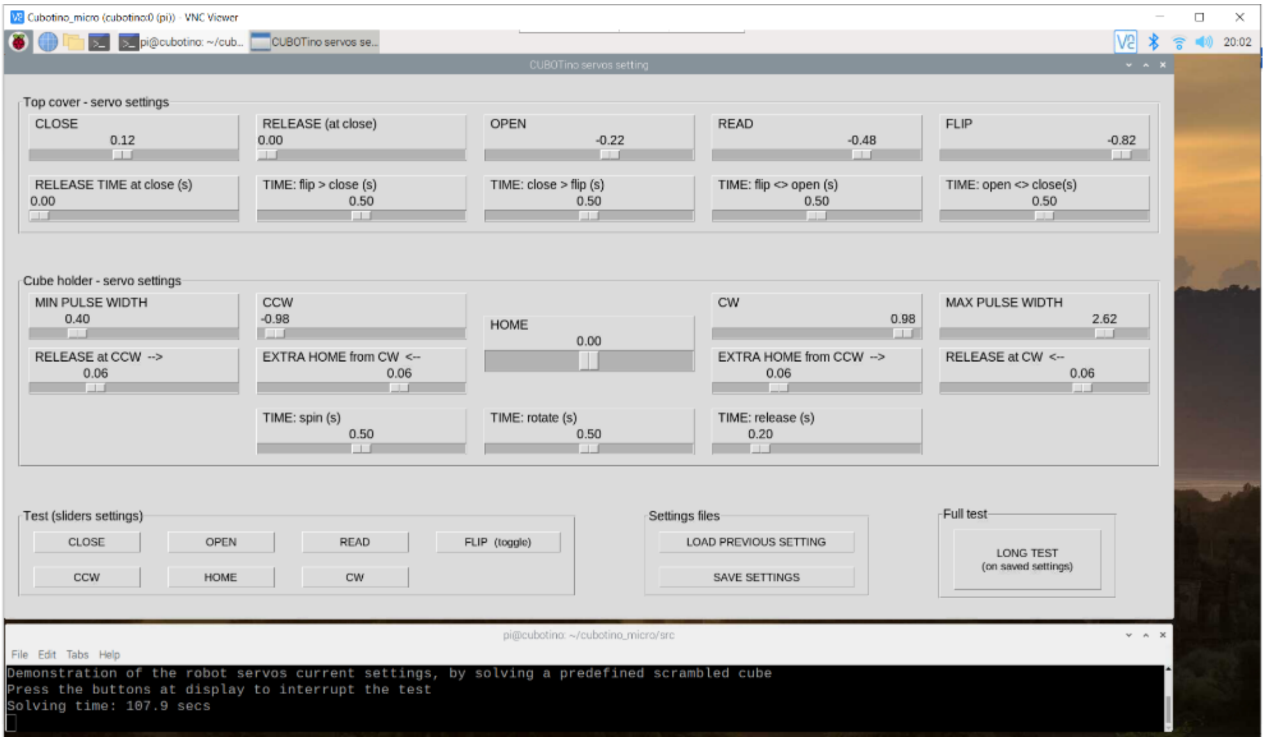
a. If the Top_cover is not in close or open positions, the Bottom_servo related widgets are blocked.
b. The Bottom_servo related buttons, at Test sliders settings, mimic what the robot does:
    - When the **Top_cover is in close position**, the positions "CCW", "Home" and "CW" are reached by first making 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:

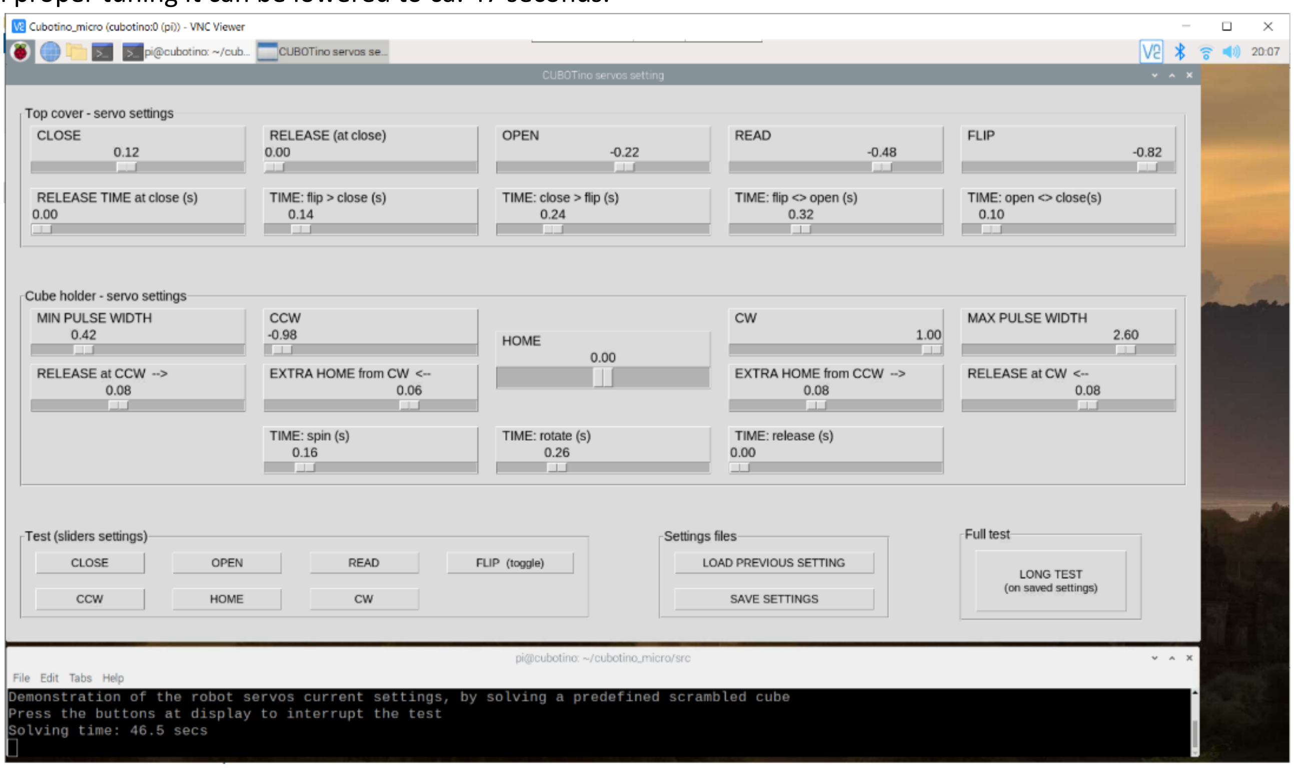
1. Press OPEN button to position the Top_cover to the open position: This activates the Cube holder – servo settings widgets.
2. Search the mix possible "Min Pulse Width":
    a. Verify the CCW slider to be at -1.00.
    b. Press CCW button.
    c. Decrease the Min Pulse Width slider, with decrements 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:

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

<u>LONG TEST:</u> 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.



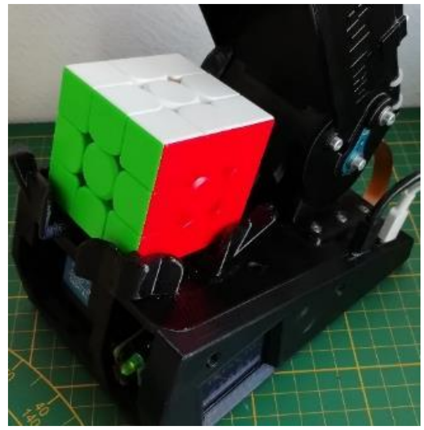
25. Tuning the camera via GUI

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

1. Different kinds of cubes:

For the Cubotino_micro project, the assumption is that only "frameless" cubes are used.
Anyhow, the cube facelets detection algorithm is the one from Cubotino Autonomous, and considering the possibility to use cubes with and without the frame:

classic (with stickers) classic (in-molded) frameless



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

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

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

2. Frame cropping:

Cropping parameters are set in a (json) text file: Cubotino_m_settings.txt
PiCamera position, and its Field of Viev (FoV), are likely to read both top and back cube faces.

Cubotino_m.py file is delivered with no cropping effect (variables set to zero): this to help the camera assembly angle to be set to get the cube top face centered, as shown on the left picture below.

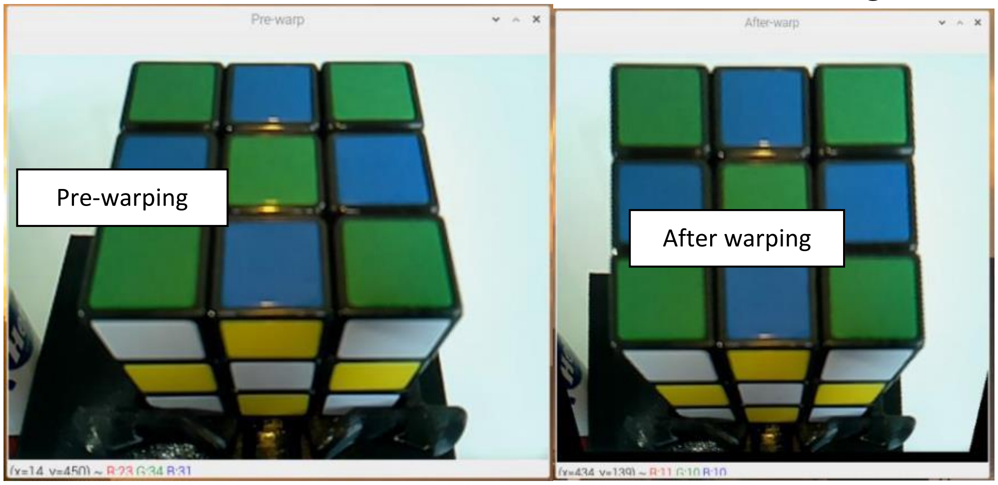


Image cropping must be tuned, to reduce the image to the region of interest (ROI).
Be noted the image cropping is made before warping it; Pictures on this page have been made by inverting these processes, to better show the potential problem.
Image warping does not prevent the risk to have some of facelets at back face, to be detected as part of the top face.
Another reason to set proper cropping parameters is to reduce the image size and gaining process speed.

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

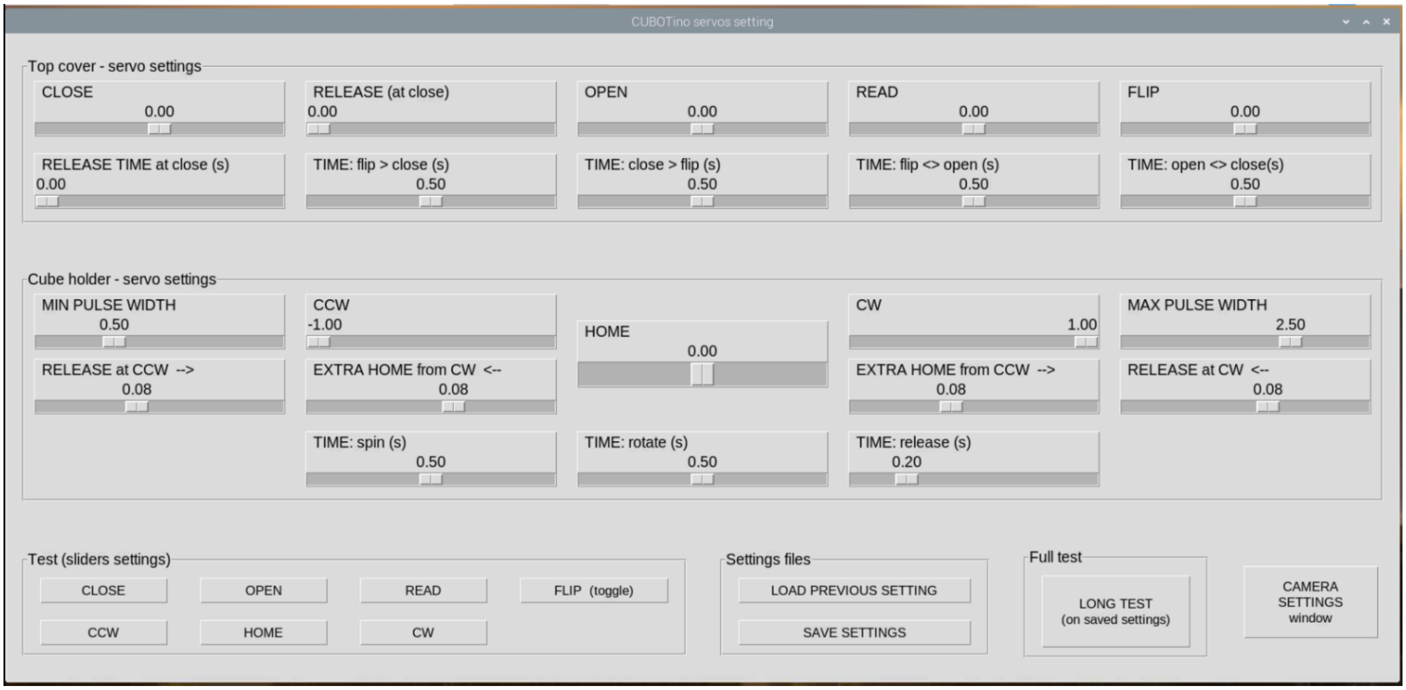
The cropping parameters identify the bandwidth, in pixels, to be removed from the four image sides.

3. Setup the camera with the GUI:

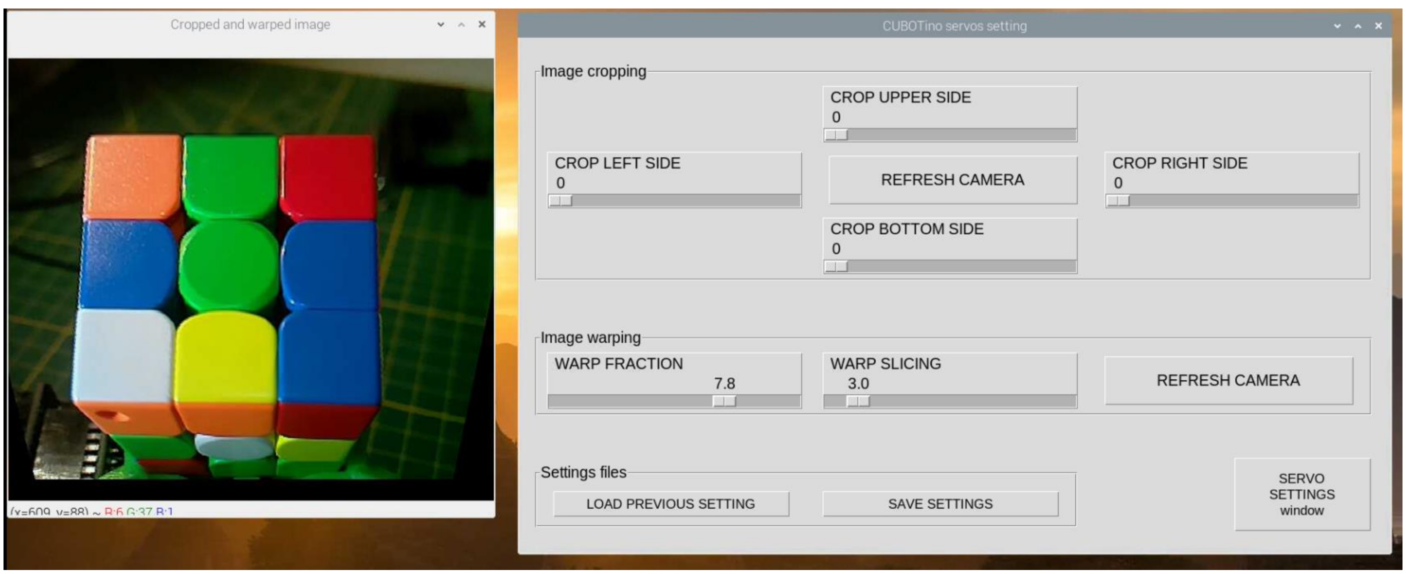
Access the tuning GUI, as done earlier for tuning the servos:

1.- Enter the cube folder by typing *cd cubotino_micro/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 servos_setting_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 3 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	

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.

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 there is 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.

Congratulations!

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

To preserve all your work to get there, 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.

26. 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:**
   Connect a 5V power source to the microUSB connector.
   In my case the servos work flawlessly with a phone charger rated 2A and the suggested power-bank/cable.
   Do not connect phone smart-charger, those that can deliver a voltage higher than 5.1V.

4. Run the Cubotino_m.py script

a. Access src folder: *cd cubotino_micro/src*

b. Activate the virtual environment: *source .virtualenvs/bin/activate*

c. Run the script: *python Cubotino_m.py*

d. Below arguments can be added (without parameters):

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

Only for test 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:

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

Examples:

python Cubotino_m.py - -debug - -F_deg
python Cubotino_m.py - -cv_wow - -F_deg
python Cubotino_m.py - -timer

7. **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. When the display shows the two options “SOLVE” and “SCRAMBLE”, press the upper button, next to “SOLVE”.
    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 one of the buttons for about 1 second.

8. **Start a scrambling cycle:**
    a. Position the cube on the cube holder.
    b. Cube layers should be reasonably aligned.
    c. When the display shows the two options “SOLVE” and “SCRAMBLE”, press the lower button, next to “SCRAMBLE”.
    d. The robot reacts by energizing the LED.
    e. The scrambling process can be interrupted at any time by pressing one of the buttons for about 1 second.

9. **Stop a scrambling or a solving cycle:**
    The scrambling or solving process can be interrupted at any time by pressing one of the buttons for about 1 second.
    In case the push button is maintained pressed longer, a warning is presented to the display.
    Very long pressing time (5 to 6 secs) of a button, is interpreted as intention to quit Cubotino-M.py script.

10. **Raspberry Pi shut down:**
    Please note that 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.
       Wait until the RPI “working” LED is off.
    b. If the robot has proved to work without errors, un-comment last row at Cubotino_m_bash.sh file (*halt -p*); This means the SBC will automatically shut down when the *Cubotino_m.py* script ends.
       To quit the *Cubotino_m.py* script, keep pressed one of the buttons long enough (ca 6 seconds) until the ‘SHUT DOWN’ appears on display then release the button.
       The SBC will close the open applications and files.
       If the button 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 go off.
       Through the Structure hole check if the green LED of the Raspberry Pi Zero is off; This typically takes an additional 10 seconds, afterward the power supply can be safely removed.

    If the cover_self_close parameter has been changed to ‘true’, the top cover will be closed automatically at the Raspberry Pi shutdown (at python script closure).
    This action is anticipated by some info on the display: Please be aware this might pose a risk to your kids if they have their hand on the way while the cover closes!

10. **Un-power the robot:**
    Detach the robot from the power supply.

