How to make

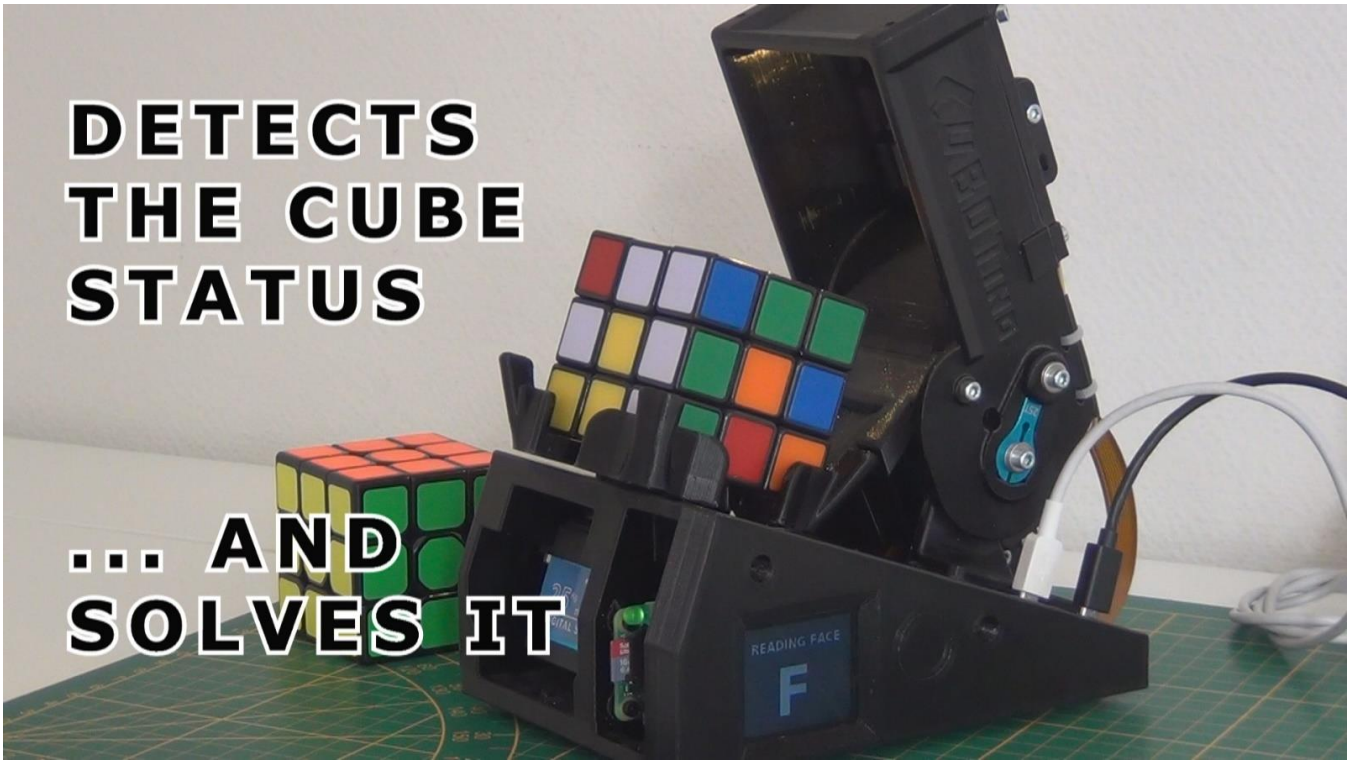
CUBOTino autonomous: A small, 3D printed, Rubik's cube solver robot

https://www.instructables.com/CUBOTino-Autonomous-Small-3D-Printed-Rubiks-Cube-R/
https://github.com/AndreaFavero71/cubotino

Andrea Favero, Groningen (NL)

27/10/2022 Rev. 3.10 (always check if a newer version is available)

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



This is an autonomous Rubik's cube solver robot, the "Top version" of CUBOTino series. (the Cubotino Base version can be seen at: https://youtu.be/ZVbVmCKwYnQ)

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

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

This is one of the smallest autonomous robots, for Rubik's cube solving, from those not requiring a special cube.

1. Instructions: Order and organization

This document is organized in about 30 chapters, divided into 5 main sections:

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

Sections:

1. Supplies

2. Connections_board & Raspberry Pi setup
    a. Make the connections_board
    b. Setup the Raspberry Pi
    c. Test the servos rotation range
    d. Set the servos to mid position

3. 3D print and assembly
    a. Print the parts
    b. Assemble the robot
    c. Using a Raspberry Pi 3 or 4

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

5. Info (my preferred part 😊, yet not strictly needed to build the robot)
    a. Project background
    b. High level information
    c. Robot solver algorithm
    d. Computer vision
    e. Colour detection strategy
    ….and much more

Use the Summary links to quickly reach the chapters

1) Safety

Energize the robot only via USB ports having a class 2 insulation from the power supply net; Raspberry Pi power supply and phone charger normally have this safety feature: Check it for your own safety.
Despite the robot mechanical force is limited, it must be operated only under adult supervision.
If you build and use a robot, based on this information, you are accepting it is on your own risk.

2. Manage expectations

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

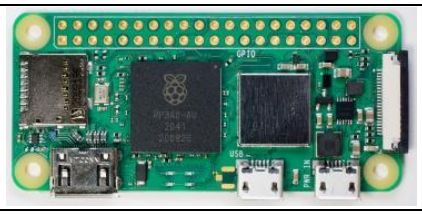
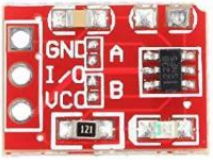
- servos
- arm positioning to the servo
- cube dimensions
- print quality

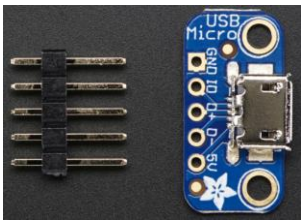
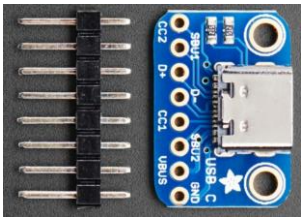
But hey, don’t worry …. Other makers have successfully tuned their own Cubotino, and you will too 😊

… and now, let’s start!

Summary

2) Supplies

Q.ty	Part	link to the shop I used	Cost (euro)	Notes
2	Servos I used in first place: TD-8325MG (**180deg** 25Kg metal, **Pulse width 1to2ms**) and metal arm "25T" **Check the (better) alternative servos on next pages before ordering**	https://www.aliexpress.com/item/32298149426.html?gatewayAdapt=glo2ita&spm=a2g0o.9042311.0.0.5d1e4c4d14Qjaz	25 (2 servos + 2 arms)	 max 31mm protrusion under the flange
1	Raspberry Pi Zero2 (WH needed, yet the header can be bought at side)	https://www.sossolutions.nl/	18 (W version)	
1	microSDHC 16GB	https://www.dataio.nl/sandisk-ultra-micro-sdhc-16gb-uhs-i-a1-met-adapter/	8	
1	PiCamera V1.3	https://www.amazon.nl/gp/product/B01M6UCEM5/ref=ppx_yo_dt_b_asin_title_o05_s00?ie=UTF8&th=1	7.5	
1	30cm cable Raspberry Pi Zero/Camera	https://www.amazon.nl/gp/product/B079H33VCM/ref=ppx_yo_dt_b_asin_title_o05_s01?ie=UTF8&th=1	5	
1	SPI TFT 128x160 Pixels Display (1.77 inch) with ST7735 driver	https://www.amazon.nl/gp/product/B078JBBPXK/ref=ppx_yo_dt_b_asin_title_o03_s00?ie=UTF8&th=1	8	
1	Led module 3W	https://www.aliexpress.com/item/1005001984730729.html?spm=a2g0o.cart.0.0.58be3c00SgxRBT&mp=1	2.6 (4 pcs)	
1	2.5-5.5V TTP223 capacitive touch switch button self-locking module for Arduino	https://www.amazon.nl/-/en/gp/product/B07BVN4CNH/ref=ppx_od_dt_b_asin_title_s01?ie=UTF8&psc=1	4 (5 pcs)	

1	AMS1117-3.3 DC 4.75V-12V to 3.3V Voltage Regulator	https://www.amazon.nl/gp/product/B07MY2NMQ6/ref=ppx_yo_dt_b_asin_title_o00_s00?ie=UTF8&psc=1	6 (5 pcs)	
2	USB MICRO-B BREAKOUT BOARD	https://www.kiwi-electronics.nl/nl/usb-micro-b-breakout-board-3908	3.8	
1	Alternative solution: USB Type-C BREAKOUT BOARD	https://www.adafruit.com/product/4090	3.5	
~500g	Filament 1.75mm		~10	Suggested PETG, yet other material will do the job

Electrical small parts:

Q.ty	Part	Notes
1	Prototype board	To connect all the parts
1	40pin (2x20) GPIO male header	In case you could not get the WH version of Raspberry Pi
1	40pin (2x20) GPIO female header	To connect the Connections board to Raspberry Pi Zero 2
1x8	Female Header	To connect the display to the Connections board
4x3	Male Headers 90deg	To connect the servos, touch pads, Led module
2x3	Female Headers	To connect the touch pads, Led module
3	Capacitor 16V 220uF	To limit voltage drop when servos are activated
1	Heat sink for Raspberry Pi	

Screws:

Quantity	Dimension	Head type	Info
1	M4x20	Cylindrical	Pivot for Top_cover
~ 30	M3x12	Cylindrical	
~30	M3x12	Conical	
4	M2.5x10	Cylindrical	Rpi to Structure
4	M2.5x4	Cylindrical	Micro-usb break-out boards to PCB cover

Power supply:

- If micro-USB: 2x 2A phone charger with micro-USB cable, nowadays abandoned into some drawers at home….
- If USB type C: 1x 3A

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

Alternative servos

Option 1:

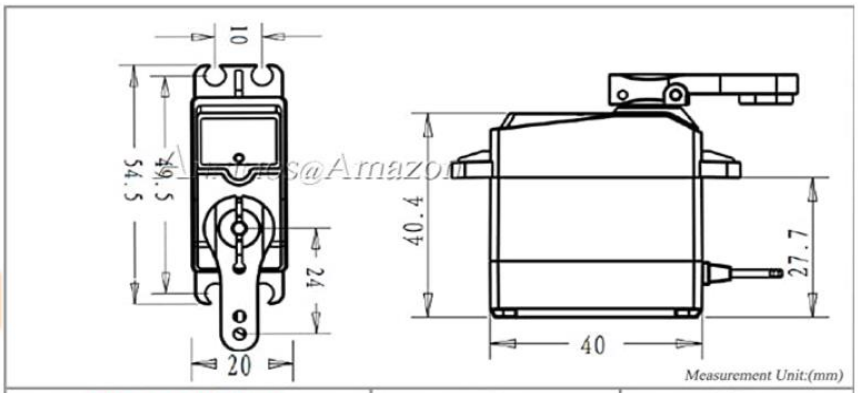
It is possible to use servos having 270deg rotation combined with an extended Pulse Width (from 500µs to 2500 µs): When I started this project, I didn't know about the extended pulse width… a missed opportunity I would call it now! Rotation and Pulse Width ranges makes this servo a much better choice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

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

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

As reference this servo has these characteristics:

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



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

For the tests I've ordered the 180° version from Amazon Netherlands:
https://www.amazon.nl/gp/product/B01MU78A29/ref=ppx_yo_dt_b_asin_title_o02_s01?ie=UTF8&psc=1

On the received servo, the real rotation range is about 192°, therefore still ok for the robot.
I've measured a resolution <1° (vs 1.8 of the servos I've initially used, and suggested), making easier the angles settings.
When moving from a 180° to a 270° version, the angle resolution should still be around 1.4°: Again better than the servo I initially used and suggested.

Option 2 (a cheaper servo model):

- The MG996R servo is enclosure in a plastic body, while having the reduction gears made by metal (MG might mean Metal Gear) as well as a metal bush at the outlet gear.
- Torque is ca 10Kg*cm, therefore much lower than the metal version I've initially used.
- It looks like this model is rather common for 180° rotation range, and rather rare for the 270°.
- The several online shops I've verified don't specify the PWM range.
  I ordered them to Amazon Netherland https://www.amazon.nl/AZDelivery-Versnellingsbak-Helikoptervliegtuigen-compatibel-Inclusief/dp/B07H88DB8R/ref=asc_df_B07H88DB8R/?tag=nlshogostdde-21&linkCode=df0&hvadid=594732236372&hvpos=&hvnetw=g&hvrand=8695186734418167946&hvpone=&hvptwo=&hvqmt=&hvdev=c&hvdvcmdl=&hvlocint=&hvlocphy=9103947&hvtargid=pla-653808722062&th=1
  The servos I got have a Pulse Width ranging from 500µs to 2500 µs.
- Because of the lower rated torque, and less metal parts, these servos are cheaper, and apparently widely available; It is typically sold width a set of plastic arms.

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

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

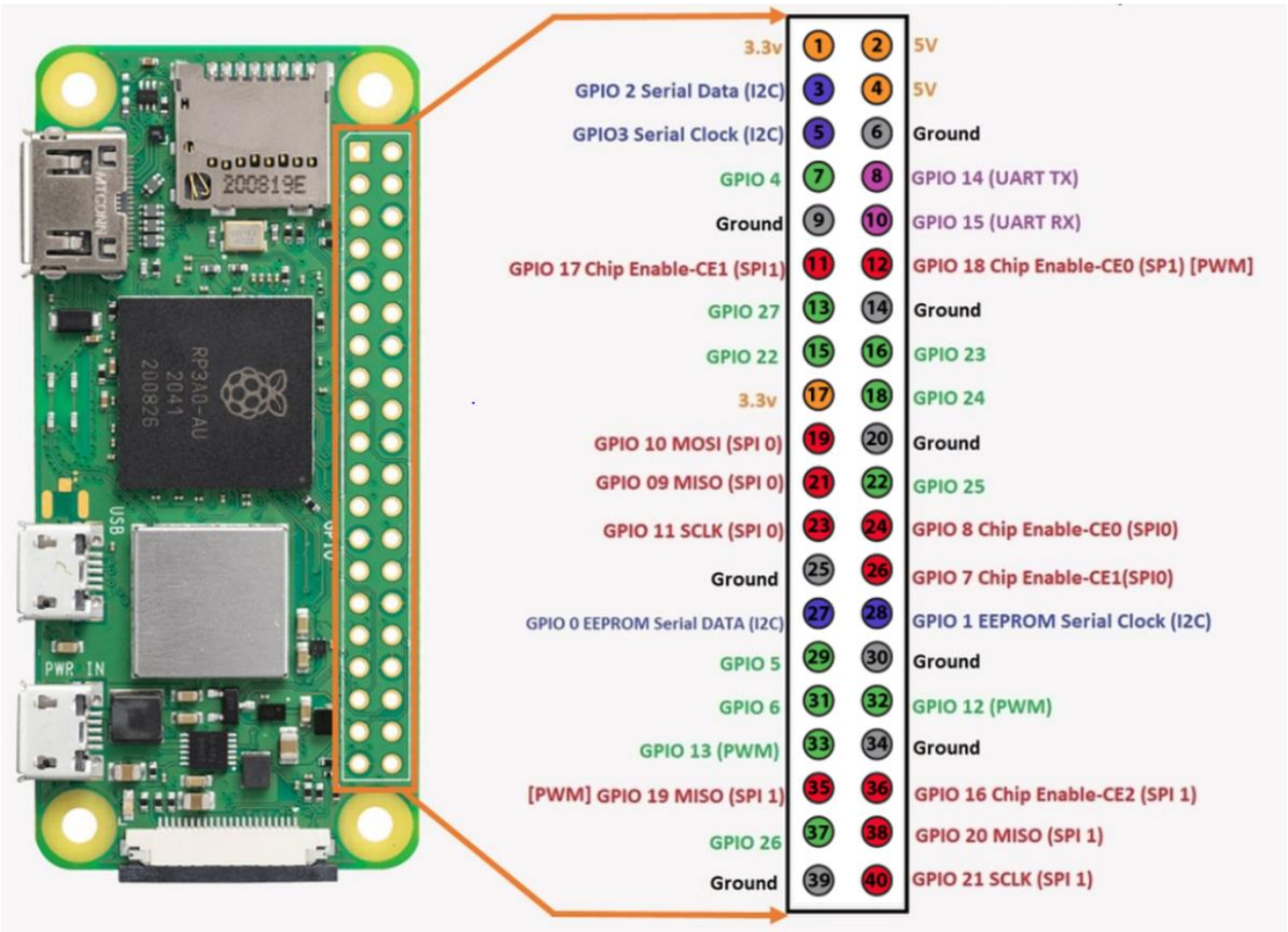
Top servo:

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

As reference:

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

3) Rasperry Pi Zero 2 GPIO pinout



Used pins:

Pin	Label/GPIO	Purpose
2, 4	5V	Energize the SBC board
25, 6	GND	Energize the SBC board (Note: GND continuity between pins 25 and 6 is made by the Raspberry Pi board)
8	GPIO 14 (UART TX)	Led of Rpi ON status
7	GPIO 4	8 Display backlight
13	GPIO 27	6 Display RS (Register select)
15	GPIO 22	5 Display reset
24	GPIO 8 (CHIP ENABLE SPI 0)	7 Display CS (Chip select)
19	GPIO 10 MOSI (SPI 0)	4 Display SDA (Serial Data Pin)
23	GPIO 11 SCLK (SPI 0)	3 Display SCK (Serial Clock)
12	GPIO 18 [PWM]	PWM Led at Top_cover
32	GPIO 12 (PWM)	PWM servo Top_cover
33	GPIO 13 (PWM)	PWM servo Cube_holder
37	GPIO 26	Touch button

4) Connections board

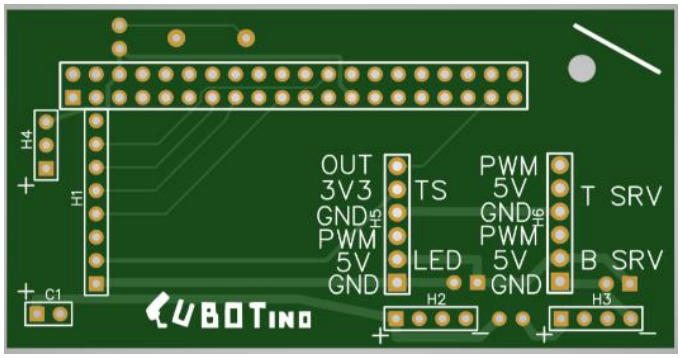
There are some options for the Connections board:

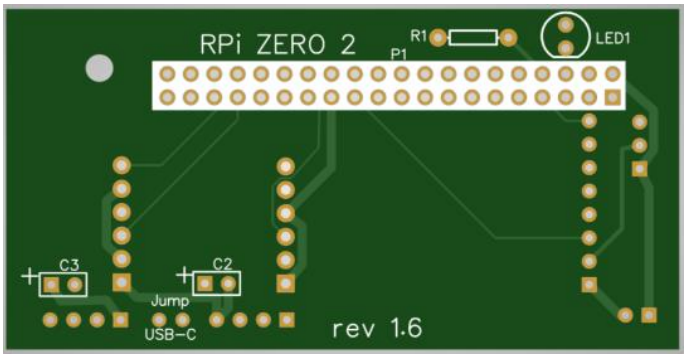
A) Chad (Vanblokland) has drawn and made available the Gerber files to order the pcb to a pcb manufactures; On the received board you'll 'just' solder the few components needed.
B) You can make it by your own via a (perfboard) prototyping board
C) You can energize it via 2 x 2A microUSB power supply,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

A) Notes for the printed circuit board:

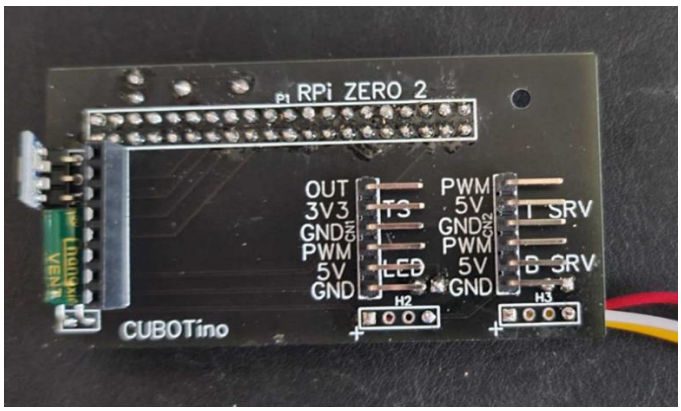
1. This cover the V1.1 and V1.6 revision, that have been positively tested; V1.6 replaces the V1.1
2. Gerber files are available at https://github.com/AndreaFavero71/cubotino/tree/main/connections_board/gerber

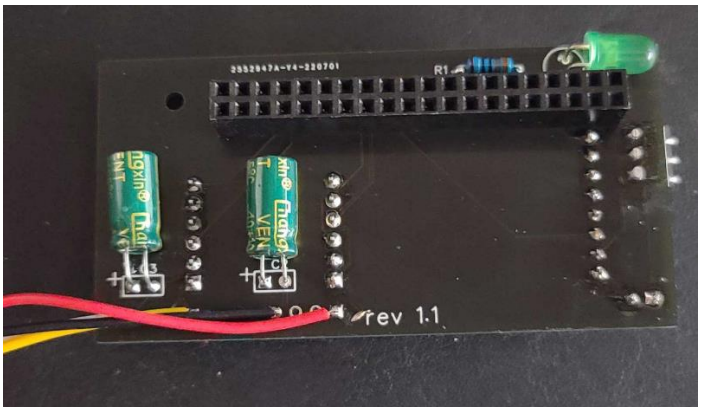
   To download the Gerber files folder from GitHub:
   1. Via this link a helpful online tool opens: https://downgit.github.io/
   2. Paste the address of the Gerber folder
   3. Click Download, and a zip file of the folder will be downloaded into your download folder.
3. For the pcb order check: https://docs.easyeda.com/en/PCB/Order-PCB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





Just as example:

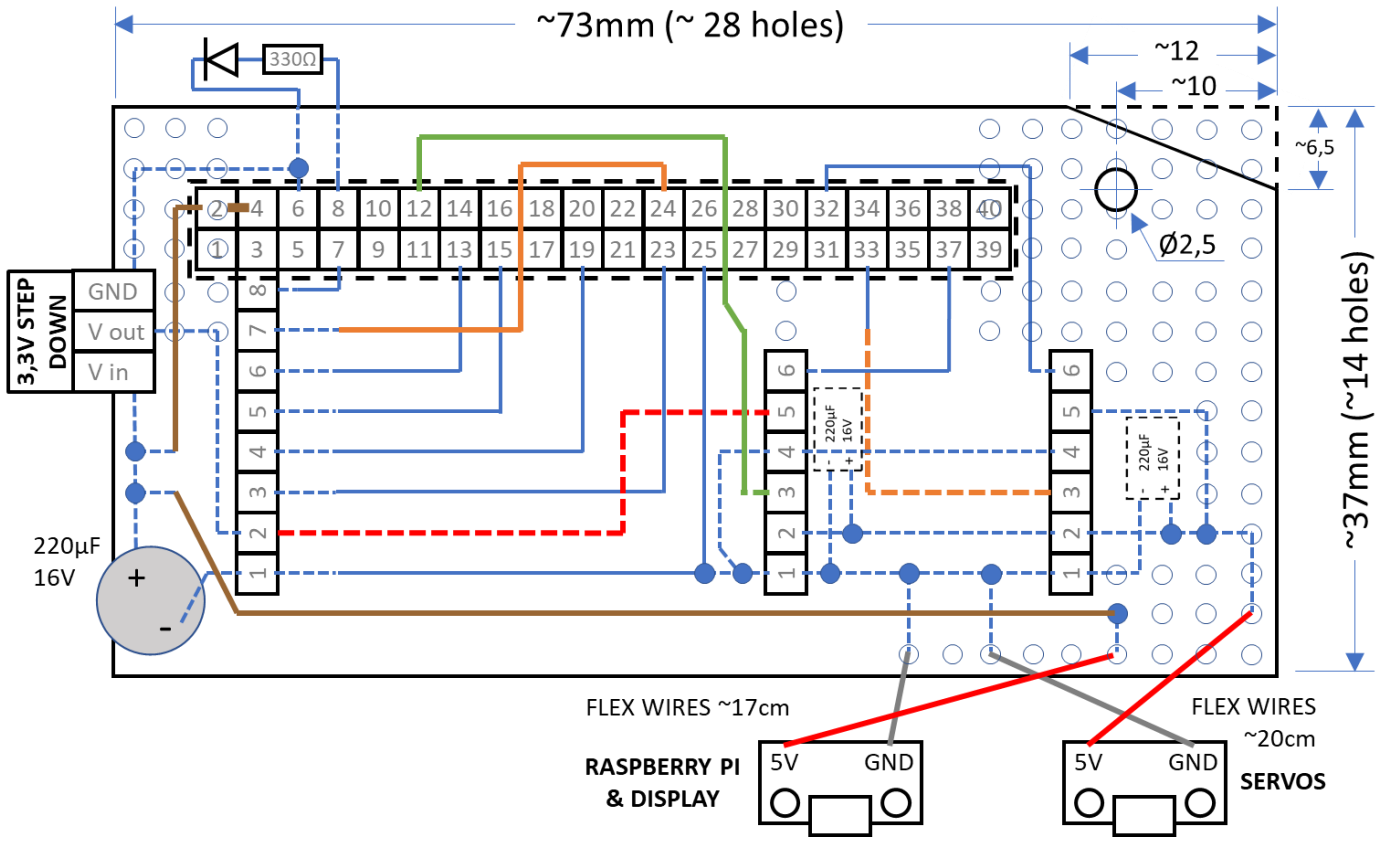


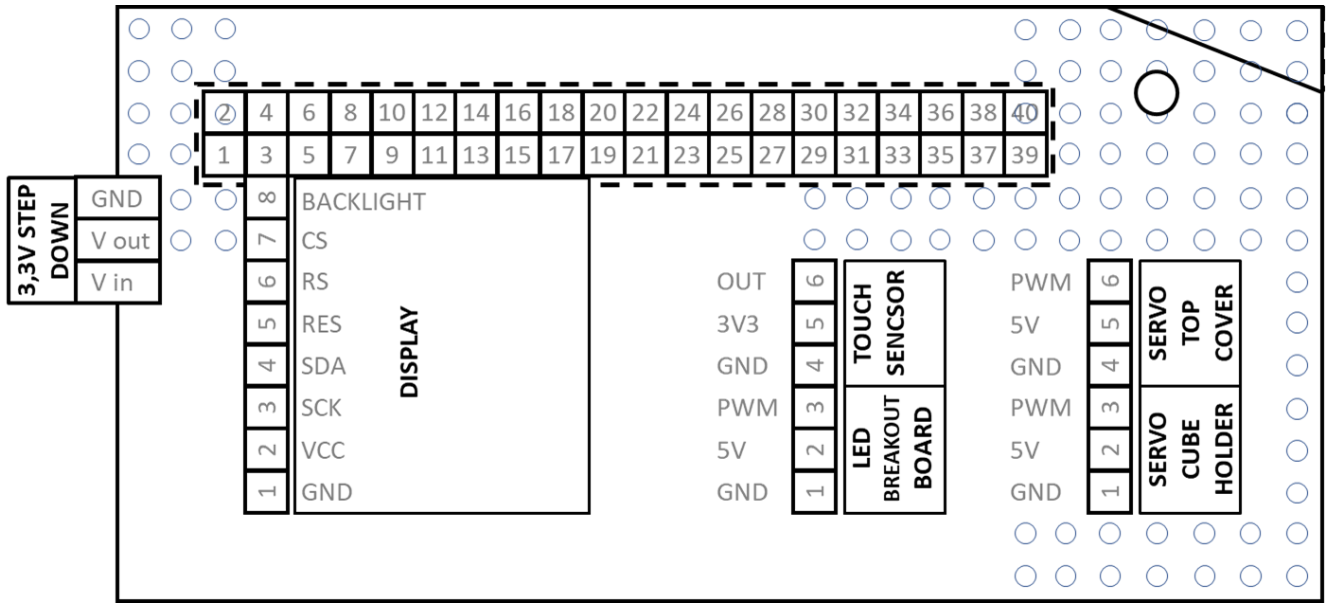


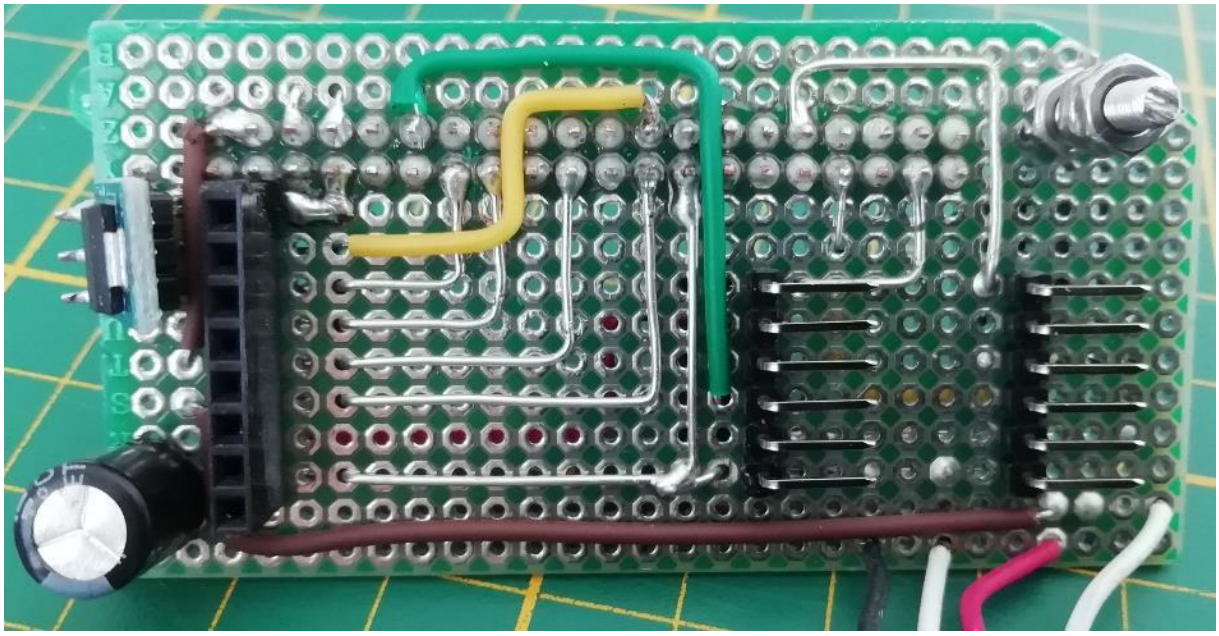
B) Notes for the prototyped board:

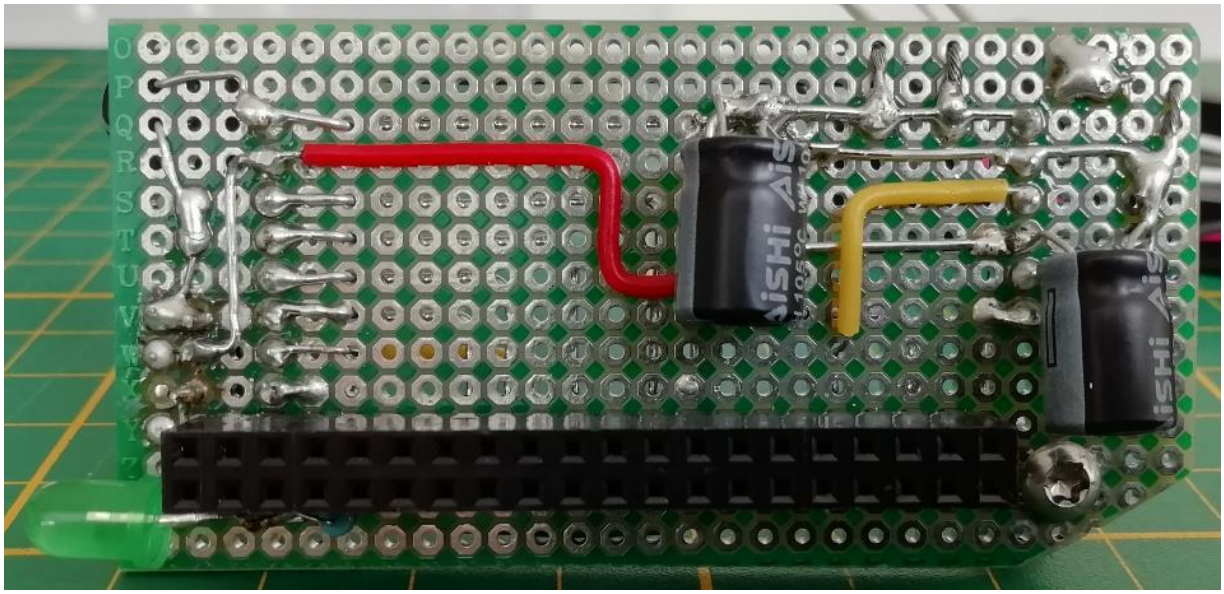
1. The prototype board should be of a double face type.
2. Start by positioning the 2x20 header, at 3rd hole from the corner.
3. Count the holes for the other parts positioning.
4. Dashed lines for parts/wires on the opposite side.
5. Blue lines are wires without insulation.
6. Red, Brown, Orange and Green lines are insulated wires.
7. Use insulated wires when crossing other lines, also when these are on the board opposite side.
8. Filled dots are connections.
9. 15mm is the max height for the 3V3 step down board, and the capacitor close to the display connector.
10. Ø2,5mm hole is for a M2,5mm bolt (and 3 nuts) to support the display. The highest nut should be at ~10.5mm from the board surface.
11. Add the last 2 capacitors, close to the servo connectors, once the board has been tested.