11. 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_m_bash.sh file and the Cubotino_m.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:

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

```
pi@raspberry:~ $ ps aux
USER        PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
pi          624  0.0  0.2   4488   808 ?        Ss   16:24   0:00 /usr/bin/ssh-ag
pi          638  0.9  0.3   8760  1296 tty1     S+   16:24   0:00 -bash
pi          642  0.2  0.9  43400  3692 ?        Ssl  16:24   0:00 /usr/lib/gvfs/g
root        657  0.0  0.2   7676  1048 ?        S    16:24   0:00 bash -l /home/p
pi          664  0.2  0.9  56752  3384 ?        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
```

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

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

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

12. “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_m.py:

argument	parameter	notes
- - cycles	Int > 0	• Quantity of consecutive scrambling and solving cycles. • If not provided, the robot will wait for commands from the buttons.
- - pause	Int >0	• 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	• 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_m.py --cycles 10* will scramble and solve the cube 10 times.
- *python Cubotino_m.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_m.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:

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


```
#######################    END  SOLVING CYCLE  1   ##########################

Next cycle: [.............                                      ] 29.5%   14s
```

27. Automatic start

When 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 $ sudo crontab -e

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

Choose 1-3 [1]:
```

Un-comment the last row
MAILTO=""
@reboot su - pi -c "/usr/bin/vncserver :0 -geometry 1280x720"
#@reboot su - pi -c "/usr/bin/vncserver :0 -geometry 1920x10800"

#@reboot /bin/sleep 5; bash -l /home/pi/cubotino_micro/src/Cubotino_m_bash.sh > /home/pi/cubotino_micro/src/Cubotino_m_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_m_bash.sh* file.

The *Cubotino_m_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_m.py*" to "*python Cubotino_m.py - -timer - -F_deg*"

29. Rpi shut down via display button

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

From the folder */home/pi/cubotino_micro/src* , edit the file with *sudo nano Cubotino _m_bash.sh*

and uncomment the *#halt -p* command.

```
#!/usr/bin/env bash

##########   Andrea Favero,  01 March 2023  ######################################################
#  This bash script activates the venv, and starts the Cubotino_m.py script, after the Pi boots.
#  When the python script is terminated wiithout errors (long button press), the Pi shuts down
#  (check notes below before uncommenting the "halt -p"command)
#################################################################################################

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

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

# runs the robot main script (--fast option requires a good cube holder tuning)
# python Cubotino_m.py --fast
python Cubotino_m.py

# 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_m.py exited on request"
        echo ""
    else
        echo ""
        echo "Cubotino_m.py exited with error"
        echo ""
    fi

else
    echo ""
    echo "Successfully executed Cubotino_m.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
```

Note: Do not uncomment:

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

30. 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. Cube layer (bottom, or central vertical) doesn't align nicely.
2. Top cover usage to flatten the cube.
3. Cube status detection error.
4. Robot stuck on reading the same face.
5. Cube status detection error.
6. The program doesn't work as intended.
7. PiCamera focus.
8. Updating the Cubotino software.
9. Raspberry Pi freezing (memory management).
10. Raspberry Pi Wi-Fi dropping.
11. Servos stop working.

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

This is probably the most difficult part of the tuning process, perhaps of the complete project.
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 slight 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.
The GAN 330 keychain 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.
   - d) evaluate if sufficient friction increment.

2 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 lowering 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.

Be noted the Release should be minimized to 0.02 max 0.04, differently the Top_cover will not properly constrain the mid cube layer.

3 Cube detection error:

It is returned when the interpreted cube status isn't coherent, meaning not having 9 facelets per color 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-colored 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 center, 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_m_settings.txt the frameless_cube parameter is not 'true' or 'auto'.
6. 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.

4 Robot stuck on reading the same face, until timeout:

a. When the frameless_cube is set 'false' (classic cube type), the cube status detection algorithm must find 9 facelets with given characteristics before changing face; If the robot doesn't change the cube face, it is because some of the pre-conditions aren't met (at least 9 facelets, areas of the facelets, distance between the facelets, etc)
b. When the frameless_cube is set 'true' or 'auto', the cube status detection algorithm must find 5 to 7 facelets with given characteristics before changing face; The remaining facelets are estimated for the position, not the color.
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 there is a lot of ambient light 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_m.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 facelet (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 PiCamera_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 facelet: 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.

5 Cube status detection error

1 The first possible reason related to the logo presence on the White center facelet:

With a nail pull the facelet off.
Lay a piece of sandpaper (grit > 800) on a hard and smooth surface.
Gently slide the facelet with the logo on the sandpaper, with circular movements
Frequently check when the logo is sufficiently removed.

2 Cube's facelet and light reflection (cube status detection):

Detection of edges, as well as colors, can be largely affected by light reflection made by the facelets.
I assume we are all using the GAN330 cube, that doesn't require to mat the facelets.
Sometimes the problem relates to shadow on part of the cube face: Change the bot orientation or make a uniform shadow by using a panel to shield the light.

6 **PiCamera focus:** See specific chapter.

7 Program doesn't work as intended:

This is a difficult topic, as my coding skills are rather limited …..
A good starting point is to get some feedback from the script:

a. Run Cubotino_m.py with --debug argument; This variable is used by many functions to print out info to the terminal.
b. Check the prints.
c. If the printout doesn't suggest much to you, share it at the Instructables chat.

When the problem seems more related to the servo program:

a. Edit Cubotino_m_servos.py
d. At about row 71 change the Boolean “s_debug” to True. This variable is used by many functions to print out info to the terminal.
e. Run Cubotino_m_servos.py (activate the venv first and recall typing python in front). This code activates the servos like solving a predefined scrambled cube.
f. Check the prints.
g. Run Cubotino_m.py and let it call the Cubotino_m_servos.py.
h. Check the prints.
i. If the prints out don’t suggest much to you, share it at the Instructables chat.

8 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”, Step 1 in “Appendix 1 – Additional Information” for details.

9 Raspberry Pi freezing (memory management):

In the 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 written 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 12 in section 2 of the present document for details.

10 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.

11 Servos stop working:

Today, 19/03/2023, I have experienced a second servo failure.

First failure:

1. Happened to the Top servo.
2. happened with about 250 full solving cycles (logged data) and large yet unknown quantity of tests.
3. happened when I was trying to minimize the robot solving time, without taking care to let the servos cooling down.
4. I gave full responsibility to myself.
5. I exchanged the servo with a new one, from the batch (I bought a kit of 4 pieces)

Second failure

1. Happened to the bottom servo.
2. happened with about 450 full solving cycles (logged data) and large yet unknown quantity of tests.
3. Happened with very low duty cycle, therefore not a real temperature related issue.
4. I exchanged the servo with the last new one I had still available.

Because the 2nd failure happened without overheating the servo, I decided to open the first servo to check whether I could see something obvious.

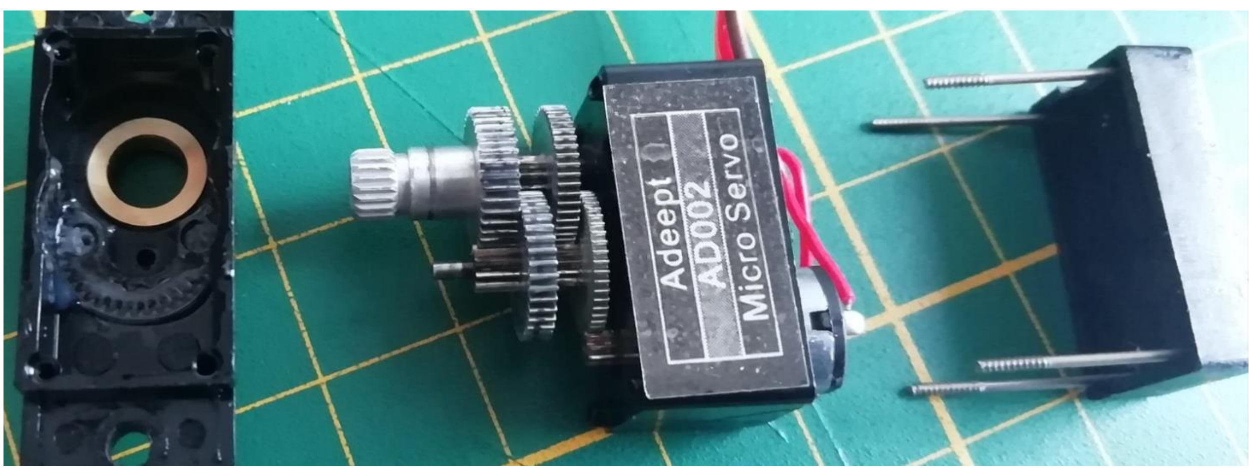
The most relevant aspects I could observe:

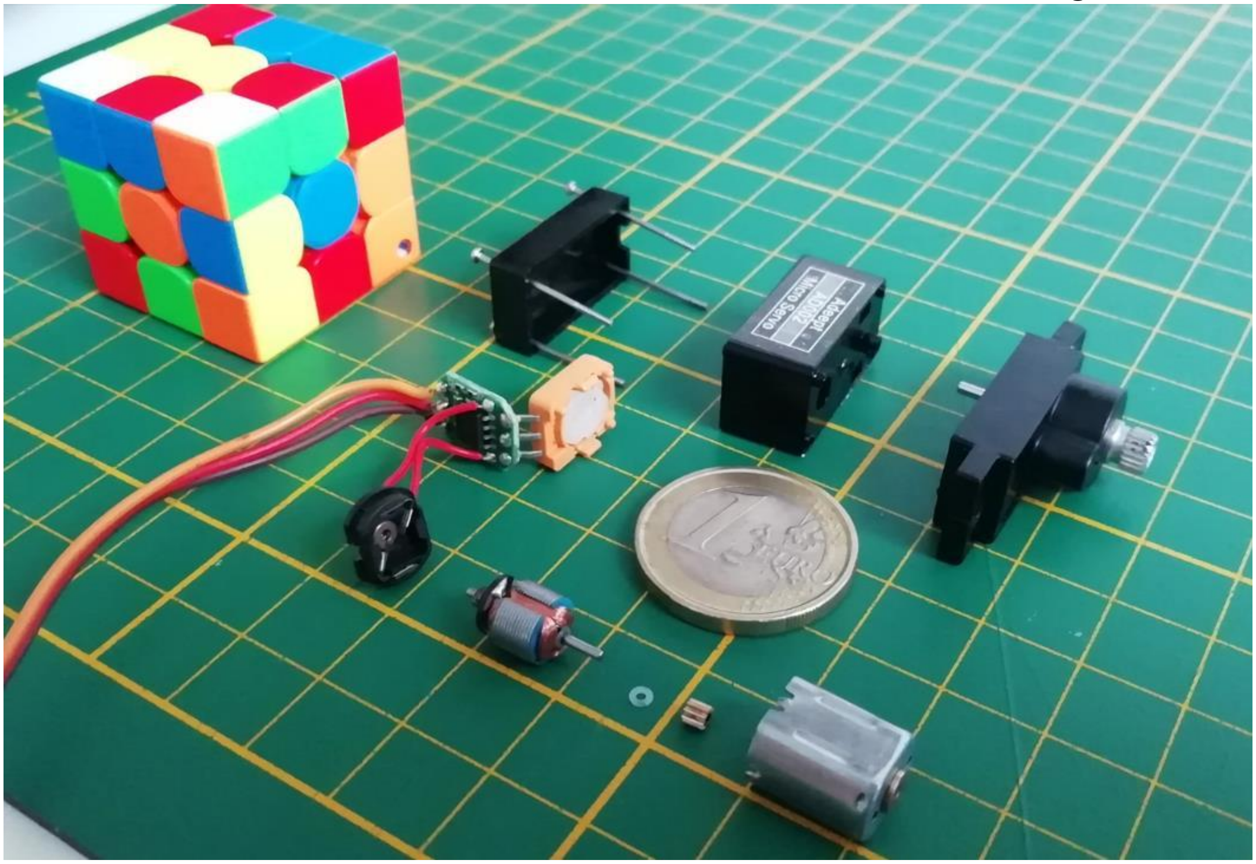
1) The overall construction quality is great:
    a. All gears are made of metal.
    b. Brass bushes in all the needed locations.
    c. Custom potentiometer with snap hooks fitting well in position.
    d. Large magnets, with very clean edges.
    e. Rotor coil very well made.

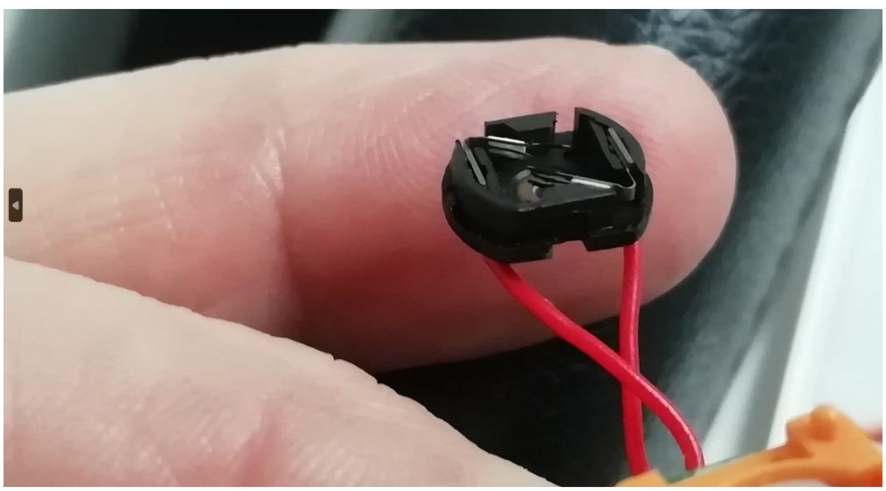
2) Brushes were at the end of life. The motor, and consequently the brushes, are extremely small and were partially broken.

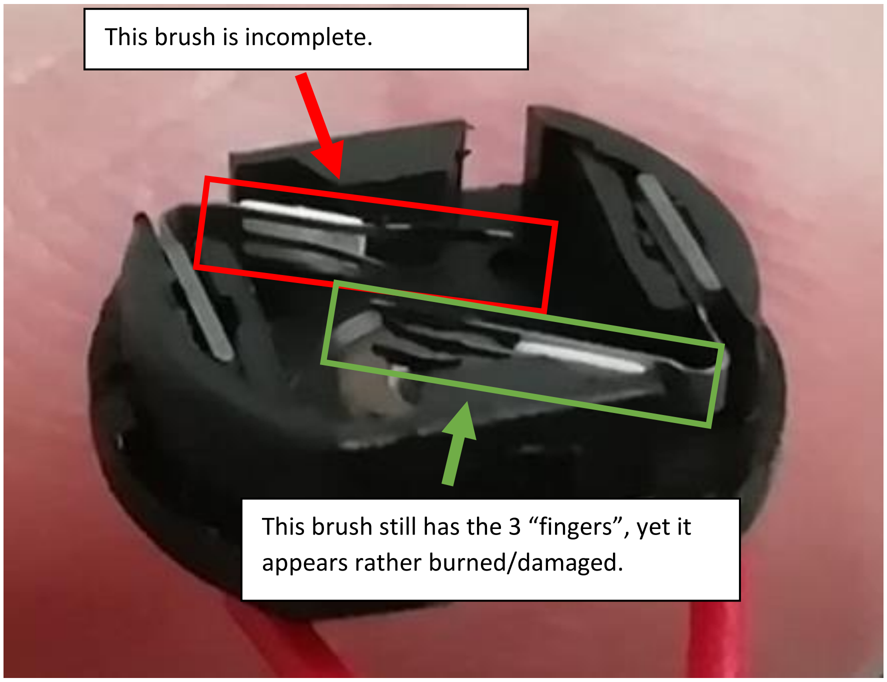
Because of the overall very good construction quality, I have sent an email to the manufacturer, with the aim to:

1 check if there are versions with higher reliability.
2 Set expectations on what to expect from these tiny servos.

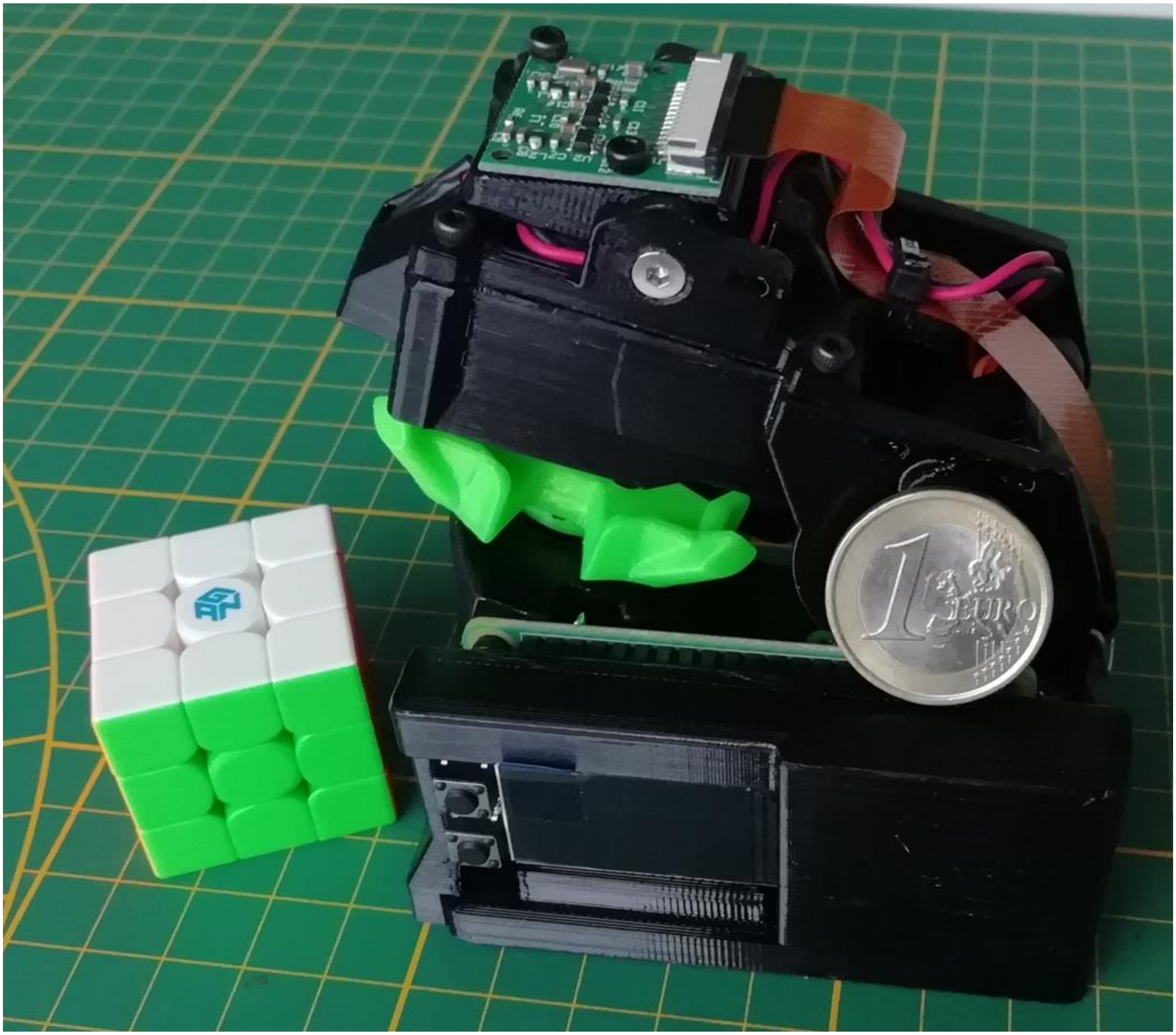


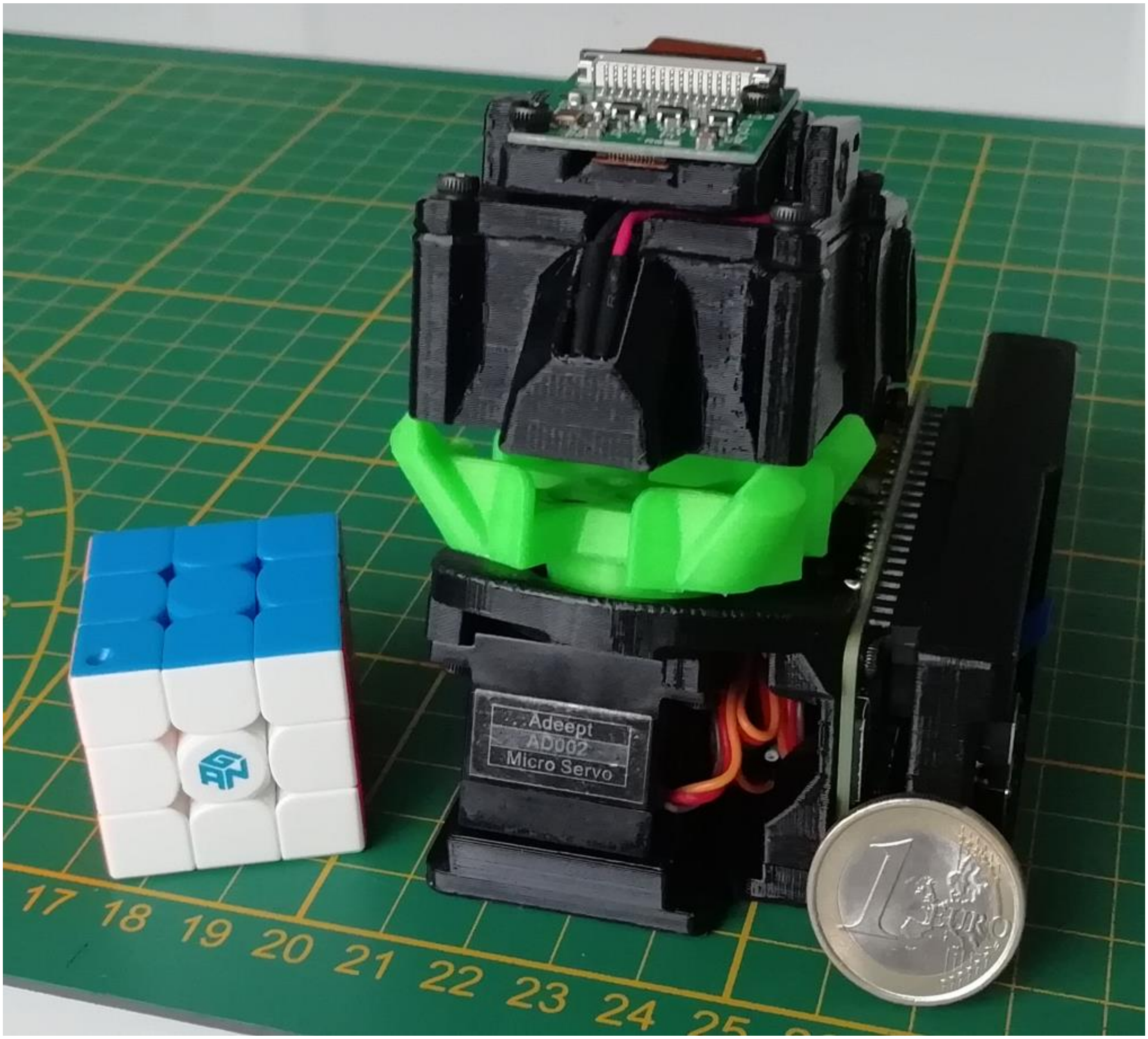

Adeept
Micro Servo

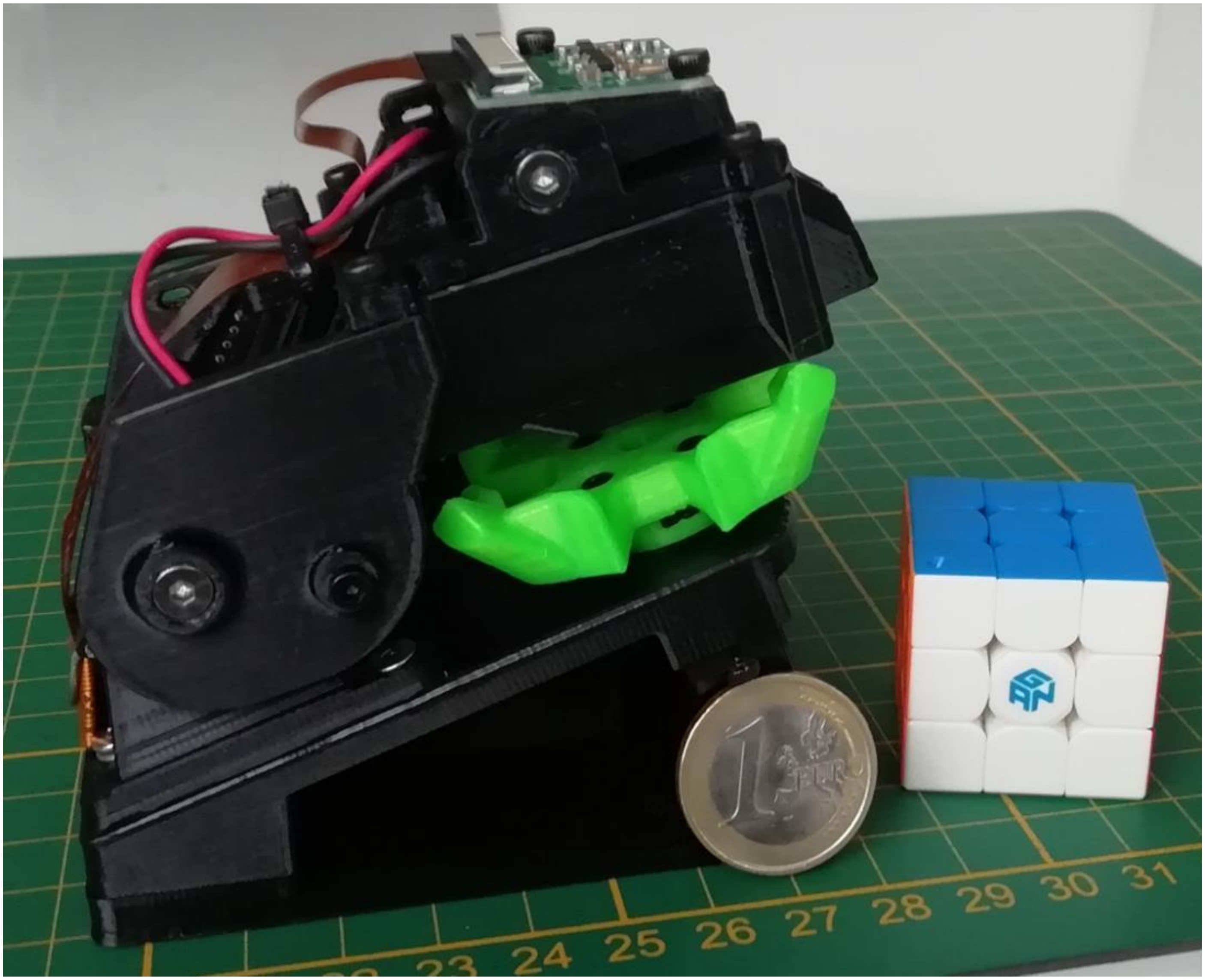


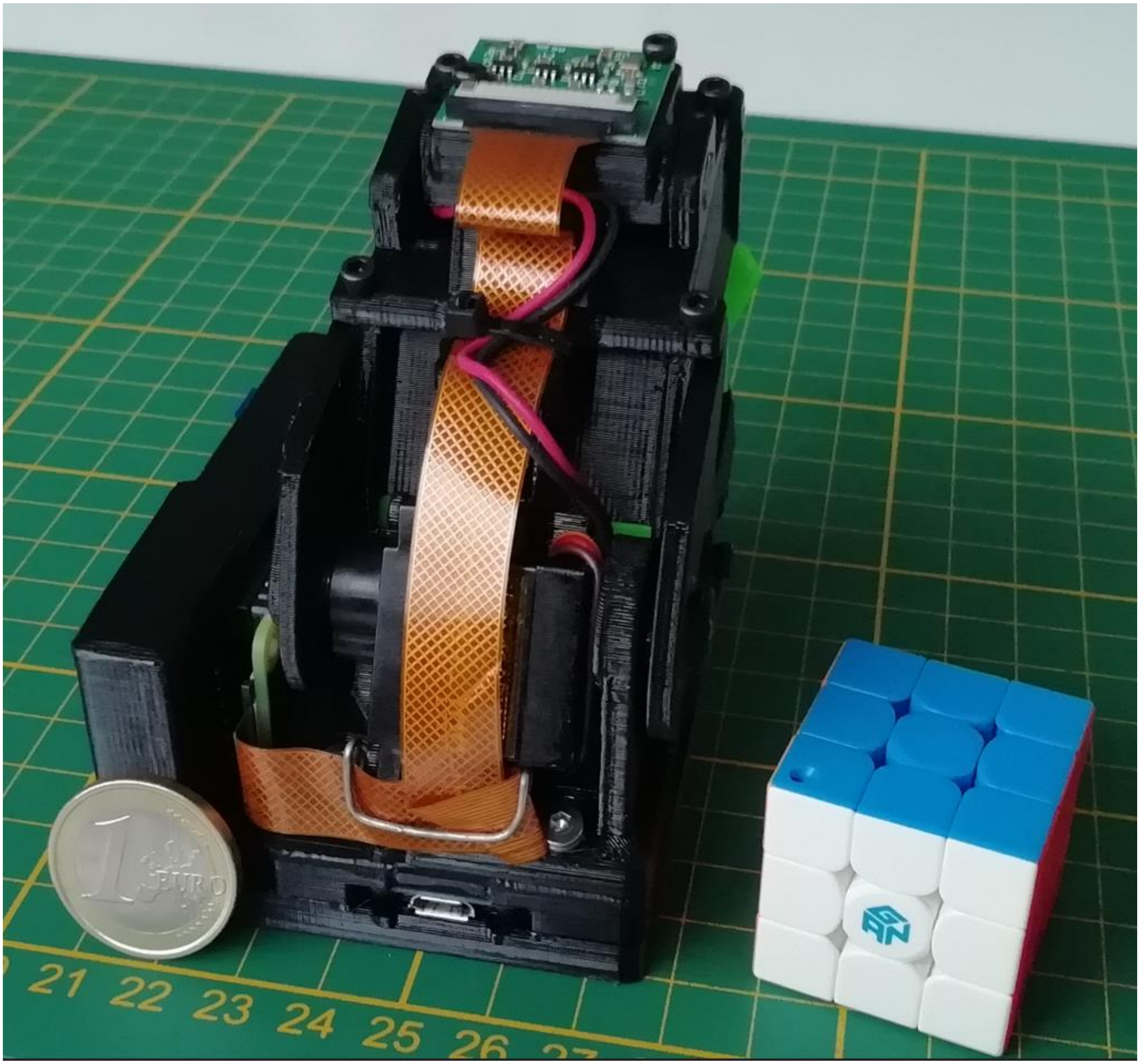

This brush is incomplete.
This brush still has the 3 “fingers”, yet it appears rather burned/damaged.

31. Collection of robot’s pictures




Adeept
AD002
Micro Servo


23 24 25 26 27 28 29 30 31



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_micro/*
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_micro folder.
2. Personal settings are "Cubotino_m_settings.txt and Cubotino_m_servos.settings.txt"
3. The robot also makes a backup copy of your settings every time the Cubotino_m.py is started; This gives you one more chance to recover your previous settings (at least until Cubotino_m.py is executed after the update); In that case remember to rename the backup files:
   Cubotino_m_settings_backup.txt → Cubotino_m_settings.txt
   Cubotino_m_servo_settings_backup.txt → Cubotino_m_servo_settings.txt

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 specific files are copied into */home/pi/cubotino_micro/src* folder, by the installation process:

File	Purpose	Notes
Cubotino_m.py	Main robot script	
Cubotino_m_moves.py	Translates the cube solution (Singmaster notation) in robot moves	
Cubotino_m_servos.py	Manages the servos and LED.	
Cubotino_m_servos_GUI.py	GUI to help with servos setting	
Cubotino_m_display.py	Manages the display	
Cubotino_m_set_picamera_gain.py	Manages the Camera gains settings	
Cubotino_m_Logo_265x212_BW.pdf	Cubotino logo, for the display	
Cubotino_m_settings_manager.py	Class interacting with the setting files	From rev 1.0 (15th March 2024)
Cubotino_m_settings_default.txt	Json file with settings for Cubotino_m.py script	Default values, to start the tuning
Cubotino_m_settings_AF.txt		Optimized values for my robot
Cubotino_m_servo_settings_default.txt	Json file with settings for Cubotino_m_servos.py script	Default values, to start the tuning
Cubotino_m_servo_settings_AF.txt		Optimized values for my robot
Cubotino_m_bash.sh	Bash file to start-up the robot script automatically after Raspberry Pi boots	
get_macs_AF.py	Makes settings files names based on how many mac addresses are listed at macs_AF.txt	Allows me (AF) to manage specific settings files for my robots while keeping generic start-up settings files for other makers
macs_AF.txt	Lists of macs addresses	

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

Library	Forced version	Scope and notes
Kociemba solver (twophase)	1.1.1	Kociemba solver for the (almost optimum) cube solution: https://github.com/hkociemba/RubiksCube-TwophaseSolver This library is made by 20 python files and 19 Lookup tables files; Most of the files are copied to */home/pi/cubotino_micro/src/twophase*
numpy	1.21.4	Image array manipulation and many other functions
picamera[array]		Outputs camera images in array format
st7735	0.0.4.post1	Driver for the display (purpose to have the installation compatible with Autonomous version too)
st7789	0.0.4	Driver for the display
RubikTwoPhase	1.1.1	Kociemba TwoPhase solver

getmac	0.8.3	Get MAC address (useful when more Cubotino bots with different settings)

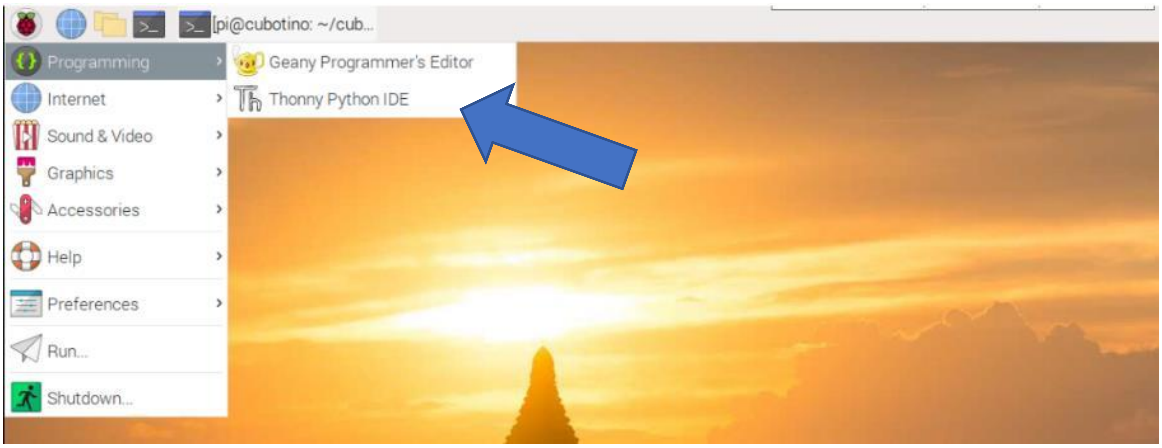
3. Set Thonny IDE interpreter

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.

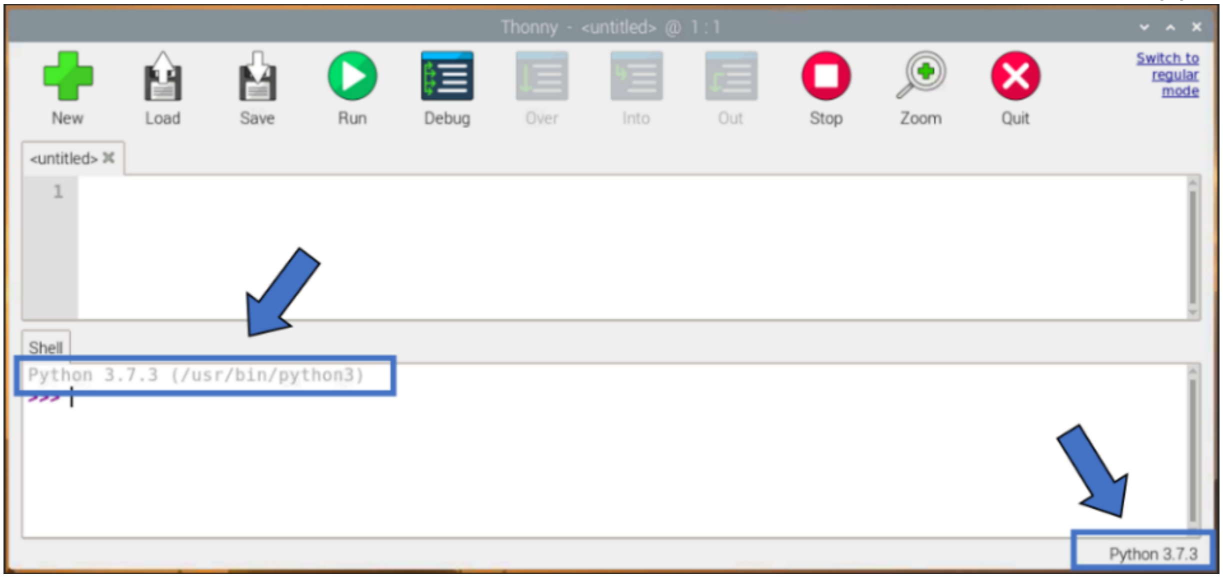
Thonny is part of the Raspberry Pi installation (according to the 'Setting up Raspberry Pi' procedure):

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.

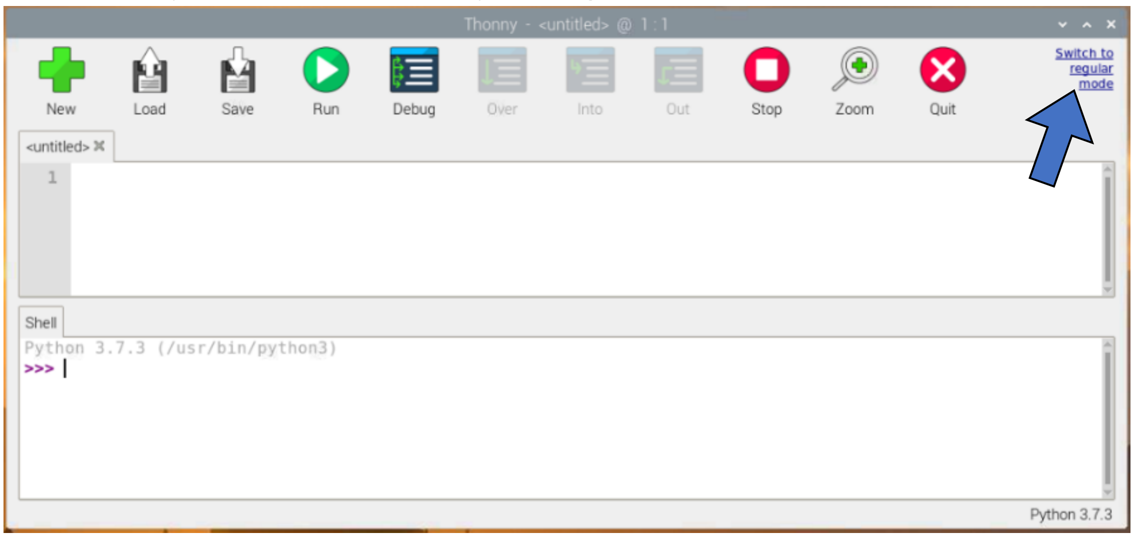


Setting up Thonny IDE interpreter, to work with the venv, it will be handy to tune the parameters hard coded in the scripts.

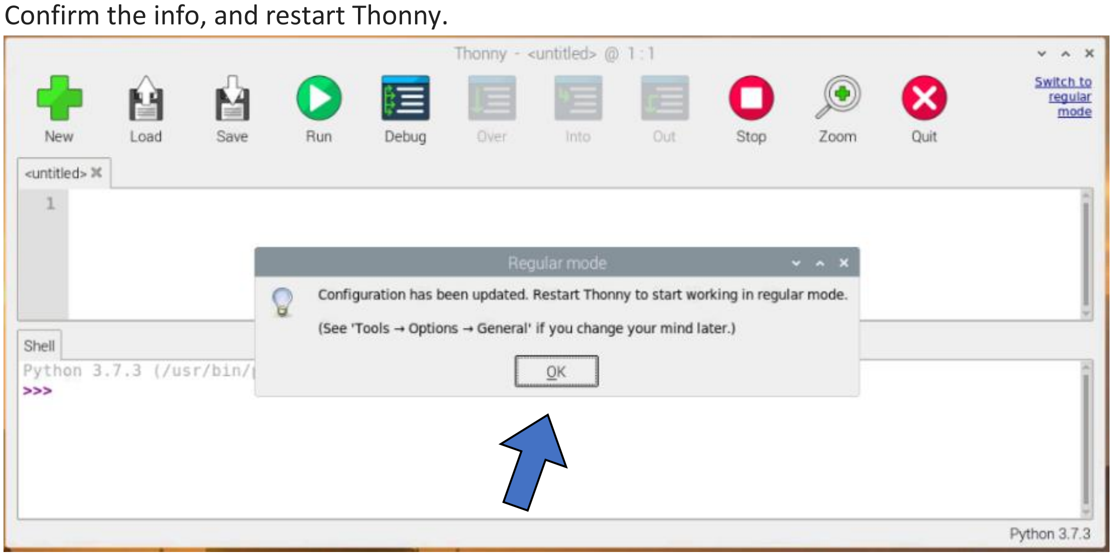
Thonny opens with the standard interpreter (/usr/bin/python3), and if you run Cubotino_m.py it won't find the libraries….



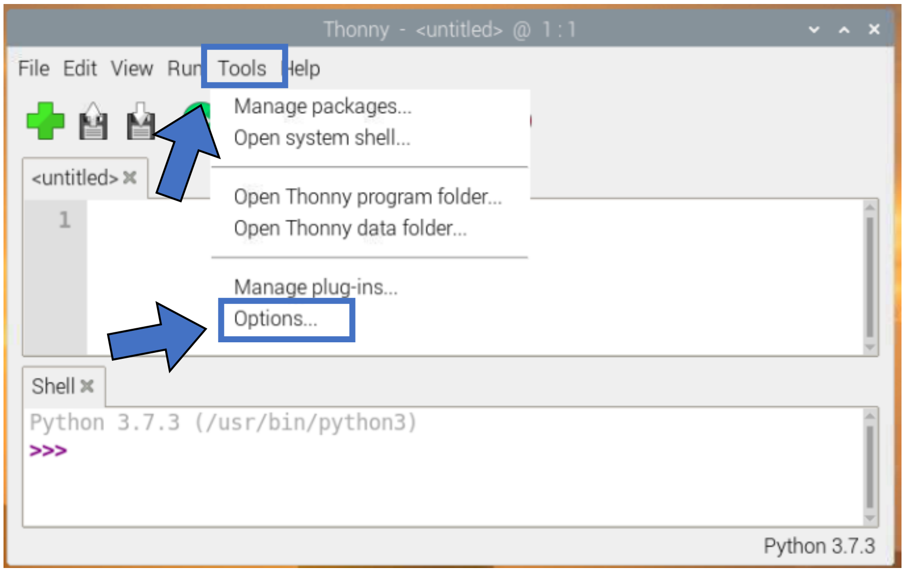
To have the Option menu, it is necessary to change the mode:



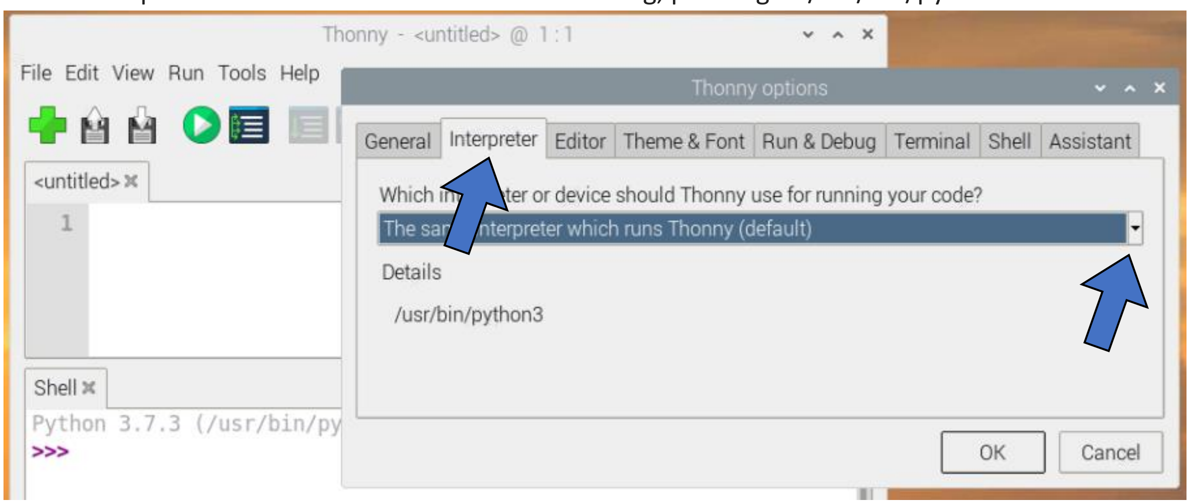
Confirm the info, and restart Thonny.



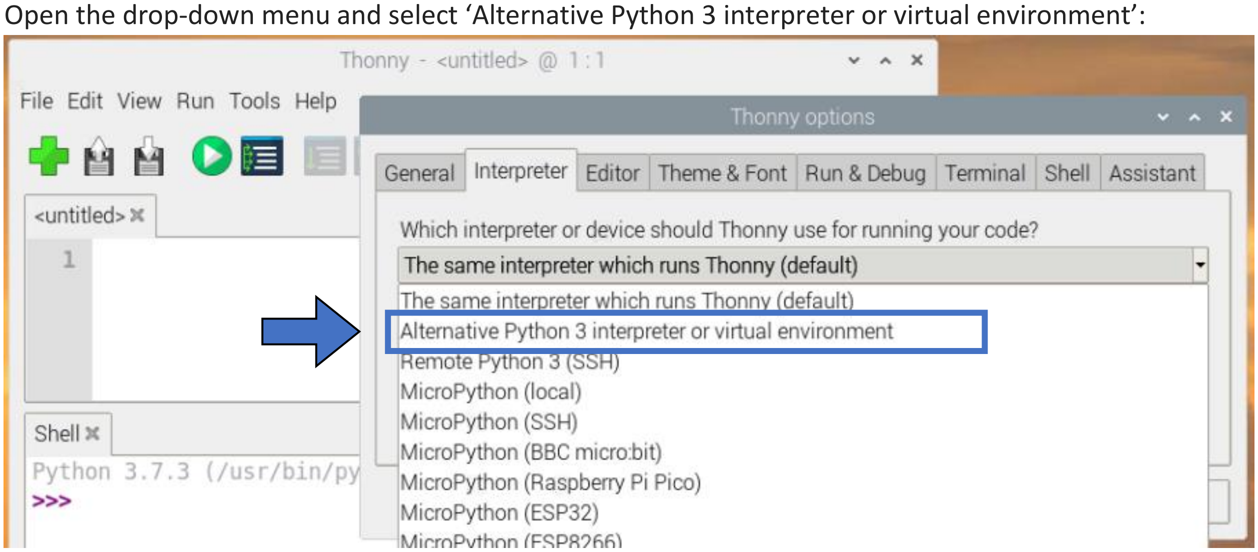
Restart Thonny, and select Tool, Option



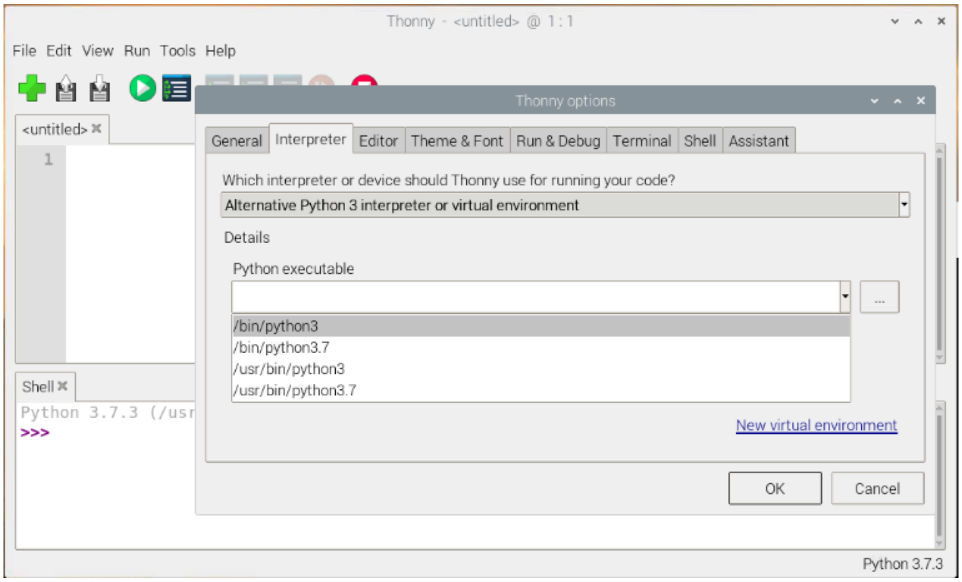
Select Interpreter where it is shown the default setting, pointing to /usr/bin/python3.



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

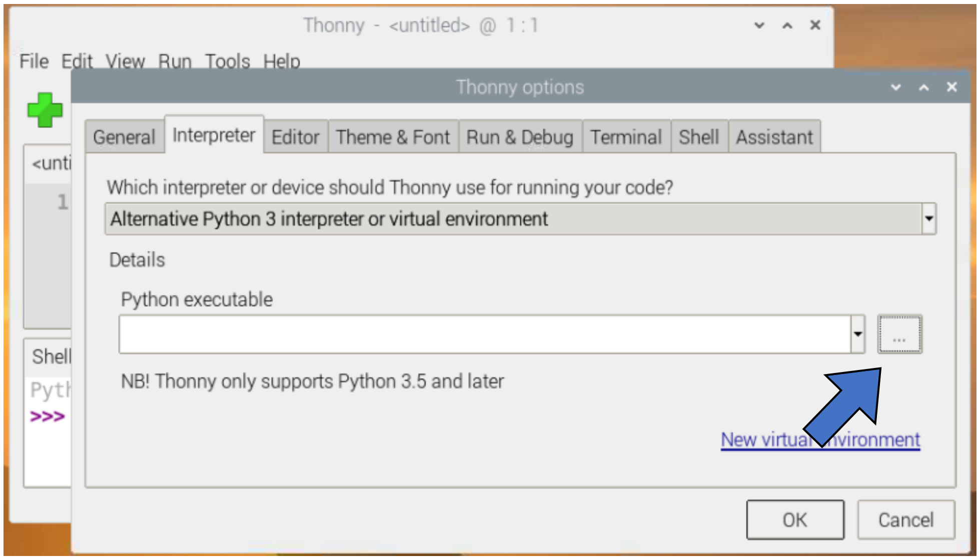


Open the drop-down menu and if ‘/home/pi/cubotino_micro/src/.virtualenvs/bin/python3’ is listed just select it.

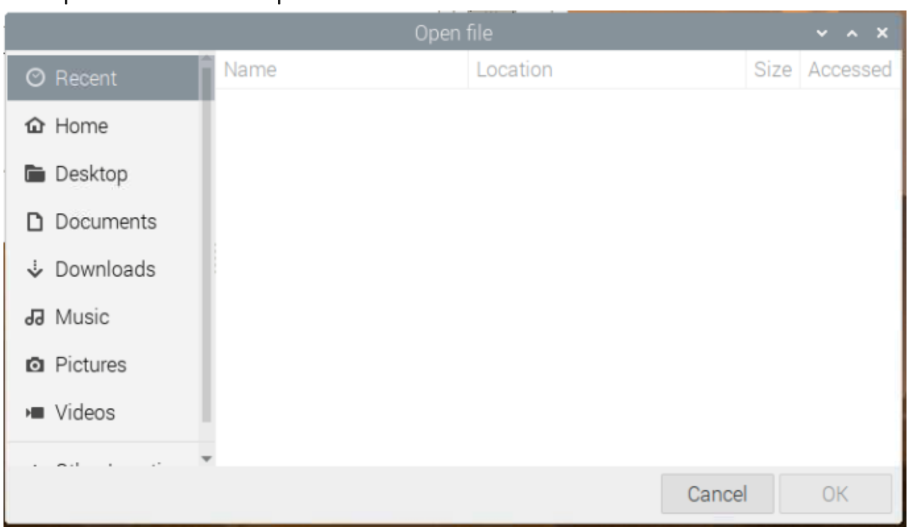


Thonny - <untitled> @ 1 : 1
File Edit View Run Tools Help
<untitled>
Shell
Python 3.7.3 (/usr
>>>
Thonny options
General
Interpreter
Editor
Theme & Font
Run & Debug
Terminal
Shell
Assistant
Which interpreter or device should Thonny use for running your code?
Alternative Python 3 interpreter or virtual environment
Details
Python executable
/bin/python3
/bin/python3.7
/usr/bin/python3
/usr/bin/python3.7
New virtual environment
OK
Cancel
Python 3.7.3

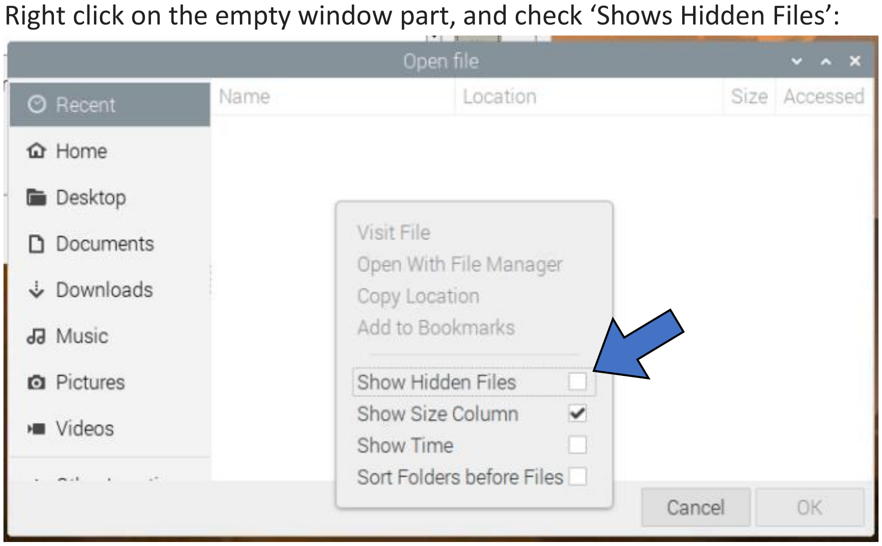
If ‘‘/home/pi/cubotino_micro/src/.virtualenvs/bin/python3’ is not listed, select the browse button:



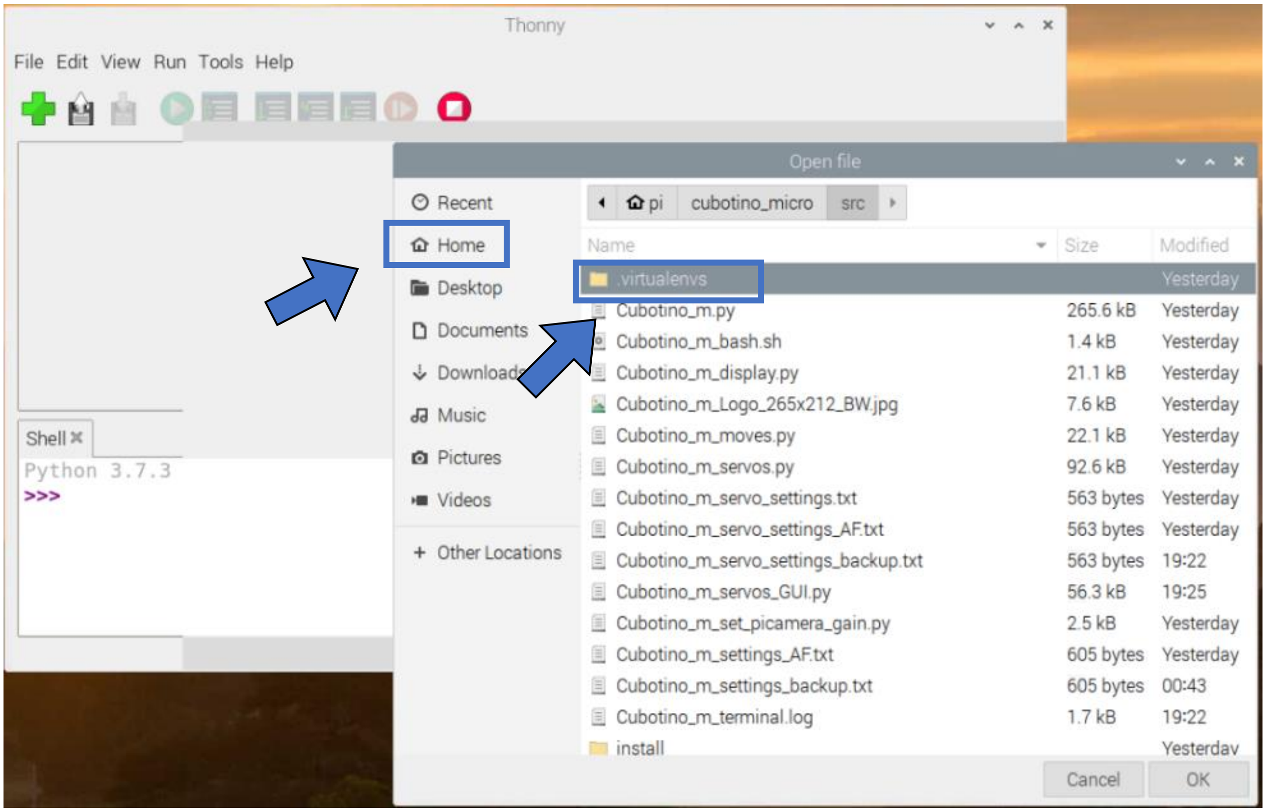
An Open file window opens:



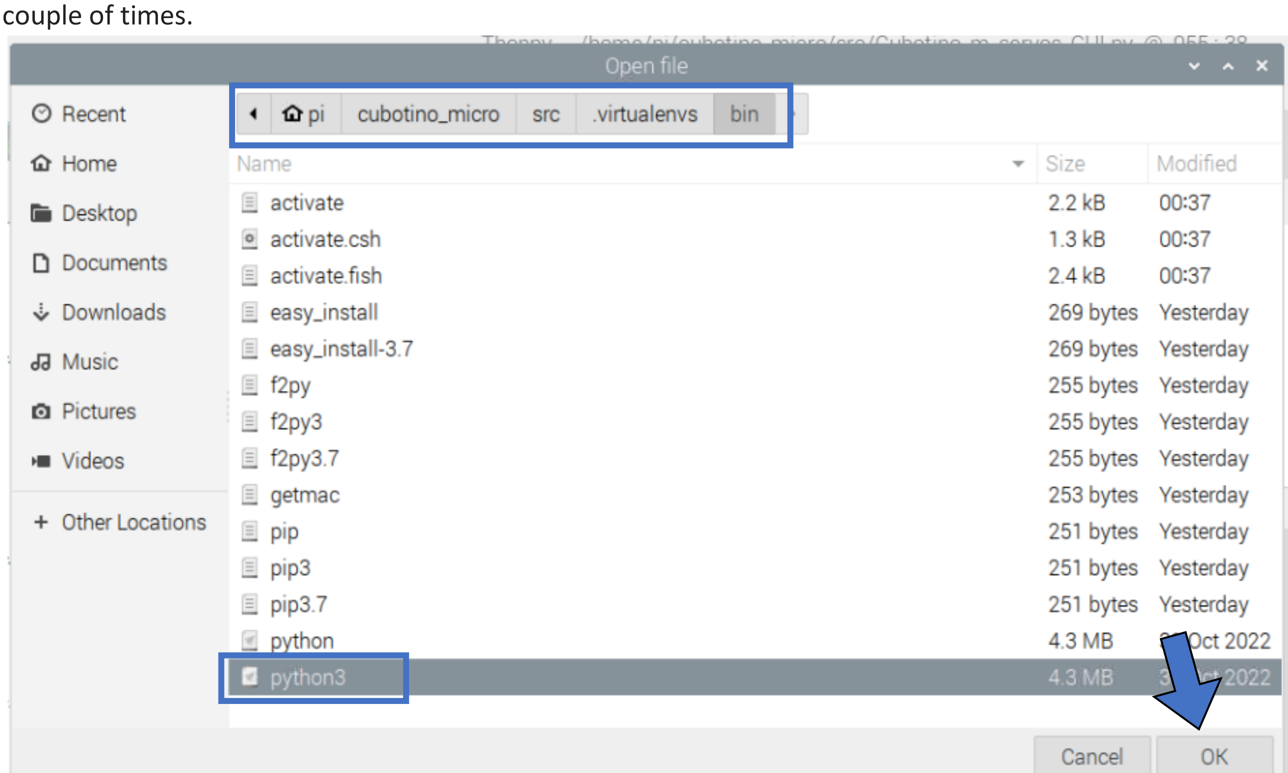
Right click on the empty window part, and check ‘Shows Hidden Files’:



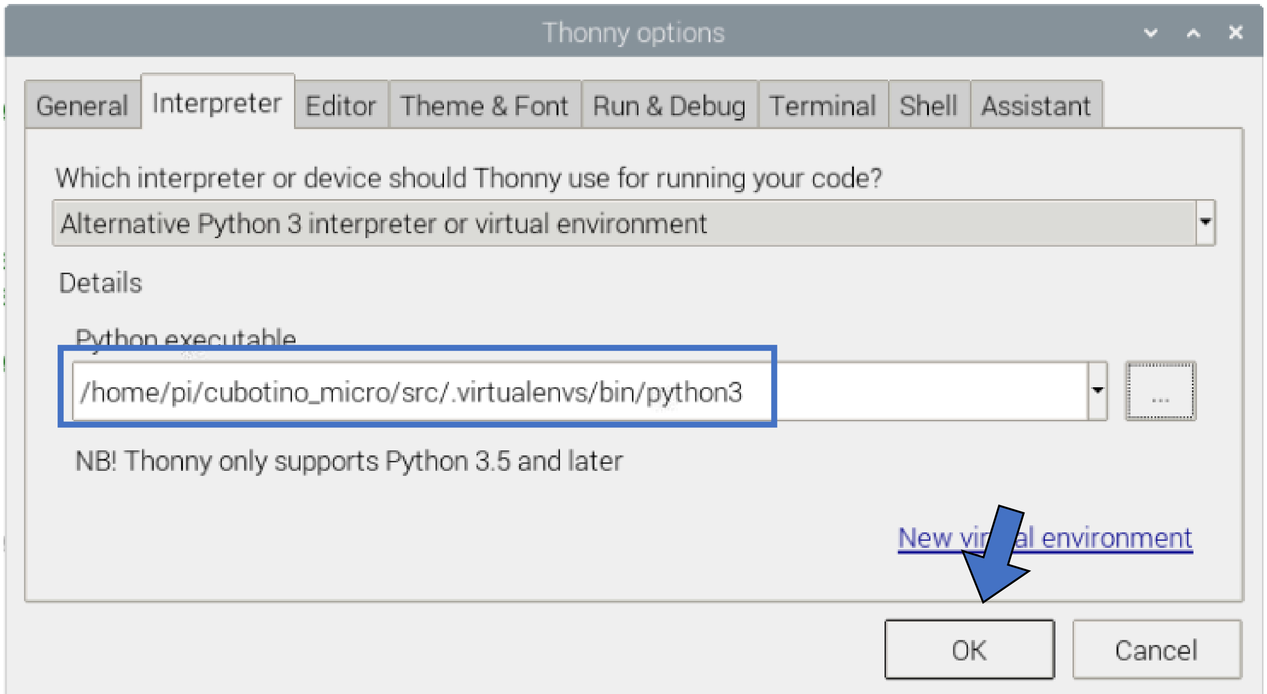
Select Home, .virtualenvs should appears (Note: all folders and files starting with a dot are hidden type)
The path should be ''/home/pi/cubotino_micro/src/.virtualenvs"



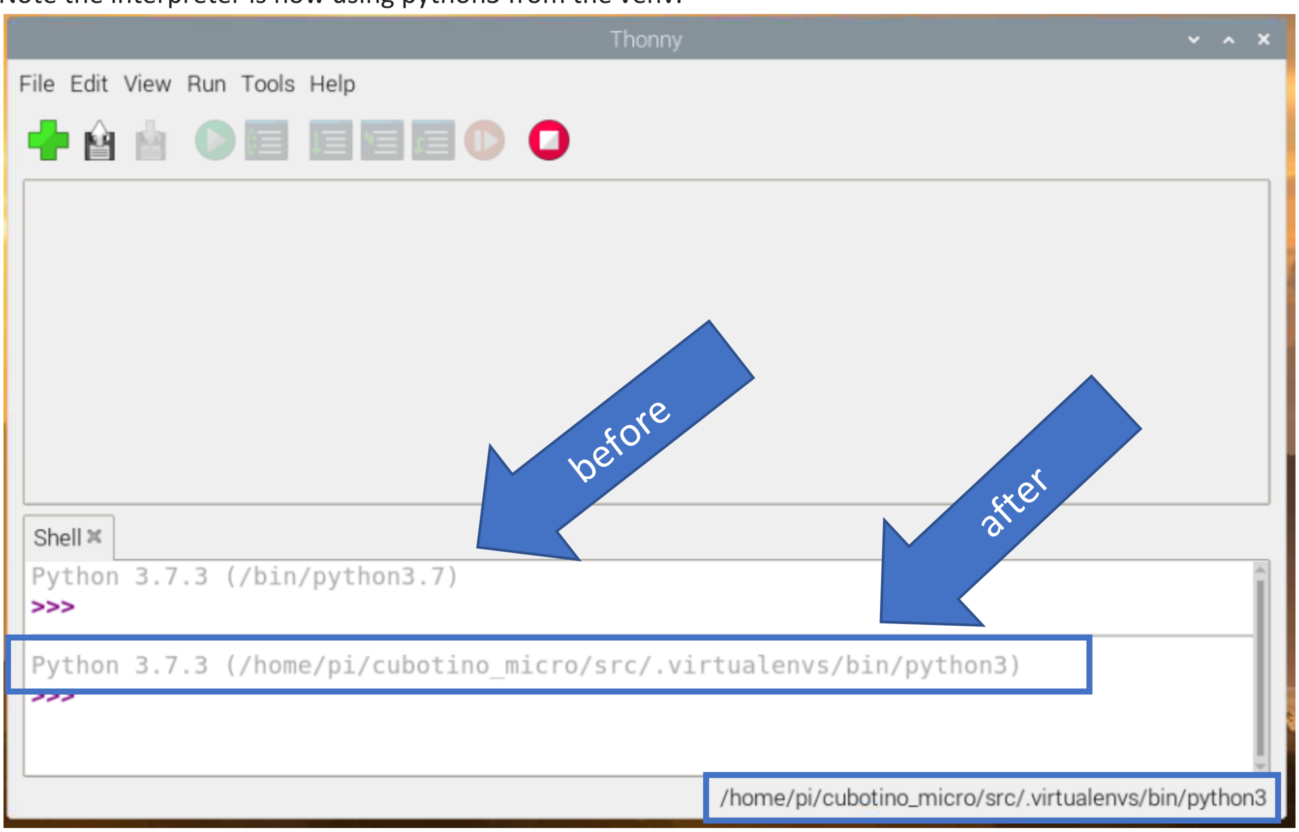
Select python3 from the path: '/home/pi/cubotino_micro/src/.virtualenvs/bin/python3' and confirm a couple of times.



Confirm one more time.



Note the interpreter is now using python3 from the venv.



Notes, to get this change proposed as default:

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

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_micro/src*) and activate the venv (*source .virtualenvs/bin/activate*); Attention to the dot in front of virtualenvs.
4. run the script *python Cubotino_m_servos.py --tune*; Attention to the space in between '-'

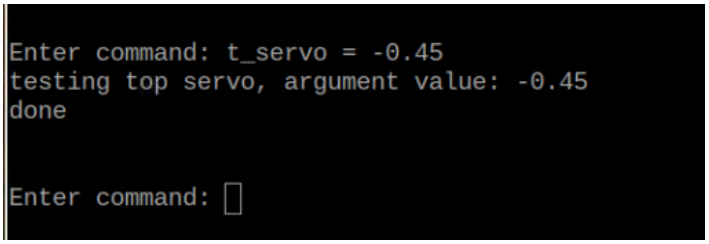
```
File  Edit  Tabs  Help
pi@cubotino:~ $ cd cubotino_micro/src
pi@cubotino:~/cubotino_micro/src $ source .virtualenvs/bin/activate
(.virtualenvs) pi@cubotino:~/cubotino_micro/src $ python Cubotino_m_servos.py --tune
```

5. 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: 
```