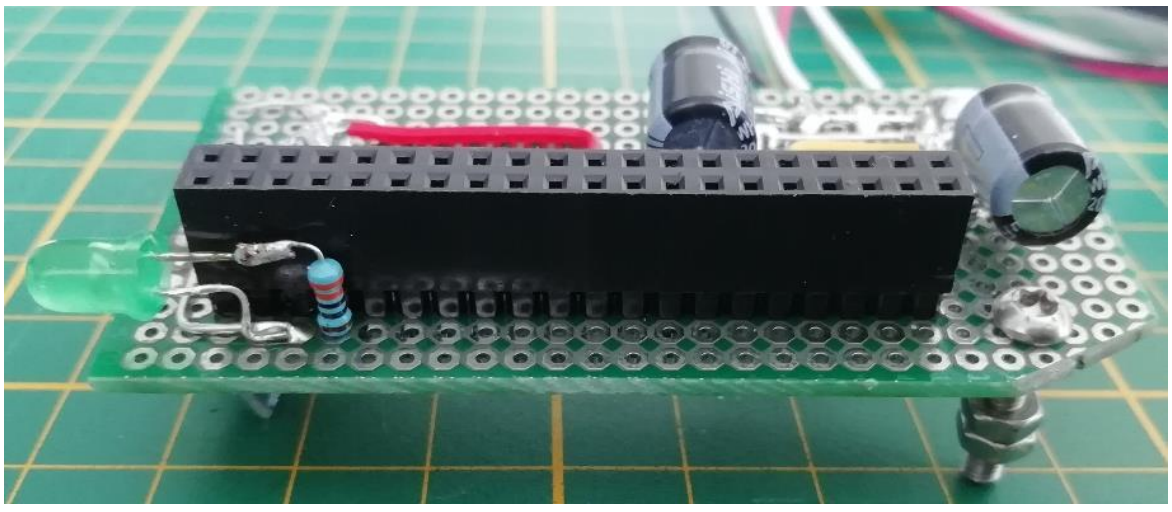
Front view (the 2x20 header is on the opposite side):

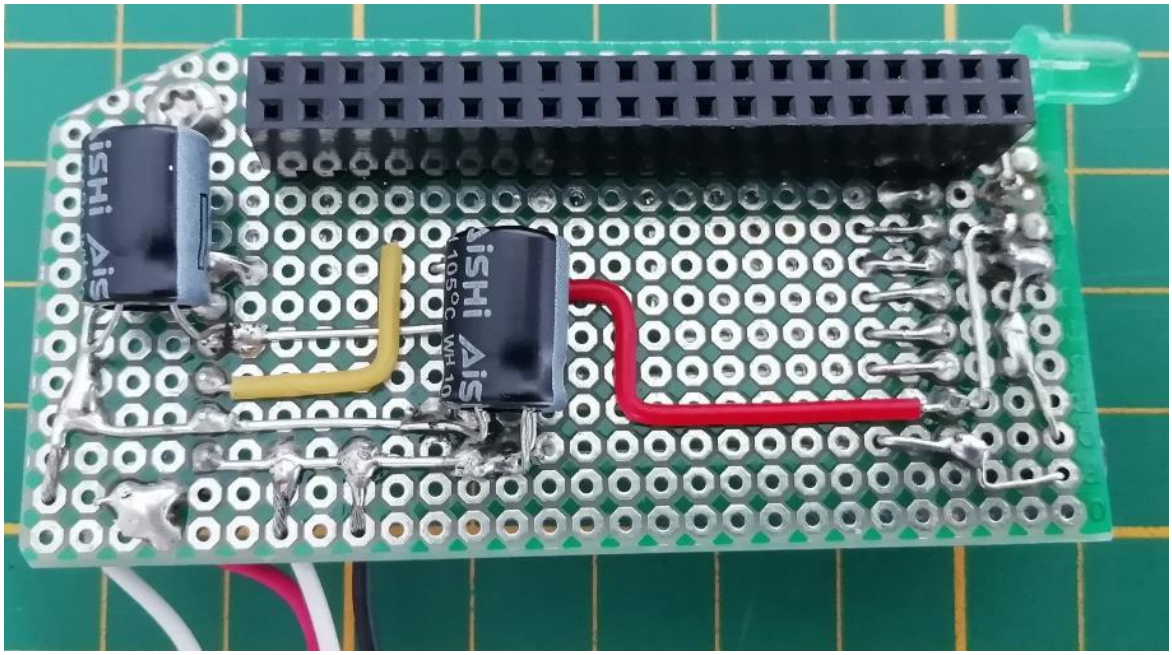


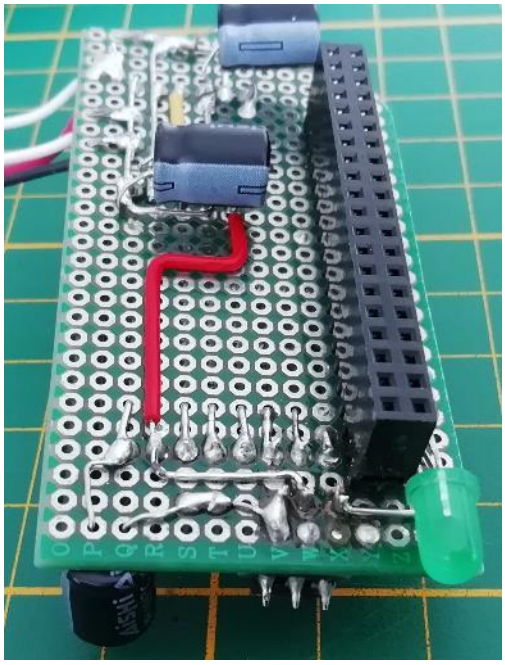




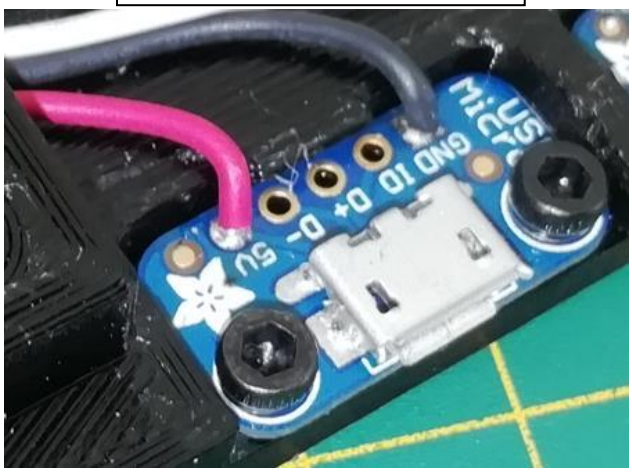








microUSB breakout board:
Wire soldering orientation



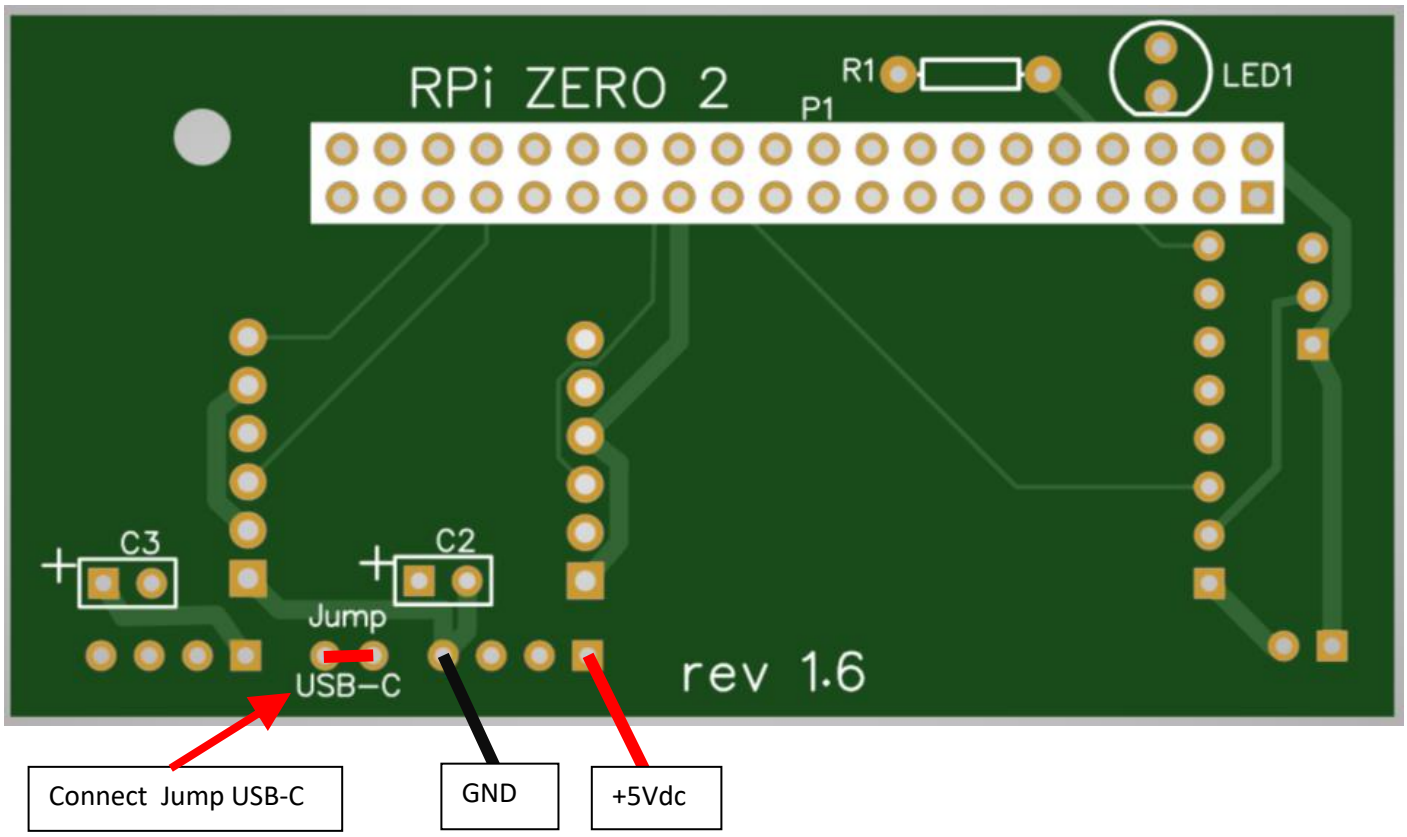
Wiring	Wire length approx. (cm)
USB breakout board	**~18**
Touch sensor breakout board	**~18**
3W Led breakout board	**~35**

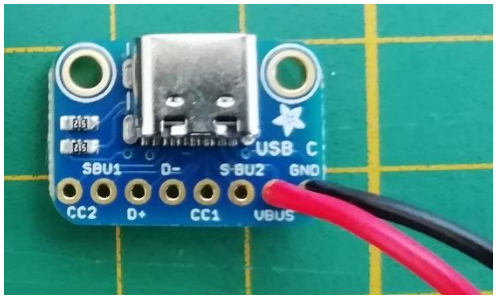
Connecting a single power supply input (board V1.6):

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

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

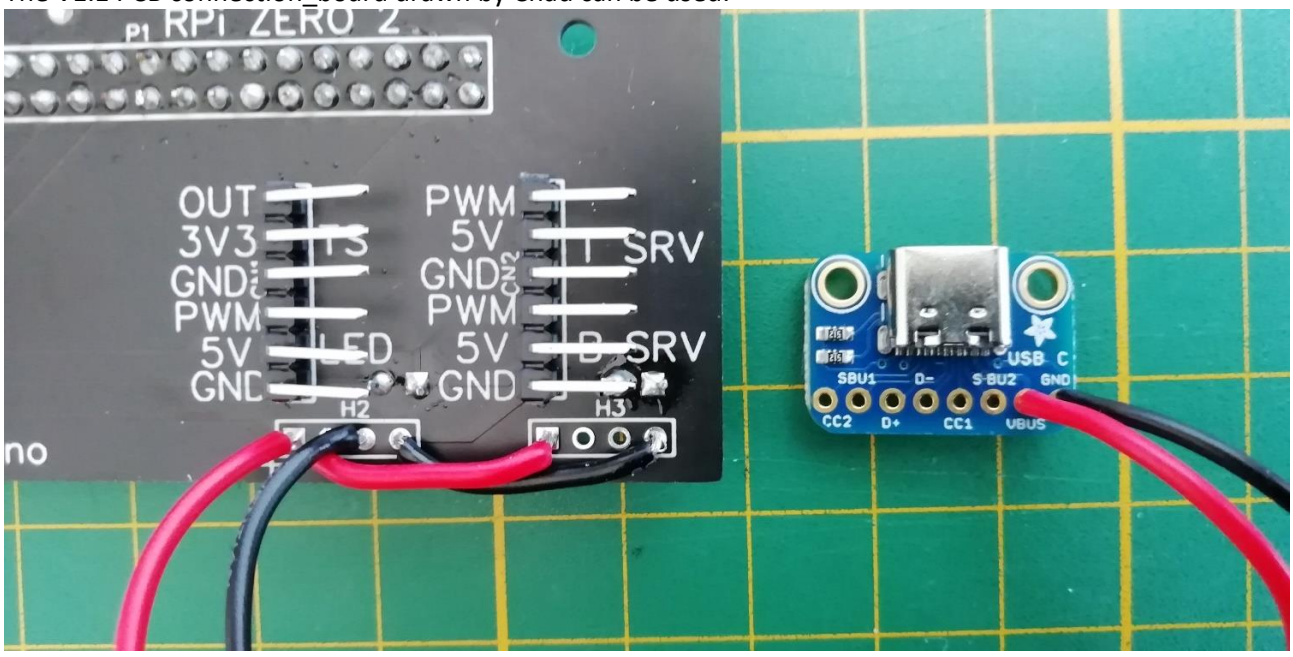
1. Connect the USB-C breakout board to either H2 or H3
2. Connect the “Jump USB-C” pads



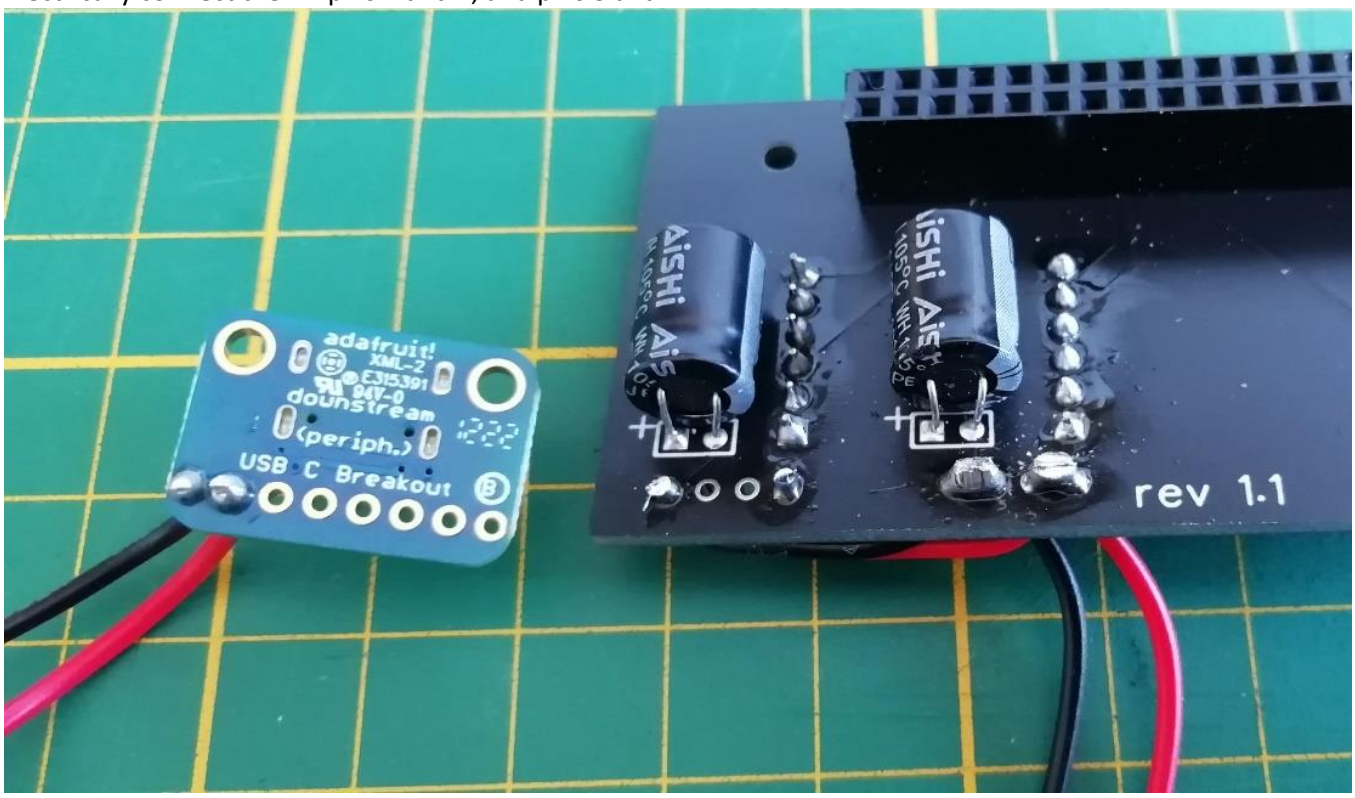


Connecting a single power supply input (board V1.1):
USB type-C connector is rated 3A, therefore possible to use a single power supply connections for the robot.

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



Notes:
At H2 and H3 the soldering pads #2 and #3 are not connected.
Make use of these to mechanically constrain the wires ah H2 pins 2 and 3
Electrically connect the H2 pins 1 and 2, and pins 3 and 4:



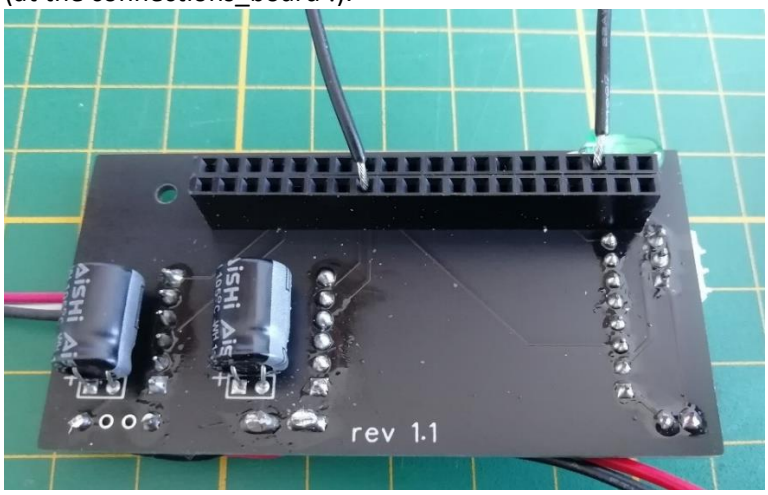
Testing the connections_board before connecting it to the raspberry Pi:

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

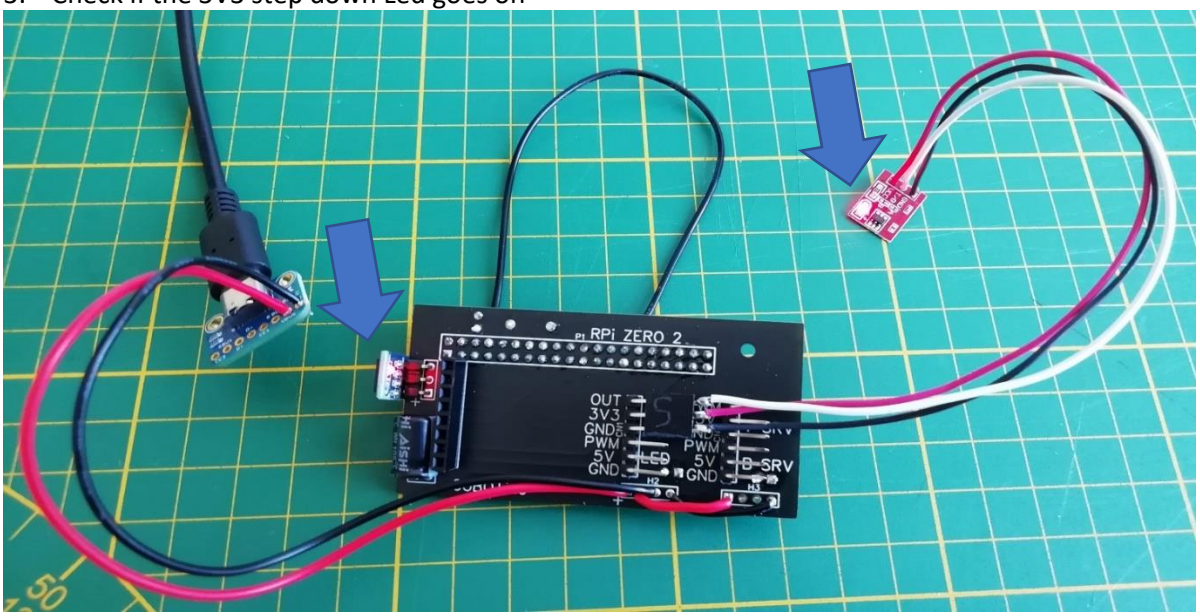
1. Connect the Touch_sensor breakout board.
2. Energize the board, via the USB type-C break out board or via the micro-USB breakout board connected to H2
3. Touch the touch_sensor pad and check if the breakout board Led goes on.

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

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



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



5) Setting up Raspberry Pi Zero 2

Notes:

1. Starting from 30th June 2022 only the simplified installation (via GitHub repository) is described in this document.
2. In case you're interested in the manual installation, please refer to the document "How_to_make_CUBOTino_autonomous_robot_20220625.pdf" at Instructables site or GitHub.

The simplified installation method forces some differences compared to the manual installation method previously described; Some of these differences are:

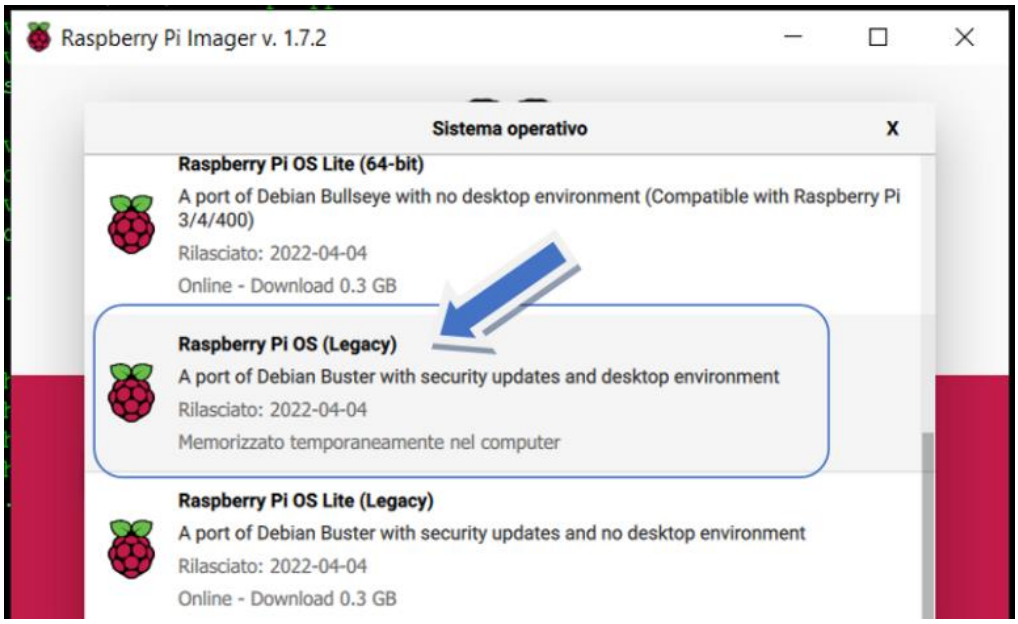
- Hostname changed from raspberry to cubotino
- Main folder changed from /home/pi/cube to /home/pi/cubotino
- Robot specific files moved from /home/pi/cube to /home/pi/cubotino/src
- Virtual environment method

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

Step1: Download the Raspberry Pi Imager, from https://www.raspberrypi.com/software/

Step 2: From the Raspberry Pi Imager

a. Selection of Raspberry Pi OS: under the "raspberry Pi OS (other)", and choose "RASPBERRY PI OS (LEGACY)"
   The reasons to select this 'special' version are explained at
   https://www.raspberrypi.com/news/new-old-functionality-with-raspberry-pi-os-legacy/



b. Select the SD card

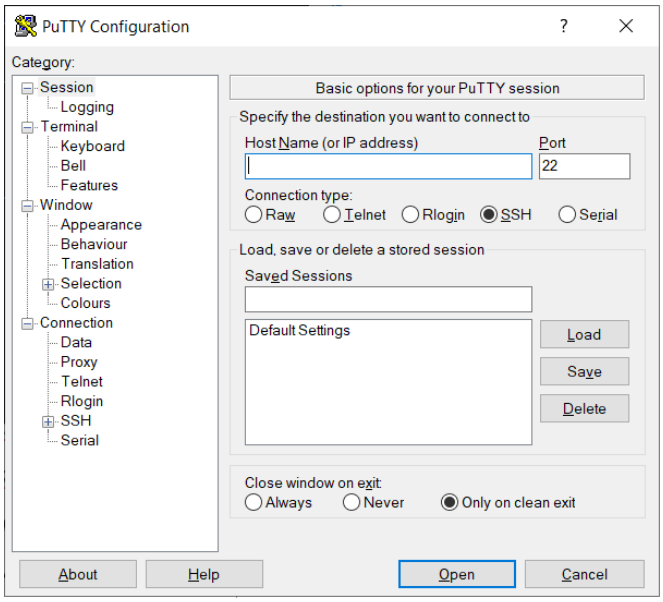
c. On settings, enter:
    a. *cubotino*.local
    b. check SSH
    c. enter *pi* as username
    d. enter a password (my choice has been *raspberry,* as no sensitive data, but you are encouraged to use a more secure password)
    e. check set WiFi
    f. enter your SSID
    g. enter your network password
    h. enter your Country code
    i. set your local settings
    j. set ejects at the end
d. write the OS, that takes around 10 minutes

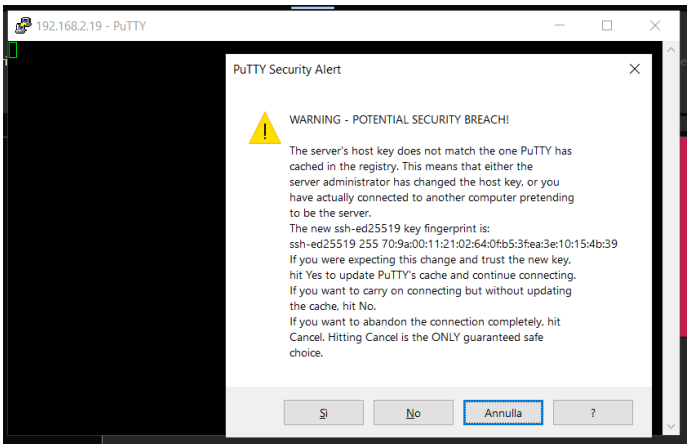
Step 3: Insert the SD card into the Raspberry Pi, and power it.
First boot takes longer, up to two minutes; Wait until the led stop blinking

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

Step 4b: In case you cannot connect via *ssh pi@cubotino.local*, search the Raspberry Pi IP address (different tools for that, i.e. Advanced IP Scanner)

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)

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

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

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

In one or few minutes, depending on the internet connection, files will be cloned into raspberry Pi:

```
pi@cubotino:~ $ git clone https://github.com/AndreaFavero71/cubotino.git
Cloning into 'cubotino'...
remote: Enumerating objects: 235, done.
remote: Counting objects: 100% (167/167), done.
remote: Compressing objects: 100% (145/145), done.
remote: Total 235 (delta 96), reused 36 (delta 16), pack-reused 68
Receiving objects: 100% (235/235), 90.74 MiB | 2.02 MiB/s, done.
Resolving deltas: 100% (102/102), done.
Checking out files: 100% (52/52), done.
pi@cubotino:~ $
```

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

Step 7: Start the installation:

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

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

The simplified installation is now completed

The below folder structure will result at Raspberry Pi:

/home/pi/cubotino/	Is the main robot 21older
/home/ pi/cubotino/connections_boardl	contains the gerber files
/home/ pi/cubotino/doc	contains the How_to_buid… .pdf file
/home/ pi/cubotino/extra	contains the link to the Instructables page of this robot
/home/ pi/cubotino/images	contains the Cubotino logo image for the display
/home/ pi/cubotino/stl	contains all the robot stl files
/home/ pi/cubotino/stp	contains all the robot stp files
/home/ pi/cubotino/scr/	contains all the robot specific files
/home/ pi/cubotino/src/twophase	contains the lookup tables for Kociemba solver
/home/ pi/cubotino/src/install	contains the setup.sh bash file: Do not use this file!

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

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

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

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

Step 8: Updating the installation:

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

1. Enter cubotino folder : *cd ~/cubotino*
2. Type *git status* to check if there are updates
3. In case there are updates available, and you want to install them:
    i. type *git reset --hard* to discharge local files changes
    ii. type *git pull* to receive the updates

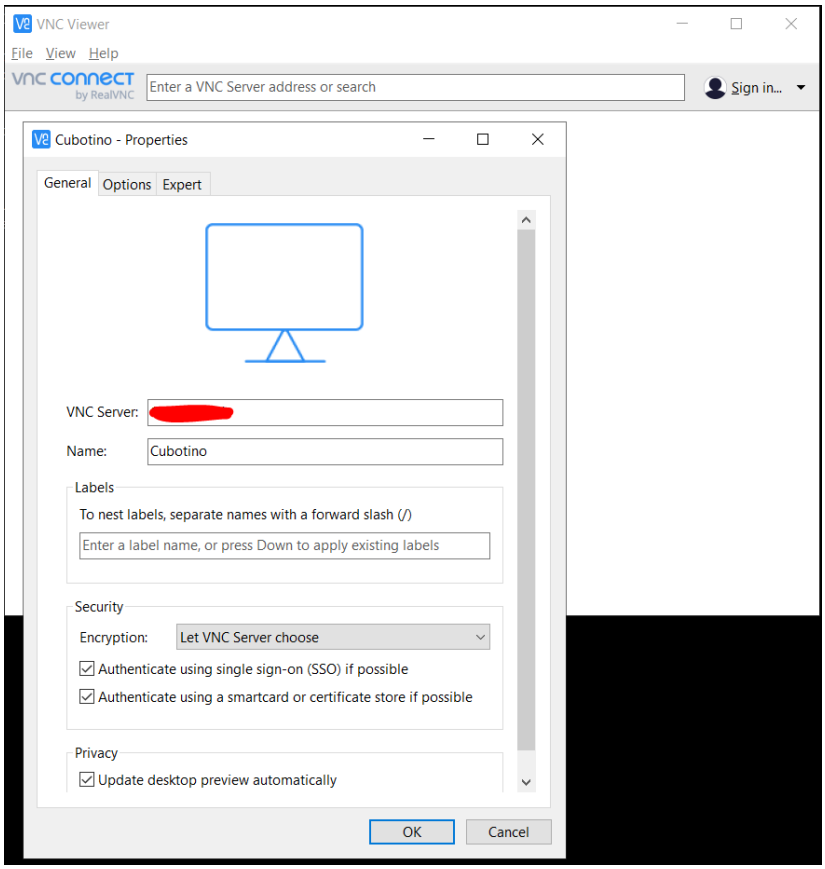
Note: Please remember to take notes (or save a copy) of your personal settings (Cubotino_T_settings.txt and Cubotino_T_servo_settings.txt) before updating the robot package;
In case you forgot…, the robot makes a backup copy for you at every boot; In this case rename the backup files:

Cubotino_T_settings_backup.txt → Cubotino_T_settings.txt
Cubotino_T_servo_settings_backup.txt → Cubotino_T_servo_settings.txt

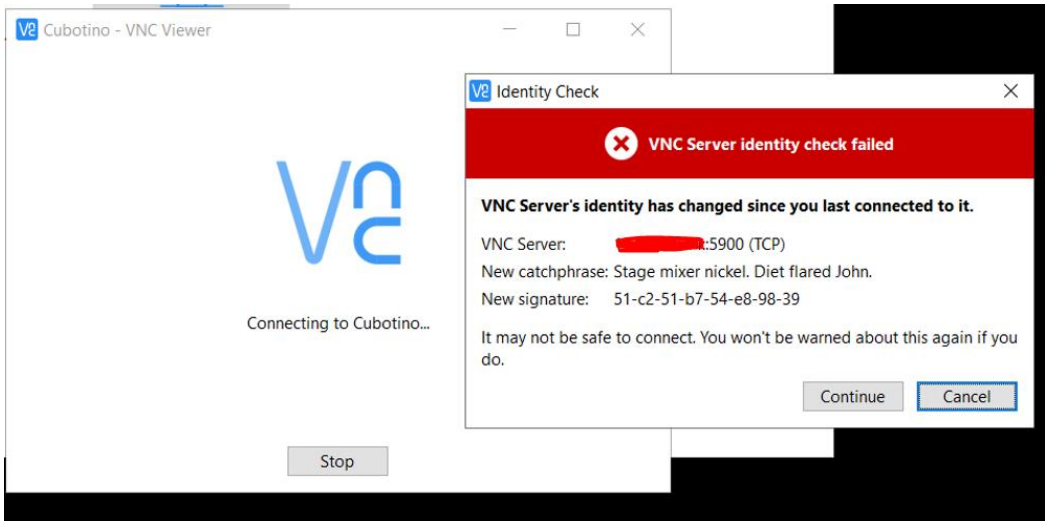
Please bear in mind the updates are based on my robot; There might be other changes you've made: Those have to be handled by you another time.
I can think of the display orientation if you've a Raspberry Pi 3 or 4, and you've oriented the display differently.
You might also have personalized some prints to the display: Recall to save notes/snippets.

Step 9: Make a new connections at VNC Viewer

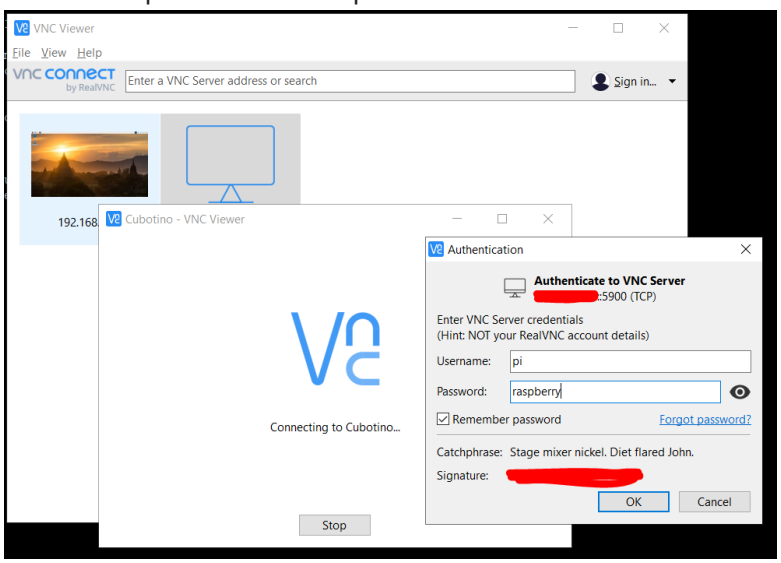
Make a new connection: File, New connections …., insert the IP address at VNC Server, and a Name



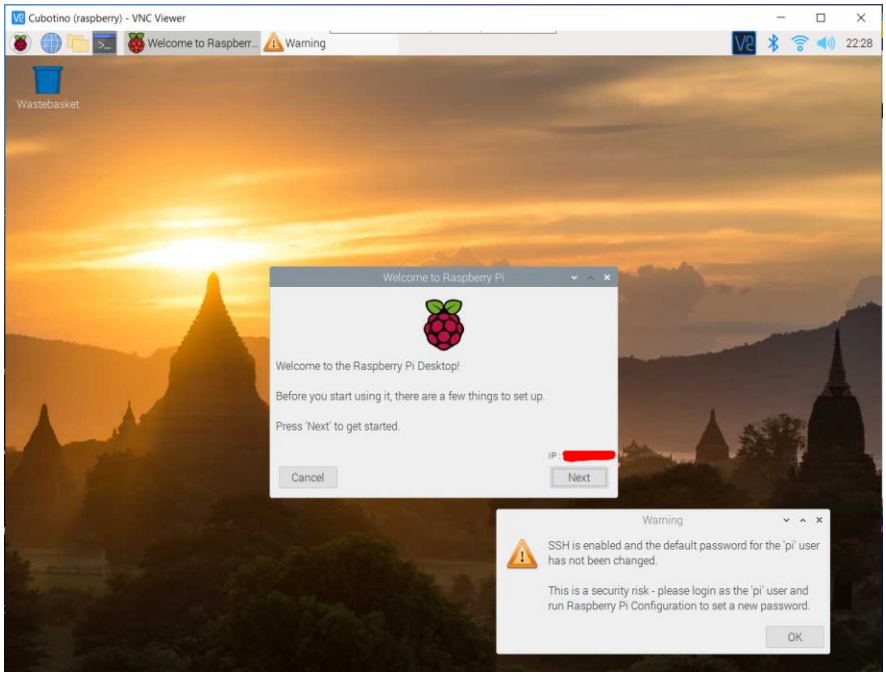
Confirm the warning



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

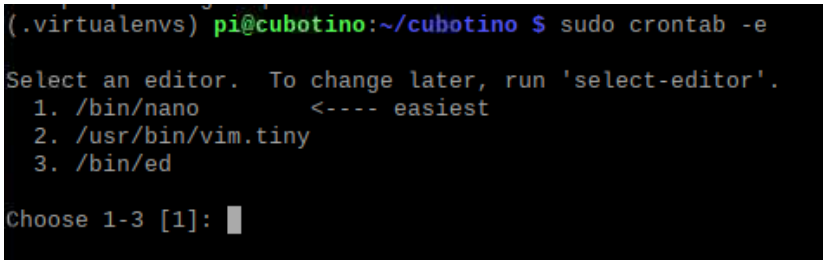
Complete the settings

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

Step 10: Change the monitor size in case you don't feel comfortable with the proposed resolution

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

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



Change the geometry on the line:

@reboot su – pi -c \"/usr/bin/vncserver :0 -geometry 1280x720\";

(note: 1920x1080 fits my laptop screen, and it allows me to work easily)

Step 11: Make a backup image of the microSD

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

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

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

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

See 'Servos test and set to mid position

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

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

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

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

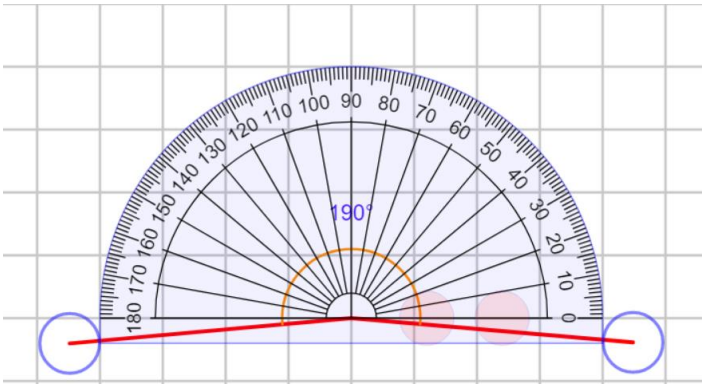
Argument set, and related value, can be passed to Cubotino_T_servos.py, to play with the servos angle and especially to set them to the mid position before the assembling.