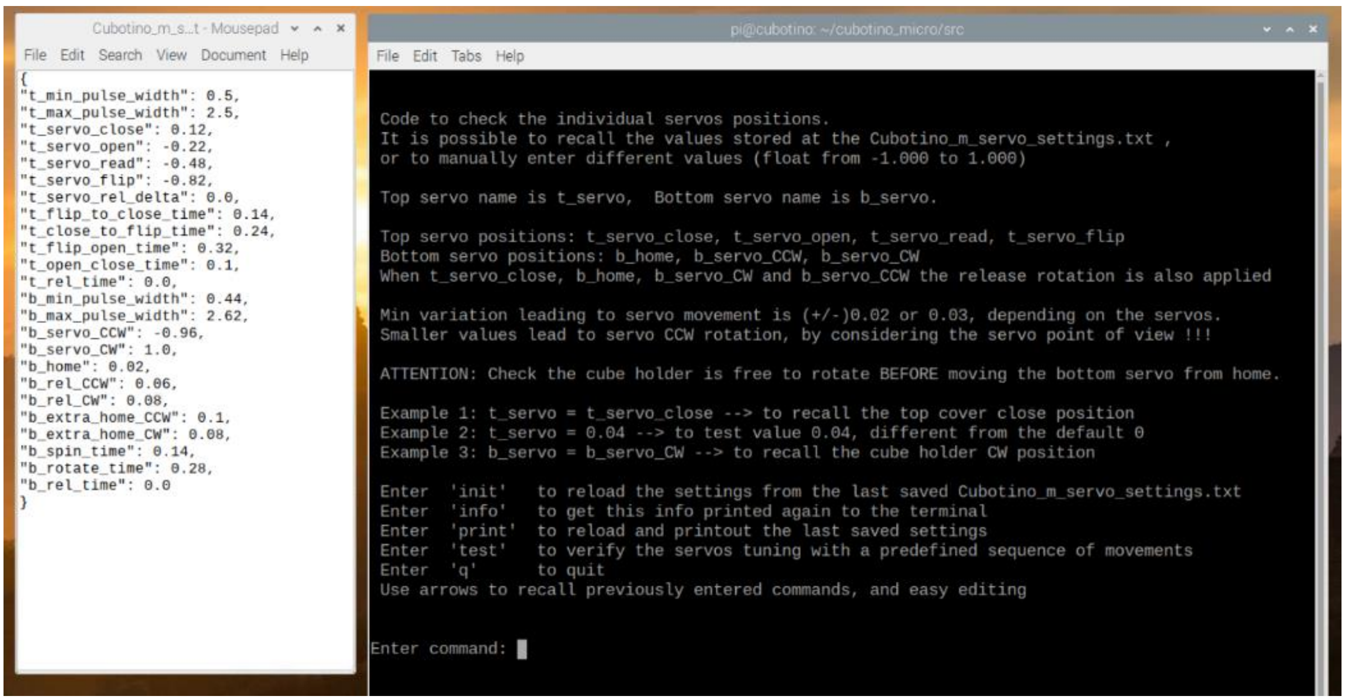
6. it is possible to recall the settings stored at Cubotino_m_servo_settings.txt as well as to enter different target values (value should be a float ranging from -1.000 to 1.000).

7. After the 'Enter command:' type the below commands to test the servos positions:
   - t_servo = t_servo_close (a backward rotation is applied if t_servo rel_delta is not zero)
   - t_servo = t_servo_open
   - t_servo = t_servo_read
   - t_servo = t_servo_flip
   - b_servo = b_home (a backward rotation is applied if b_extra_home is not zero)
   - b_servo = b_servo_CCW (a backward rotation is applied if b_extra_sides is not zero)
   - b_servo = b_servo_CW (a rotation backward is applied if b_extra_sides is not zero)
8. To adjust the Top_cover and/or the Cube_holder positions, you might enter the value instead of the saved parameters; Check the example below.



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

Enter command:
```

9. Once the position(s) are satisfactory, edit the *Cubotino_m_servo_settings.txt* and save the file to apply the new settings; It is suggested to keep open the Cubotino_m_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_m_s...t - Mousepad
File Edit Search View Document Help
{
"t_min_pulse_width": 0.5,
"t_max_pulse_width": 2.5,
"t_servo_close": 0.12,
"t_servo_open": -0.22,
"t_servo_read": -0.48,
"t_servo_flip": -0.82,
"t_servo_rel_delta": 0.0,
"t_flip_to_close_time": 0.14,
"t_close_to_flip_time": 0.24,
"t_flip_open_time": 0.32,
"t_open_close_time": 0.1,
"t_rel_time": 0.0,
"b_min_pulse_width": 0.44,
"b_max_pulse_width": 2.62,
"b_servo_CCW": -0.96,
"b_servo_CW": 1.0,
"b_home": 0.02,
"b_rel_CCW": 0.06,
"b_rel_CW": 0.08,
"b_extra_home_CCW": 0.1,
"b_extra_home_CW": 0.08,
"b_spin_time": 0.14,
"b_rotate_time": 0.28,
"b_rel_time": 0.0
}
```

```
pi@cubotino: ~/cubotino_micro/src
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:
```

To verify if everything goes well:

10. If you type *info* , instead of a command after the 'Enter command:', guidance is printed again on the screen.
11. If you type *init* , instead of a command after the 'Enter command:', the new settings from Cubotino_m_servo_settings.txt are re-applied to the robot.
12. If you type *print* , instead of a command after the 'Enter command:' the latest values saved at Cubotino_m_servo_settings.txt are printed to the screen.
13. 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.
14. run the script *python Cubotino_m_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.
15. 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.
16. 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 = -04".*

On the Cubotino_m_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_m_settings_default.txt* and *Cubotino_m_servo_settings_default.txt*
Note: These two files will be automatically duplicated on your local repository, and renamed as: *Cubotino_m_settings.txt* and *Cubotino_m_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_m_settings_default.txt (becoming Cubotino_m_settings.txt in your local repository) and Cubotino_m_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	240	240	Int	Display width (in pixels)
disp_heigh	135	135	Int	Display height (in pixels). The variable name has a typo, yet this has become its name….
disp_offsetL	40	40	Int	Display offset on width **Left** (in pixels)
disp_offsetT	53	53	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	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 abs(1-kl) from the average of the previous readings. 0.95 stops the AUTO mode when all the parameters have a max deviation of 5% from the average
x_l	0	70	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.