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

Position the servo arm as per below picture:

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

The angle can be evaluated by printing a protractor:



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

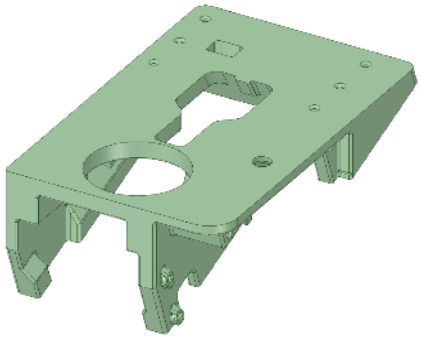
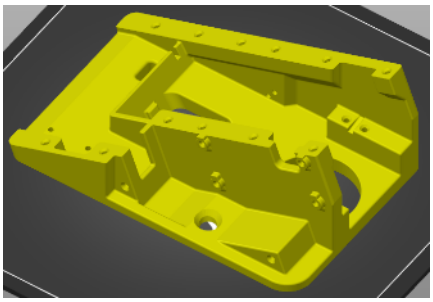
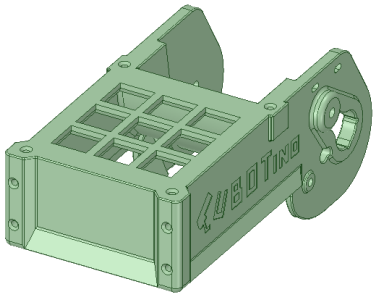
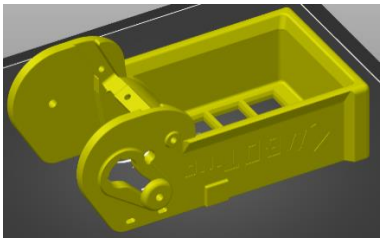
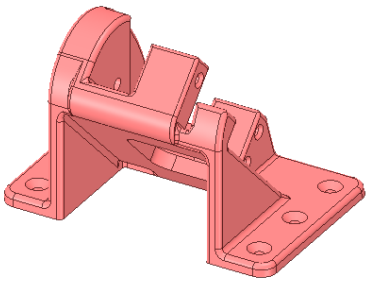
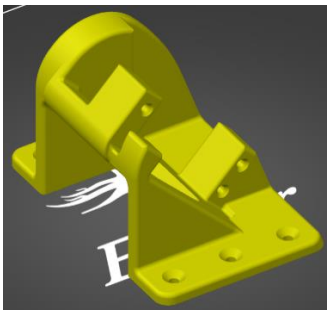
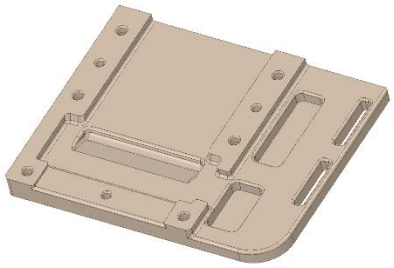
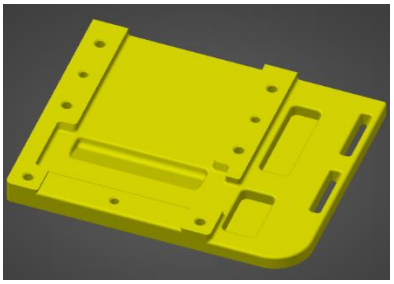
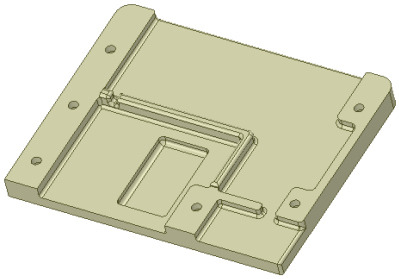
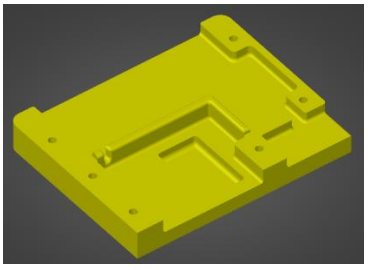
6) 3D printed parts

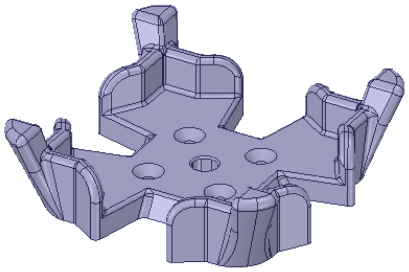
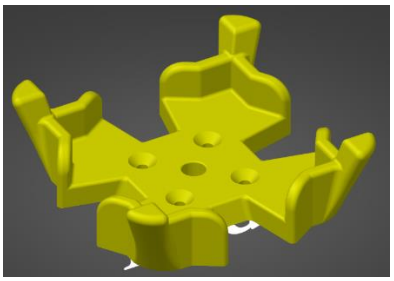
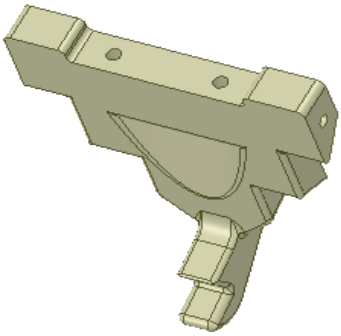
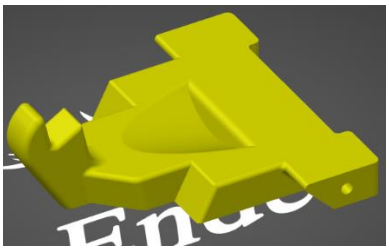
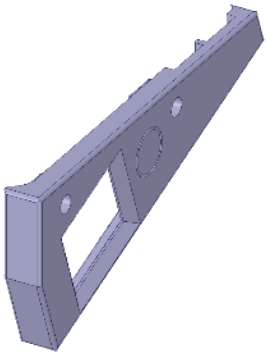
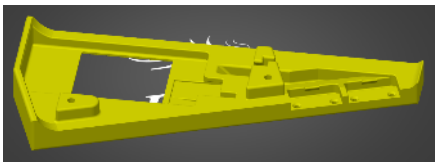
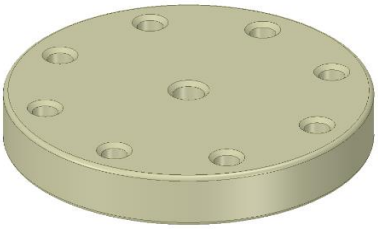
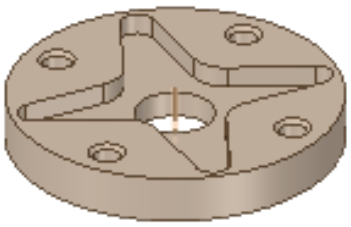
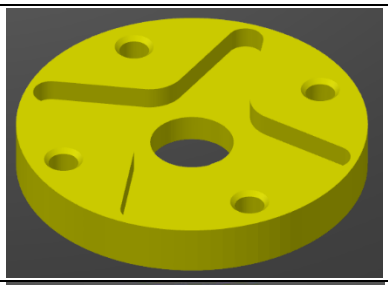
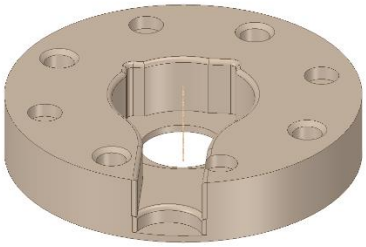
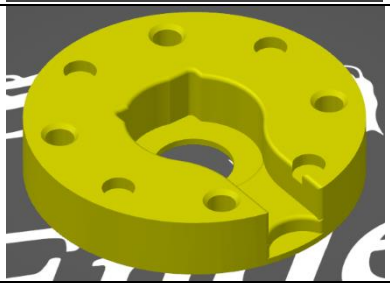
See notes below for filament quantity, and printing time …

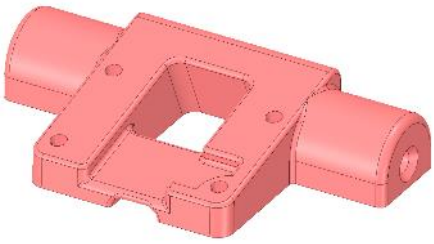
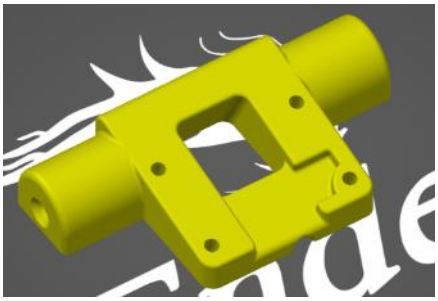
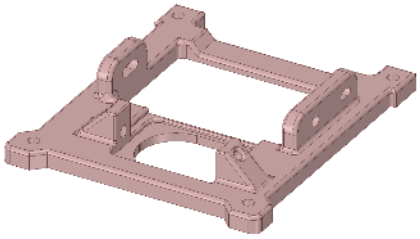
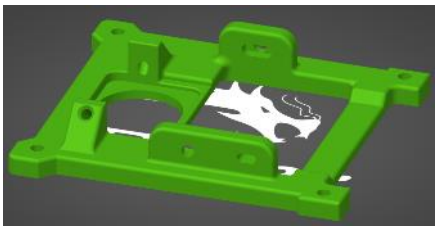
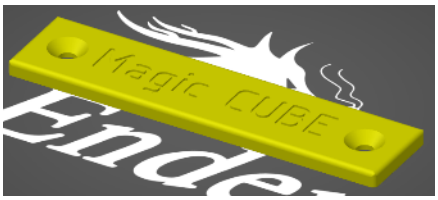
Ref	Part	Filament		Printing time	Version specific parts
		Meters	Grams		
1	Structure	49	145	14h33m	
2	Top_cover	39,5	117	12h26m	
3	Hinge	17,2	51	5h37m	
4	Baseplate front	12,2	36	3h33m	
5	Baseplate back	12,6	37,3	3h36m	
6	Cube_holder	11,6	34,4	3h36m	
7	Cube_lifter	5,5	16,2	1h48m	
8	Servo_axis_sup (or its alternative)	2,4	7,2	0h55m	
9	Servo_axis_inf (or its alternative)	2,4	7	0h52m	
10	PCB_cover_display (or its alternative)	14	41.4	4h13m (4h27m)	Top specific
11	PiCamera holder	2.2	6.5	0h50m	Top specific
12	PiCamera holder frame	6.1	18	2h17m	Top specific
13	Personaliz_plate	1.5	5	0h30	Top specific
	TOTAL	**175m**	**692g**	**51h30m**	

Notes:

1. The biggest part is the Structure, and it easily fits on printers with plate size of 200x200mm.
2. Filament length is based on Ø1.75mm.
3. Filament weight is based on PETG density (1.23g/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
4. The filament quantity, and printing time, in the table should be considered as an upper limit. After printing the parts for the base version, total weight was closer to 400grams than 466grams estimated by the slicer SW.
5. Possible to upgrade the Base version, by printing 3 (or 4) more parts, requiring ~ 8h.
6. All parts have been designed to be printed without supporting the overhangs.
7. Some parts have been split, for easier and better 3D printing.
8. The suggested part orientation for the 3D print is showed on below Table.
9. The **stl** files are available at
    - https://www.instructables.com/CUBOTino-Autonomous-Small-3D-Printed-Rubiks-Cube-R/
    - https://www.thingiverse.com/thing:5407226
    - https://github.com/AndreaFavero71/cubotino/tree/main/stl
10. The **step** files are only available at https://github.com/AndreaFavero71/cubotino/tree/main/stp

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

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

PiCamera holder		
PiCamera frame		
Personaliz_plate Or Personaliz_plate01	 	

7) Assembly steps

Before assembling the robot:

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

Assembly order are initially as per the Cubotino_base_version:

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

Starting from here, the steps are different from the Cubotino_base_version

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

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

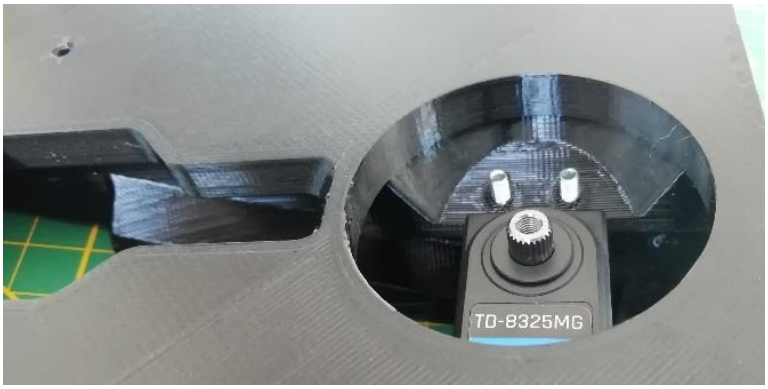
8) Assembly details

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

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

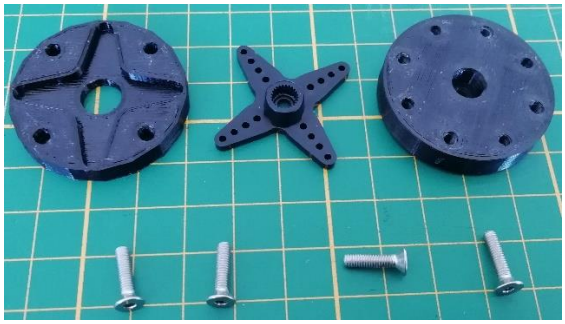
Couple of notes:

- Before tightening the four screws, checks if the servo output gear is well centred to the Structure hole.
- To limit the protrusion of the below two screws, add a couple of washers to these screws; This to prevent eventual interference with the servo_axis_inf part.



Step5 (sandwich Servo_axis_inf / servo arm / Servo_axis_inf):

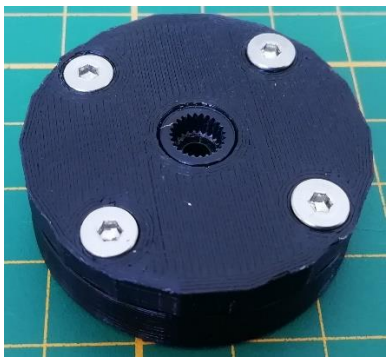
4x M3x12mm conical head

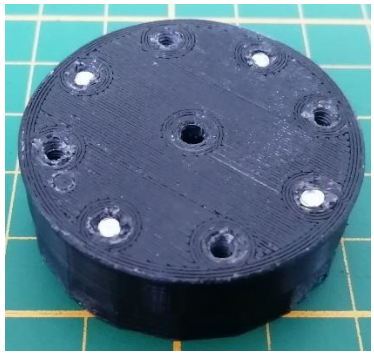


Check if the supplied X arm fits with the indentation of Servo_axis_inf part:

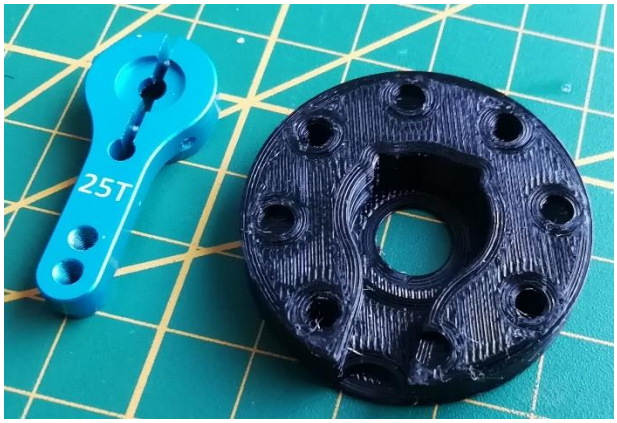
If you don't have a X arm to make it fit, please print the alternative parts (Servo_axis_inf and Servo_axis_sup) designed to fit the aluminium "T25" arm (arm has to be reduced in length).

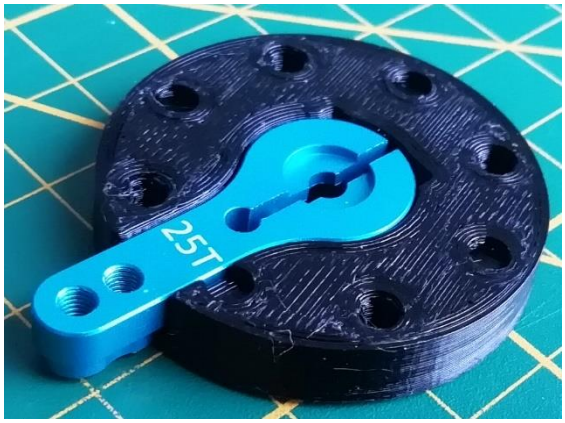
Make the sandwich





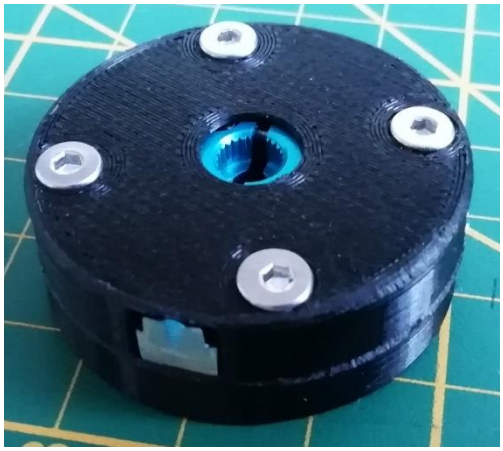
Step5a Alternative servo axis assembly:





Cut the protruding part of the “T25” arm
Adjust its screws to be able to enter the servo outlet gear

Make the sandwich as per Step5
4x M3x12mm conical head



Step6 (Assemble the Cube_holder to Servo_axis assembly):

4x M3x12mm conical head



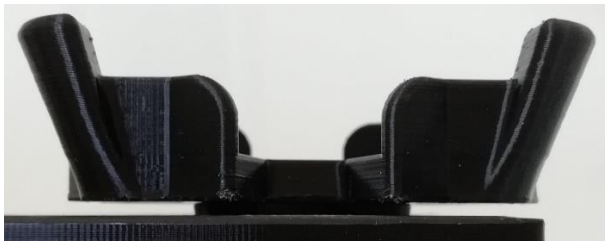
Step7 (Assemble the Cube_holder assembly to the bottom servo):

Try to not rotate the Servo output gear during this step: Gently try to feel the teeth coupling, and if the Cube_holder is not well aligned, retract it, rotate the Cube_holder by about 90 degrees and check again.

Servo output, and Servo arm, have even number of teeth: For sure there is one good coupling

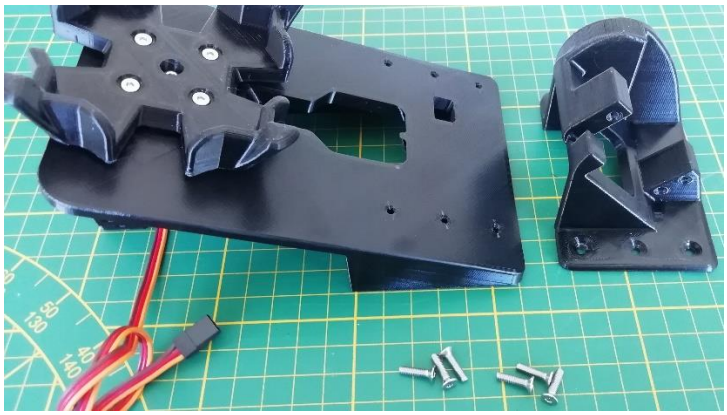

1x M3x12mm cylindrical head

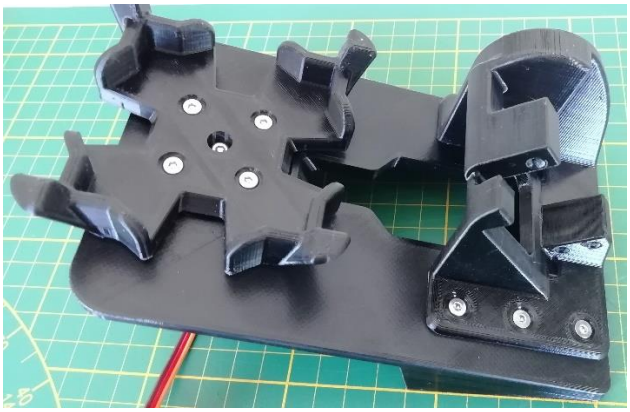




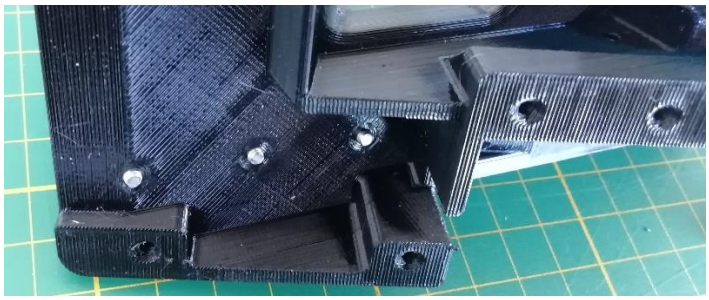
Step8 (Assemble the Hinge to the Structure):

6x M3x12mm conical head





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



Step9 (Assemble the "T25" servo arm to the upper servo):

Place the "T25" arm along the main Servo axis (Servo should be prepared upfront with the gear at middle angle).

Close the two tiny screws at the arm

Step10 (Insert the Top_servo assembly into the Top_cover slot):

1x M3x12mm cylindrical head

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

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

Note: The screw should not protrude from the Structure plane; Add some washers under the screw head if needed

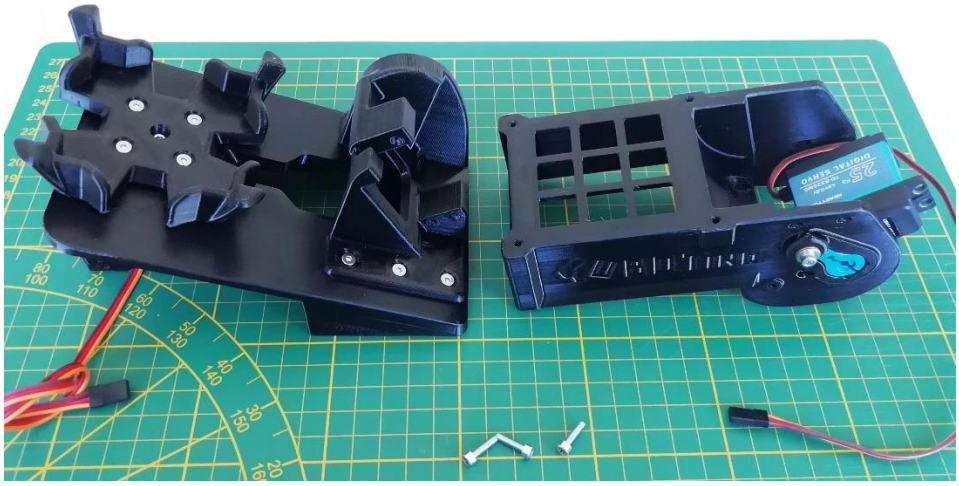
Rotate the servo by about 45deg, to facilitate next steps

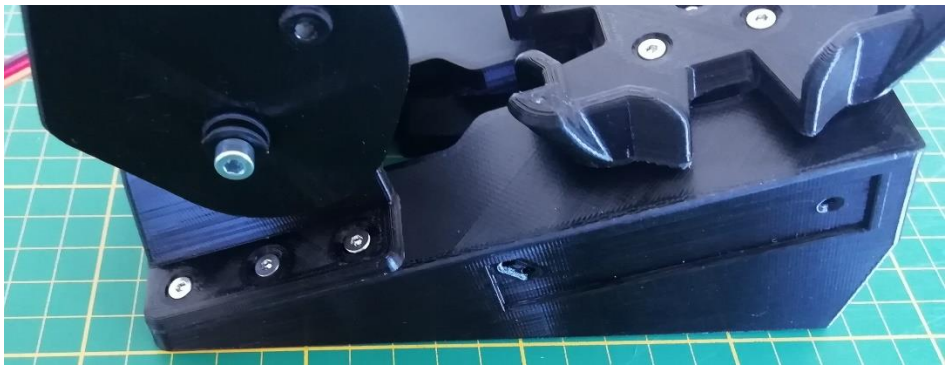
Step11 (Assemble the Top_cover assembly to the Hinge):

1x M4x20mm cylindrical head

3x M3x12mm cylindrical head

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

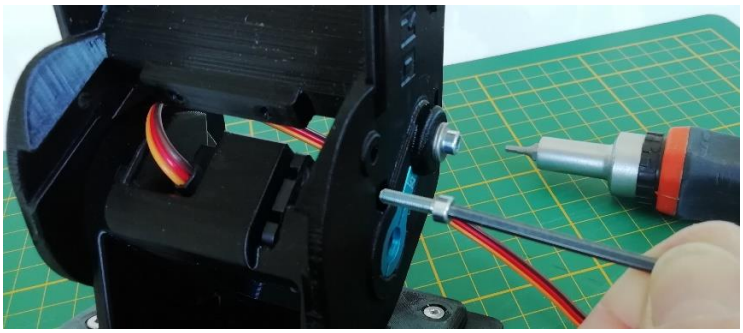




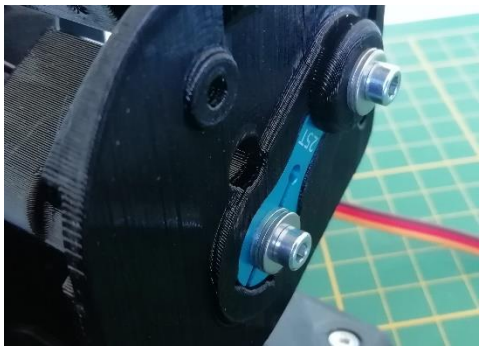
Do not fully tighten the two M3x12mm, until also the third screw is positioned

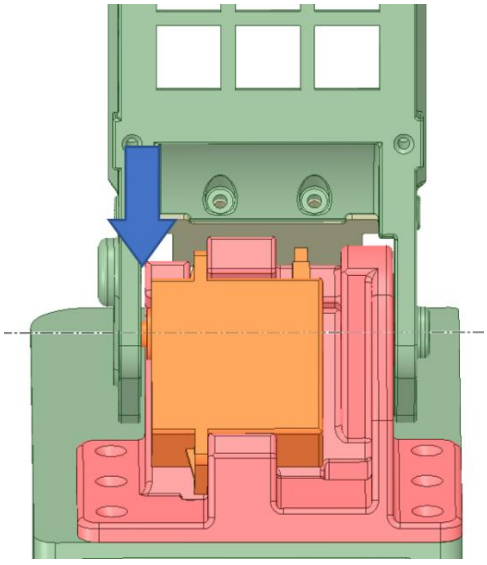
Step12 (Complete the top servo assembly to the Hinge):

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



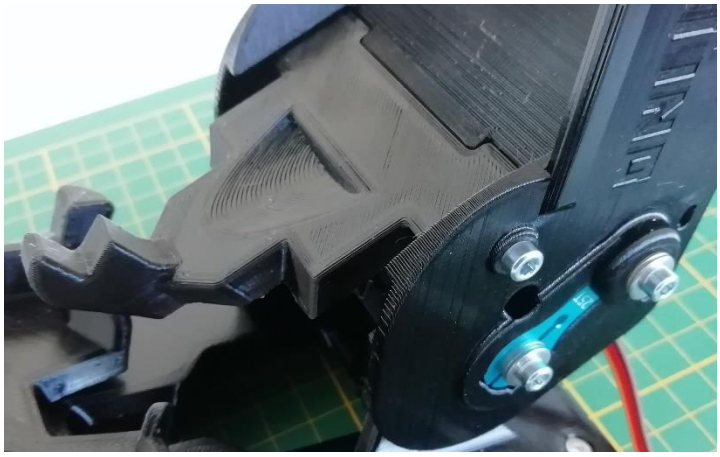
1x M3x12mm cylindrical head (add washers if the screws doesn't push on the "T25"Arm)

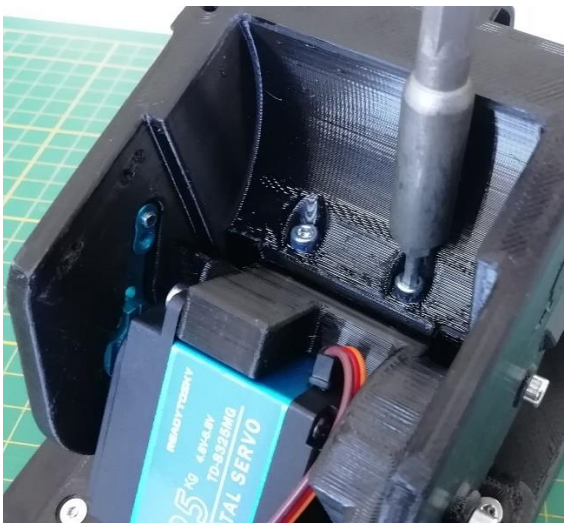




Step13 (Assemble the Lifter to the Top_cover):
4x M3x12mm cylindrical head

Slide the Lifter into the Top_cover slots

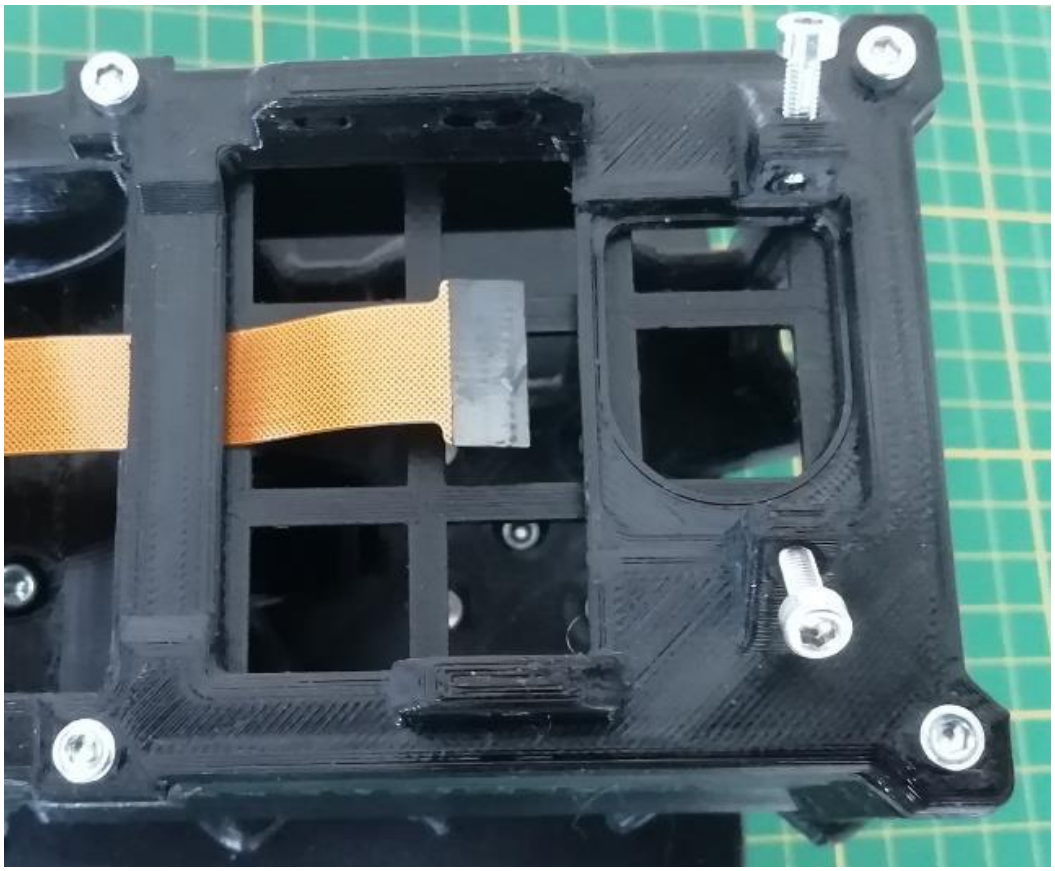




Step14 (Assemble the PiCamera holder frame to the Top_cover):

4x M3x12mm cylindrical head

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

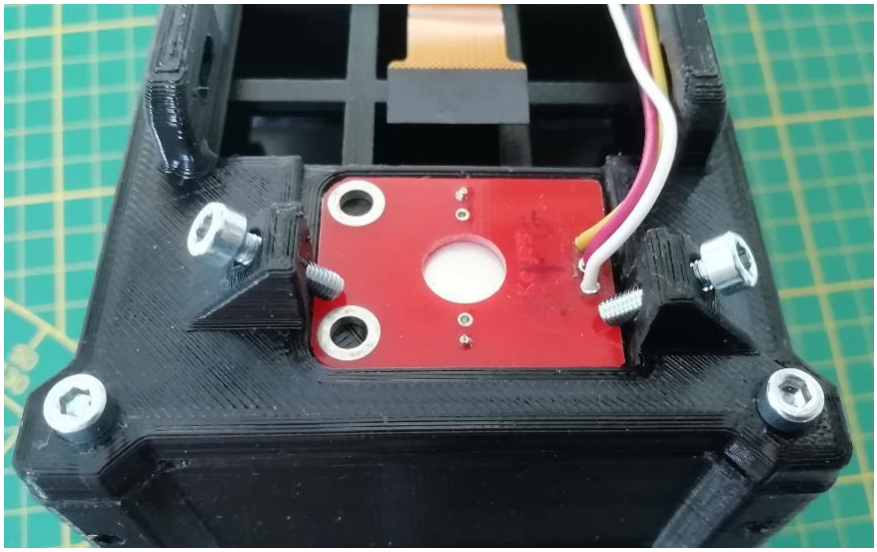


Step15 (Assemble the LED breakout board):

Add 2x M3x12mm cylindrical head

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

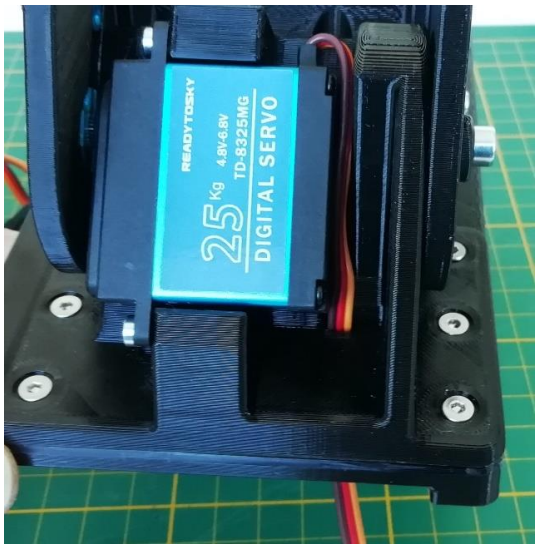
Stop screwing once the screw tip touches the board

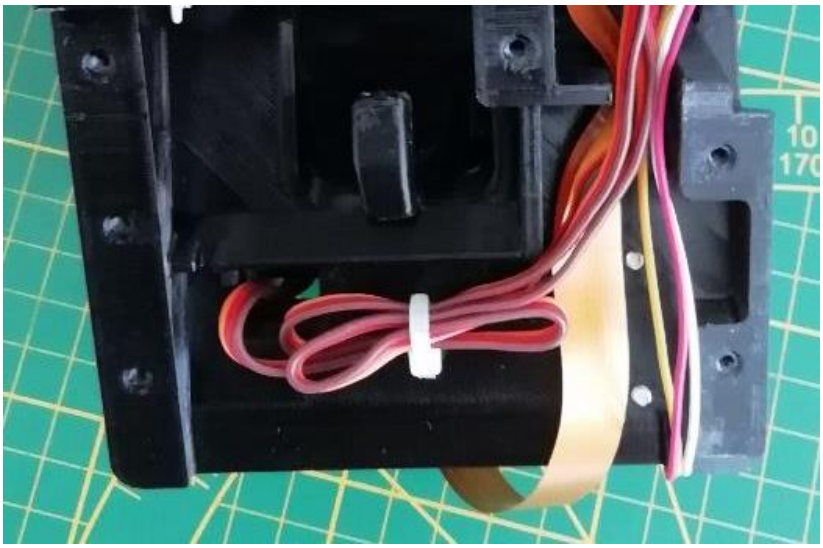


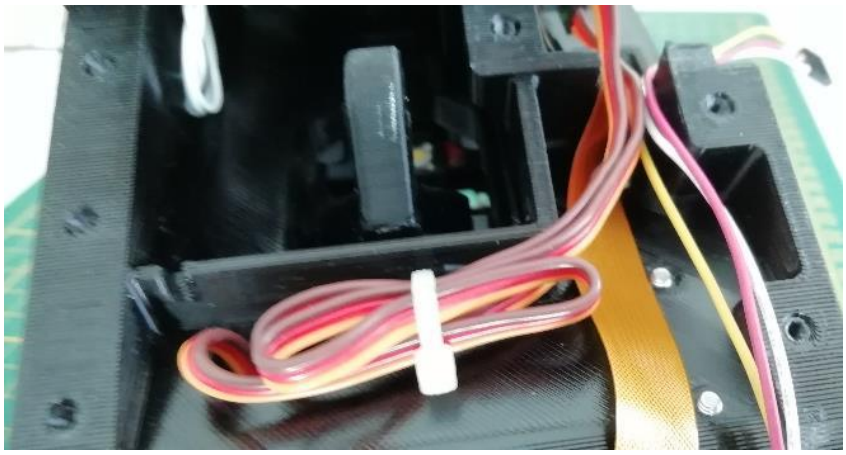
Step17 (Position the cables, and assemble the Baseplate_rear):
6x M3x12mm conical head (4 screws at corners are sufficient)

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

Dress the cables and close the Baseplate_rear.