Cubotino_m_settings_default.txt (becoming Cubotino_m_settings.txt in your local repository) and Cubotino_m_settings_AF.txt as reference)

Parameter (dict key)	**Default value**	**AF value**	**Data type**	**Info**
x_r	0	60	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	30	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	80	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	7.8	7.8	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	3	3	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).
sv_max_moves	20	20	int	Max number of moves requested to the Kociemba solver. When the solver finds a solution matching this movement quantity, that solution is returned even before the timeout expiration (sv_max_time).
sv_max_time	2	2	float	Timeout, in seconds, for the Kociemba solver. The best solution found within the timeout is returned at timeout end, even if it doesn't match the desired max quantity of moves.
collage_w	1024	1024	int	Image width for the unfolded cube file. This parameter determines the image collage realization, and it makes possible to save all images with the same size.
marg_coef	0.1	0.07	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).

Cubotino_m_settings_default.txt (becoming Cubotino_m_settings.txt in your local repository) and Cubotino_m_settings_AF.txt as reference)

Parameter (dict key)	Default value	AF value	Data type	Info
cam_led_bright	0.5	0.5	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_m_settings.txt belongs to an older version (absence of this parameter), 0.5secs will be used as default.
built_by	(empty)	(empty)	string	From rev 0.4: String with the maker's name added to the Cubotino logo. (only when the string is not empty)
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	From rev. 1.0: Timer in seconds, to switch to Fix Coordinates System (FCS). This value is increased when ZeroW and/or screen connection. Before this timer elapse, the cube facelets are searched via edge detection, afterward the facelets position are based on fix coordinates. The fix coordinates are automatically generated, on saucerful cycles (facelets detected by edges). The coordinates are saved on a text file.

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_m_servo_settings_default.txt (becoming Cubotino_m_servo_settings.txt in your local repository) and Cubotino_m_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	0.50	0.14	float	Time for t_servo to move from Flip to Close position
t_close_to_flip_time	0.50	0.22	float	Time for t_servo to move from Flip to Close position
t_flip_open_time	0.50	0.30	float	Time for t_servo to move from Flip to Open position and vice versa
t_open_close_time	0.50	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_m_servo_settings_default.txt (becoming Cubotino_m_servo_settings.txt in your local repository) and Cubotino_m_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_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_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_spin_time	0.50	0.14	float	Time for the Cube_holder to spin ~90° (cube not constrained)
b_rotate_time	0.50	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

Note: On Cubotino_m.py and Cubotino_m_servos.py, the string “#(AF ” is placed as comment start, where the above listed parameters are used.

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 idea behind this model has been MINIMIZING the overall side.
The trigger has been the idea to make evident how relatively easy is to make a compact robot, by using the CUBOTino mechanical principle.

CUBOTino micro is a derived model from the CUBOTino series.
It reuses the CUBOTino suffix name as it shares the mechanical concept and most of the code.
Differently from the Top and Base versions, it deviates from the platform concept (reusing parts).

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.

Project targets for CUBOTino robot series, in red those that still apply to the micro version:

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

3. Robot name

I've started 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, ro**BOT** 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	• 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	• Cube status detection at the robot. • Cube solution (Kociemba solver) generated at PC.	Stand-by, see notes below
Top	Autonomous	• Cube status detection at the robot, via a vision system. • Cube solution (Kociemba solver) generated at the robot.	Finalized (June 2022) Link
Micro	Autonomous	• This essentially is scaled down from the Top version. • Deviates from the platform concept	Finalized (March 2023) Link
Pocket	Autonomous	• This essentially is the Top version adapted for a 2x2x2 Rubik's cube (~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

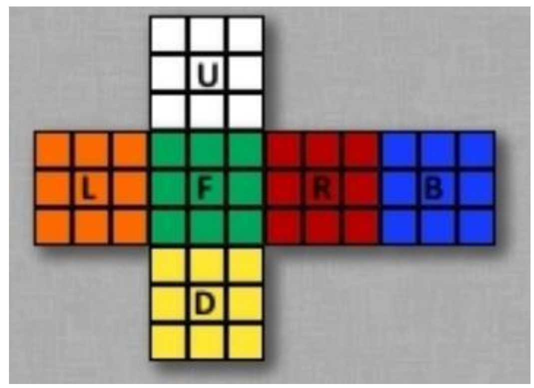
I spent some time on this method, but the quality of my welding turned out to be insufficient; I'm even doubting if it really fits the overall project scope, as this method involves too many skills.

Anyhow I've some other ideas for the Medium version….

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

5. High level info

1. The robot is based on a Raspberry Pi Zero (2WH or WH) 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 size of 30mm (GAN 330 keychain); Quite remarkably the vision part is rather forgiven of the two holes made on the cube facelets to connect the cube to the keychain.
8. Cube notations are from David Singmaster, limited to the uppercase (one "external layer rotation" at the time): https://en.wikipedia.org/wiki/Rubik%27s_Cube#Move_notation
9. 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



10. Cube solver uses the Hegbert Kociemba, two-phase algorithm in its fully developed form with symmetry reduction and parallel search for different cube orientations", with an almost optimal target:
    - intro: https://www.speedsolving.com/threads/3x3x3-solver-in-python.64887/
    - Python script: https://github.com/hkociemba/RubiksCube-TwophaseSolver
11. 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.
12. 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.
13. The robot detects the cube status on cubes with and without the black frame around the facelets.
14. In case the facelets' edges are difficult to be detected, after a predefined time a Fix Coordinates System (FCS) is used. Facelets location based on fix coordinates to pick the facelets' color info. The FCS is automatically generated, and updated, each time a cube is successfully scanned via edges detection technic.

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

6. Construction

The micro version has been scaled down from the CUBOTino Autonomous; Targets considered:

1. solving a Rubik cube without changing it for special gripping.
2. low cost.
3. simplicity.
4. **as compact as possible 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** Zero 2 SBC (the computer part)
- **OpenCV** (an open-source library for computer vision; From https://en.wikipedia.org/wiki/OpenCV: OpenCV is a library of programming functions mainly aimed at real-time computer vision).
- **PiCamera** (a camera module, highly integrated with Raspberry Pi)

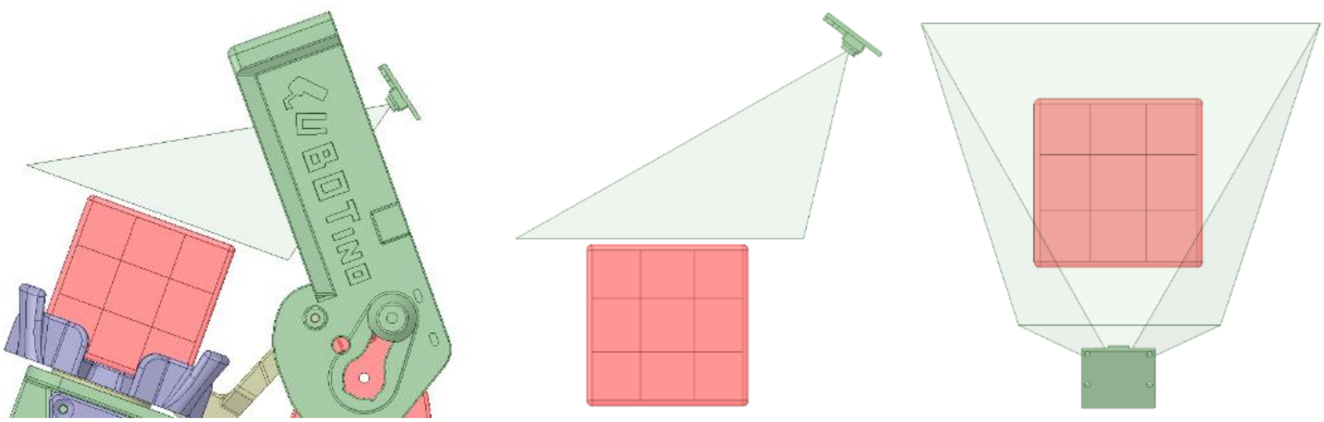
In which the python script '**Cubotino_m.py'** is responsible for the interaction with these elements.

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 facelet 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 facelet, by evaluating if fitting a square shape.

To solve the above problems:

1. A controllable LED has been placed close to the camera; This 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 facelet edge analysis, below an example:

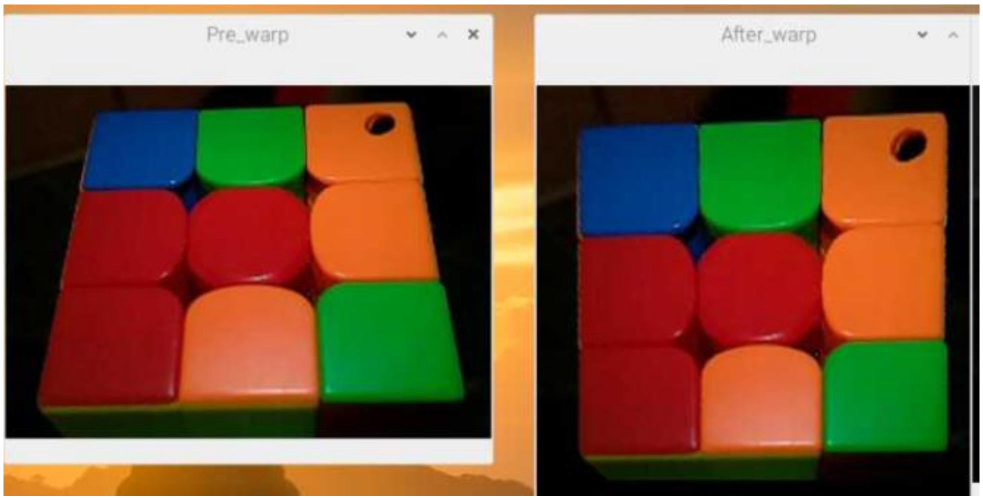
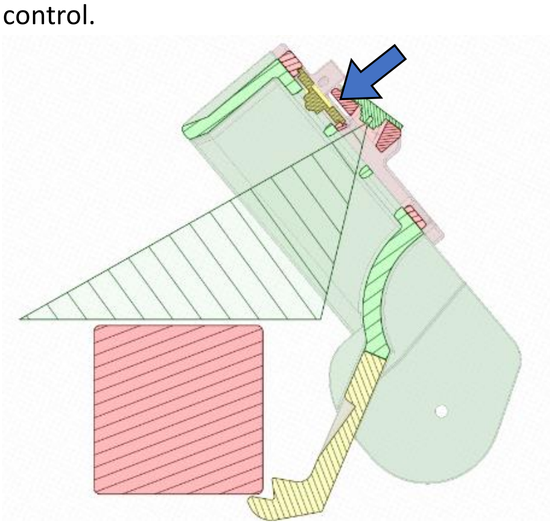


Image warping requires some computation power, but it does not affect the overall timing.

B. Taking consistent images

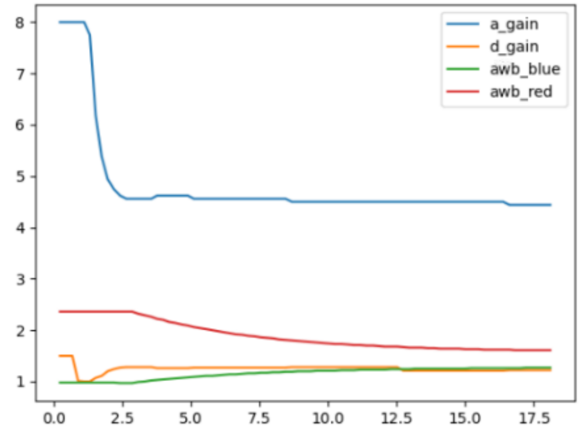
This is a crucial aspect for proper color analysis.
The light source addition is a good way to mitigate the environmental light conditions, typically out of our control.



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 already well set.

To cover these situations, a so called 'warm-up' function is implemented in Cubotino_m.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-551cceba80f0

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 a facelet.

1. To facilitate the contours selection, it is convenient to approximate them (those with more than 4 vertices).
2. From the approximated contours, those not having 4 vertices are discharged.
3. The remaining contours are ordered to have the first vertex on top left.
4. The approximated and ordered contours are then evaluated on:
    a) Area that should be within pre-defined thresholds.
    b) Max area deviation, from the median one
    c) Max sides length difference, from a pre-defined threshold
    d) Max diagonals length difference, from a pre-defined threshold
    e) Max distance from the central one; This step includes ordering the 9 contours, according to their center coordinates.
    f) Quantity of contours left, after discharging those not ok.
5. The first 9 contours, passing through this process, are then used as masks; These masks are applied on the colored warped image, as guidance for the facelets position.
   In case the frameless_cube is set ‘true’ or ‘auto’, as soon as 7 contours are detected the remaining two are estimated for their position.
6. The ‘accepted’ contours are plot over the colored warped image, as visual feedback.

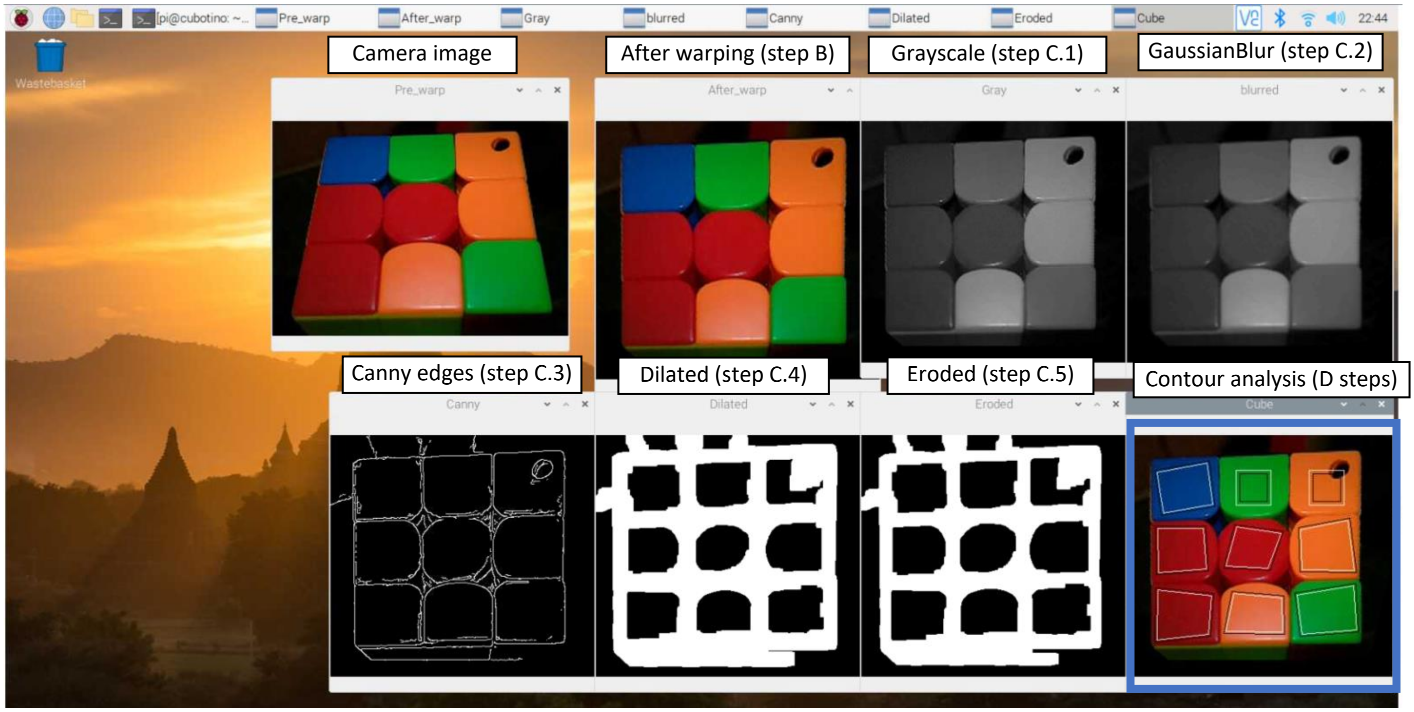
E. Colors retrieved:

On each facelet, are the retrieved 2 main info:

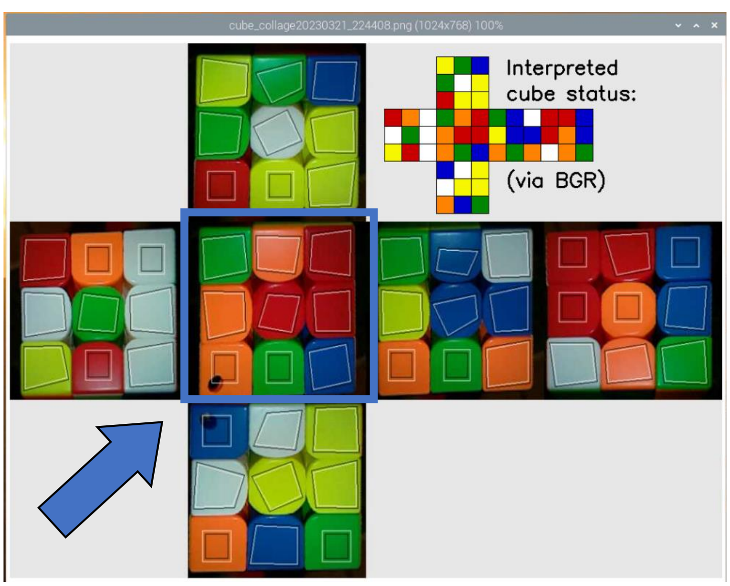
1. Average BGR, for a portion of the facelet around the detected contour center.
2. Average HSV, based on the average BGR.

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_m.py --cv_wow* , a FHD screen/setting needed)



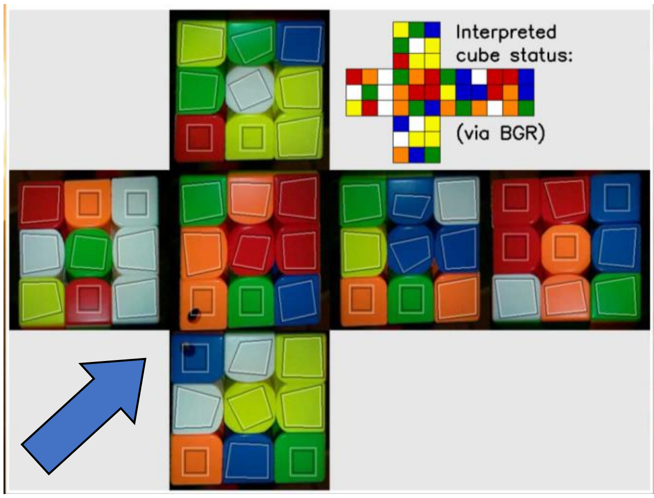
Not all images are oriented as per user point of view: Sides UBDF are 180deg rotated.
The described image analysis process is repeated for the 6 cube faces.
The last processed image of each side, the one with the 'accepted' contours, is stored in RAM.
Once the full cube status is detected, these images 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.