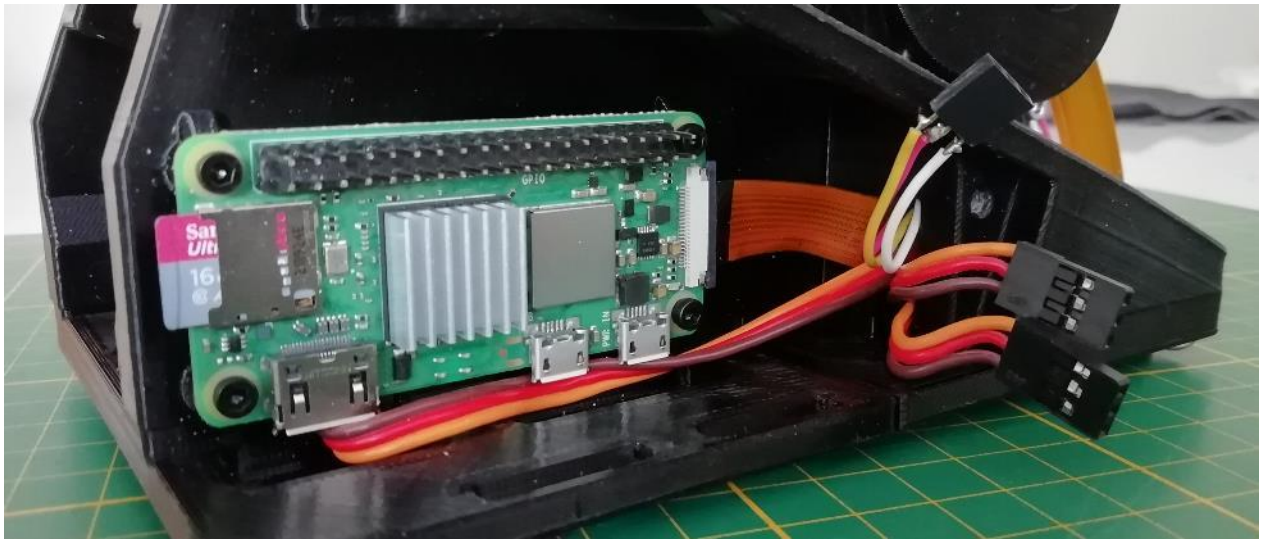






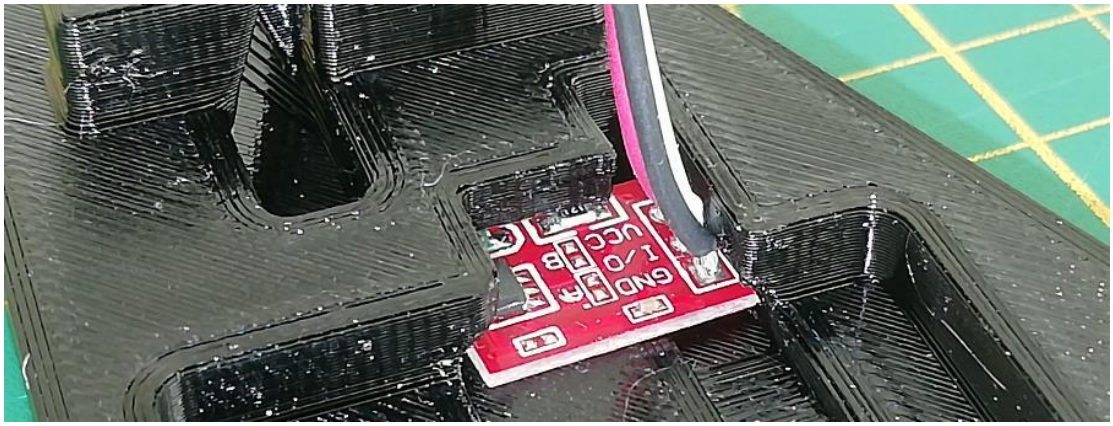
Step17 (Connect the PiCamera flex cable to Raspberry Pi Zero2 board, and fix it to the Structure)

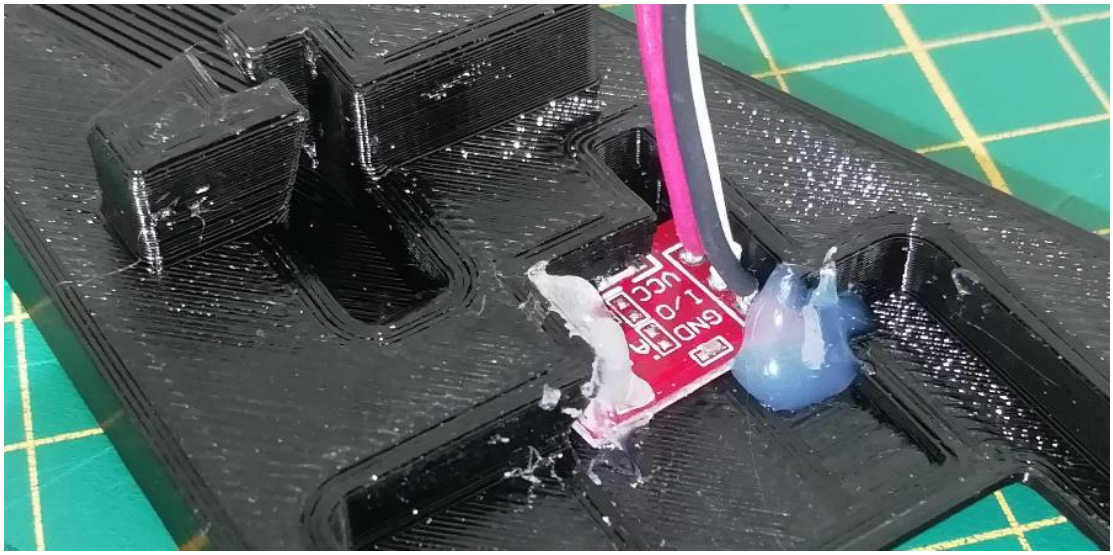
4x M2.5x10mm (or M2.5x8mm) cylindrical head



Step18 (Fix the Touch_sensor to the PCB cover):

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

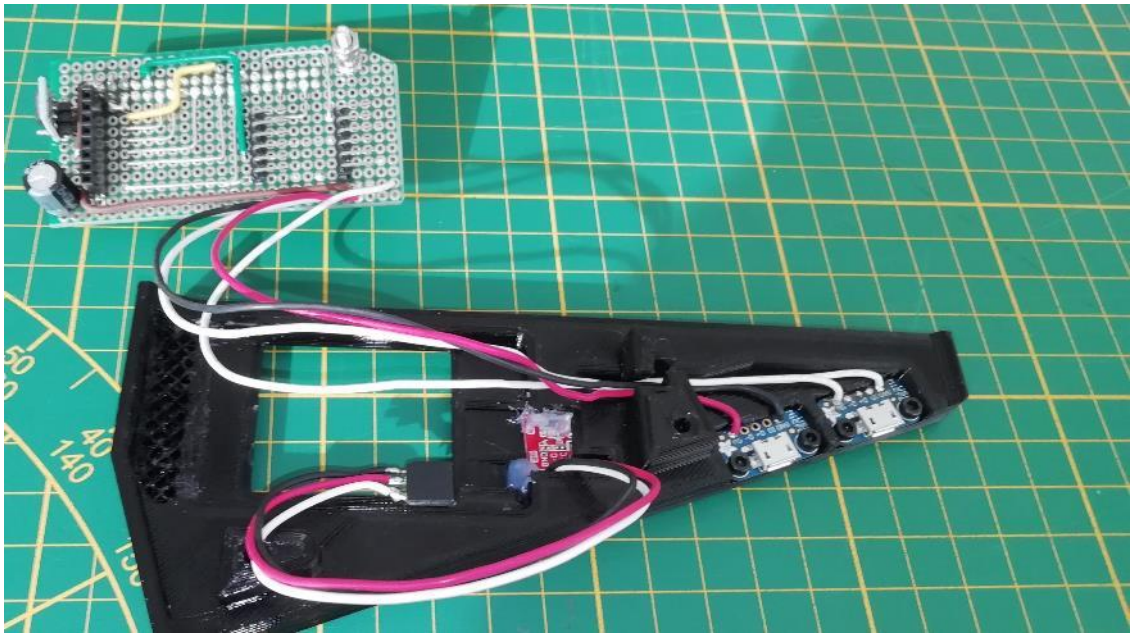




Step19 (Fix the microUSB breakout board to the PCB_cover):

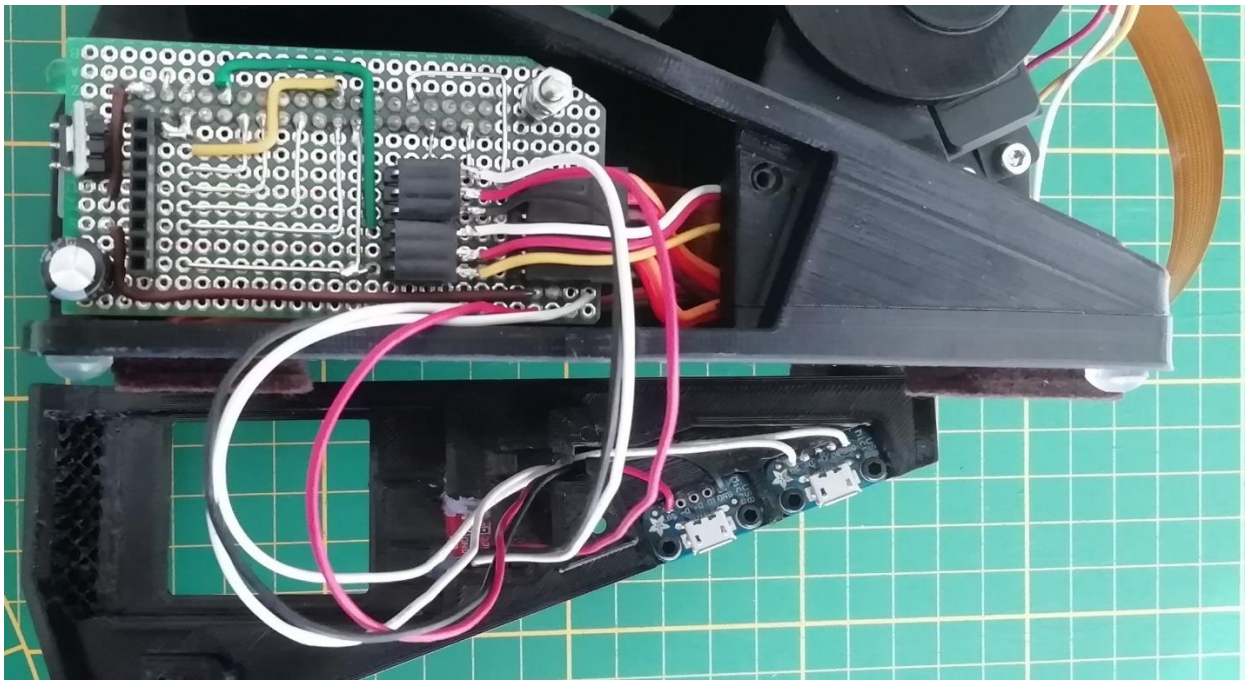
2x M2.5x4mm cylindrical head

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



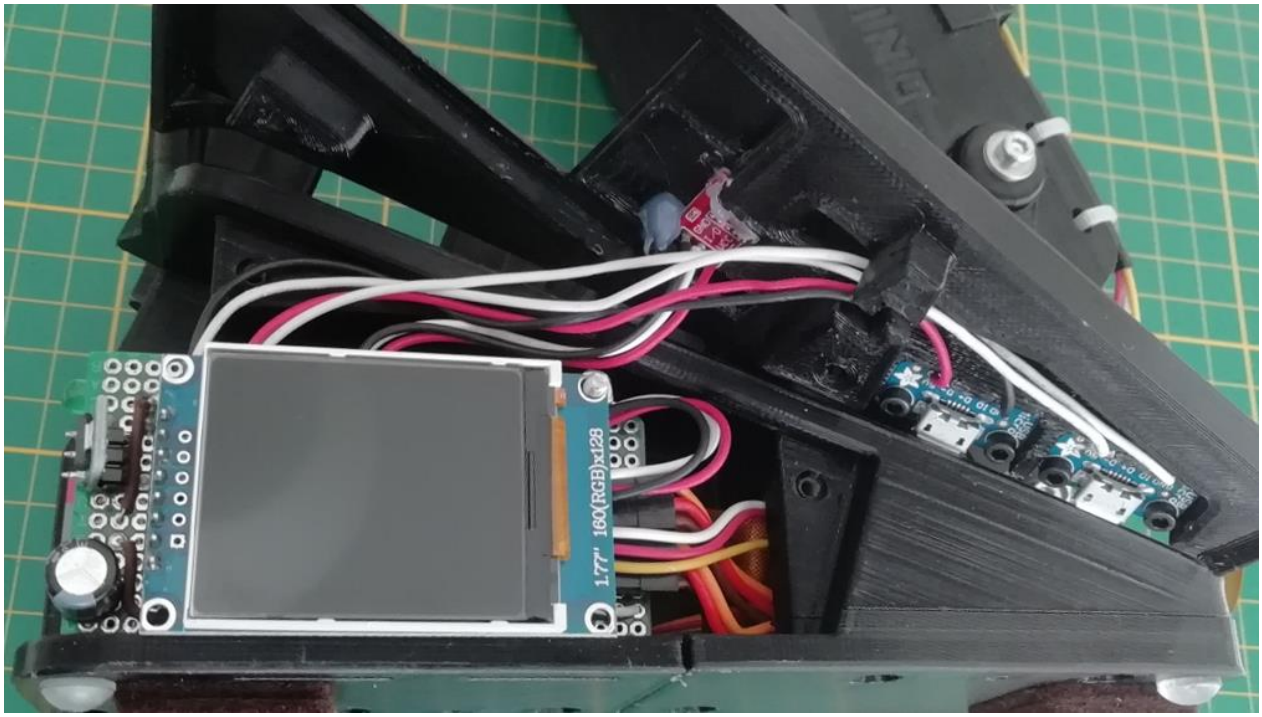
Step20 (Connect the servos, LED breakout board and Touch sensor):

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



Step21 (dress the cable and connect the display):

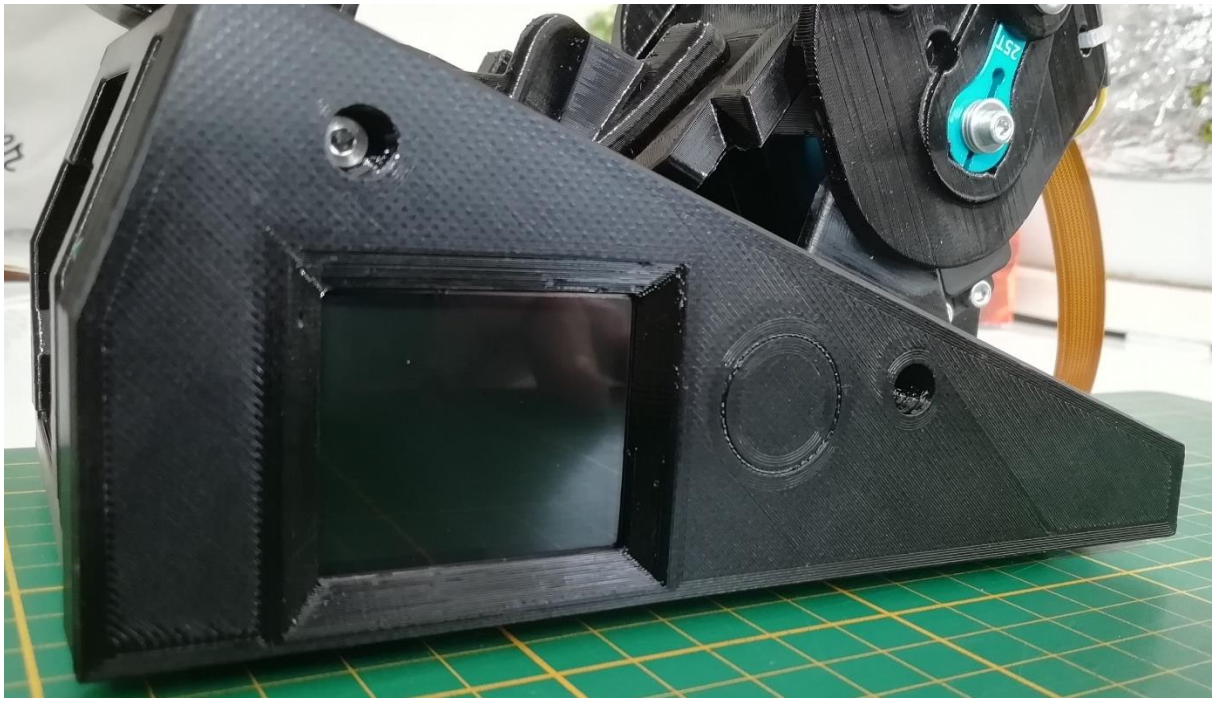
Cables at the microUSB breakout boards have to be well positioned into the groove



Step22 (Assemble the PCB_cover):

2x M3x12mm conical head

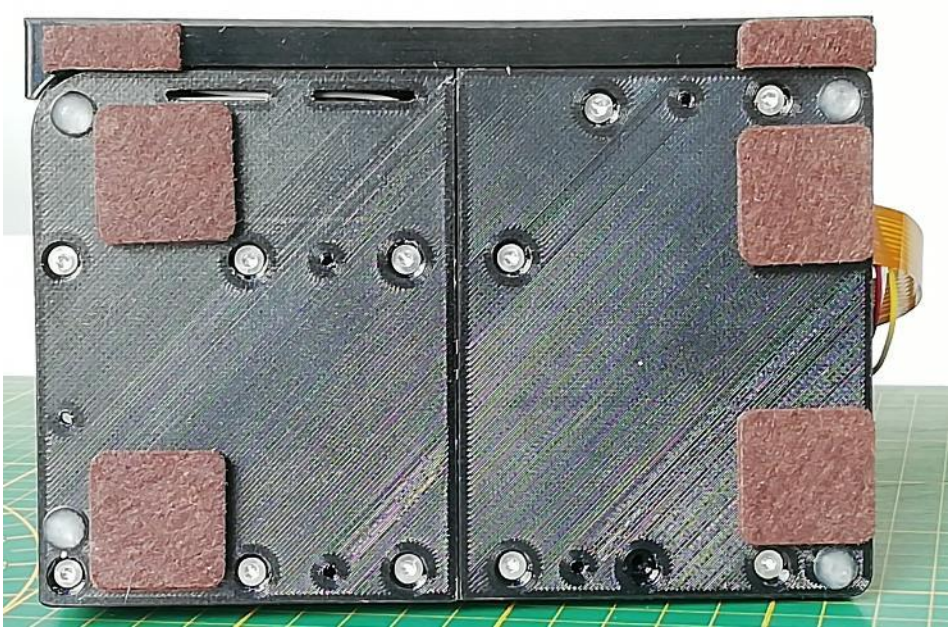
Attention to don't squeeze cables against the Structure



Step23 (Assemble the Baseplate_front to the Structure):

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

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



Step24 (Stick the PiCamera to the board):

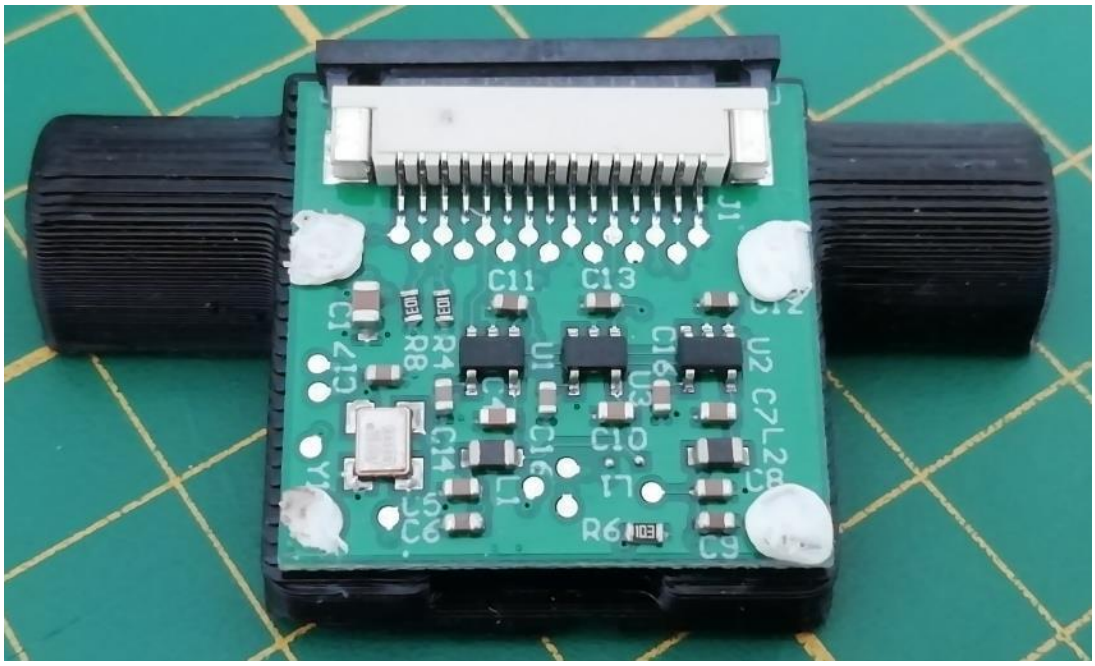
Underneath the picamera there is a little piece of self-adhesive tape.

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

Step25 (Assemble the PiCamera module to the PiCamera holder):

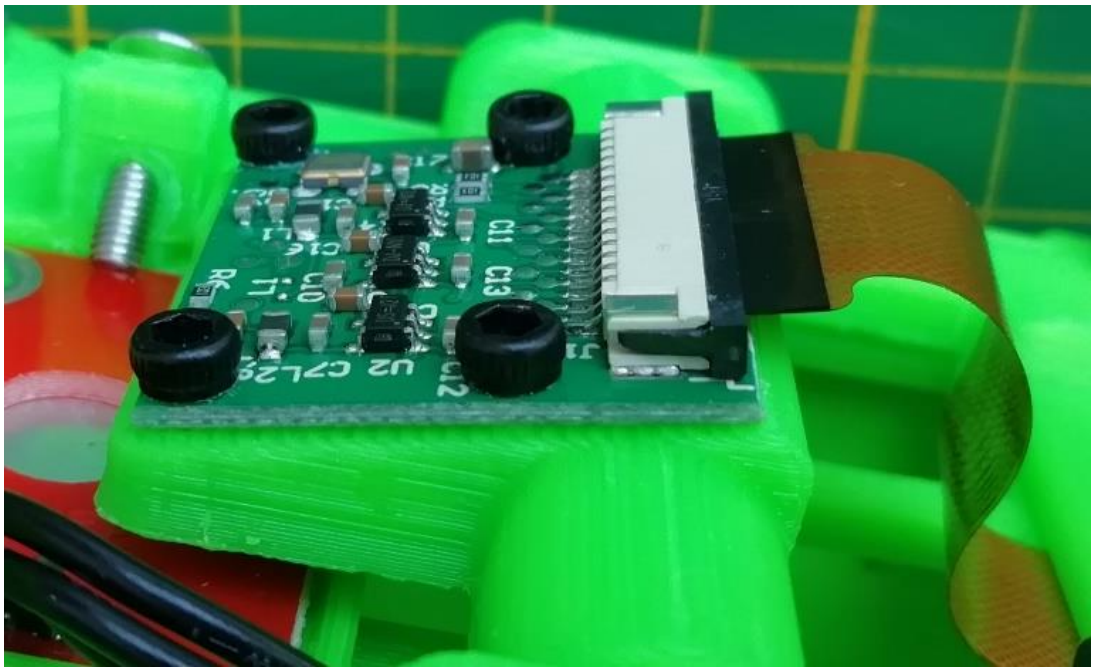
Necessary some little pieces of filament Ø1.75mm.

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



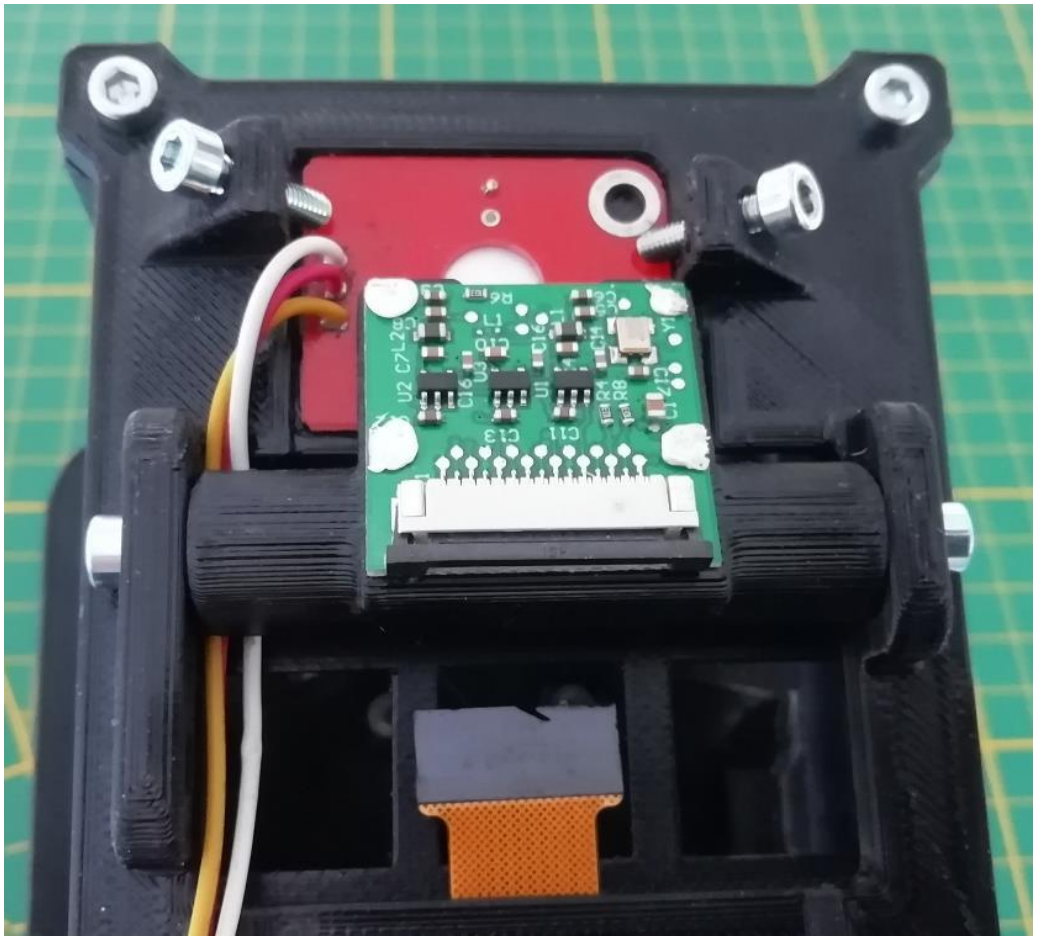
Alternatively, 4x M2x3mm screws can be used or 4x M2.5mm screws can be used.

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



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

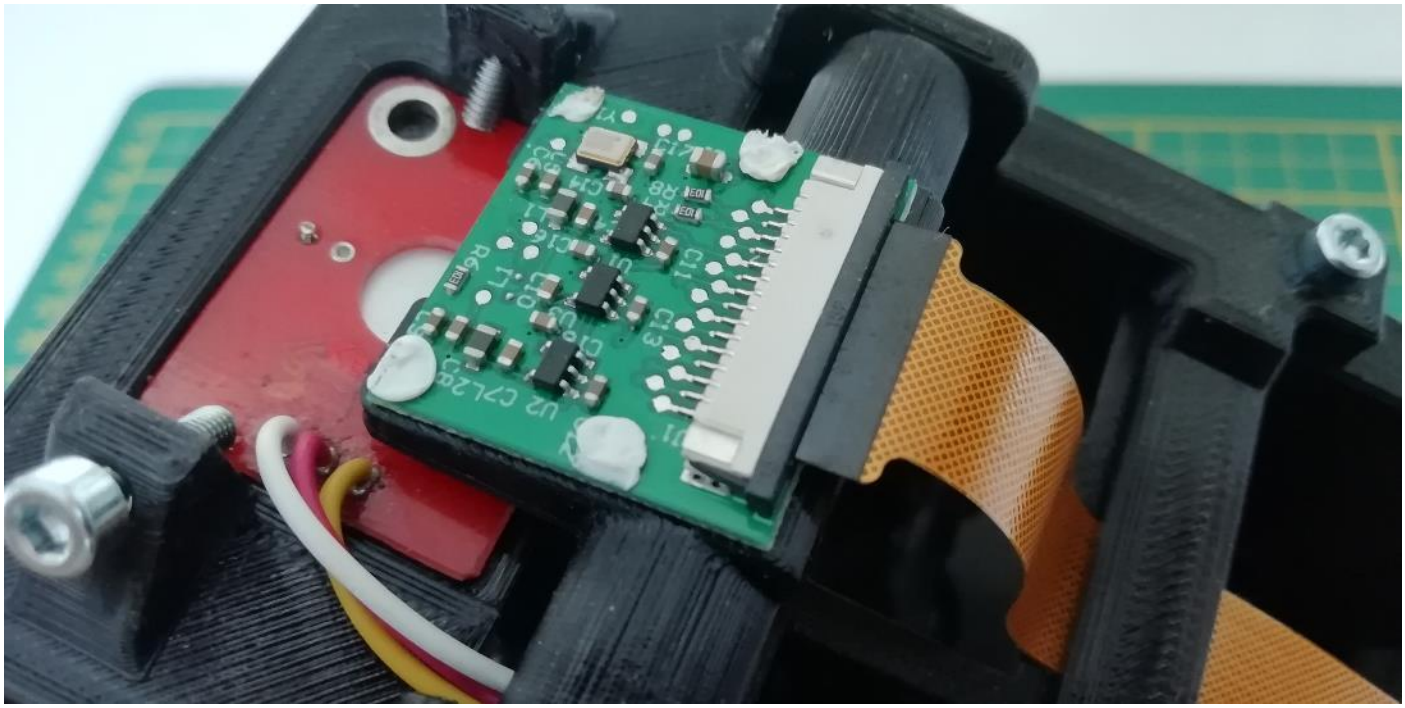
2x M3x12mm cylindrical head



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

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

Step27 (Connect the flex cable to the PiCamera module):



Step28 (Personalize the formal Stop plate):

The Structure has a recess, meant to hold a metal plate working as a Stop touch sensor on the Base version.
On this version, the stop function is available on the PCB_cover, making that recess quite useless and unesthetic.
I made available two alternative stl files of a plate to 3D printed as cover such a recess; One plate is neutral, the second one has ‘Magic CUBE’ words engraved.
Of course, you might consider using the neutral one as baseline to personalize your own Cubotino 😊.

9) Raspberry Pi 3 or 4

Cubotino_Top_version has been designed in late winter 2021, by considering the Raspberry Pi Zero 2 an easy to source and relatively cheap (ca 15$) component ….

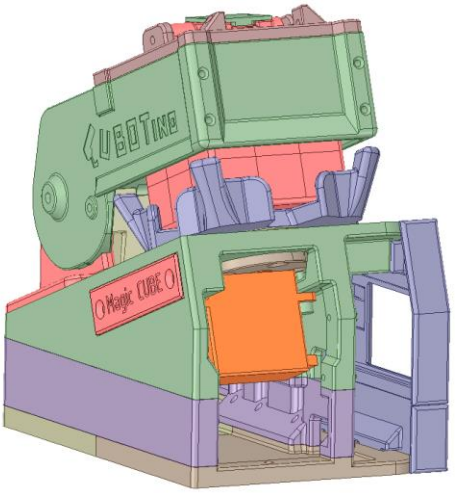
Unfortunately, the market situation has changed: Raspberry Pi Zero 2 boards are scarcely available and are gold priced.

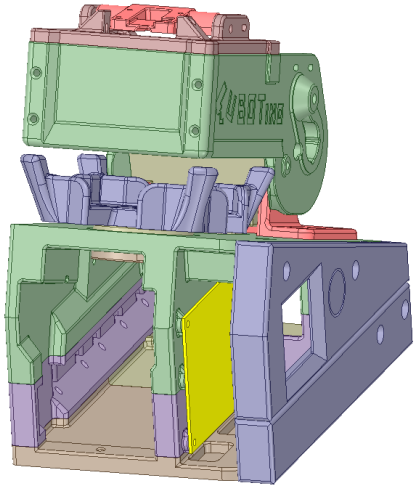
If you have a Raspberry Pi 3 or 4, a possible solution is to increase the robot height by 26mm to embed that larger SBC.
Some notes:
The connections_boards need to be slightly smaller, to prevent interference with some of the raspberry Pi 3 or 4connectors.

The height increment can be gained via 3 extension parts to be 3D printed and screwed to the other parts; This choice allows to shrink the robot in future, in case the smaller boards will be back reasonably priced again.

This approach has been already suggested with Rev. 2.2 on 18/06/2022, yet at that moment I hadn't any "large" Raspberry Pi board available, and I did not realize the interferences with the boards; On that period, I had very little time to work things out, and decided to remove the proposal in less than a week.

Below a few images of how the robot will look like:





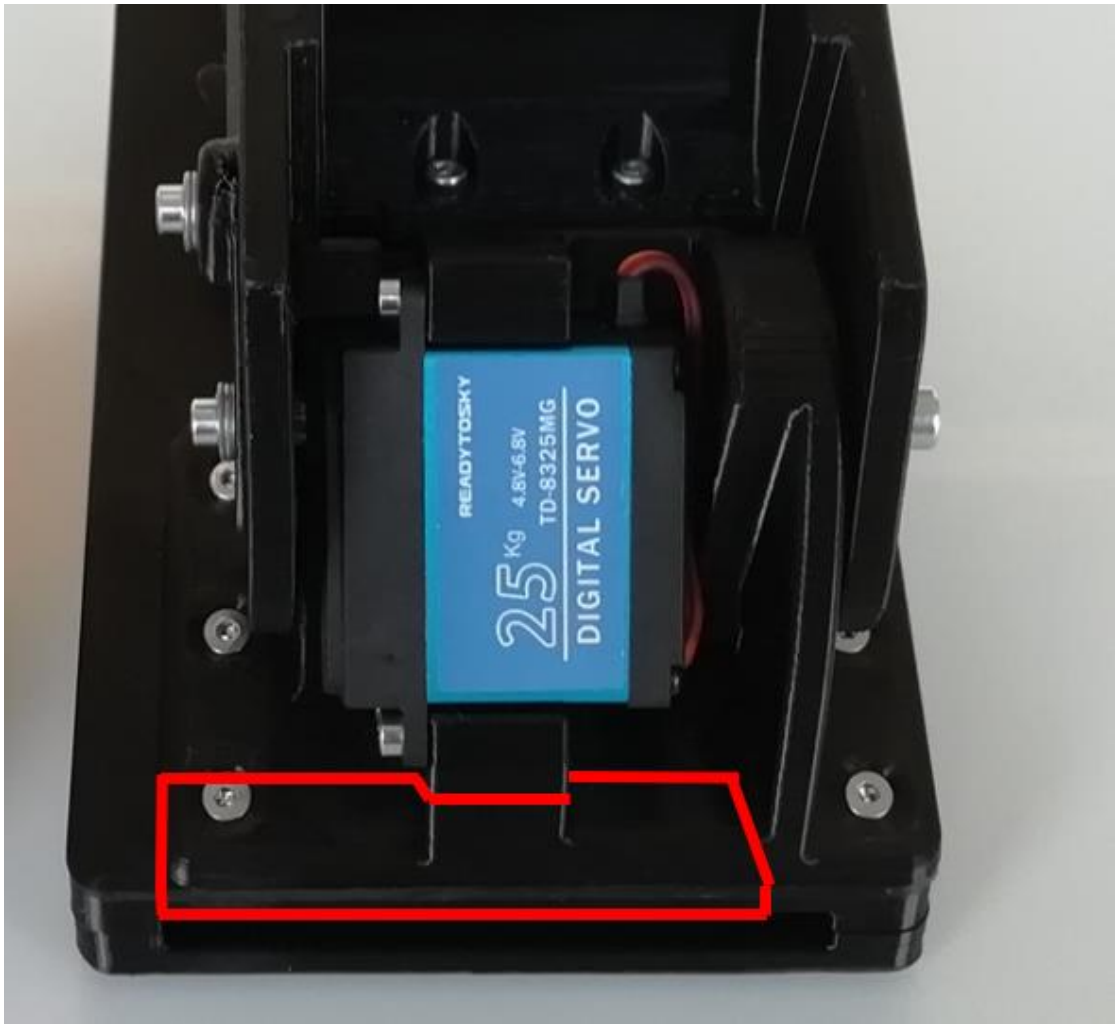
And a picture from nrdlrd Maker:

Raspberry Pi 3 and 4 have “standard” CSI camera, meaning the flex cable differs from the one at supply list for Raspberry Pi Zero (and Zero2).

Raspberry Pi 3 and 4 have the CSI camera port in a less convenient position than Raspberry Pi Zero: This makes the 30cm flex cable just enough …. meaning it might be just not enough.

The obvious option is to buy a longer cable, meaning the commercial 450mm or 500mm

In case you already ***have*** a 30cm cable that is just enough, and you’d like to reduce cable tension, you might consider modifying a couple of parts: Structure and Hinge



This is quite labour intensive if you file/cut the parts, or you revise the files and re-print.

The Hinge currently has 6 fixing screws, considering some screws might be less effective; The removal of one screw is clearly not a problem.
As said, the suggested solution is to buy a longer cable (400 or 450mm), yet this might be a more creative solution.

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

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

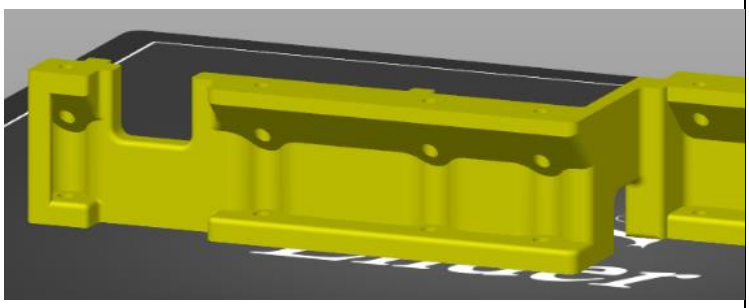
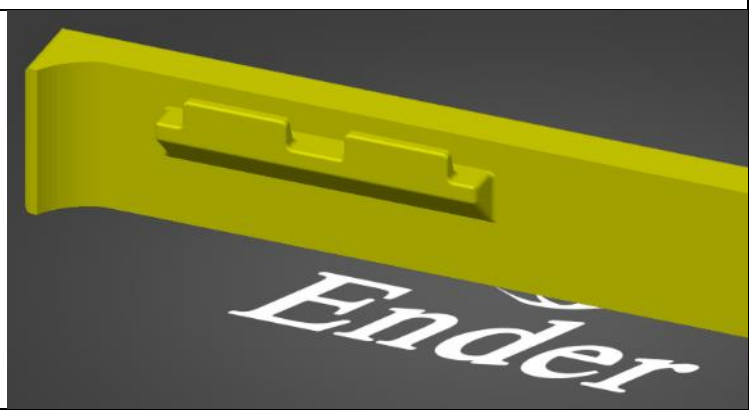
Filament quantity, and printing time …

Notes:

Still valid the notes on chapter 3D printed parts (i.e. no support needed).

Above Filament consumption and printing time are based on a quality printing setting; If these parts are really meant to be temporary in your case, a much lower quality could be considered.

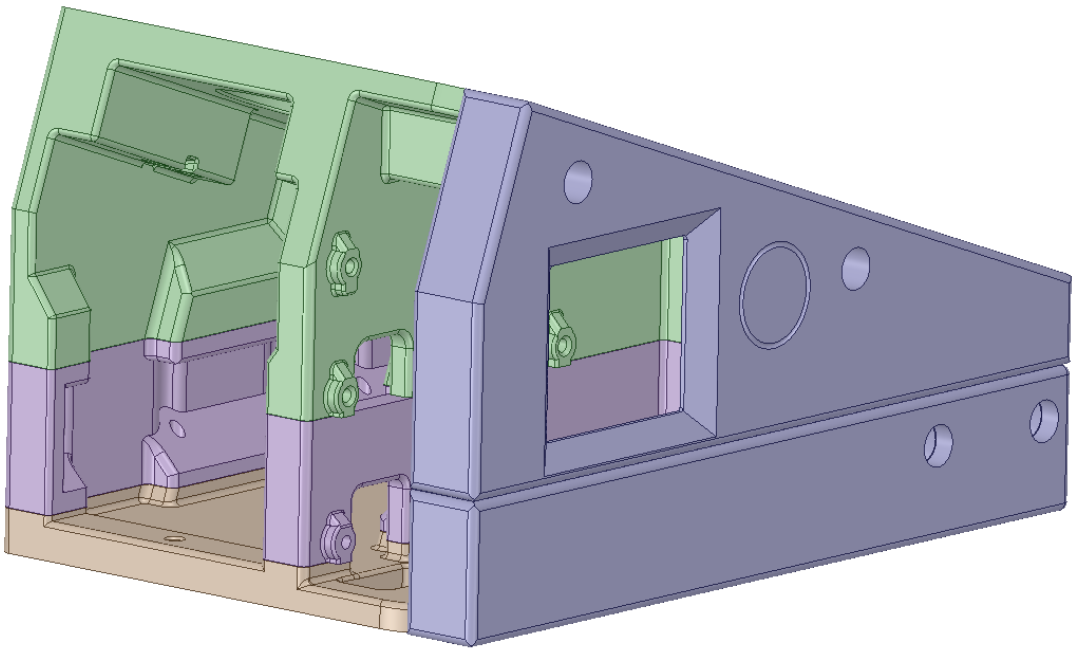
The suggested part orientation for the 3D print is showed on below Table.

Part name	3D print orientation
Extension_left	
Extension_middle	
Extension_right	

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

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

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



Assembly:

The three additional parts are connected by screws.

Use M3x12mm cylindrical, to connect the Extension_left and the Extension_middle toward the Structure.

Note: In case you don't have Allen keys with "spherical" head, to reach the screw under an angle, it might be necessary to use screws with conical head; M3 screws with conical head uses smaller Allen key, that can be used through the holes straight in front.

Use M3x12mm conical head, to connect the Extension_right toward the Extension_middle

Raspberry Pi 3b or 4b must be assembled by keeping the GPIO connector on top, like for the Raspberry Pi Zero 2

Do not tight much the screws of the board, as there might be small electronic parts of Rpi 3b or Rpi 4b in contact with the second unused screw seat for Rpi Zero 2.

The bottom_servo cable must be dressed through the recess between the Extension_middle and the Base_front.

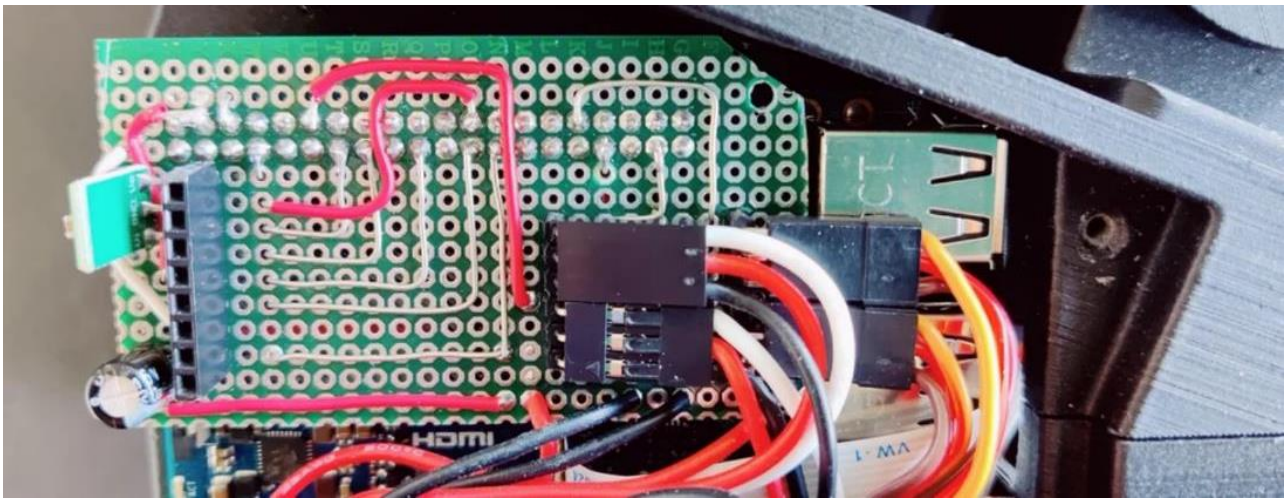
10) The Extension_left and the Extension_middle parts should be assembled after Step4 (of Assembly details chapter), therefore after assembling the bottom servo.

The Extension_right has to be assembled before assembling the Base_front and before assembling the PCB_cover_display.

-
- Adapting the connections**_board, to fit Raspberry Pi 3 (or 4) component layouts**
-
- If you make the connections_board, out of a proto-board, the important changes are:
- limit the protrusion of the proto-board, to the right side, to max 4 / 4.5 holes from the last used pin of the Raspberry Pi GPIO connector.

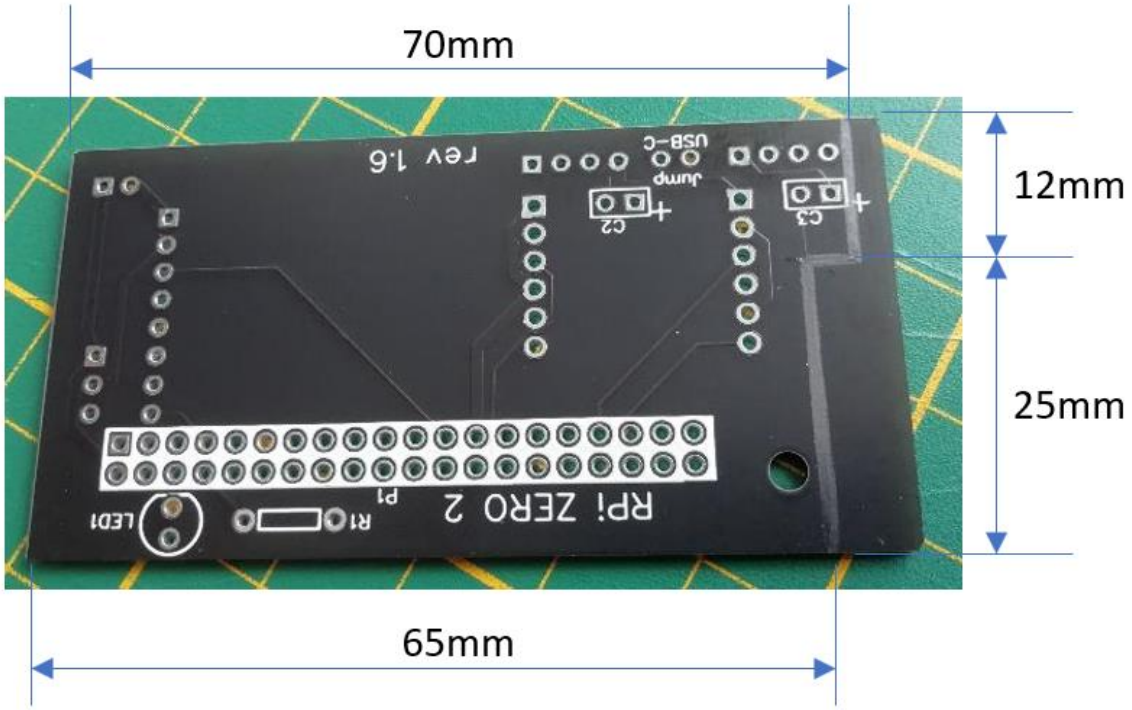
Solder the C3 capacitor directly to pins 1 and 2 of the (H6) header for the bottom servo.

Below an image from Maker nrdlrd, who had used a Rapberry Pi 3:

29. 

30.
31.
32.
33. The V1.6 board can also be used, in combination with Raspberry Pi 3 or 4, by cutting out part of the right side.
34. In case of Raspberry Pi 3 the board can have a straight cut, to reduce the board length from 74mm to 70mm.

35. In case of Raspberry Pi 4 the board requires a slightly more complex cut:



36.

37.

38.
39. Notes
40. Solder the C3 capacitor directly to pins 1 and 2 of the (H6) header for the bottom servo
41. With reference to the PiCamera flexible cable, I'm not fully sure the 30cm version to be sufficient
42. The M3x16mm screw, meant to keep the display parallel to the Connections_board and the PCB_cover, need to be fixed to the display hole instead of the connections_board hole.
43. Verify the absence of short circuits between the connections_board and the Raspberry Pi 3 or 4 connectors, in particular for the C3 positive solder pad
44.

46. Tuning
47.
48. As anticipated, be prepared the robot won't magically work right after assembling it: Tuning is needed!
49. This has to do with differences between each robot, in particular:
50. servos
51. arm positioning to the servo
52. cube dimensions

print quality

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

53. In case things are getting too difficult, check out the Instructables chats (there are chances other people have asked the same questions); Differently, consider dropping specific questions to that chat.
54.
55.
56. General:
57. There are parameters that are expected to be differently tuned on each robot.
58. These parameters are grouped into two (json) text files: See Parameters and settings chapters.
59. Some of those parameters are quite likely to require tuning, because each robot will slightly differ from others:
60. Servo angles, and servo timers
61. Frame Cropping, as Top_cover angle dependent and PiCamera assembly angle dependent
62. Other parameters in the json files, aren't so likely to be tuned, but it might be something you'd like play with 😊.
63.
64.
65. Setting servos angles:
66. The servos at supplies list have 180° of rotation, that is more than sufficient for the lifter and Top_cover angle of this robot, and it should be right sufficient to the Cube_holder; I don't suggest buying 270° servo as this will affect the angle resolution.
67. Apart from tolerances between different servos, one variation source is the connections between the servo arms, and the servo's outlet gear, having many possible positions (I believe there are 25 teeth).
68. This means the reference angles set on Cubotino_T_servo_settings.txt working fine on my robot, are not necessarily the best choice on other systems: **These parameters must be tuned on each system!**
69.
70. Servos are controlled on angle, via a PWM signal (https://en.wikipedia.org/wiki/Servo_control)
71.
72. The servos at supplies list, accept a Pulse Width signal from 1ms to 2ms, wherein 1.5ms is the mid angle.
73. It is anyhow possible to use servos with different pulse width range, in that case adjust the min_pulse_width and max_pulse_width parameters for the related servo, see Parameters and setting for more info.
74.
75. The Cubotino_T_servos.py uses gpiozero library to manage the servo PWM.
76. This library uses a target servo position/angle with a parameter ranging from -1 to 1 (0 is the mid angle, value is a float), based on the Pulse Width range (i.e. from 1 to 2 ms, or from 500 to 2500us).
77.

79.
80. Detailed info on servo management:
81. On Parameters and settings chapter are listed the involved variables and the default values.

The gpiozero library is used to control the servos, with a (float) parameter ranging from -1 to 1.

The float value entered on the '–set' argument (see Servos test and set to mid position chapter), represents a normalized rotation; The applied rotation is based on the Pulse Width range. Be considered your servos might have a different Pulse Width range…

Value **-1** is the max CCW servo rotation; The library sends the minimum Pulse Width to the servo.

11) Value 1 is the max CW servo rotation; The library sends the maximum Pulse Width to the servo.

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.

Changing from a smaller value to a larger one it results to a CW rotation of the servo outlet.

On servos with a Pulse Width ranging from 1 to 2ms, every 0.02 step in the "—set" argument determines a rotation of 1.8 degrees: -0.02 rotates the servo outlet of 1.8deg CCW, while 0.02 rotates the servo outlet of 1.8deg CW.

It is convenient checking the servo rotation range before the assembly, therefore prior to have mechanical constrains on the servo rotation

In case your servos don't make 180 degrees rotation, when the '—set' parameter is changed from -1 to 1 (or the other way around), it might be the case those servos have a Pulse Width ranging from 0.5ms to 2.5ms (default range considered is from 1 to 2ms).

In this case it is necessary to change the minimum and maximum pulse width parameter at the Cubotino_T_servo_settings.txt file (t_min_pulse_with, t_max_pulse_with, b_min_pulse_with, b_max_pulse_with) with values that best fit your servos.
Save the text file and re-launch the Cubotino_T_servo.py (pulse width parameters are uploaded when this script is started)
Re-check the servo angle range: If 180 degrees are now covered set the servos to their mid position.
In case the servo for the Cube_holder makes less than 180° rotation, despite you've tried to enlarge the Pulse Width range, it is necessary to increase the rotation range to slightly more than 180° in order to get the robot working properly.

There are tutorials in internet on how to increase the rotation angle, by adding some resistors in series with the servo potentiometer (servo must be opened for this eventual change).
2k2 Ω resistors were used by Jonesee (reference https://www.thingiverse.com/make:1034575).
Andy (search for G7UHN at https://www.instructables.com/CUBOTino-a-Small-Simple-3D-Printed-Inexpensive-Rub/#discuss) has reported 'my servos had only about 90° range of motion out of the box (despite being advertised as 180° servos) so I had to modify them, adding 3k resistors to both ends of the internal (5k) potentiometer'.

Top_cover (t_**servo) angles**:

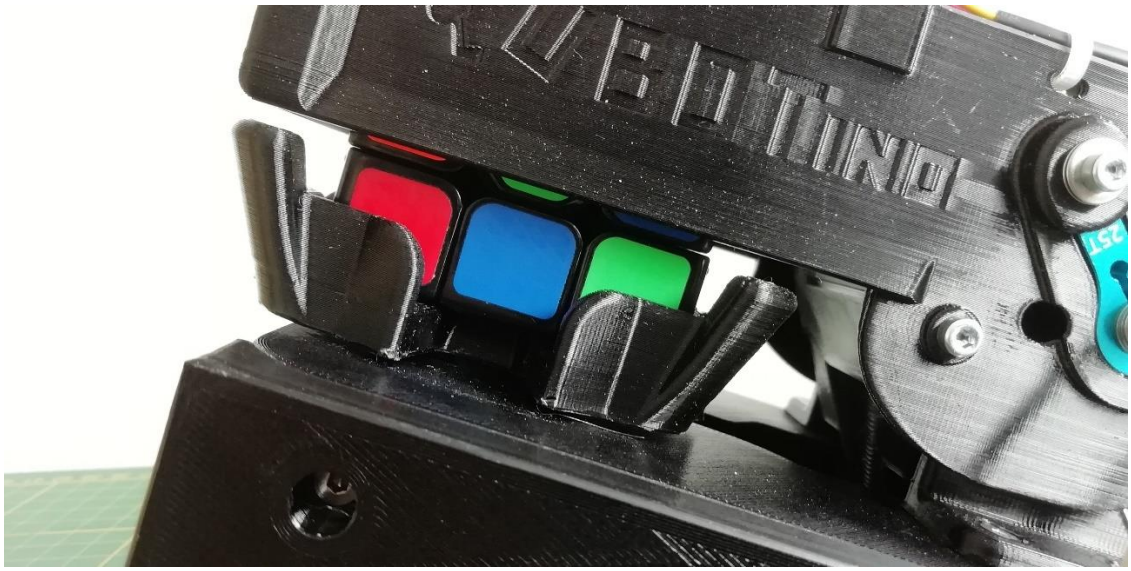
- Working angles for the servos, are set in a (json) text file: Cubotino_T_servo_settings.txt
- There are 5 defined angles (most of time mentioned as positions):
- Close: position to constrain the top and mid cube layers
- Rel: position to release tension, from cube, at Close position

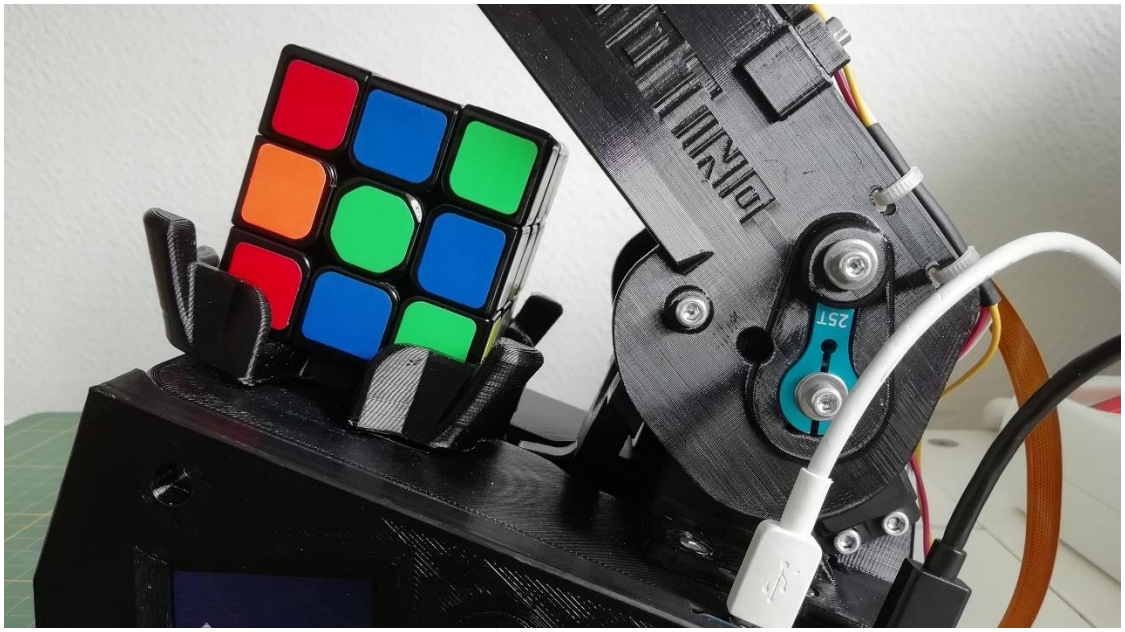
  Open: position without interferences with the cube and Cube_holder

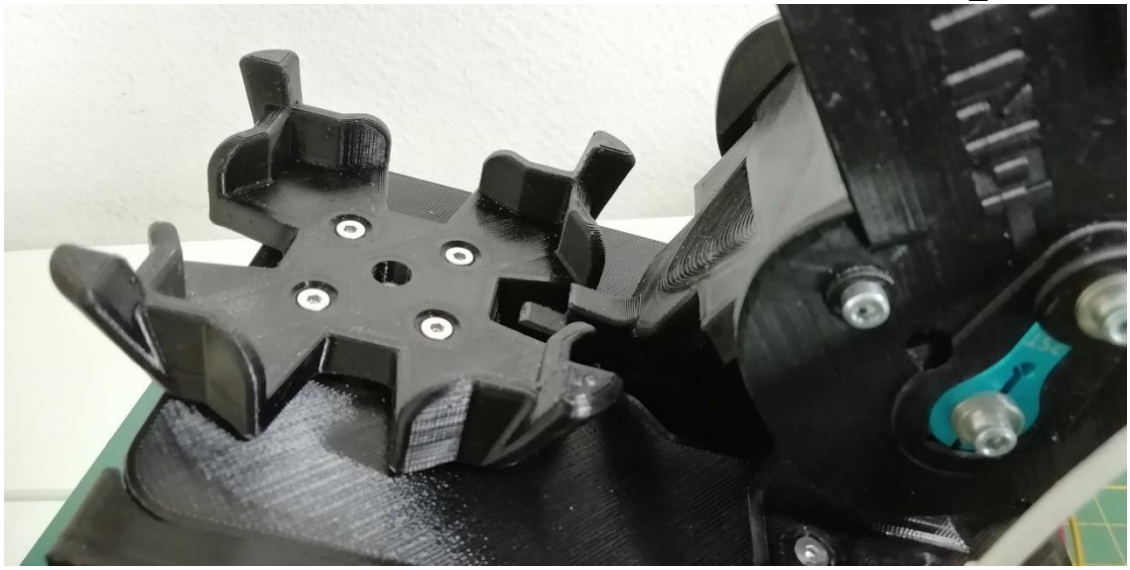
  Read: position for camera reading, with the Lifter almost touching the cube

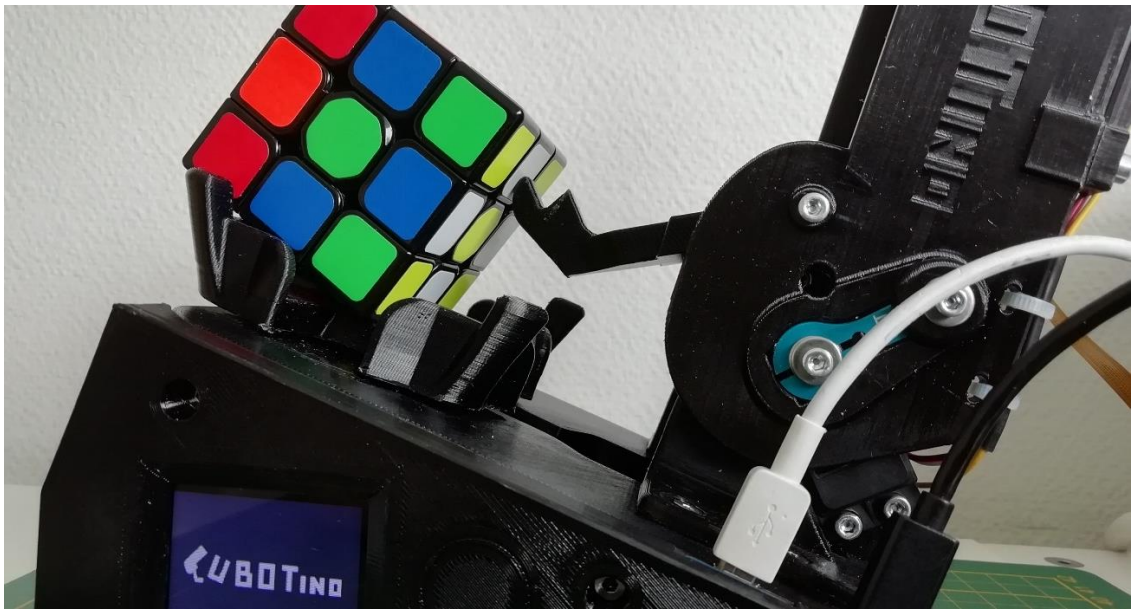
(and unfortunately constraining the Cube_holder)

Flip: position for the Lifter to flip the cube (about 2 cube layers height)








CUBOTino

Cube_holder (b_servo) angles:

There are 3 (+4) defined angles (most of time mentioned as positions):

CCW: position to spin or rotate the Cube_holder >90° CCW from Home

(Direction according to the motor point of view)

CW: position to spin or rotate the Cube_holder >90° CW from Home

(Direction according to the motor point of view)

Home: mid position between CCW and CW

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

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

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

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

The Home position has to be well centered.


FRONT

Fine **tuning servos angles (changed on 25th July 2022):**

set the Servos to the mid angle (see Servos test and set to mid position chapter)

assemble the robot

enter the cube folder (cd *cubotino/src*) and activate the venv (*source .virtualenvs/bin/activate*); Attention to the dot in front of virtualenvs

run the script *python Cubotino_T_servos.py –tune True* ; Attention to the space in between '-'

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

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

After the 'Enter command:' type the below commands to test the servos positions:

t_servo = t_servo_close

t_servo = t_servo_open

t_servo = t_servo_read

t_servo = t_servo_flip

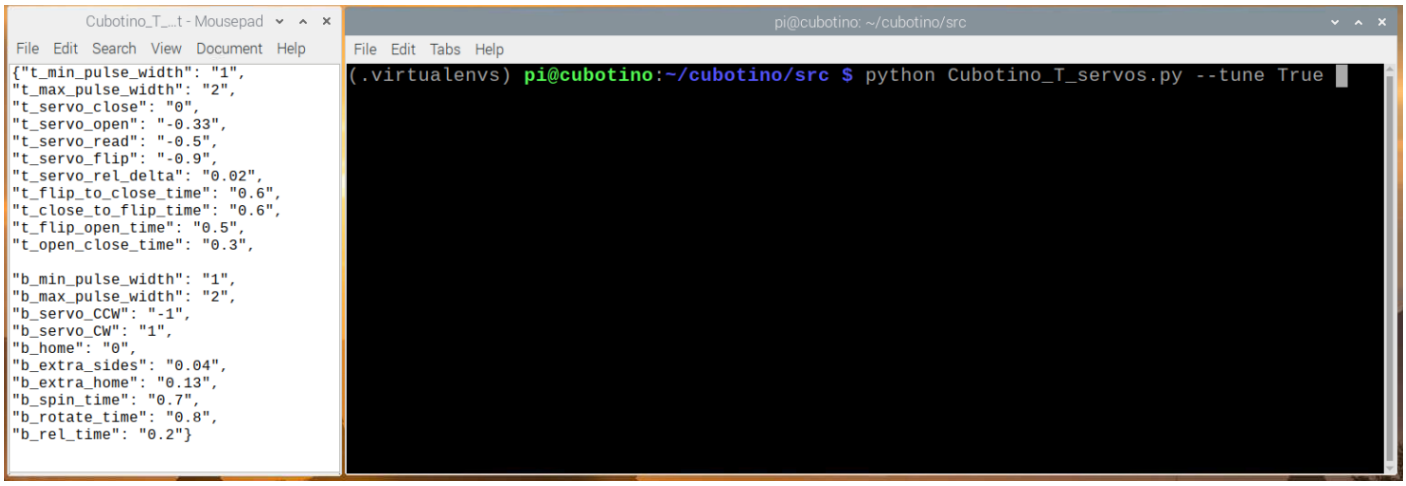
b_servo = b_home

b_servo = b_servo_CCW

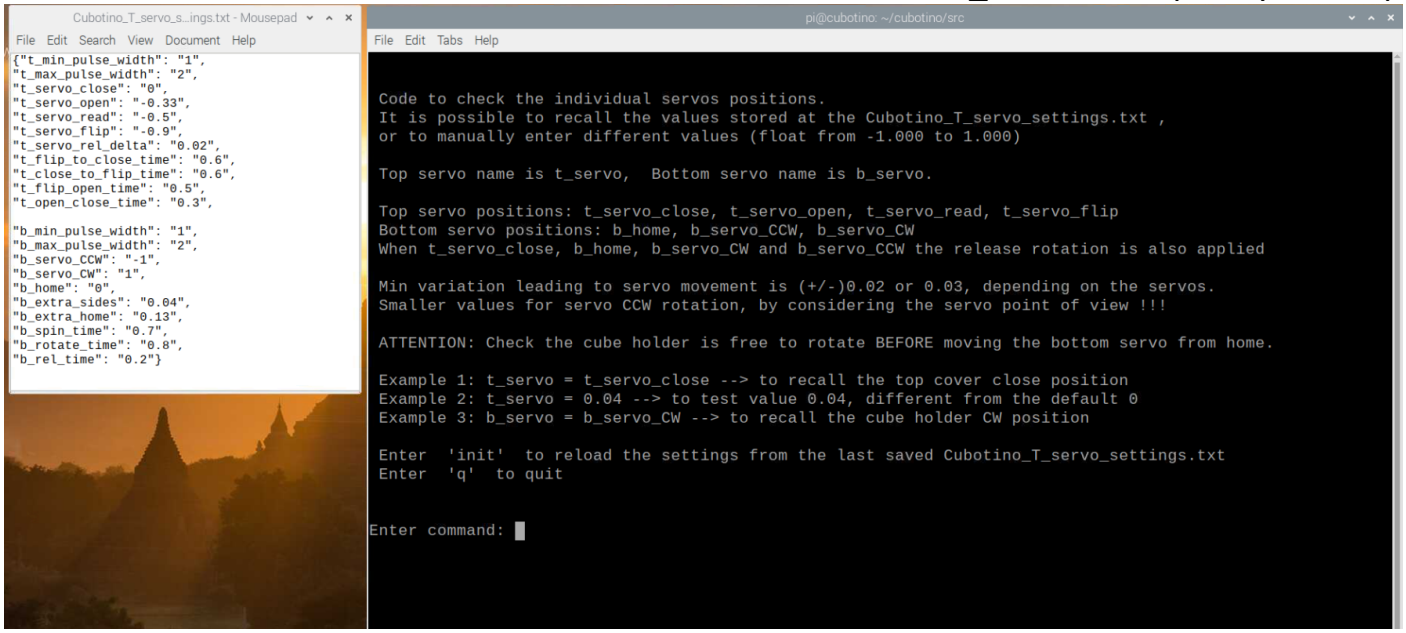
b_servo = b_servo_CW

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.

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



If you type init , instead of a command after the 'Enter command:', the python script reloads the new settings from Cubotino_T_servo_settings.txt, so to verify if everything goes well.



run the script *python Cubotino_T_servos.py* ,without any argument, to test the robot manoeuvring the cube like during a solving process; Take a close look to check if the cube handling is ok.

If the cube layers don't align well, if is suggested to apply some stickers on the cube holder: When the cube holder is home place an 'F' on the cube holder front side, 'L' on the cube holder left side and 'R' on the cube holder right side. Take a movie while the robot manoeuvres a cube and watch it back to see in which position the misalignment is generated.

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 to try 0.02 (0 + 0.02) by typing *'t_servo = 0.02'.*

In case the Top_cover is too close to the cube then the servo position requires to decrease the CW position (CW and CCW are frm the servo point of view).
To decrese 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 to try -0.04 (0 – 0.04) by typing
't_servo = - 04 '.

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

Timers for servos:

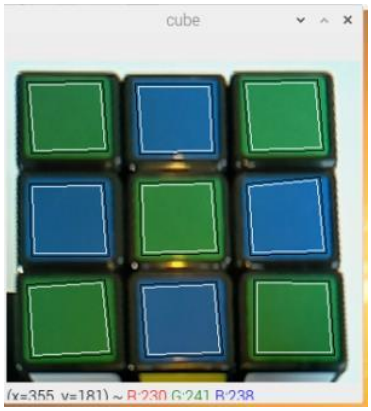
Servos don't provide feedback when they have completed the requested angular rotation; It's necessary to set appropriate waiting time in the script, to allow sufficient time for the servo to complete the action.
It will be convenient to use larger delays at the beginning, and progressively reduce them once the servo angles are adjusted to your system: The default values I've set for the default should be more than sufficient to allow the servos intended rotations.
Once your robot runs smoothly, and in case you' like to reduce the solving time, then you might start reducing those timers. As reference you can check on "Parameters and settings" for the values I'm using on my robot.
Timers for the servos, are set in a (json) text file: Cubotino_T_servo_settings.txt

Frame cropping:

Cropping parameters are set in a (json) text file: Cubotino_T_ settings.txt
PiCamera position, and its FoV, are likely to read both top and back cube faces.
Cubotino_T.py file is delivered with no cropping effect (variables set to zero): this to help the camera assembly angle to be set to get the cube top face centered: First picture below as example.

Image cropping has to 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.

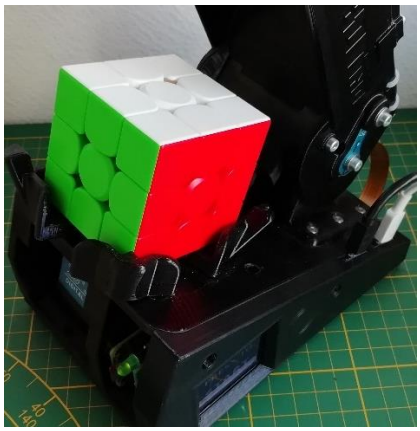
Below how the cropped and warped image should look like: Just keep a little part visible of the back face:


cube

Frameless cube:

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





The cube facelets detection algorithm has been initially developed for the classic cubes, those having the black frame around the facelets.
Starting from 6th August 2022, it is also possible to use the frameless type of cubes
At Cubotino_T_settings.txt there is a “frameless_cube” parameter, with below options:
‘false’ for the classic cube type (default value)

‘true’ for the frameless type

‘auto’ for both types,

The ‘auto’ mode is computationally more demanding, therefore it will take slightly longer (about 1 to two seconds more for the six cube faces).
In the case the Cubotino_T_settings.txt doesn’t have the key “frameless_cube”, because you are re-using an old setting files, then the new Cubotino_T.py file version will simply consider the classic cube type.

You might argue on the reason ‘auto’ is not the default choice for the cube facelets detection method.
The reason is that the used strategy for the frameless cube is less robust.
When facelets_cube is set ‘true’ or ‘auto’, as soon as there are at least 5 detected facelets, without an empty row or column, the remaining facelets will be estimated on their position.

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

Examples of estimated facelets:

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

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

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

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

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

Display initialization and test:

For installation up to 6th August 2022, the display was initialized twice; This wasn't pleasant to be seen, and potentially leading to hardware race by the code
Starting from 6th August 2022, thanks to Yannick solution, the display is initialized once, with better display management.
The new solution requires one more file: Cubotino_T_display.py to the project.
The new file can be used to test the display after its connection:
enter the cube folder (cd ~/cubotino/src)

activate the venv (*source .virtualenvs/bin/activate*); Attention to the dot in front of virtualenvs

run the script *python Cubotino_T_display* ; Attention to the space in between '-'

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

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

PiCamera focus

The V1.3 PiCamera is delivered with fixed focus.
On my first project with PiCamera I did adjust the camera focus, by following this tutorial
https://projects.raspberrypi.org/en/projects/infrared-bird-box/6
It hadn't been easy, yet I managed.
After correcting the focus, I did not glue the lens, as there still was quite some friction, preventing the lens from getting loose.

When I tried to repeat this operation on a second PiCamera I have almost destroyed it.
Both my Cubotinos have PiCamera "as received", one Cubotino has okish focus while the second one is out of focus.
In both cases the robot work properly, the only difference is the sharpness on the images the robot saves.
In case of non-satisfactory results, and if you are good with your hands, you might decide to try … on your own risk.

Parameters and settings

Parameters that are more likely to differ on each system, are into two json files:
Cubotino_T_settings.txt and Cubotino_T_servo_settings.txt

In order to provide a reference, the below json files capture the settings used on my robot:
Cubotino_T_settings_AF.txt and Cubotino_T_servo_settings_AF.txt

On below tables are listed these parameters, with the proposed value to start the tuning, the value that work on my Cubotino, and some information.
Highlighted the default settings that differ from mine

Cubotino_T_settings.**txt (and Cubotino_T_settings_AF.txt), part 1**

Parameter (dict **key)**	Default **value**	AF **value**	**Data type**	**Info**
frameless_cube	false	auto	string	Set the facelets edge detection according to the cube. Options are: 'false', 'true' and 'auto'. If the parameter key is missed (i.e. old setting files) a classic cube is considered, meaning frameless_cube = false. frameless_cube = true should be used when small logos on the cube.
disp_width	130	132	Int	Display width (in pixels); At troubleshooting for Display the explanation to make easy to adjust this parameter this parameter
disp_heigh	160	162	Int	Display height (in pixels); At troubleshooting for Display the explanation to make easy to adjust this parameter this parameter
disp_offsetL	0	-2	Int	Display offset on width **Left** (in pixels); At troubleshooting for Display the explanation to make easy to adjust this parameter this parameter
disp_offsetT	0	-2	int	Display offset on height **Top** (in pixels); At troubleshooting for Display the explanation to make easy to adjust this parameter this parameter

Cubotino_T_settings.txt **(and Cubotino_T_settings_AF.txt),** part 2

Parameter (dict **key)**	Default **value**	AF **value**	**Data type**	**Info**
x_l	0	60	Int	Image cropping at the left, before warping (in pixels). This is meant to removes a slice of the image at the **left** side, external to the cube image. around the bot, and it will increase speed.
x_r	0	80	Int	Image cropping at the right, before warping (in pixels). This is meant to removes a slice of the image at the **right** side, external to the cube image. around the bot, and it will increase speed.
y_u	0	0	Int	Image cropping at the top, before warping (in pixels). This is meant to removes a slice of the image at the **upper** side, external to the cube image. around the bot, and it will increase speed.
y_b	0	110	int	Image cropping at the bottom, before warping (in pixels). This is meant to removes a slice of the image at the **bottom** side, external to the cube image. around the bot, and it will increase speed.
warp_fraction	7	7	float	Image warping index. This parameter is used to alter the perspective from the cube face images. Smaller values increase the effect, meaning it applies a larger variation to the camera image.
77				Image cropping index, that crops the right side of the image after

Cubotino_T_settings.txt **(and Cubotino_T_settings_AF.txt), part 3**

sv_max_moves	20	20	int	Max number of moves requested to the Kociemba solver. When the solver finds a solution matching this movement quantity, that solution is returned even before the timeout expiration (sv_max_time).
sv_max_time	2	2	float	Timeout, in seconds, for the Kociemba solver. The best solution found within the timeout is returned at timeout end, even if it doesn't match the desired max quantity of moves.
collage_w	1024	1024	int	Image width for the unfolded cube file. This parameter determines the image collage realization, and it makes possible to save all images with the same size.
marg_coef	0.1	0.06	float	Defines the margin around the cube faces images. This margin is used to cut the six cube faces with some margin around, for the unfolded cube collage. The margin is calculated by multiplying the detected cube diagonal in pixels to this coefficient. The larger the value the more pixels margin around the cube 0.1 means the margin is 10% of the cube diagonal (calculated on the detected facelets contours).
cam_led_bright	0.1	0.1	float	PWM for the 3W led at Top_cover. Range from 0 (no PWM) to 1 (PWM=100%). At the Cubotino_T_servos.py the max value really delivered to the LED is 30%, therefore values > 0.3 are useless.
detect_timeout	40	40	int	Timeout, in second, for the cube status detection. If the six cube faces aren't detected within this time, the cycle is terminated with a timeout message on the display.

Parameters related to the servos;

Notes:

‘t_’ refers to Top_servo while ‘b_’ refers to Bottom_servo

“Angles’ are in gpiozero range for the Servo class (range from -1 to 1, with 0 as mid angle)

Time is in seconds

Cubotino**_T_servo_settings.txt (and Cubotino_T_servo_settings_AF.txt):**

Parameter (dict **key)**	Default **value**	AF **value**	**Data type**	**Info**
t_min_pulse_width	1	1	float	Min pulse width, in ms of the used top servo. Most of the servos accepts a slightly extended value (<1)
t_max_pulse_width	2	2	float	Max pulse width, in ms, of the used top servo. Most of the servo accepts a slightly extended value (>2)
t_servo_close	0	0	float	“Angle” for Top_cover to constrain the top and mid cube layers
t_servo_open	-0.33	-0.33	float	“Angle” for Top_cover not constraining the cube and Cube_holder
t_servo_read	-0.5	-0.5	float	“Angle” for Top_cover for PiCamera reading. Lifter almost touching the bottom cube face

Note: On Cubotino_T.py and Cubotino_T_servos.py, the string '#(AF ' is placed as comment start, where the above listed parameters are used.