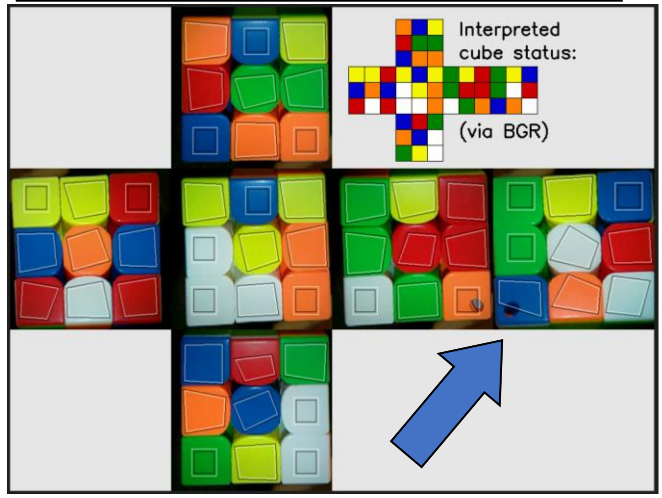
When individual cube faces are presented on screen, are always oriented as per camera point of view.
The F side highlighted on the cube sketch at side, is the cube face image that was under analysis on the above example.

F. The algorithm works rather well with this special cube, despite having two holes for the keychain connection.

Most of the times the holed facelet is one of the last two, therefore the position is estimated:



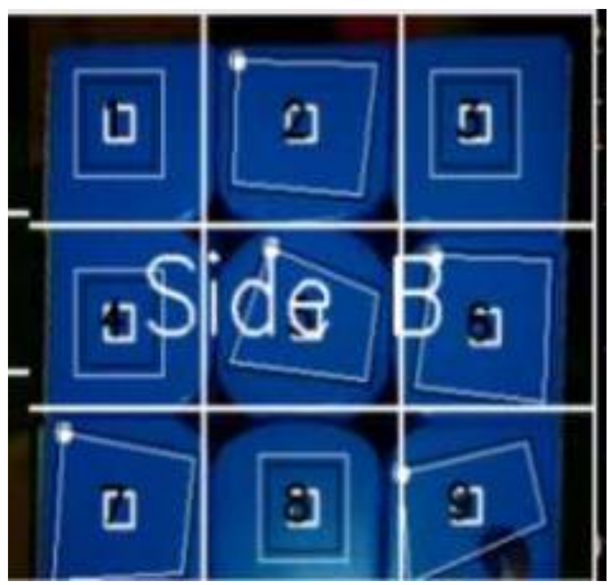
Fewer times the facelet edges are detected:



Yet, so far, I haven't seen one case with both the holed facelets detected on their contours.

Below images were obtained by adding the debug argument (- -debug) when launching the script.

The little white squares, positioned at contours centers, is the area used to retrieve the facelets colors: This little area typically remains ad the holes side:

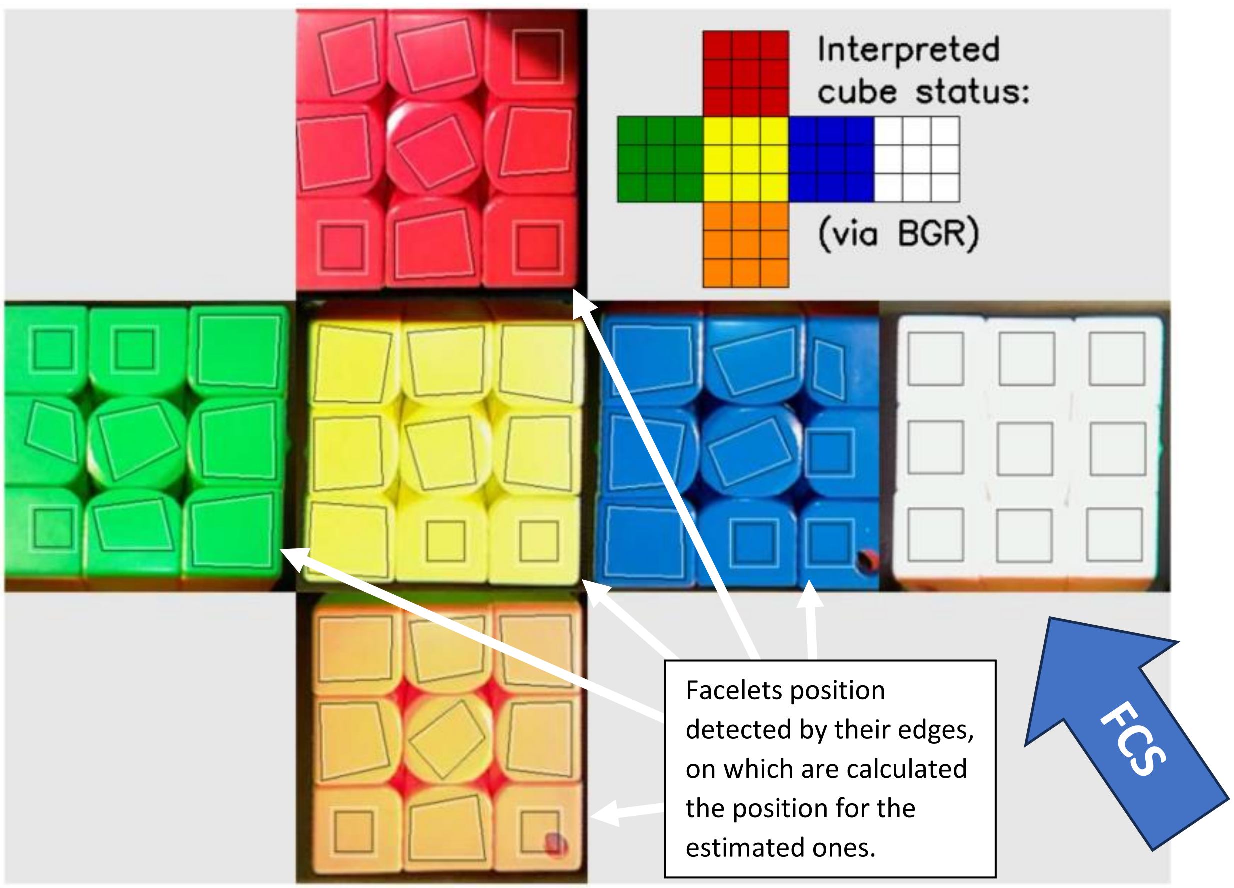


G. Fix Coordinates System (FCS):

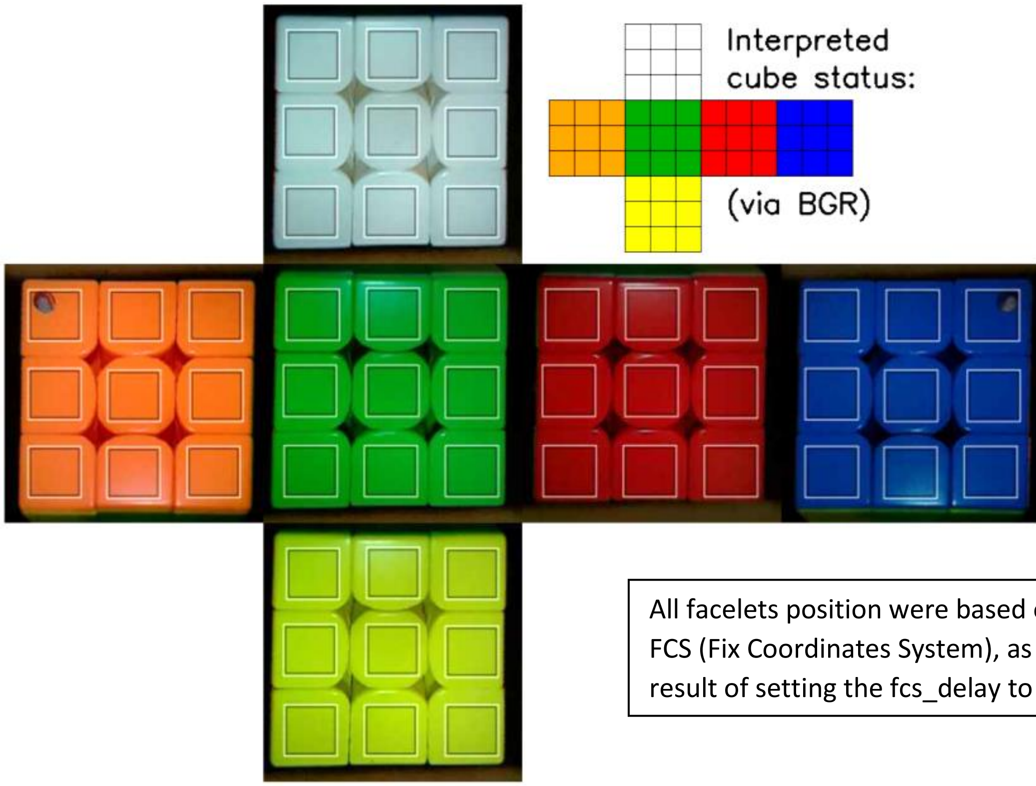
The facelets position are also estimated via an additional method (FCS), below the main aspects:

- 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_m_coordinates.txt file.
- The coordinates are loaded from the file once the Cubotino_m.py is launched.
- Coordinates are averaged from the last "n" values.
- When the Cubotino_m_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 B (back) face were done via the FCS (drastically increased the ambient light after the camera finished its setting in low light conditions).
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.

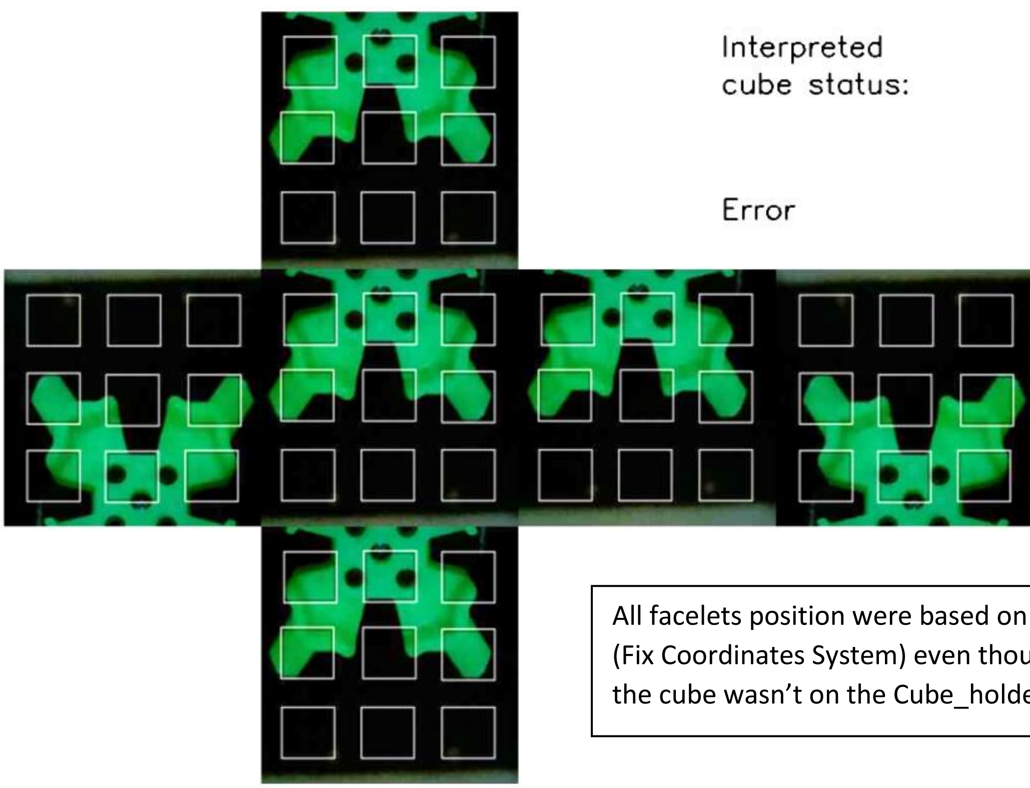


If the fcs_delay is set to 0 (zero), it gives priority to the Fix Coordinates System over the edge's detection one. If a cube is on the Cube_holder, then it works and it might gain some speed.



All facelets position were based on FCS (Fix Coordinates System), as result of setting the fcs_delay to 0

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



All facelets position were based on FCS (Fix Coordinates System) even though the cube wasn't on the Cube_holder.

H. Is all this really needed?

You might argue there is no need to search for the facelets contours, and hard coded coordinates to be sufficient.

I haven't tried the 'simpler' approach; Below the reasons I like to stick to the more complex approach:

1. The robot construction is rather basic, and the cube/camera positions cannot be expected to be very repetitive: Small angle variation of the Top_cover will result in large variation of the camera coordinates for the same facelets.
2. Below picture shows one extreme case, in which the edge detection excluded a large area affected by light reflection; This cube face was correctly interpreted!

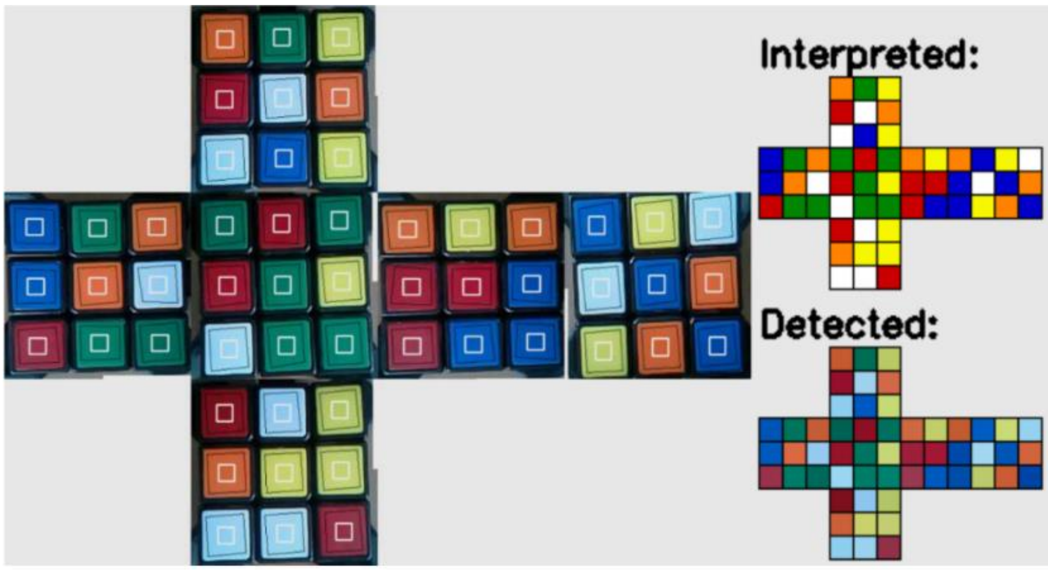
Notes:
Light reflection drastically affects the color interpretation.
The accepted contour is on the very low contour acceptable area, yet sufficiently large to don't be noise.

3. There is one more reason: Fun…. Having the facelets detected by a piece of AI is quite cool.

8. Color's detection strategy

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 9 facelets per each cube, and to know the contour center coordinates.
   b. The inner one, in white on below picture, depicts a smaller square area centered on the outer contour; This smaller area is used to:
      i. Measure the BGR average value, used for the color interpretation according to the 1st method (BGR color distance).
      ii. calculate the HSV average colors, used for color interpretation according to the 2nd method (Hue value).



2. Properties of the faces center facelet:
   On a 3x3x3 Rubik's cube, the 6 center's facelets have useful properties:
   a. These facelets don't move (fix facelets number).
   b. These facelets have (obviously) 6 different colors.
   c. Opposite faces have known colors couples, white-yellow, red-orange, green-blue (Western color code).
   This means we can make use of these 6 facelets as color reference.

3. The average HSV, detected on the 6 centers, is used to determine which color is located on the 6 centers:
   a. White facelet is the one having the largest V-S delta (difference between Value, or Brightness, and Saturation), while the yellow one is located at opposite face.
   b. The remaining 4 centers are evaluated according to their Hue, and the Hue at opposite face.
   c. Orange has very low Hue, and red should be very high (almost 180), depending on light condition, the red's Hue could "overflow" and resulting very low (few units). The red is expected to be much higher than Orange, unless it overflows … in this case both red and orange are rather small with red smaller than orange.
   d. Out of the two remaining centers, blue is the one with highest Hue, and consequently the green is also known.
4. Based on previous step, the 6 cube colors (at least their centers) have a known average HVS and therefore an average BGR color; This also informs on the cube orientation (colors) as placed on the cube-holder.

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

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

9. Robot solver algorithm

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_m_moves.py file.

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

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

The robot solver follows instead the below approach:

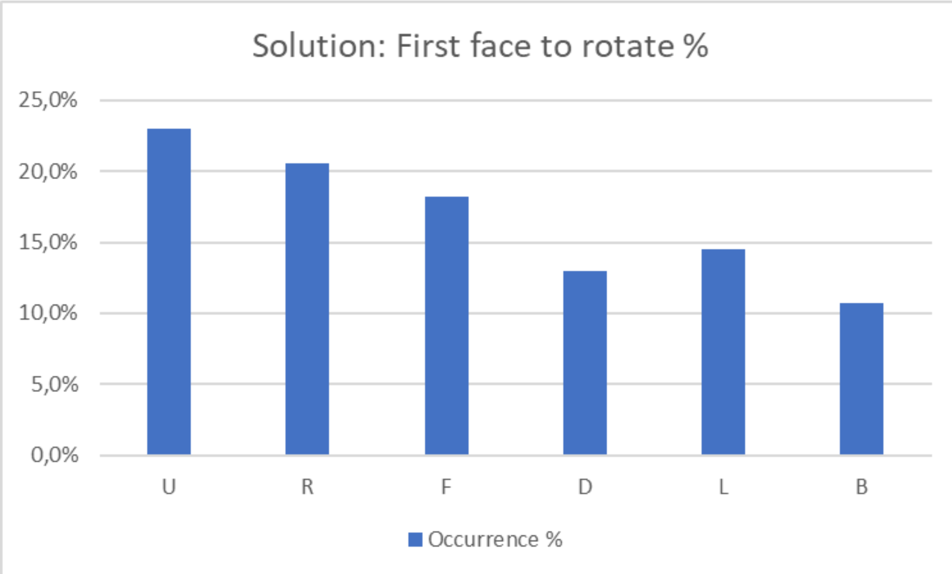
1. All the 18 possible cube moves (U1, U2, U3, ….., B1, B2, B3) the solver can return, are used as keys in a dictionary; There are 18 sets of robot movements (hard coded) associated to these keys. These robot movements consider the cube as ideally positioned: U side facing up and F side facing front.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

On November 2022 I took some time to look back at the information saved in the log files of my 3 robots.
I had a total of about 1300 manually scrambled cubes, after removing those with less than 10 rotations.
On below table the occurrences of the first rotation face, returned by the Kociemba solver:

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

~ 62%

~ 38%



Based on these cases, the Kociemba solver proposed a solution having higher chances URF to be the first move; Furthermore, the chances the first move will be on U face is higher than all the other faces.
To be noted the time given to the solver is very limited, therefore rather expected the URF faces having the large percentage.

After scanning the cube, my original and simplistic approach has been to move the cube to the "Initial position": This means the U face is facing upward, and D face is laying on the Cube_holder.
Based on the above analysis, there were only 13% chances to apply the first move (face rotation) on D face, therefore, to apply a face rotation without flipping the cube first.