This because those variables weren't initially collected in a json file; Later I decided to comment those rows instead of cancelling them.

Troubleshooting

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

Servos rotate about 180 deg, yet not more than that.

Servos rotation angle of about 90 deg, when you've ordered 180deg

Servos rotation angle of about 270deg

Servos not moving smoothly

Bottom cube layer doesn't align nicely

Top cover usage to flat the cube

Cube status detection error

Robot stuck on reading the same face

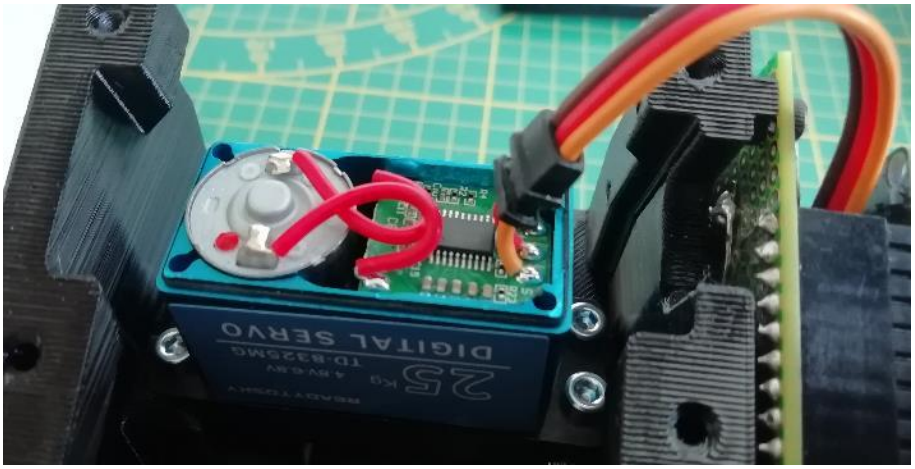
Cube's facelet and light reflection (cube status detection)

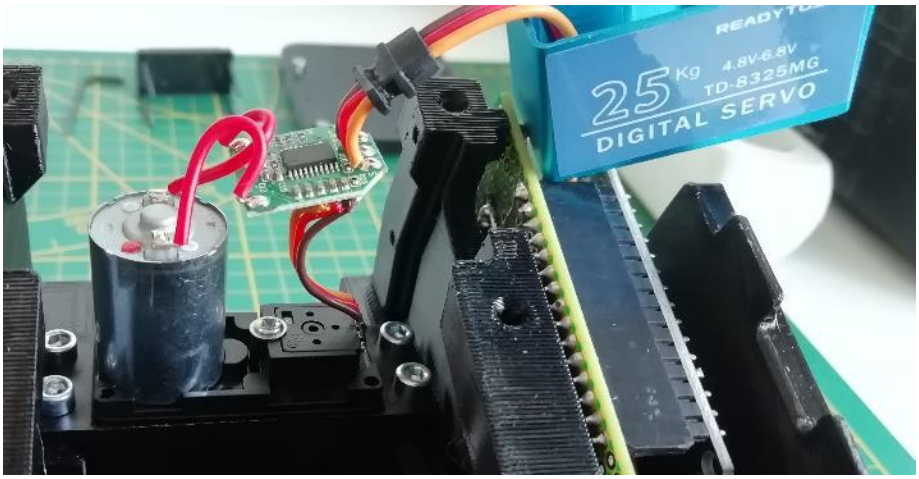
Displayed text and images are un-readable

Program doesn't work as intended

12) PiCamera focus

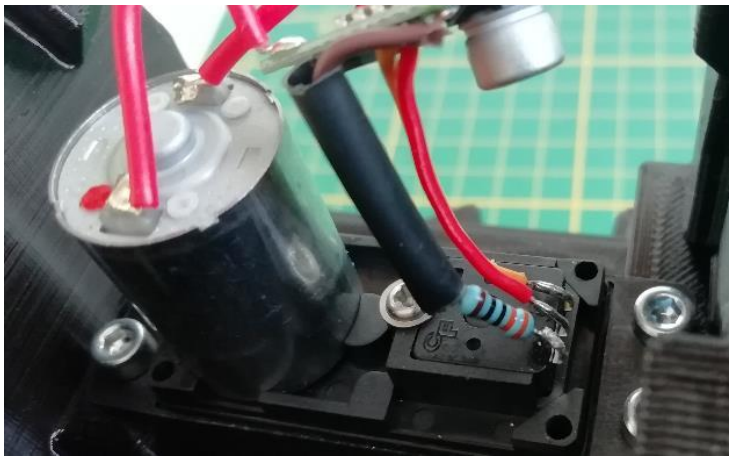
Increase the servo rotation **range (if needed)**

- It's possible to increase the rotation angle, by adding one or two resistors in series with the servo potentiometer (servo must be opened for this change). In my case I had 1 servo (out of 8 used so far) with insufficient rotation range.
-
- Open the servo (4 very long screws)
- 

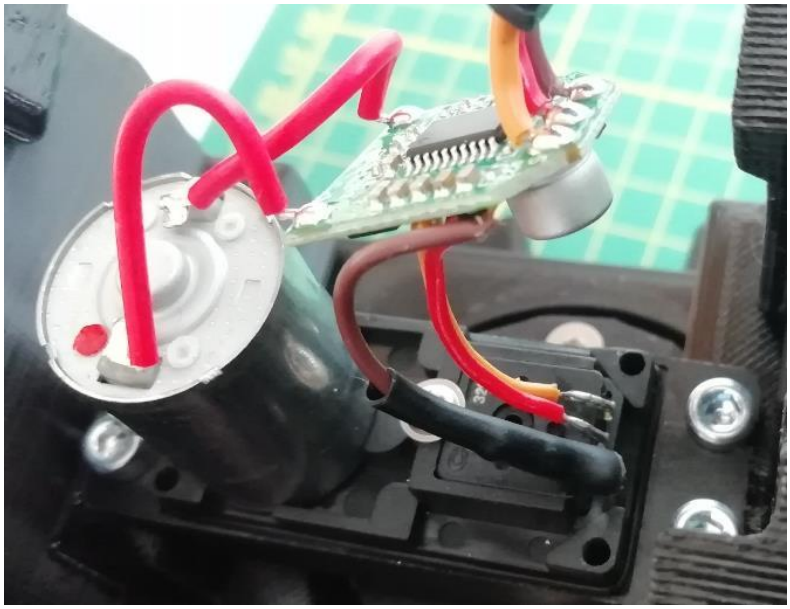
-
- Slide out the pcb, and afterward slide the case out of the way.
- 

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

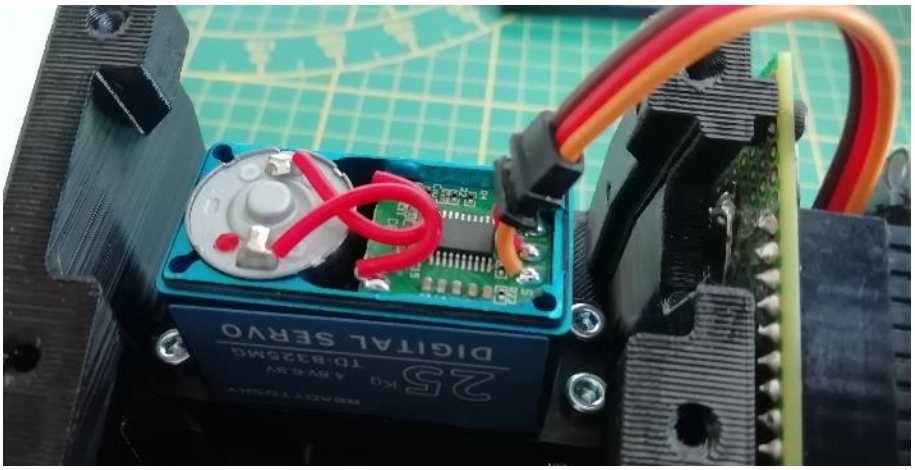
I've added 330ohm to gain about 5 degrees on a servo having pulse width from 1 to 2 ms.



Protect the soldered parts with a sleeve



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

82. 

83.
84. Place the cover and close the 4 screws

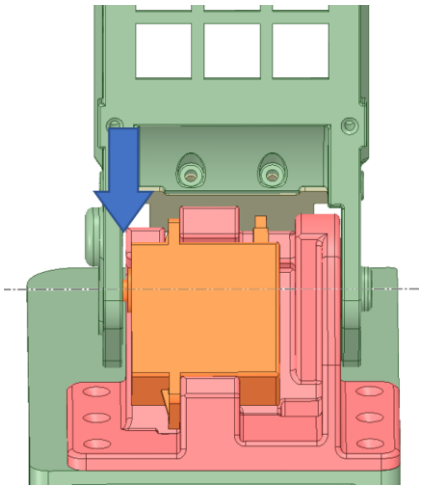
86.

87. Servos rotate about 90deg while you've ordered 180deg servos
88. This has probably to do with the Pulse Width of the received servos, ranging from 500µs to 2500µs instead of from 1 to 2 ms.
89. This project uses as default a Pulse Width range from 1m to 2ms, yet you can adjust some parameters and get your servos working fine; At Cubotino_T_servo_settings.txt change:
90. b_min_pulse_width from 1 to 0.5 (meaning the min pulse width reduces from 1ms to 0.5ms)
91. b_max_pulse_width from 2 to 2.5 (meaning the max pulse width increases from 2ms to 2.5ms)
92. t_min_pulse_width from 1 to 0.5 (meaning the min pulse width reduces from 1ms to 0.5ms)
93. t_max_pulse_width from 2 to 2.5 (meaning the max pulse width increases from 2ms to 2.5ms)
94.
95.
96. In case you've got servos with 270deg rotation, it **shouldn't be of a problem.**
97. If the Pulse Width is from 500µs to 2500µs, then the angle resolution will be acceptable; In this case correct the Pulse Width parameters as per previous Troubleshooting point.
98. In case of Pulse Width from 1ms to 2ms I don't know if the angle resolution will be acceptable for this robot.
99. After adjusting the Pulse Width parameters, in case servos 86ren't from 1ms to 2ms, the default values for servos positions should be scaled by a 0.67 factor, to scale down from 270 to 180 degrees of rotation.

Servos don't move smoothly

Don't use jumper wires, or use quality jumper wires

100. Don't use bread boards, make the Connections board instead.
101. Add the capacitors, to prevent voltage drops when servos are activated.
102. Use an at least 2A power supply for the servos.
103. Use a 20 to 25Kg/cm servo.
104. Minimize Top_cover rotation friction:
105. Ensure the hole for the M4 screw (pivot) has some gap on the Top_cover hole (Ø4.1 to Ø4.3mm).
106. Rub some candle wax on the screw.
107. Ensure the M4 screw head is not pushing toward the Top_cover; 1mm gas is suggested
108. Ensure gap presence between Top_cover inner surface and the Hinge at the servo gear outlet side, as per below arrow:

109. 

110. Bottom cube layer doesn't align nicely:
111. This is probably the most difficult part of the tuning process.
112. 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.
113. Verify if the cube Holder makes an extra rotation, at both CCW and CW directions, before stopping; If this doesn't happens:
114. 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.
115. Adapt the PWM release CCW/CW value.
116. Place the PWM release CCW/CW at zero, and test if the CCW and CW position have a slightly overstroke from the 90°. If this is not the case, check if the other servo has a larger rotation range. If still not the case:
117. 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)
118. Check in internet how to (slightly) increase the servo rotation angle (additional resistors must be soldered into the servo)
119. Verify if the cube Holder makes an extra rotation, before stopping Home; If this doesn't happen, adapt the PWM release home value.
120.
121.
122.
123. Top cover usage to flatten the cube:
124. The Top_cover isn't intended to keep pushing the cube when it's in the close position; In case the cube layers don't align nicely, by playing with the cube_Holder settings, it's possible to use the Top_cover to level the cube. By lower the Top_cover close position to have a little interference with the cube, will improve the cube layer alignment in particular after flipping the cube. In this case it will be convenient to set one or few units on *PWM release from close setting.* Via this setting is possible to release the tension between the Top_cover and the cube, after pressing it, to allow the cube_Holder to rotate with less effort.

Cube detection error:

It is returned when the interpreted cube status isn't coherent, meaning not having 9 facelets per colour or other inconsistencies. Possible causes:
Objects **on** the table (background); Objects on the table can form square like contour, interpreted as facelets by the cv. This can be solved by positioning the robot to a uniform-coloured surface, without cables and objects in for 30cm around the robot. Another good way to solve this problem is to tune the cropping parameters.

13) 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.

Too little light conditions cannot be compensated by the LED light source.

In case the cube has some prints (i.e. brand) , typically on the white center, it is suggested to carefully scratch out.

In case a frameless cube type is used (facelets without the black frame around the facelets), while at Cubotino_T_settings.txt the frameless_cube parameter is not 'true' or 'auto'.

The setting ‘auto’ to detect the status on cubes with and without the frame works better with good light conditions; If this isn’t possible, set the parameter to the specific type of cube associated to the robot.

Robot stuck on reading the same face, until timeout:

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)

When the frameless_cube is set 'true' or 'auto', the cube status detection algorithm must find 7 facelets with given characteristics before changing face; The remaining two are estimated for the position, not the colour.

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.

When the ambient light is rather hight 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.

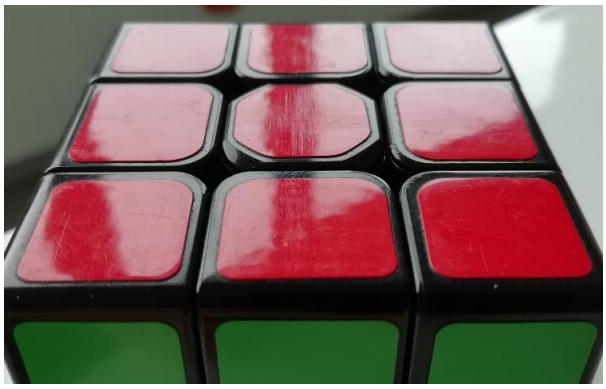
To troubleshooting is important to visualize what the camera sees; This is possible via below steps:
1) Connect to the Rpi via VNC Viewer.
2) If the robot has automatically started at the boot, two processes need to be killed as per "How to operate the robot" Step8.
3) Resize the terminal to no more than half screen, and move it to the right part of the screen.
4) Run the script from the terminal
4_1) *cd ~cubotino/src*
4_2) *source .virtualenvs/bin/activate*
4_3) *python Cubotino_T.py*
5) Press start to let the robot working, and a windows will show what the camera sees.

A contour will be drawn, over the camera image, on every location interpreted as 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 camera_support, to have margin around the top cube face.
B) facelets on the back cube side are also detected (detected means that on the image contours are drawn on the back cube facelets): Apply the cropping as explained for "frame cropping" in the "Tuning" chapter
C) the critic face has a logo on the central 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.

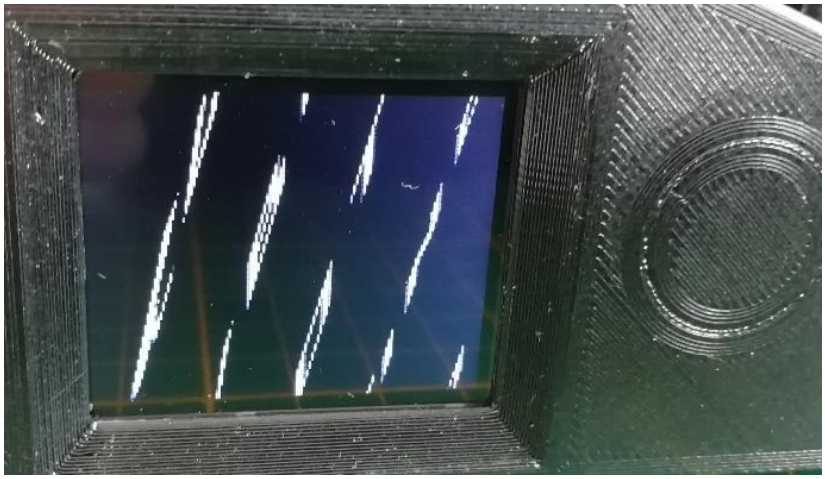
Cube's facelet **and light reflection (cube status detection):**

- Detection of edges, as well as colours, can be largely affected by light reflection made by the facelets.
- I have two cubes available, one with in-moulded coloured facelets, and the other with glossy stickers.
- On the cube with plastic facelet, I made the surface matt by using a fine grit sandpaper (grit 1000); This makes the cube status detection much more unsensitive to the light situations.



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

Display parameters are set in a (json) text file: Cubotino_T_settings.
The display I've received, from the supplies list, wasn't working as intended:



I took me some time to realize a sort of drifting…. And to find the fix.
By slightly changing the dimensions of just a couple of pixels on the display width parameter (in my case I had to set 128 instead of 130 pixels), the display was working.
Once that problem was solved, a second one became apparent: On two sides there were some 'tiny death bandwith', again for just a couple pixels.
By checking the ST7735.py code, I discovered it provides offset parameters, for the two display directions, suggesting these issues to be well known…..
By playing with both the display dimensions and the offsets, the display start working simply fine.
I don't expect all the display to have the same issue, yet in case the parameters variables are already in the code, and ready to be properly tuned.

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 feedbacks from the script:
Edit Cubotino_T.py

At __main__ change the Boolean "debug" to True. This variable is used by many functions to print out info to the terminal.

Run Cubotino_T.py

Check the prints

If the print out doesn't suggest much to you, share it at the Instructable chat

When the problem seems more related to the servo program:
Edit Cubotino_T_servos.py

At about row 85 change the Boolean "s_debug" to True. This variable is used by many functions to print out info to the terminal.

Run Cubotino_T_servos.py (activate the venv first, and recall to type python in front).This code activates the servos like solving a predefined scrambled cube.

Check the prints.

Run Cubotino_T.py and let it call the Cubotino_T_servos.py.

Check the prints

If the prints out don't suggest much to you, share it at the Instructable chat

PiCamera **focus**

The V1.3 PiCamera is delivered with fixed focus.
Until now, end of October 2022, I haven't received negative feedback from other Makers because of non-working Cubotinos caused by of focus PiCameras.

On my first project with PiCamera I did adjust the camera focus, by following this tutorial
https://projects.raspberrypi.org/en/projects/infrared-bird-box/6
It hadn't been easy, yet I managed.
After correcting the focus, I did not glue the lens, as there still was quite some friction, preventing the lens from getting loose.

When I tried to repeat this operation on a second PiCamera I have almost destroyed it.
Both my Cubotinos have PiCamera "as received", one Cubotino has okish focus while the second one is out of focus.
In both cases the robot work properly, the only difference is the sharpness on the images the robot saves.
In case of non-satisfactory results, and if you are good with your hands, you might decide to try … on your own risk.

How to operate the robot

Before starting:

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

Use a **uniform-coloured table**:

Part of the table around the robot will be captured on cube face images.
Keep some free space around, and use a uniform-coloured base, to prevent cables or other objects to be eventually detected as facelets.

Power up the **servos**:

This considers an independent power supply for the servos
Connect a 5V power source to the microUSB connector where the servos are connected.
In my case the servos work flawless with a phone charger rated 2A, but I had inconsistent movements when using a phone smart-charger rated 2A, and with a normal phone charger rated 1A.
Notes
Do not energize the servos once the SBC is up and running, if you aren't sure whether the cube holder is positioned to home.

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

Power up Raspberry Pi Zero 2:

Connect a 5V power source to the remoted microUSB connector where Raspberry Pi is connected.
Do not connect phone smart-charger, those that can deliver a voltage higher than 5.1V.
In my case the SBC works flawless with a phone charger rated 1A (note the official documentation suggest higher current).

Start **a solving cycle:**

Position the cube on the cube holder; any cube orientation is accepted.

Cube layers should be reasonably aligned.

Shortly touch the PCB_cover in front to the touch sensor (the circle suggests the touch sensor location).

The robot reacts by energizing the LED, and by indicating CAMERA SETUP on the display.

Raspberry Pi shut down:

Please be noted Raspberry Pi, like normal PC, cannot be unpowered when it is working.
To shut it down, there are few possibilities:

Connect to the Raspberry Pi via SSH, and type *sudo halt -p*

The SBC closes the open applications and files
When the SBC activity is almost done, the led at Connections board will goes off
Wait additional 10 seconds and the power supply can be safely removed.

If the robot has proved to work without errors, un-comment last row at Cubotino_T_bash.sh file (*halt -p*).

This means the SBC will automatically shut down when the Cubotino_T.py file ends.

In order to quit the *Cubotino_T.py* script, touch the Touch_button long enough (ca 6 seconds) until the 'SHUT DOWN' appears on display; The SBC will close the open applications and files.
Differently, iIf the touch sensor is released as soon as the display shows 'SURE TO QUIT?', then the robot will consider it as a stop request instead of a shut-down request.
When the SBC shut-down process is almost done, the led at Connections board will goes off.
Wait additional 10 seconds and 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 pause a risk to your kids if they have their hand on the way while the cover closes!

Un-power the servos:

This considers an independent power supply for the servos.
Servos can be unpowered at any moment.
Un-power the servo before shutting down Rpi, if you experience strange servo behaviour when Rpi goes off; The script takes care to set the servo PWM related GPIO to a fix level (low), at the shut-down, but it does not work on 100% of the times.

Running the robot from VNC Viewer:

Here is explained how to run the robot from VNC Viewer, when the python script has been started via the Bash file at Raspberry Pi boot, and 'halt -p' command is uncommented in the bash file.
Under these circumstances:
it is not an option to quit the script from the robot, by keeping the touch button pressed long, as the Raspberry Pi will shut down

it is not possible to run a 'new' script over the first one , as will conflict with Camera resources; It is necessary to quit the running python scrip first.

Steps to do:
Connect VNC Viewer to the robot

Open a terminal

Folder is not relevant

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
```

Search for python Cubotino process, and note the ID (674 on the above example); This is by far the process that takes more CPU and memory resources, making easier to find it.

Search for bash command, from root user, located above python Cubotino, and note the ID (657 on the above example)

First kill the bash process sudo kill -9 ThePIDNumberForBash (by using the above example the command will be *sudo kill -9 657*)

After kill the python process *sudo kill -9 ThePIDNumberForPythonCubotino* (by using the above example the command will be *sudo kill -9 674*)

```
pi@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 shuts off right after the Cubotino python process is killed: Not the wanted result 😊

Automatic robot start' chapter

Step 13: Set things to get the Raspberry Pi shutting off automatically, without using the PC.

See 'Servos test and set to mid position

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

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

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

7. Connect a connections arm to the non-assembled servos.
8. Connect the servos to the Connections_board.

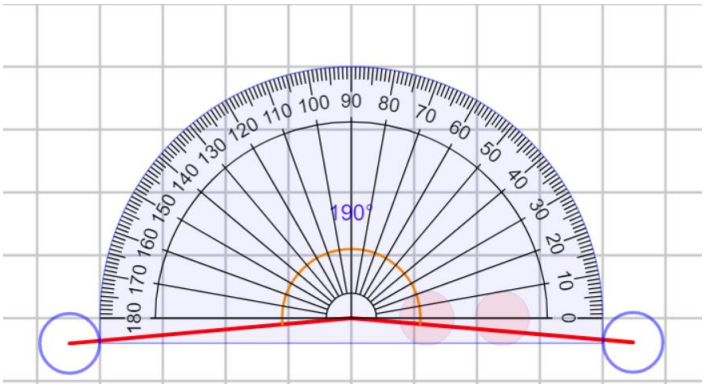
Argument set, and related value, can be passed to Cubotino_T_servos.py, to play with the servos angle and especially to set them to the mid position before the assembling.

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

Position the servo arm as per below picture:

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

The angle can be evaluated by printing a protractor:



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

14) 3D printed parts

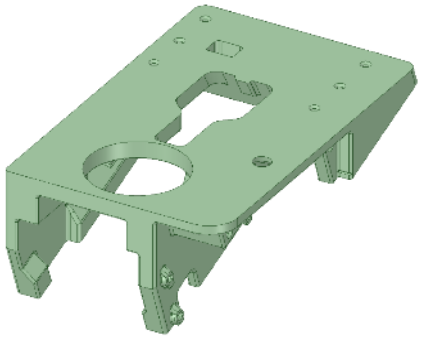
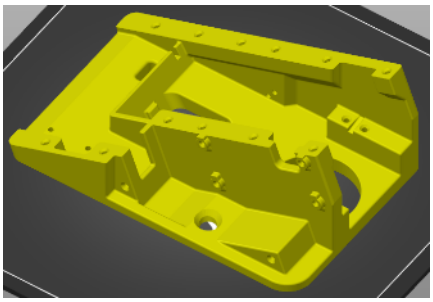
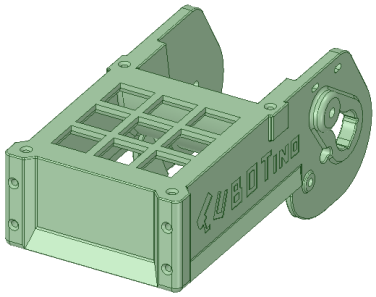
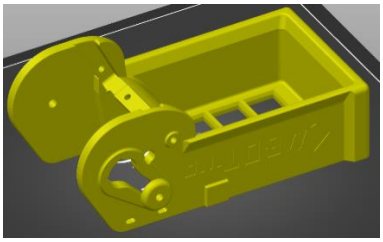
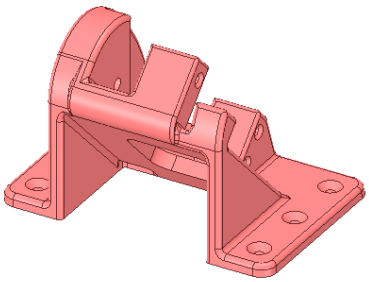
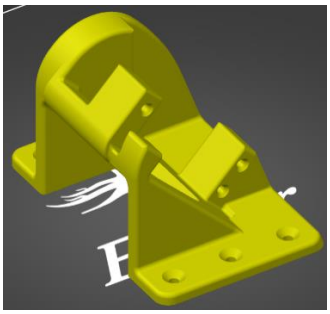
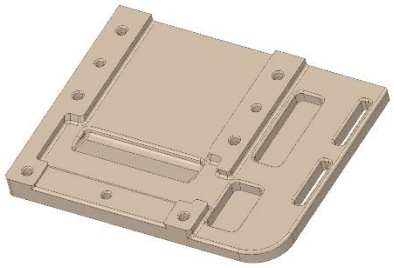
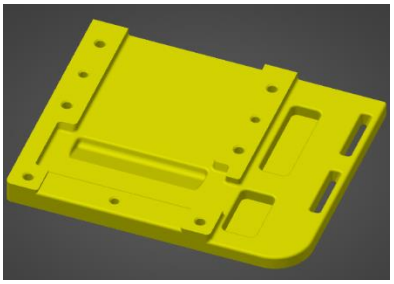
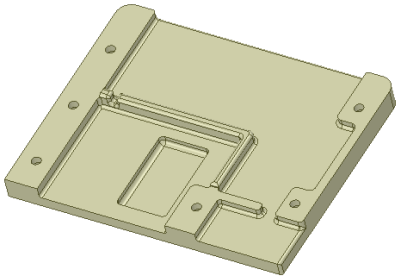
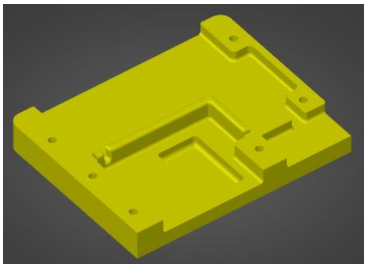
See notes below for filament quantity, and printing time …

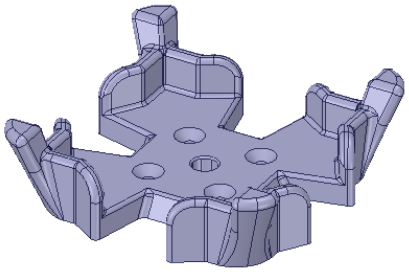
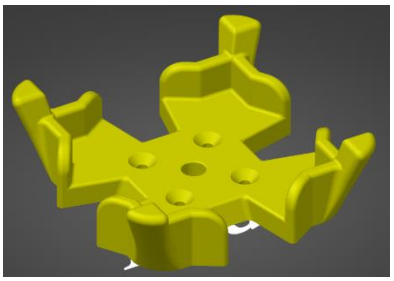
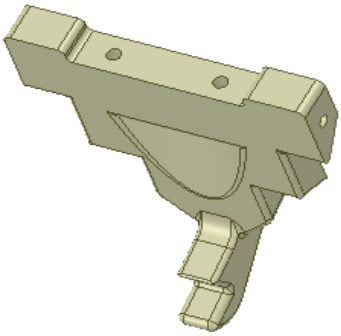
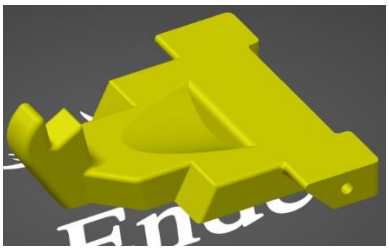
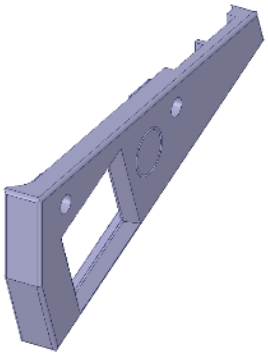
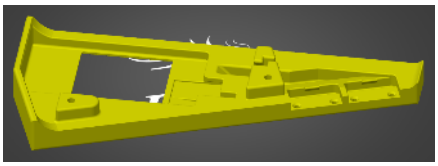
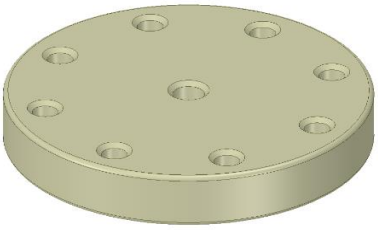
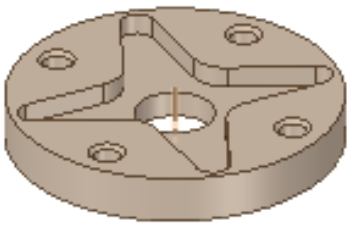
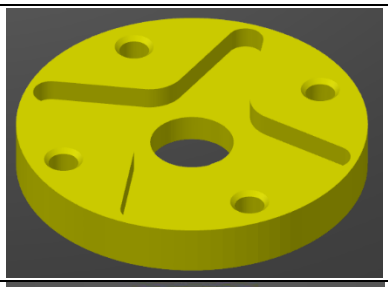
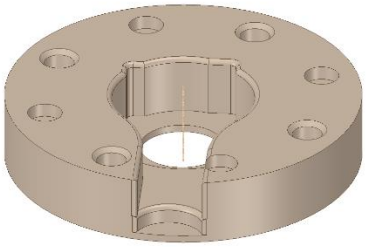
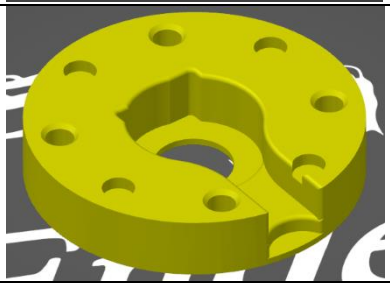
Ref	Part	Filament		Printing time	Version specific parts
		Meters	Grams		
1	Structure	49	145	14h33m	
2	Top_cover	39,5	117	12h26m	
3	Hinge	17,2	51	5h37m	
4	Baseplate front	12,2	36	3h33m	
5	Baseplate back	12,6	37,3	3h36m	
6	Cube_holder	11,6	34,4	3h36m	
7	Cube_lifter	5,5	16,2	1h48m	
8	Servo_axis_sup (or its alternative)	2,4	7,2	0h55m	
9	Servo_axis_inf (or its alternative)	2,4	7	0h52m	
10	PCB_cover_display (or its alternative)	14	41.4	4h13m (4h27m)	Top specific
11	PiCamera holder	2.2	6.5	0h50m	Top specific
12	PiCamera holder frame	6.1	18	2h17m	Top specific
13	Personaliz_plate	1.5	5	0h30	Top specific
	TOTAL	**175m**	**692g**	**51h30m**	

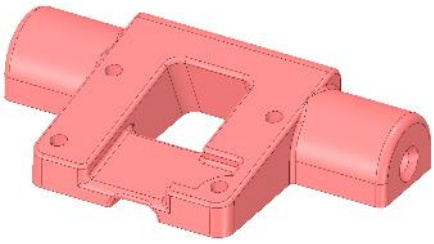
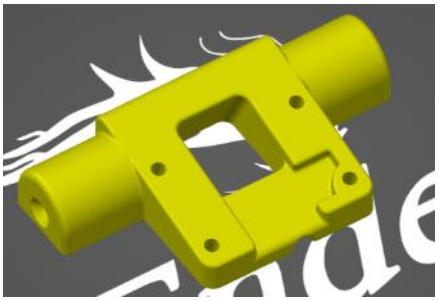
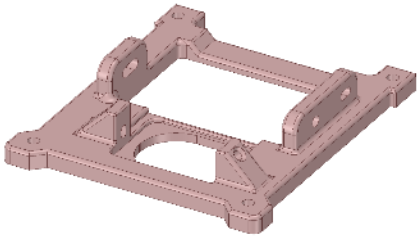
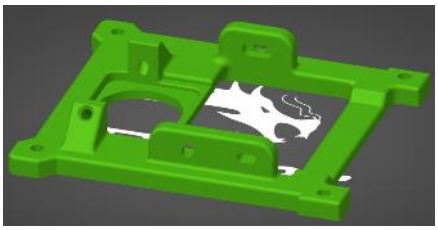
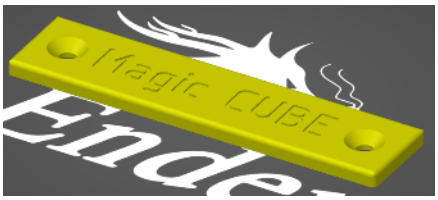
Notes:

11. The biggest part is the Structure, and it easily fits on printers with plate size of 200x200mm.
12. Filament length is based on Ø1.75mm.
13. 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:
    7. 0.2mm layers
    8. Low speed (between 25 to 40mm/s for the external parts and 1st layer)
    9. 4 layers on vertical walls
    10. 5 layers on horizontal walls
    11. 30% filling
    12. 8mm brim
14. The filament quantity, and printing time, in the table should be considered as an upper limit. After printing the parts for the base version, total weight was closer to 400grams than 466grams estimated by the slicer SW.
15. Possible to upgrade the Base version, by printing 3 (or 4) more parts, requiring ~ 8h.
16. All parts have been designed to be printed without supporting the overhangs.
17. Some parts have been split, for easier and better 3D printing.
18. The suggested part orientation for the 3D print is showed on below Table.
19. The **stl** files are available at
    - https://www.instructables.com/CUBOTino-Autonomous-Small-3D-Printed-Rubiks-Cube-R/
    - https://www.thingiverse.com/thing:5407226
    - https://github.com/AndreaFavero71/cubotino/tree/main/stl

20. The **step** files are only available at https://github.com/AndreaFavero71/cubotino/tree/main/stp

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

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

PiCamera holder		
PiCamera frame		
Personaliz_plate Or Personaliz_plate01	 	

15) Assembly steps

Before assembling the robot:

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

Assembly order are initially as per the Cubotino_base_version:

125. Screw the bottom servo to the structure
126. Prepare the sandwich Servo_axis_inf / servo lever / Servo_axis_inf
127. Assemble the Cube_holder to the Servo_axis assembly
128. Assemble the Cube_holder assembly to the bottom servo
129. Assemble the Hinge to the Structure
130. Assemble the “T25” servo arm to the upper servo, only after knowing the servo is on its middle position.
131. Insert the Top_servo assembly into the Top_cover slot
132. Assemble the Top_cover assembly to the Hinge
133. Complete the top servo assembly to the Hinge
134. Assemble the Lifter to the Top_cover

Starting from here, the steps are different from the Cubotino_base_version

135. Assemble the PiCamera holder frame to the Top_cover
136. Assemble the LED breakout board to the Top_cover
137. Position the cables, and assemble the Baseplate_rear
138. Connect the PiCamera flex cable to Raspberry Pi Zero2 board, and fix it to the Structure
139. Fix the Touch_sensor to the PCB cover
140. Assemble the USb breakout borad to the PCB_cover
141. Connect the servos, LED breakout board and Touch sensor
142. dress the cable and connect the display
143. Assemble the PCB_cover
144. Assemble the Baseplate_front to the Structure
145. Stick the PiCamera to its board, via the self-adhesive tape underneath the camera.
146. Assemble the PiCamera module to the PiCamera_holder
147. Assemble the PiCamera holder to the PiCamera holder frame
148. Connect the flex cable to the PiCamera module
149. Personalize the formal Stop plate, transformed to a personalization plate in this robot vesion

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

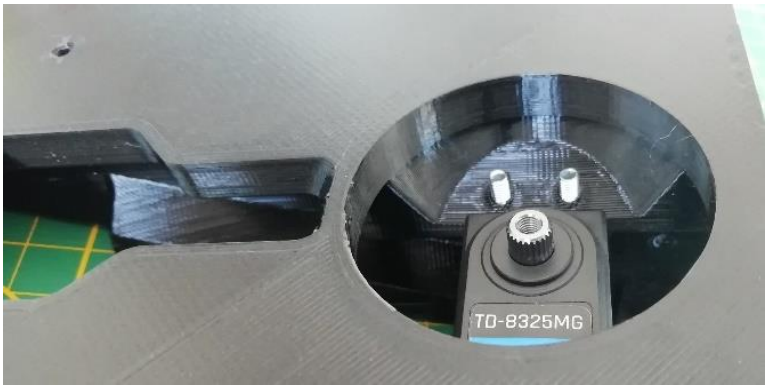
16) Assembly details

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

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

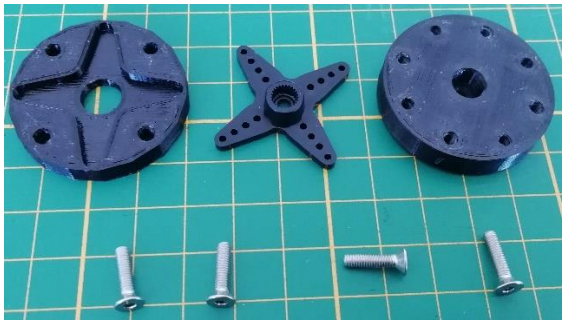
Couple of notes:

- Before tightening the four screws, checks if the servo output gear is well centred to the Structure hole.
- To limit the protrusion of the below two screws, add a couple of washers to these screws; This to prevent eventual interference with the servo_axis_inf part.



Step5 (sandwich Servo_axis_inf / servo arm / Servo_axis_inf):

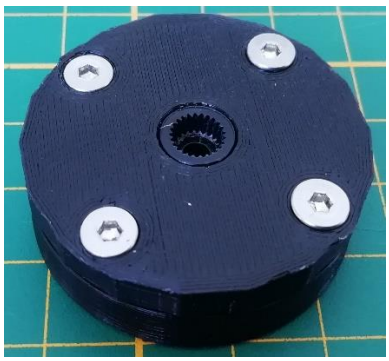
4x M3x12mm conical head

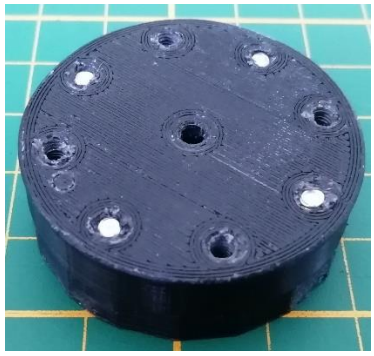


Check if the supplied X arm fits with the indentation of Servo_axis_inf part:

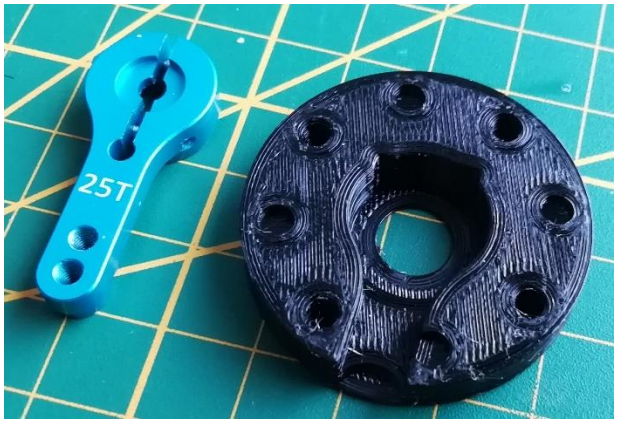
If you don't have a X arm to make it fit, please print the alternative parts (Servo_axis_inf and Servo_axis_sup) designed to fit the aluminium "T25" arm (arm has to be reduced in length).

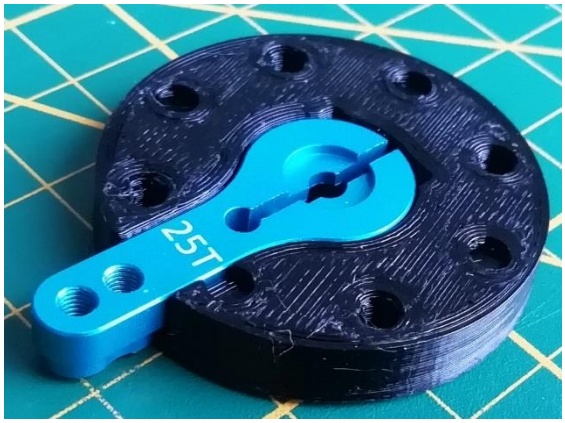
Make the sandwich





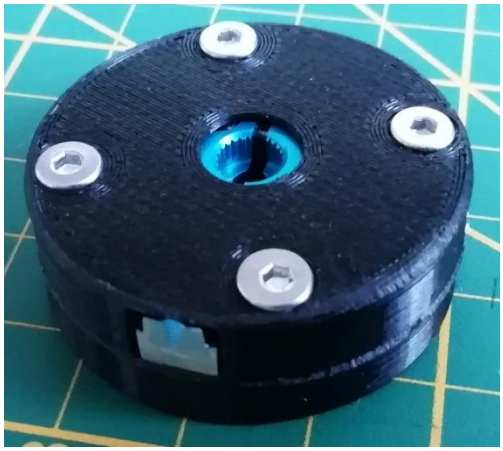
Step5a Alternative servo axis assembly:





Cut the protruding part of the “T25” arm
Adjust its screws to be able to enter the servo outlet gear

Make the sandwich as per Step5
4x M3x12mm conical head



Step6 (Assemble the Cube_holder to Servo_axis assembly):

4x M3x12mm conical head



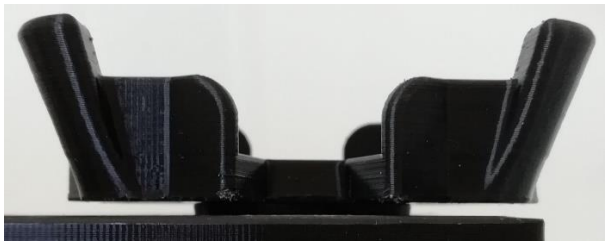
Step7 (Assemble the Cube_holder assembly to the bottom servo):

Try to not rotate the Servo output gear during this step: Gently try to feel the teeth coupling, and if the Cube_holder is not well aligned, retract it, rotate the Cube_holder by about 90 degrees and check again.

Servo output, and Servo arm, have even number of teeth: For sure there is one good coupling

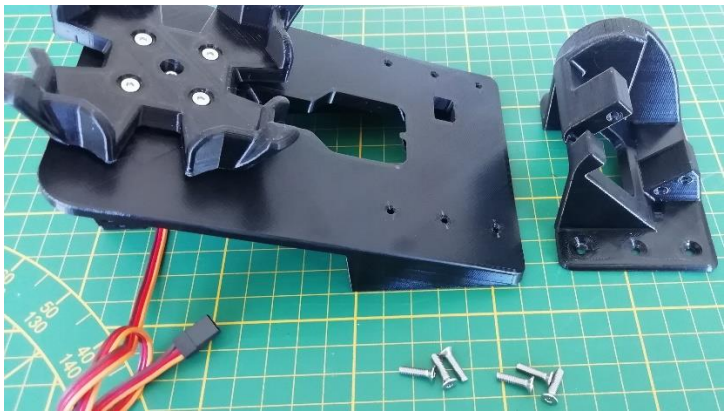

1x M3x12mm cylindrical head

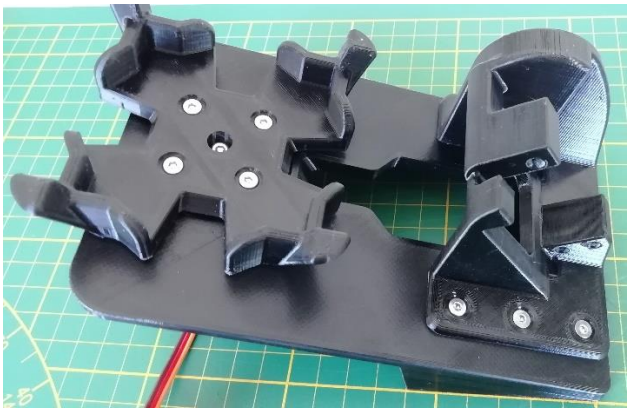




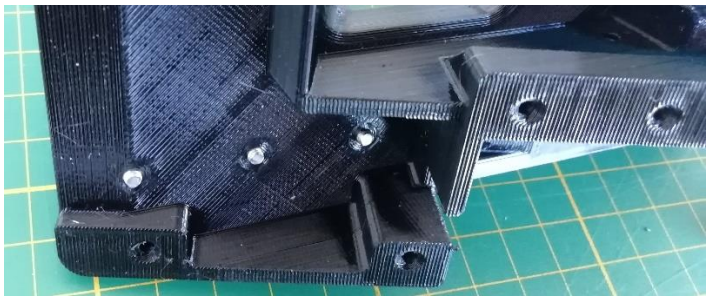
Step8 (Assemble the Hinge to the Structure):

6x M3x12mm conical head





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



Step9 (Assemble the "T25" servo arm to the upper servo):

Place the "T25" arm along the main Servo axis (Servo should be prepared upfront with the gear at middle angle).

Close the two tiny screws at the arm

Step10 (Insert the Top_servo assembly into the Top_cover slot):

1x M3x12mm cylindrical head

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

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

Note: The screw should not protrude from the Structure plane; Add some washers under the screw head if needed

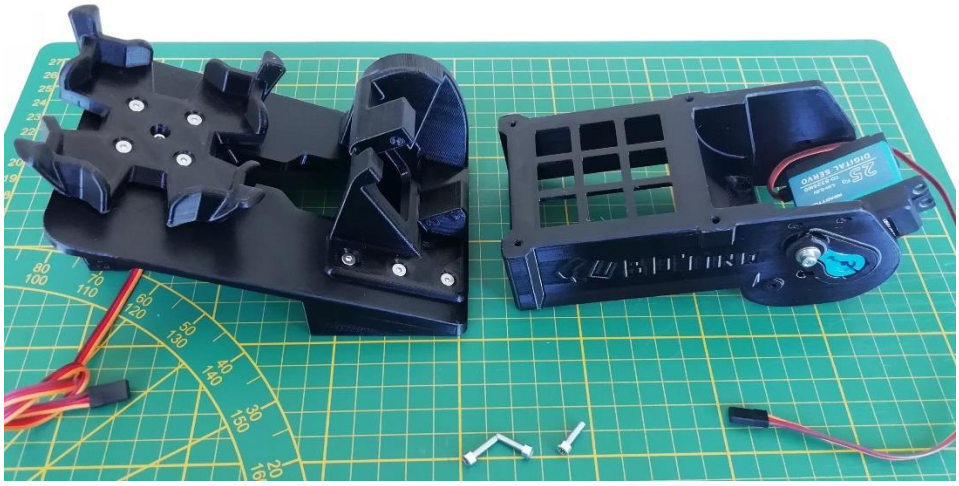
Rotate the servo by about 45deg, to facilitate next steps

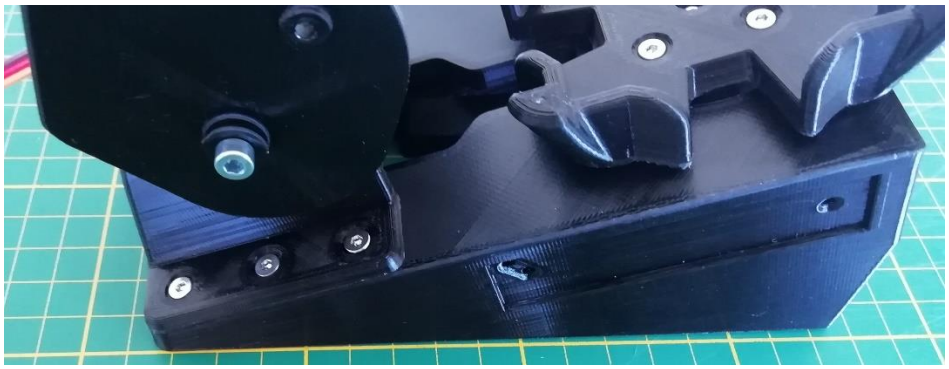
Step11 (Assemble the Top_cover assembly to the Hinge):

1x M4x20mm cylindrical head

3x M3x12mm cylindrical head

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

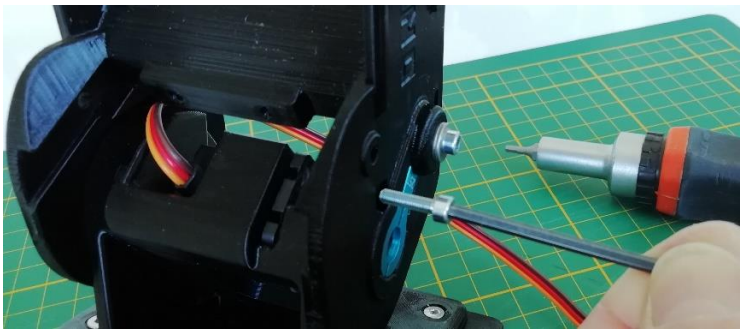




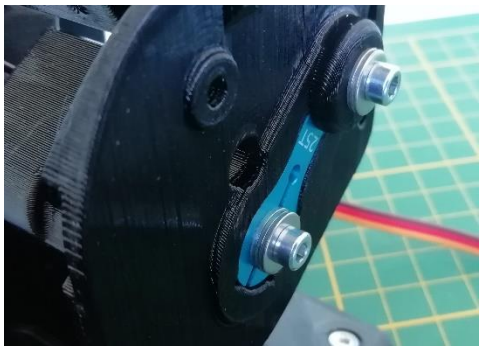
Do not fully tighten the two M3x12mm, until also the third screw is positioned

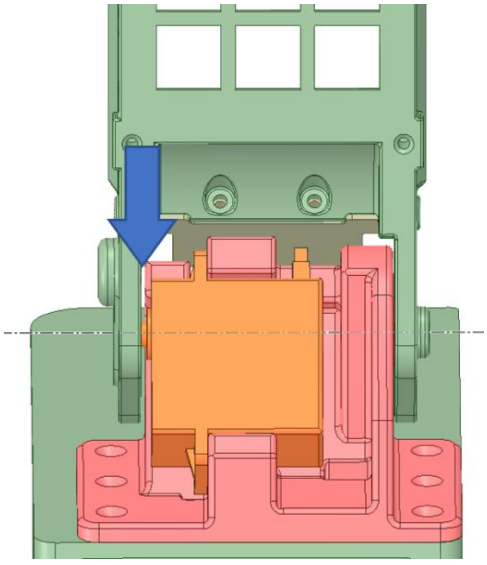
Step12 (Complete the top servo assembly to the Hinge):

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



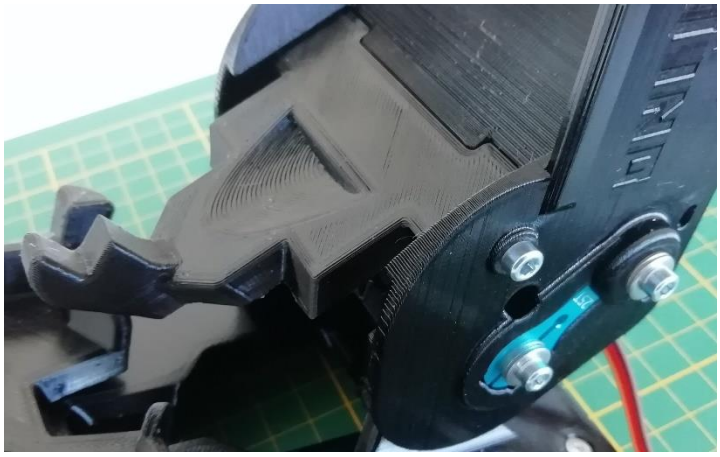
1x M3x12mm cylindrical head (add washers if the screws doesn't push on the "T25"Arm)

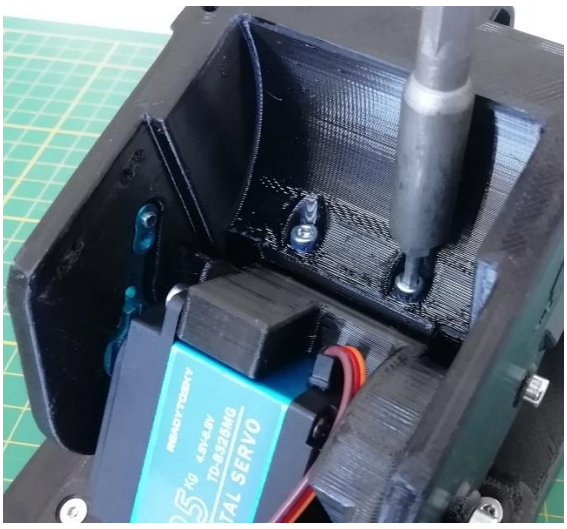




Step13 (Assemble the Lifter to the Top_cover):
4x M3x12mm cylindrical head

Slide the Lifter into the Top_cover slots

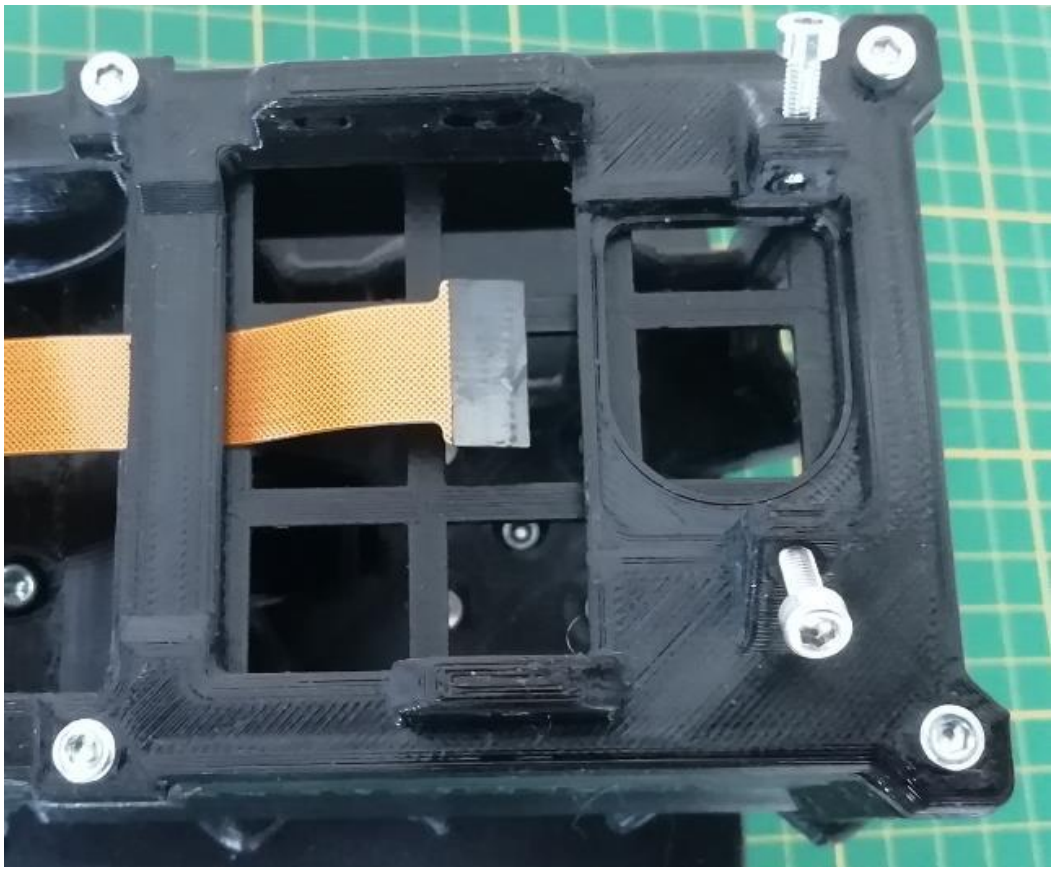




Step14 (Assemble the PiCamera holder frame to the Top_cover):

4x M3x12mm cylindrical head

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

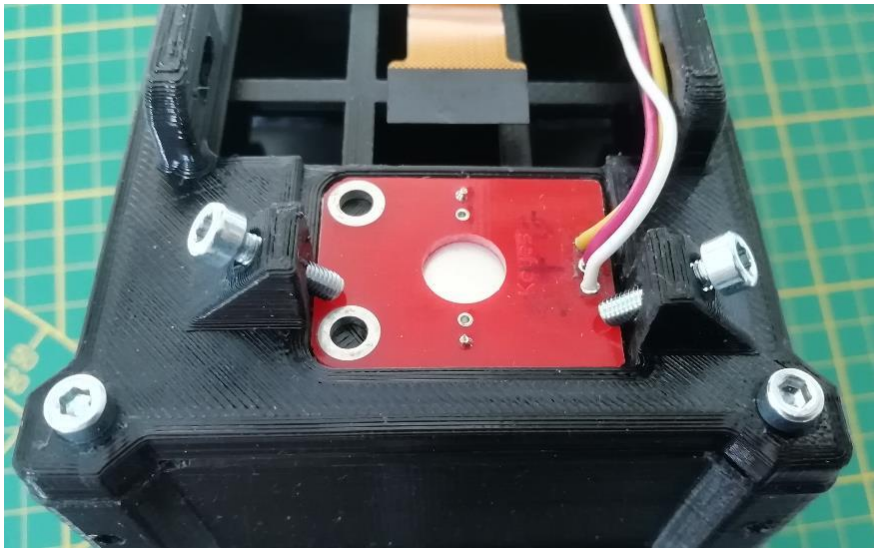


Step15 (Assemble the LED breakout board):

Add 2x M3x12mm cylindrical head

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

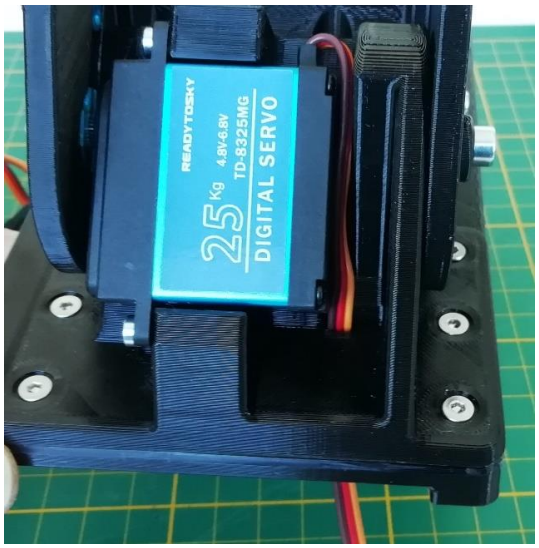
Stop screwing once the screw tip touches the board

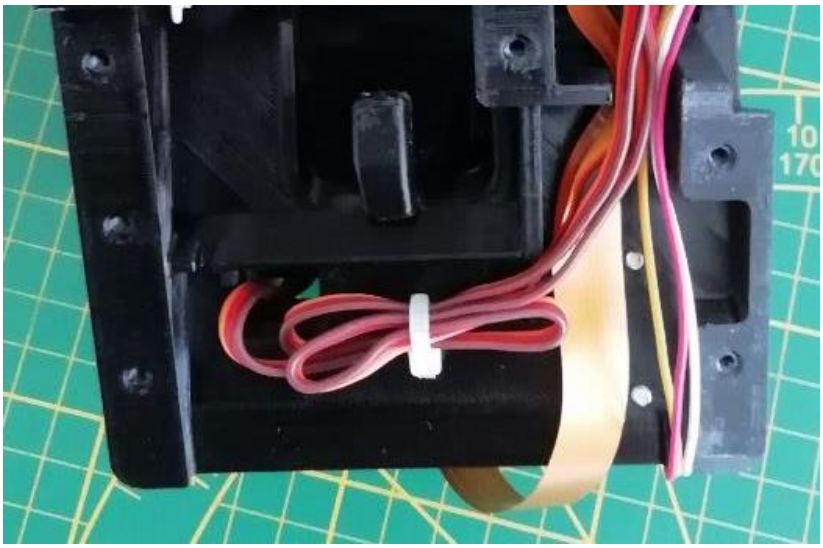


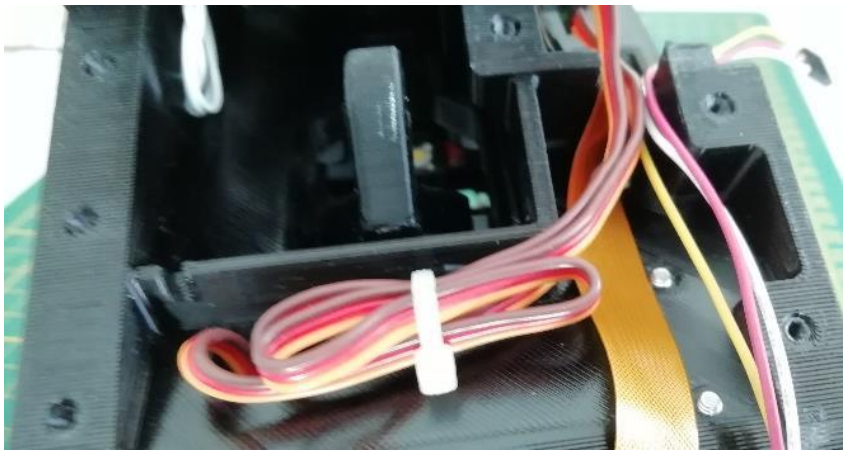
Step17 (Position the cables, and assemble the Baseplate_rear):
6x M3x12mm conical head (4 screws at corners are sufficient)

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

Dress the cables and close the Baseplate_rear.

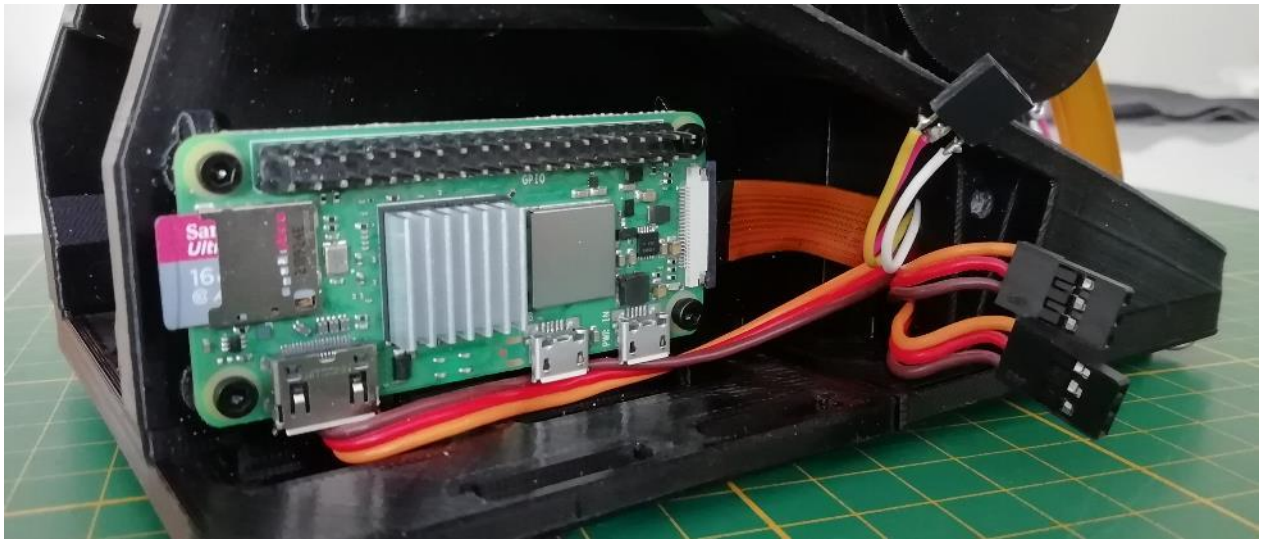






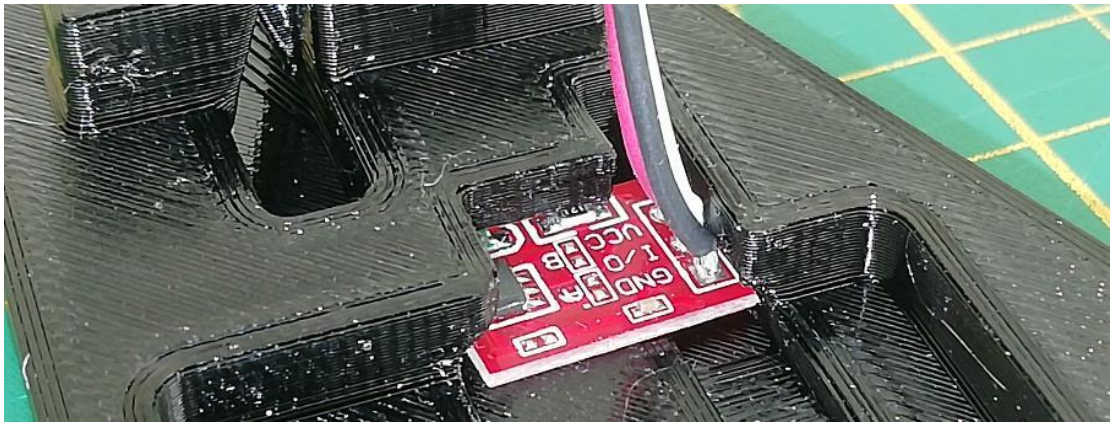
Step17 (Connect the PiCamera flex cable to Raspberry Pi Zero2 board, and fix it to the Structure)

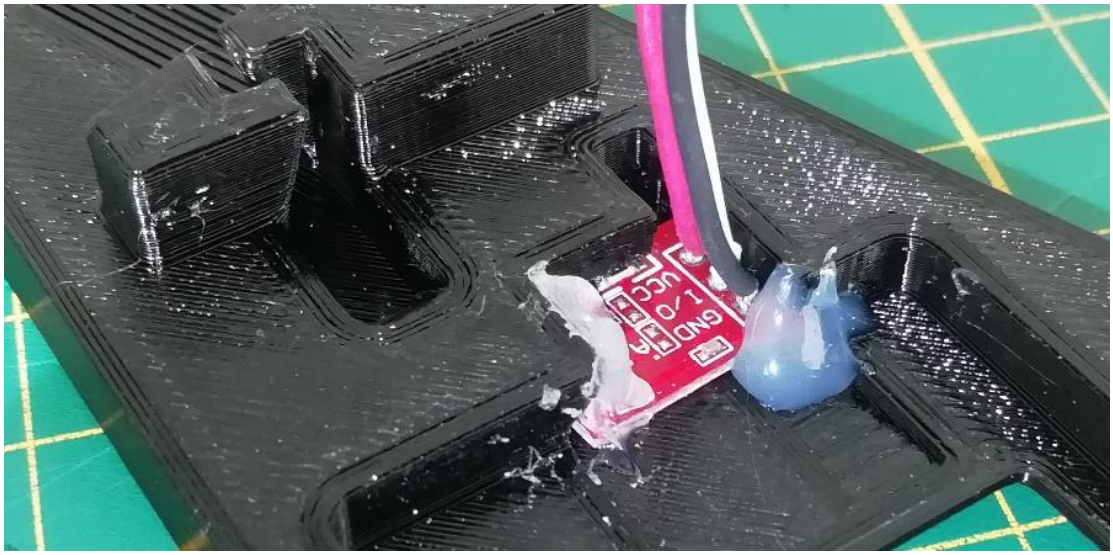
4x M2.5x10mm (or M2.5x8mm) cylindrical head



Step18 (Fix the Touch_sensor to the PCB cover):

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

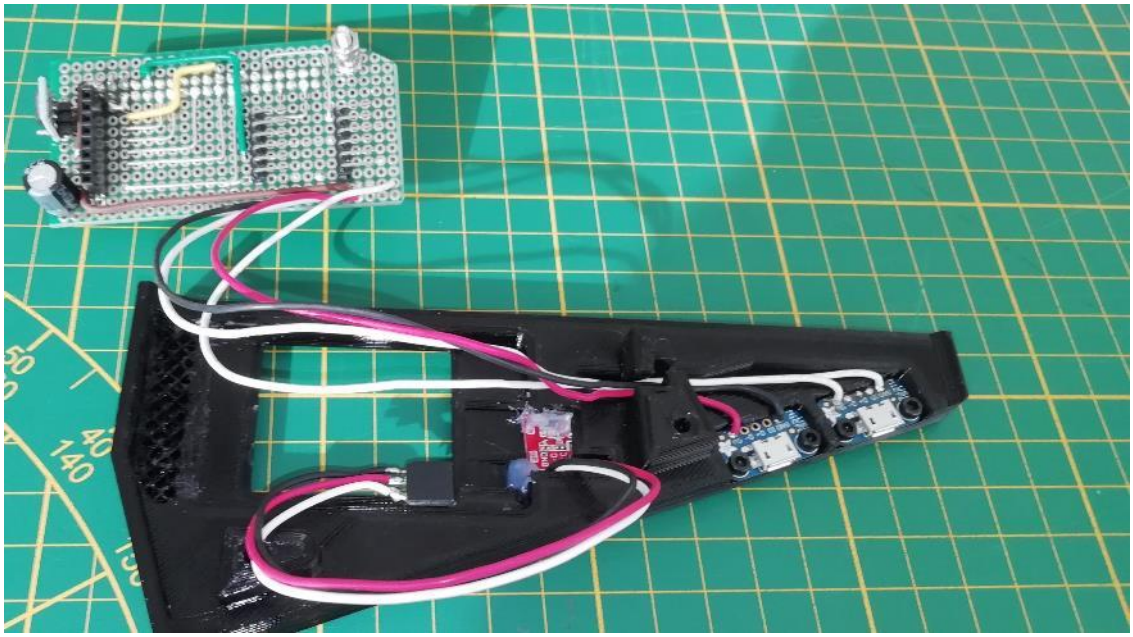




Step19 (Fix the microUSB breakout board to the PCB_cover):

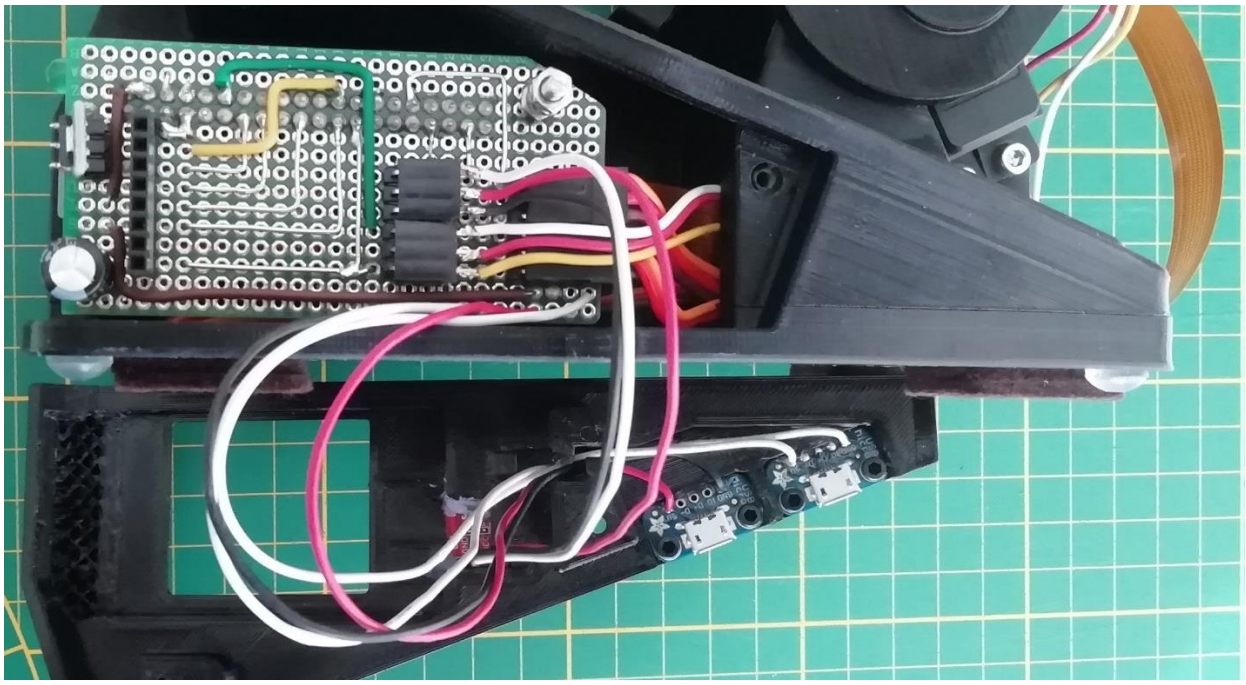
2x M2.5x4mm cylindrical head

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



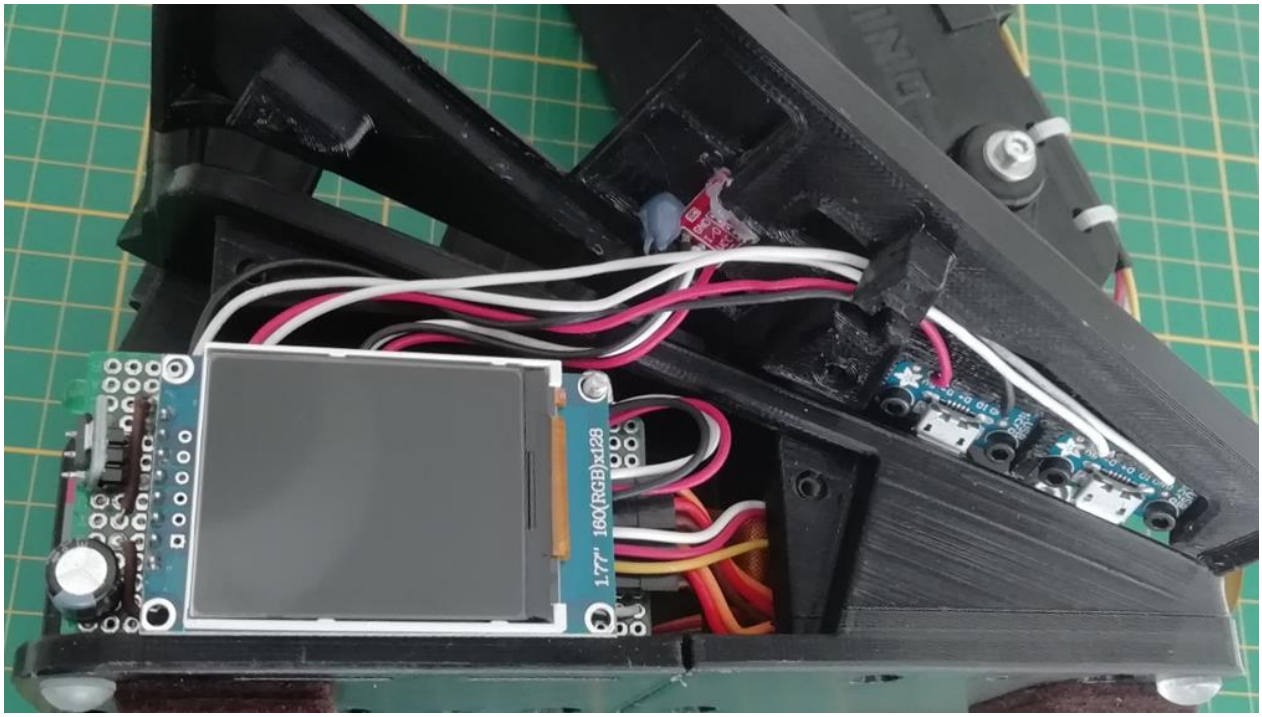
Step20 (Connect the servos, LED breakout board and Touch sensor):