On CUBOTino micro, after scanning the cube, the cube is not anymore moved to the "Initial position".
This means the cube is laying on its R face (20% chances to be the first face to rotate) and with a single flip the U face will be on the Cube_holder.
With this new approach it should be possible to save some robot movements ….

10. Python main scripts, high level info

1. *Cubotino_m.py* is the main python script on the robot, this script imports other custom files.
2. *Cubotino_m_settings_manager.py* (from 15th March 2024, rev. 1.0) 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_m.py* is started (eventually automatically at the Raspberry pi boots), the script checks if there are monitors connected. The monitor can also be via VNC, i.e., with VNC Viewer. The presence/absence of a monitor is needed to use/skip commands requiring graphical screen communication. This prevents errors, further than having a better experience.
4. Kociemba solver is tentatively imported from different locations; venv, active folder and 'twophase' sub-folder under active folder.
5. The script uses "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 9 facelets, all matching the filters criteria, are retrieved from a single image.
   b. (See Computer Vision chapter for more info) 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. (See Computer Vision chapter for more info) When determining the cube status, according to the facelets color; The analysis starts with a first method determining each (side and corner) facelet color, based on the color distance from the colors of the 6 centers. In case the cube status obtained with this first method is not coherent, then a second method is called. The second method uses the Hue value of each (non-white) facelet, by comparing it to expected (predefined) Hue ranges, adapted upon the Hue measured on the 6 centers. In case the second method doesn't provide a coherent cube status, then an error message is returned, and relevant info logged in a text file.
6. Kociemba solver:
   Kociemba solver is uploaded at the start; In case of multiple cubes solving, no need to reload it.
   The detected cube status, with URF notations, is sent to the Kociemba solver.
   The solver, with the chosen parameters, returns the best-found solution within the time-out; The solver doesn't provide the absolute best solution, as it is too computational (and time) expensive, yet it typically returns a solution with 20 movements or less. Very rarely, the solution has 21 movements, mostly because of the chosen time-out of 'only' two seconds.
   The solver returns an error if the cube status is not coherent; This info is then used to attempt the second color assignment method, or to stop by providing error feedback to the display.
7. From cube solution to robot movements:
   (see Robot solver algorithm chapter for more info) Cube solutions, in Singmaster notation, sent to Cubotino_m_moves.py that returns a (long) string with the sequence robot movements. Movements are Spin, Rotate, Flip.
8. From cube robot solution to robot movements:
   Robot solution string, in Cubotino notation, is sent to Cubotino_m_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
CubeStatus(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
CubeStatus	Cube status, in URF notation: i.e., RLFDUUDBBLFRURRBDDRDDFFLURULDRFDFLUFDLBRLRFLBUBFUBBLBU
CubeSolution	Cube solution string, in Singmaster notations: i.e., D2 L2 F2 R3 F2 L1 D2 F2 R3 U2 F1 L1 F3 R1 B2 F3 D3 L2 F2 (19f)
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.

Notes:

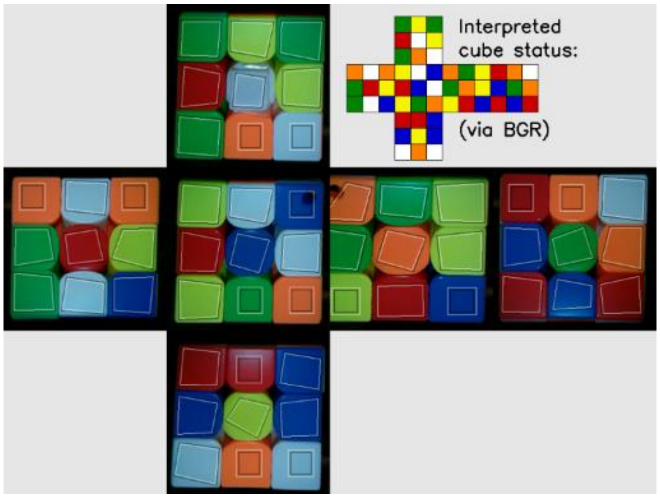
1. The folder Cube_*data_log* is made from the folder where *Cubotino_m.py* is running.
2. The logged data is saved in the *Cubotino_solver_log.txt* file.
3. Text file uses tab as separator.

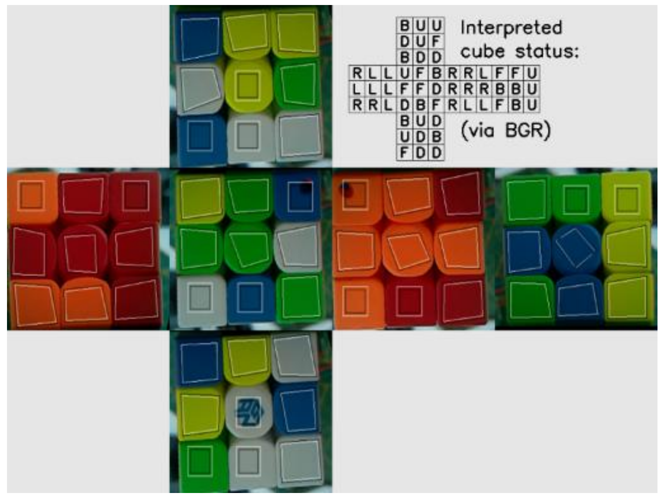
Further than saving data in the text file, a picture of the unfolded cube status is also saved.

- Folder: CubesStatusPictures
- Images: cube_collage_date_time.png

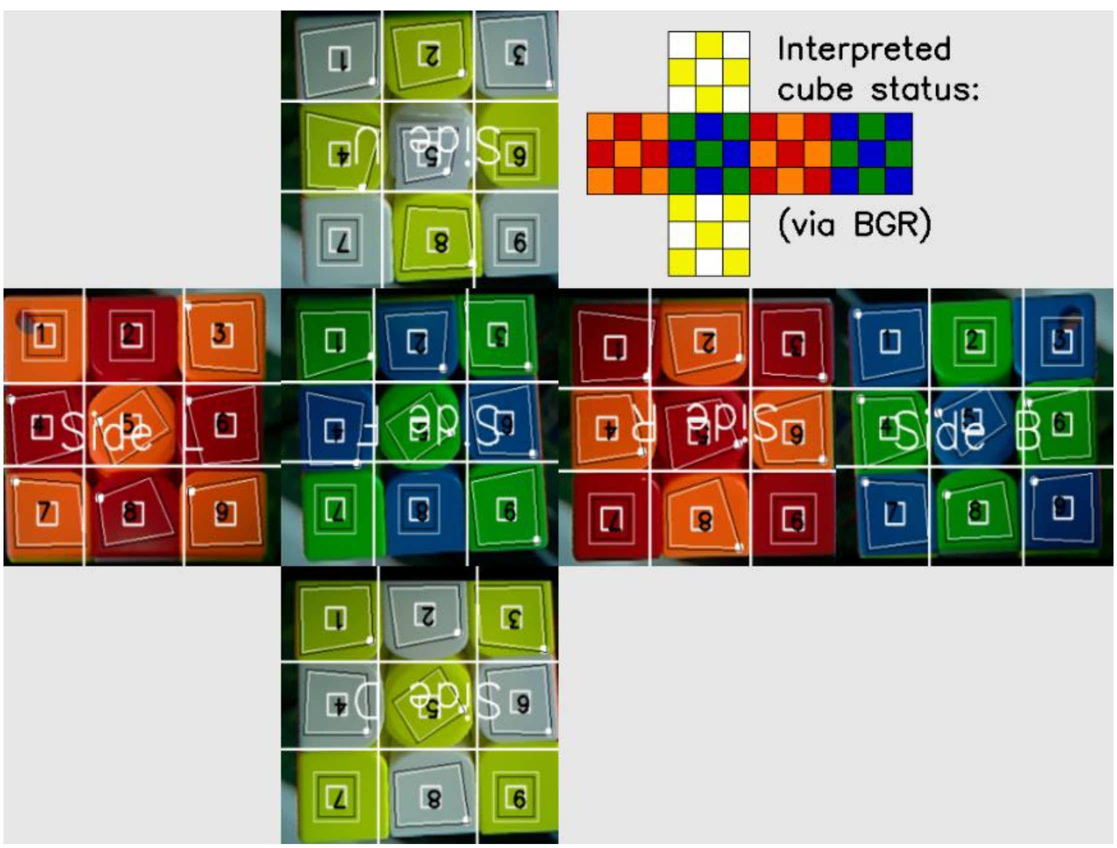
The image is a collage of the cube faces during the cube status detection.
On below two examples, the Interpreted cube status sketch shows two different representations:

- On the left when the algorithm correctly detects the 6 centers colors
- On the right, the algorithm uses the faces letters instead of color, because the logo on the white facelets prevent the algorithm from converging to a proper color association.





In below example the "- - debug" argument was used, that adds some graphical elements:



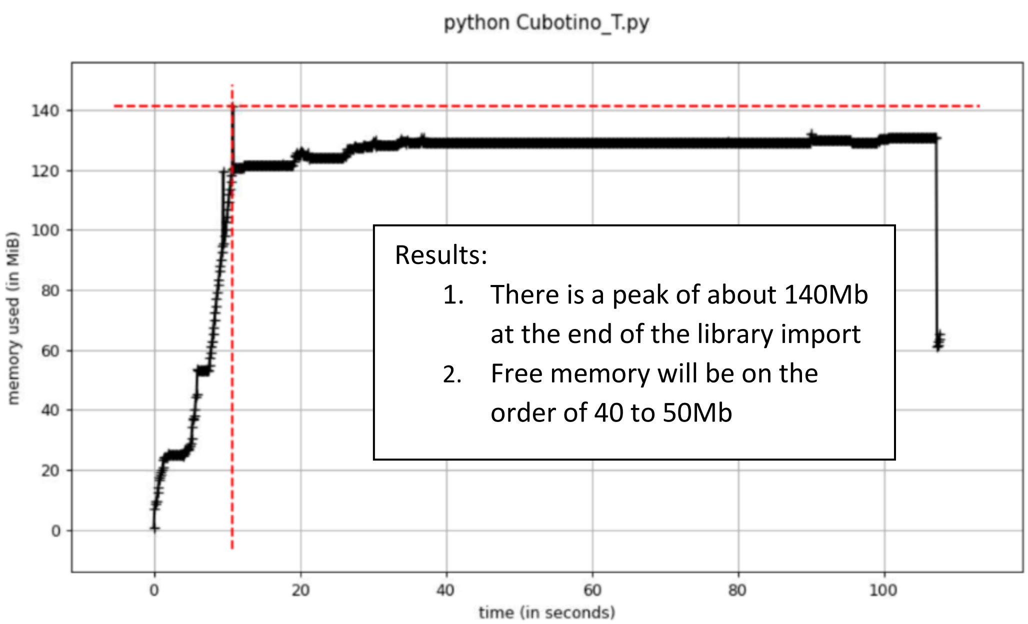
10. Animation:
After solving a cube, a simple animation is plot on the display.
The animation reproduces all the cube facelets movements that the robot has applied.
At the animation end, a solved cube appears, with the same orientation the real cube has on the Cube_holder.
The animation:
    - can be 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_m.py, the available memory is ~ 180Mb (with some little swapping):

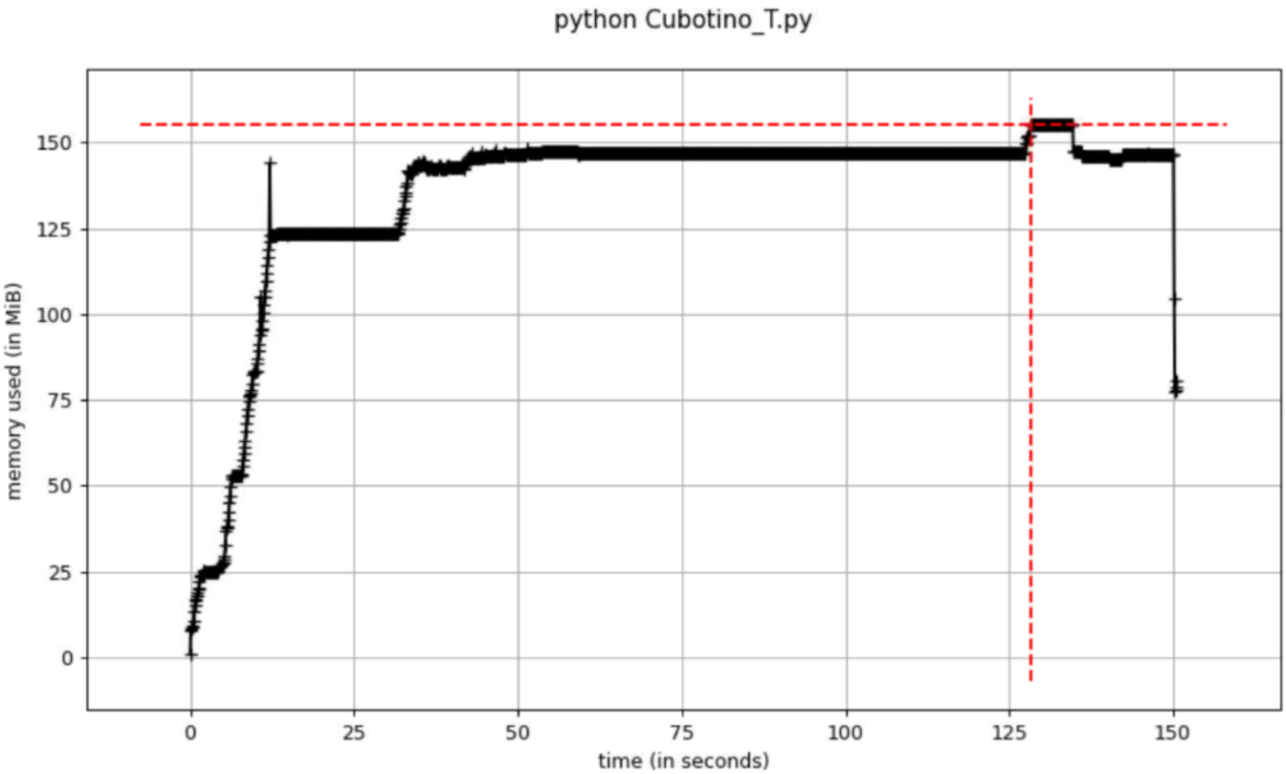
```
(cv) pi@raspberry:~/cube $ free -h
              total        used        free      shared  buff/cache   available
Mem:          364Mi       114Mi       133Mi        21Mi       116Mi       179Mi
Swap:          99Mi        86Mi        13Mi
(cv) pi@raspberry:~/cube $
```

Without VNC Viewer there is somehow lower memory usage; Below plot includes a full cycle:
    - import libraries.
    - read and solve a scrambled cube.
    - quit the script.



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

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

The project has been developed / tested with:

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

11. ChatGPT

January 2023

About 10 minutes ago I have 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 😊

12. 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/EbOHhvg2tJE), 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/CUBOTtino-Micro-the-Worlds-Smallest-Rubiks-Cube-So/).

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!

13. Credits

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

Credits 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.

14. 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.

Contacts:

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

15. Revisions

Rev	Date	Notes
0	11/03/2023	First release
0.1	19/03/2023	Modified Cubotino_m_servos.py: • to release the PWM to the servos’ pins, when quitting the script and after the “Long test” at Cubotino_m_servos_GUI. Modified Cubotino_m.py: • to be compatible with the updated GUI Modified Cubotino_m_servos_GUI.py: • added a window to adjust cropping and warping parameters. Instructions: • additional info on image cropping and warping adjustment via the GUI. • info on servos fine tuning via CLI. • Servos’ reliability (see troubleshooting).
0.2	21/03/2023	Changed method to match settings specific files to my (AF) robots; Modified: • Cubotino_m.py • Cubotino_m_display.py • Cubotino_m_servos.py • Cubotino_m_servos_GUI.py
0.3	25/03/2023	Connection board: • Uploaded gerber files to GitHub. • Modified the instal.sh file; Added command to swap the ACT LED from Raspberry Pi to the Connections_board. Updated instructions: • How to order the Connections_board via PCBWay • How to activate the ACT-LED (see Settings chapter)
0.4	26/03/2023	Added the possibility to add the Maker name on the Cubotino’s logo; Modified: • Cubotino_m.py • Cubotino_m_display.py Updated instructions: • Parameters and settings

FOLLOW AT NEXT PAGE

Rev	Date	Notes
0.5	06/05/2023	**3D parts** Added an extensible base that increases stability to the robot; New STL and STEP files added to the GitHub repository: • Baseplate_addition • Baseplate_L_leg • Baseplate_R_leg **Scripts** 1. After the cube status detection, Top_cover is positioned to “open” instead of “read” pos. 2. When the robot is operated without “--fast” argument, it now makes 2 steps to move the Top_cover from the Open position to the Flip position. 3. Solved a bug on the mac address system, used to match different settings on different robots of mine. 4. Modified files: • Cubotino_m.py • Cubotino_m_display.py • Cubotino_m_servos.py • Cubotino_m_servos_GUI.py • get_macs_AF.py **Instructions** 1. Corrected some errors in the instructions, i.e., Cubotino_m_servos.py has no arguments. 2. Added info for the additional (and extensible) baseplate. 3. Added some more info on the Connection_board.
0.6	28/11/2023	**Instructions** How to flash the right OS (Buster) as it is not anymore listed as Legacy at Raspberry Pi Imager.
0.7	12/01/2024	**Instructions** 1. Fully revisited to improve content and readability, thanks to Denis! 2. Additional info / options for the ACT-LED (on perfboard)

FOLLOW AT NEXT PAGE

Rev	Date	Notes
1.0	15/03/2024	A rather large revision: 1. Added the Fix Coordinate System, when edges detection takes too long. 2. Rationalize settings management preventing overwritten by ‘git pull’. 3. Added a facelets color animation on the display, after a successful run. **New file:** *Cubotino_m_settings_manager.py* **Modified files**: *Cubotino_m.py* • Added a facelets color animation on the display (after a successful runs). • Added the FCS (Fix Coordinates System) algorithm, to retrieve facelets colour based on fix coordinates, when the facelets are not detected within fcs_delay time. • Added FCS (Fix Coordinates System) to the info to log at Cubotino_solver_log.txt. • Adapted to the new settings management system. *Cubotino_m_display.py* • Added the animation plotting function. • Improved efficiency on Cubotino logo management. • Adapted to the new settings management system. • Solved bug on progress bar (for compatibility with newer Pillow versions). **Other modified files** (scripts using settings from the *settings.txt files) *Cubotino_m_servos.py, Cubotino_m_servos_GUI.py* **Renamed files:** • From: Cubotino_m_settings.txt → to: Cubotino_m_settings_default.txt • From: Cubotino_m_servo_settings.txt → to: Cubotino_m_servo_settings_default.txt **Instructions:** • Explained how the Fix Coordinate System (FCS) works. • Explained how the settings files are managed and corrected their name at the Parameters and settings. • Added the argument ‘- - no_animation’ at the “How to operate the robot” chapter.