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



Step21 (dress the cable and connect the display):

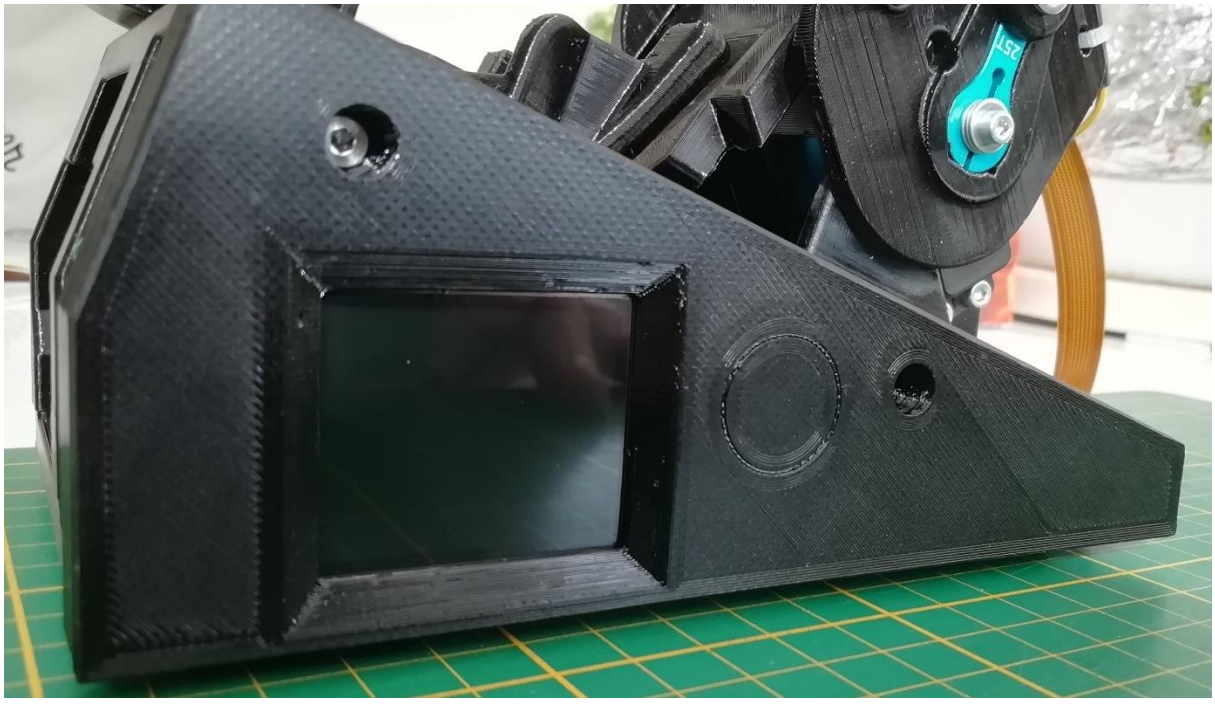
Cables at the microUSB breakout boards have to be well positioned into the groove



Step22 (Assemble the PCB_cover):

2x M3x12mm conical head

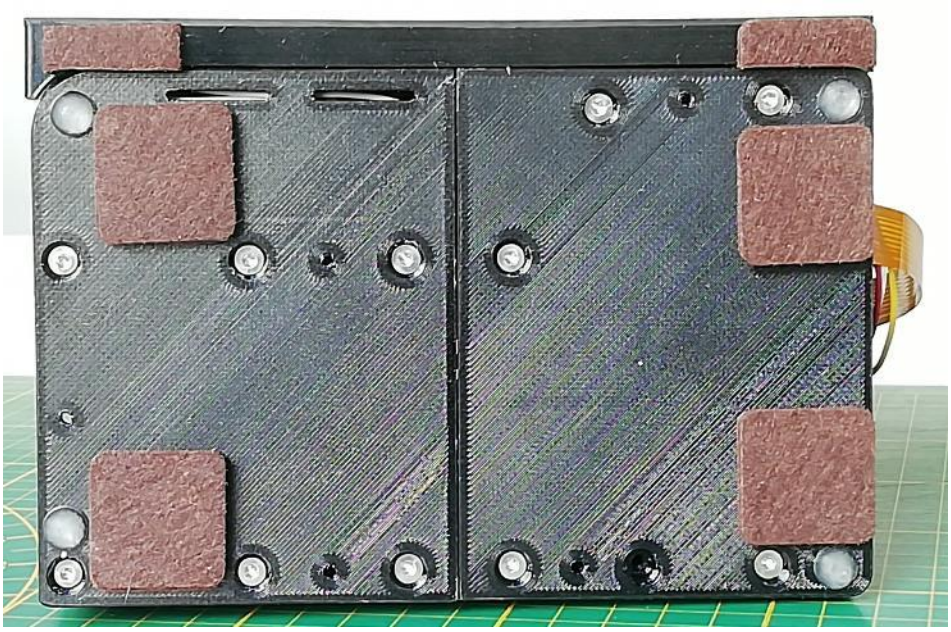
Attention to don't squeeze cables against the Structure



Step23 (Assemble the Baseplate_front to the Structure):

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

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



Step24 (Stick the PiCamera to the board):

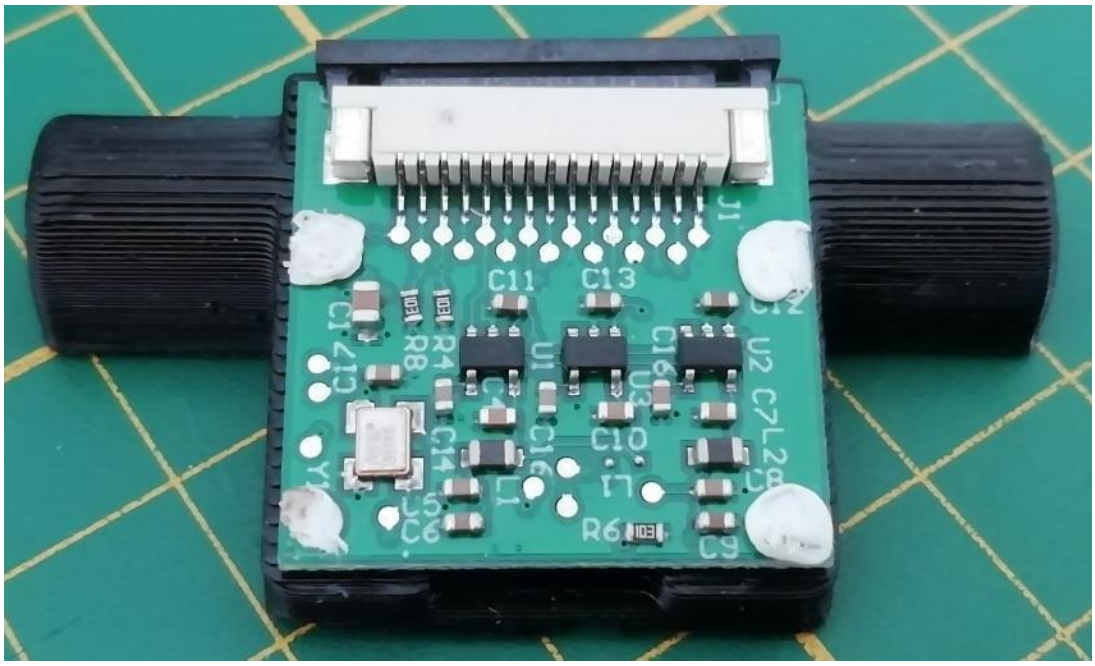
Underneath the picamera there is a little piece of self-adhesive tape.

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

Step25 (Assemble the PiCamera module to the PiCamera holder):

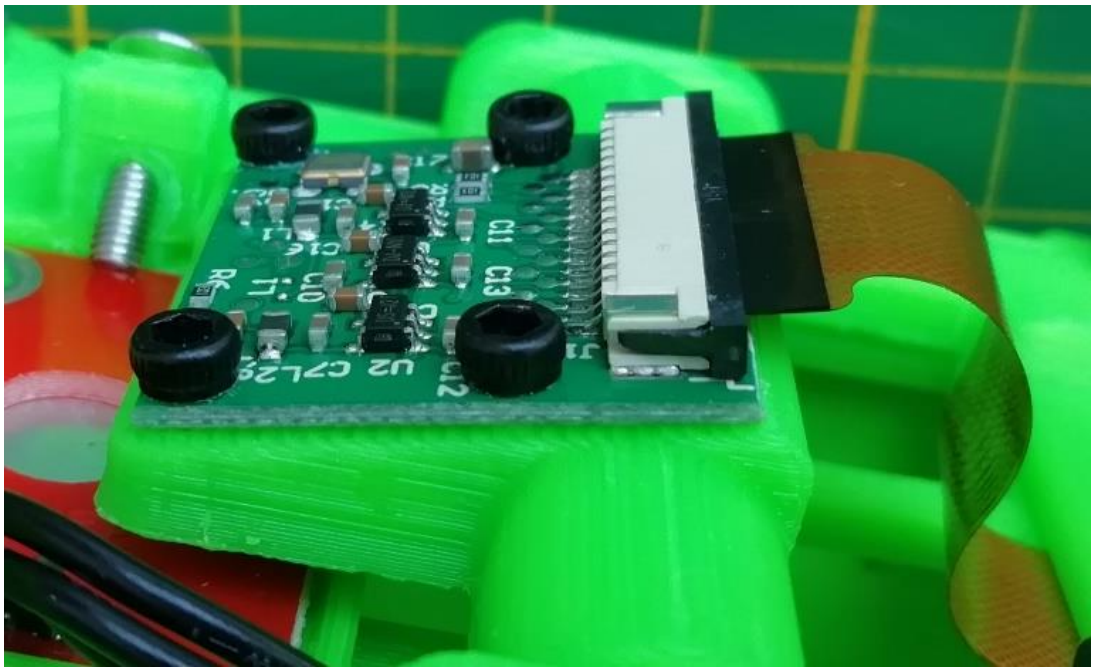
Necessary some little pieces of filament Ø1.75mm.

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



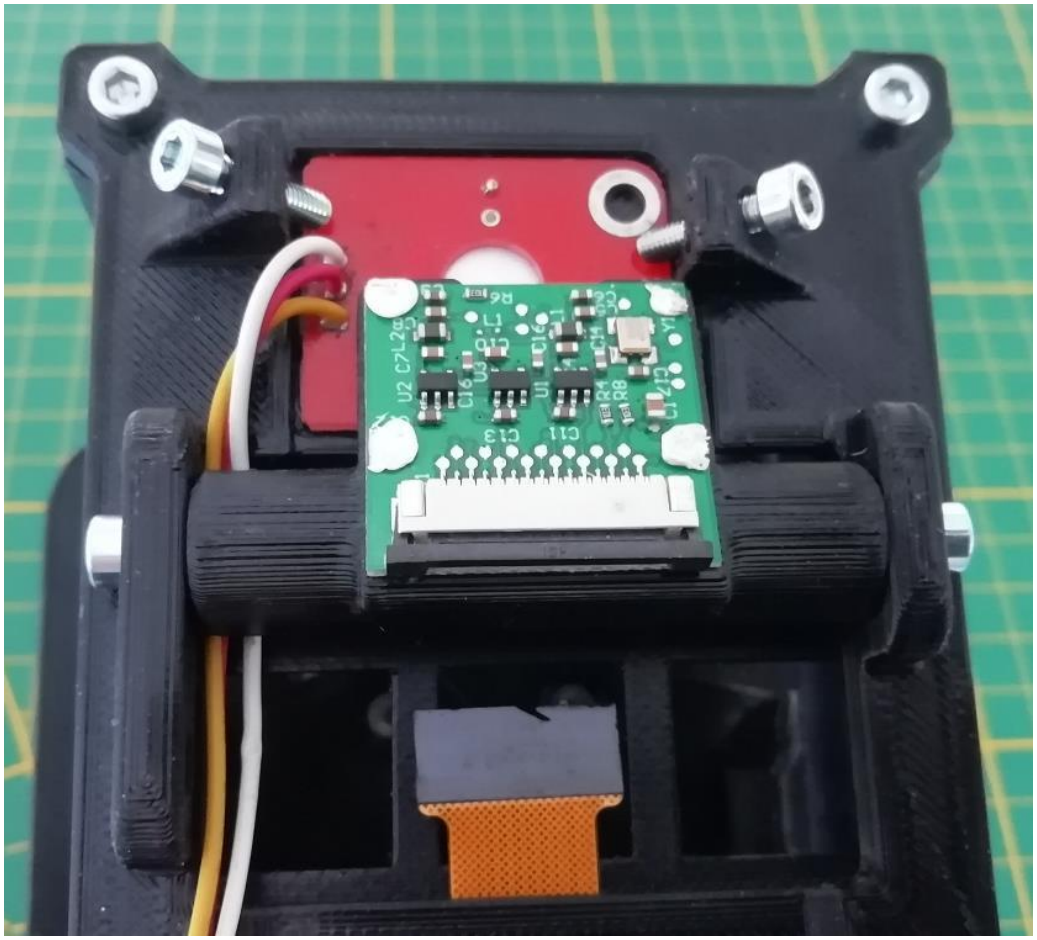
Alternatively, 4x M2x3mm screws can be used or 4x M2.5mm screws can be used.

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



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

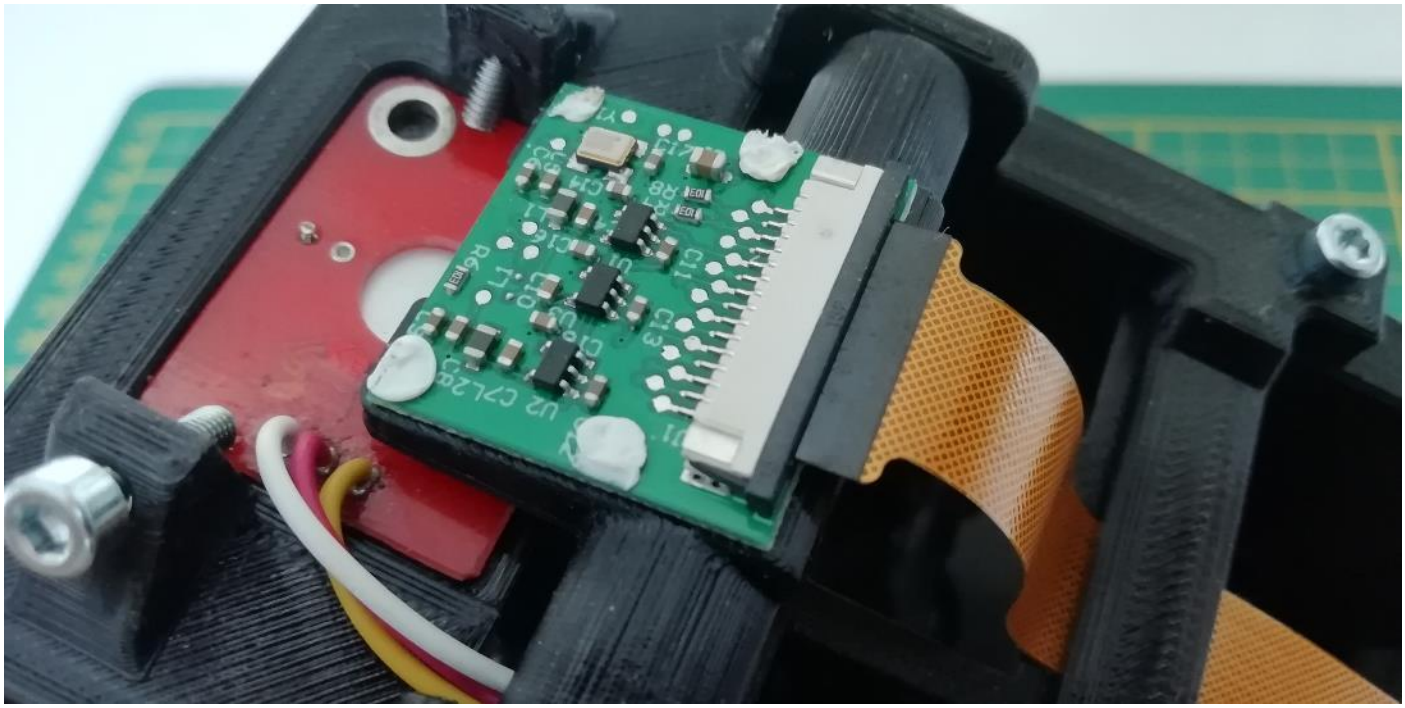
2x M3x12mm cylindrical head



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

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

Step27 (Connect the flex cable to the PiCamera module):



Step28 (Personalize the formal Stop plate):

The Structure has a recess, meant to hold a metal plate working as a Stop touch sensor on the Base version.
On this version, the stop function is available on the PCB_cover, making that recess quite useless and unesthetic.
I made available two alternative stl files of a plate to 3D printed as cover such a recess; One plate is neutral, the second one has ‘Magic CUBE’ words engraved.
Of course, you might consider using the neutral one as baseline to personalize your own Cubotino 😊.

17) Raspberry Pi 3 or 4

Cubotino_Top_version has been designed in late winter 2021, by considering the Raspberry Pi Zero 2 an easy to source and relatively cheap (ca 15$) component ….

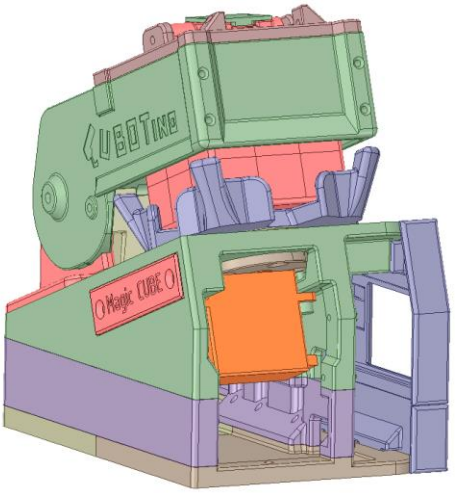
Unfortunately, the market situation has changed: Raspberry Pi Zero 2 boards are scarcely available and are gold priced.

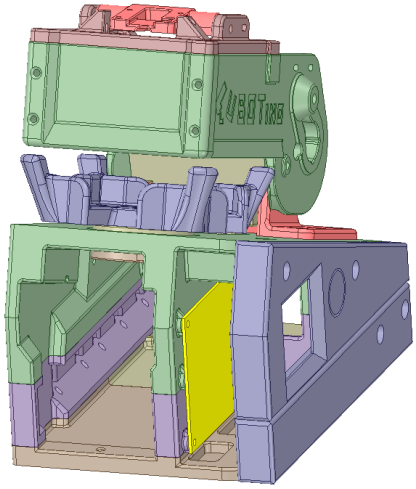
If you have a Raspberry Pi 3 or 4, a possible solution is to increase the robot height by 26mm to embed that larger SBC.
Some notes:
The connections_boards need to be slightly smaller, to prevent interference with some of the raspberry Pi 3 or 4connectors.

The height increment can be gained via 3 extension parts to be 3D printed and screwed to the other parts; This choice allows to shrink the robot in future, in case the smaller boards will be back reasonably priced again.

This approach has been already suggested with Rev. 2.2 on 18/06/2022, yet at that moment I hadn't any "large" Raspberry Pi board available, and I did not realize the interferences with the boards; On that period, I had very little time to work things out, and decided to remove the proposal in less than a week.

Below a few images of how the robot will look like:





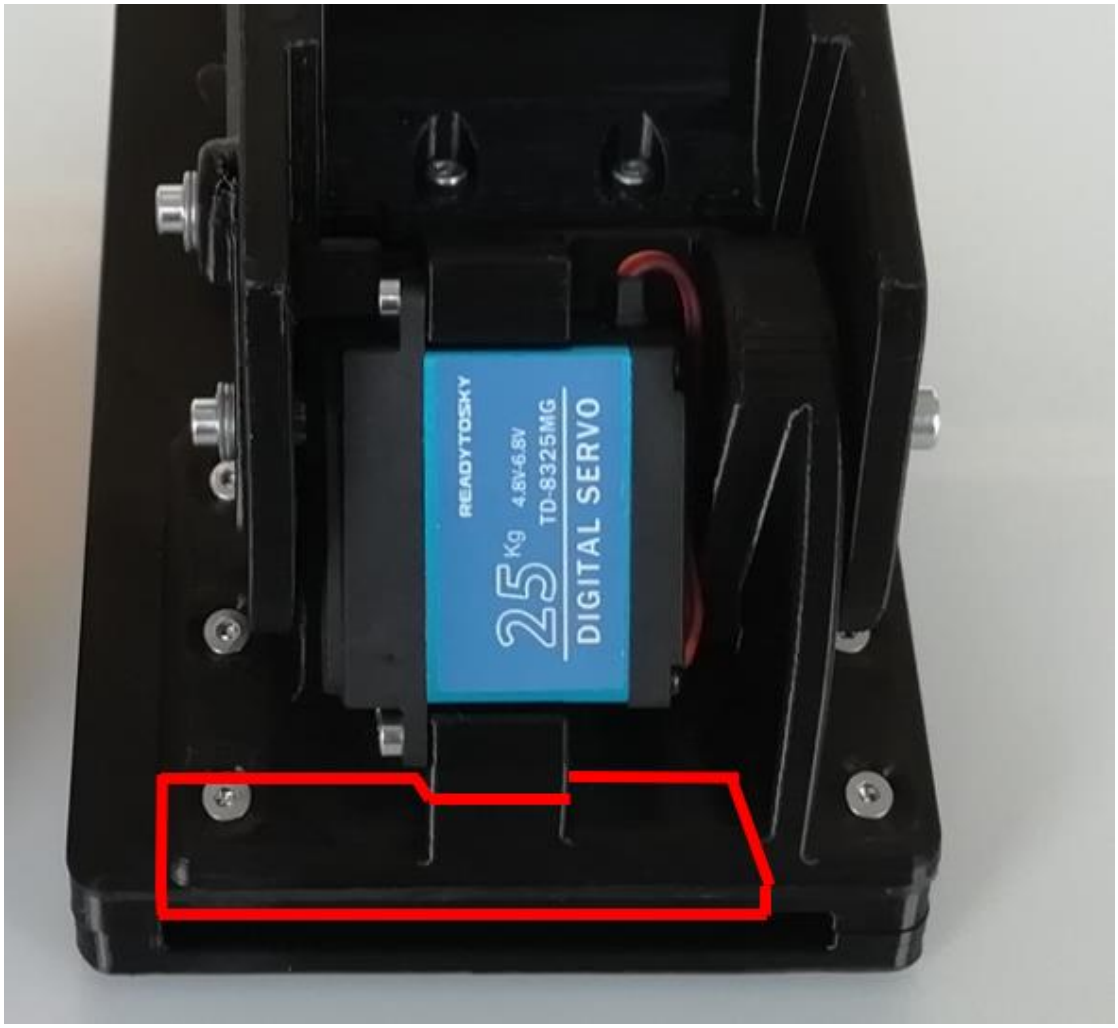
And a picture from nrdlrd Maker:

Raspberry Pi 3 and 4 have “standard” CSI camera, meaning the flex cable differs from the one at supply list for Raspberry Pi Zero (and Zero2).

Raspberry Pi 3 and 4 have the CSI camera port in a less convenient position than Raspberry Pi Zero: This makes the 30cm flex cable just enough …. meaning it might be just not enough.

The obvious option is to buy a longer cable, meaning the commercial 450mm or 500mm

In case you already ***have*** a 30cm cable that is just enough, and you’d like to reduce cable tension, you might consider modifying a couple of parts: Structure and Hinge



This is quite labour intensive if you file/cut the parts, or you revise the files and re-print.

The Hinge currently has 6 fixing screws, considering some screws might be less effective; The removal of one screw is clearly not a problem.
As said, the suggested solution is to buy a longer cable (400 or 450mm), yet this might be a more creative solution.

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

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

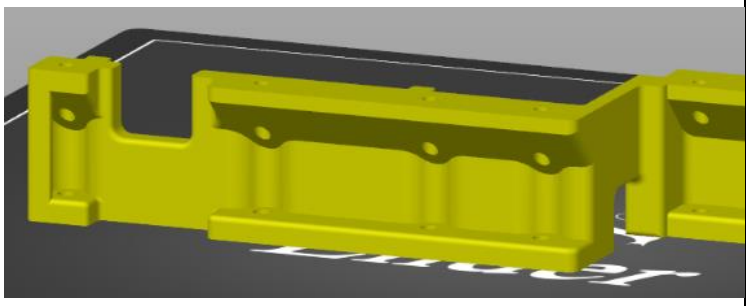
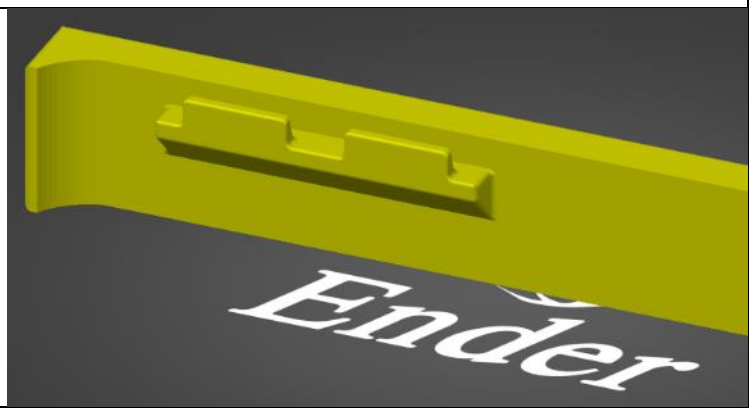
Filament quantity, and printing time …

Notes:

Still valid the notes on chapter 3D printed parts (i.e. no support needed).

Above Filament consumption and printing time are based on a quality printing setting; If these parts are really meant to be temporary in your case, a much lower quality could be considered.

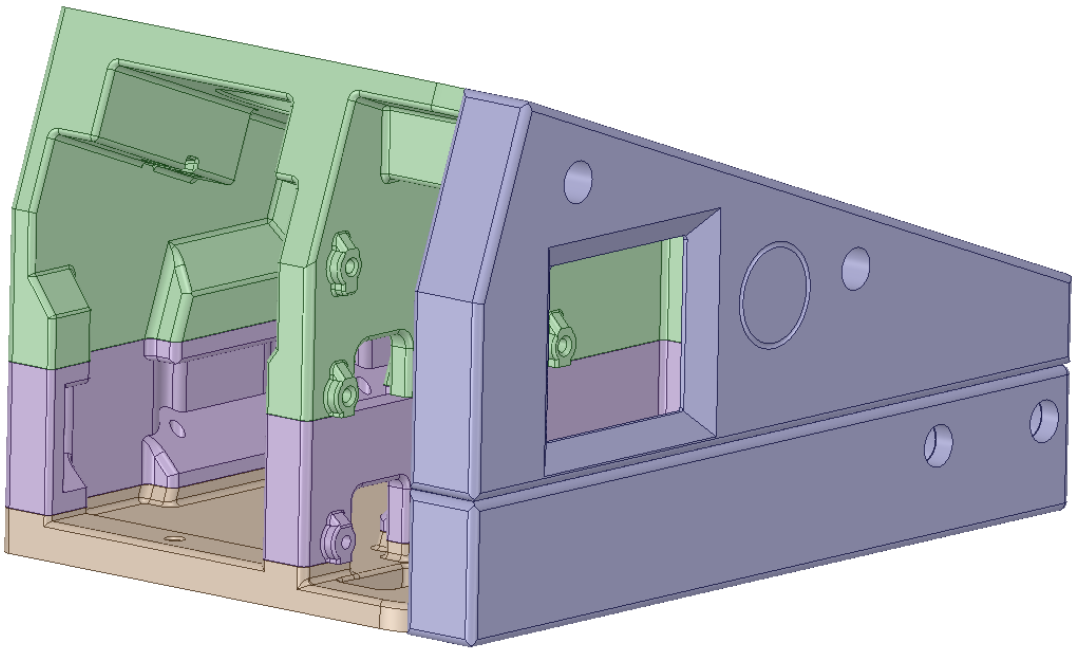
The suggested part orientation for the 3D print is showed on below Table.

Part name	3D print orientation
Extension_left	
Extension_middle	
Extension_right	

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

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

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



Assembly:

The three additional parts are connected by screws.

Use M3x12mm cylindrical, to connect the Extension_left and the Extension_middle toward the Structure.

Note: In case you don't have Allen keys with "spherical" head, to reach the screw under an angle, it might be necessary to use screws with conical head; M3 screws with conical head uses smaller Allen key, that can be used through the holes straight in front.

Use M3x12mm conical head, to connect the Extension_right toward the Extension_middle

Raspberry Pi 3b or 4b must be assembled by keeping the GPIO connector on top, like for the Raspberry Pi Zero 2

Do not tight much the screws of the board, as there might be small electronic parts of Rpi 3b or Rpi 4b in contact with the second unused screw seat for Rpi Zero 2.

The bottom_servo cable must be dressed through the recess between the Extension_middle and the Base_front.

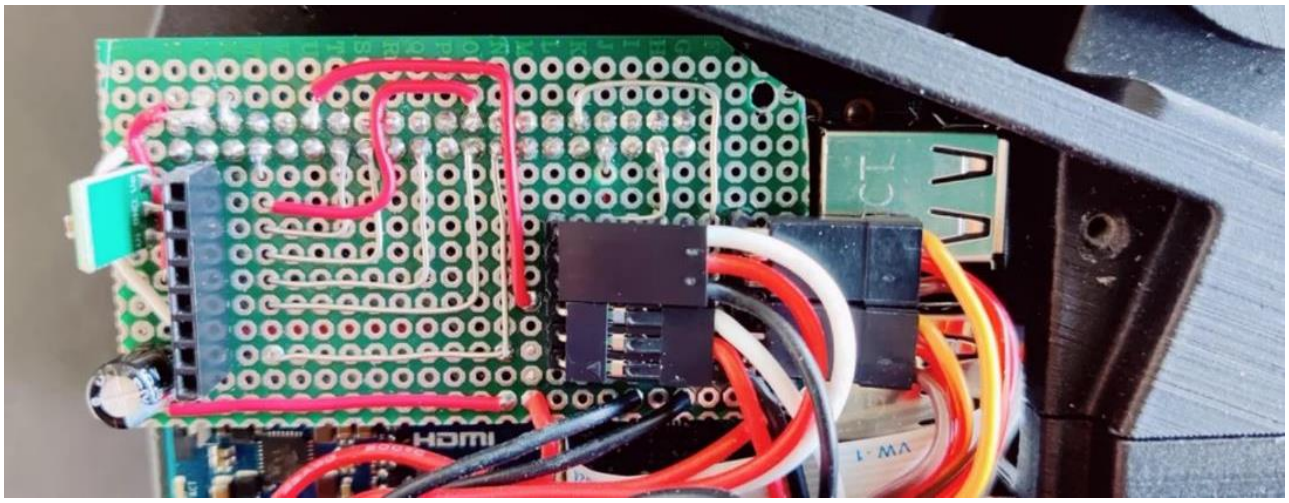
18) The Extension_left and the Extension_middle parts should be assembled after Step4 (of Assembly details chapter), therefore after assembling the bottom servo.

The Extension_right has to be assembled before assembling the Base_front and before assembling the PCB_cover_display.

-
- Adapting the connections**_board, to fit Raspberry Pi 3 (or 4) component layouts**
-
- If you make the connections_board, out of a proto-board, the important changes are:
- limit the protrusion of the proto-board, to the right side, to max 4 / 4.5 holes from the last used pin of the Raspberry Pi GPIO connector.

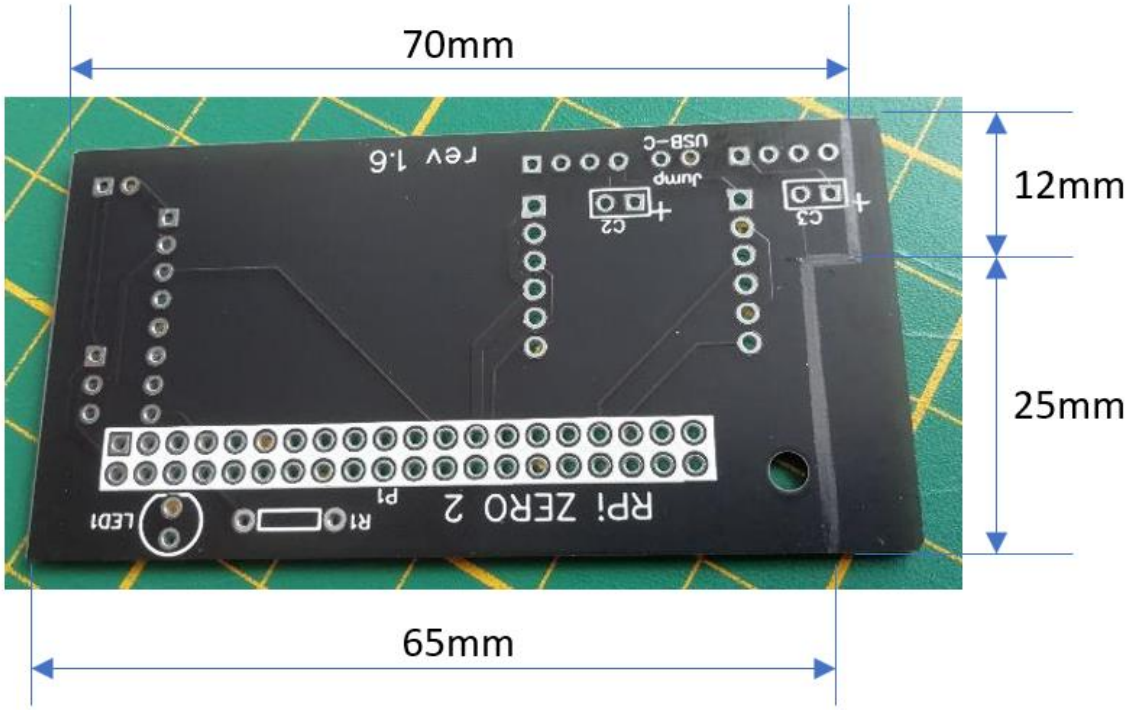
Solder the C3 capacitor directly to pins 1 and 2 of the (H6) header for the bottom servo.

Below an image from Maker nrdlrd, who had used a Rapberry Pi 3:

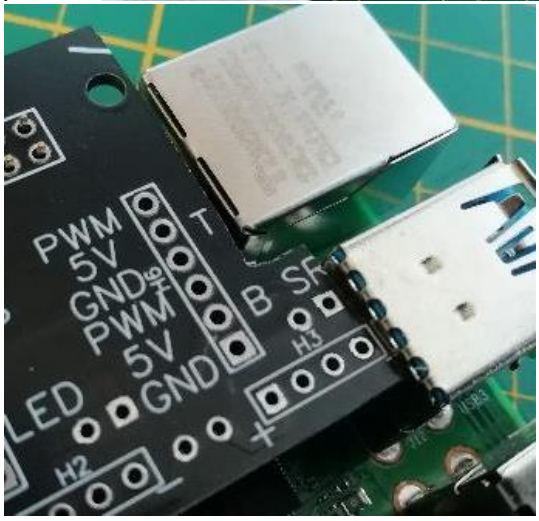


150.
151.
152.
153.
154. The V1.6 board can also be used, in combination with Raspberry Pi 3 or 4, by cutting out part of the right side.
155. In case of Raspberry Pi 3 the board can have a straight cut, to reduce the board length from 74mm to 70mm.

156. In case of Raspberry Pi 4 the board requires a slightly more complex cut:



157.



158.

159.

160. Notes

161. Solder the C3 capacitor directly to pins 1 and 2 of the (H6) header for the bottom servo

162. With reference to the PiCamera flexible cable, I'm not fully sure the 30cm version to be sufficient

163. The M3x16mm screw, meant to keep the display parallel to the Connections_board and the PCB_cover, need to be fixed to the display hole instead of the connections_board hole.

164. Verify the absence of short circuits between the connections_board and the Raspberry Pi 3 or 4 connectors, in particular for the C3 positive solder pad

165.

Tuning

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

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

servos

arm positioning to the servo

cube dimensions

print quality

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

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

General:

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

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

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

Servo angles, and servo timers

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 😊.

Setting servos angles:

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

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

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

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

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

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

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

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

198.

200.
201. Detailed info on servo management:
202. On Parameters and settings chapter are listed the involved variables and the default values.

The gpiozero library is used to control the servos, with a (float) parameter ranging from -1 to 1.

The float value entered on the '–set' argument (see Servos test and set to mid position chapter), represents a normalized rotation; The applied rotation is based on the Pulse Width range. Be considered your servos might have a different Pulse Width range…

Value **-1** is the max CCW servo rotation; The library sends the minimum Pulse Width to the servo.

19) Value 1 is the max CW servo rotation; The library sends the maximum Pulse Width to the servo.

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.

Changing from a smaller value to a larger one it results to a CW rotation of the servo outlet.

On servos with a Pulse Width ranging from 1 to 2ms, every 0.02 step in the "—set" argument determines a rotation of 1.8 degrees: -0.02 rotates the servo outlet of 1.8deg CCW, while 0.02 rotates the servo outlet of 1.8deg CW.

It is convenient checking the servo rotation range before the assembly, therefore prior to have mechanical constrains on the servo rotation

In case your servos don't make 180 degrees rotation, when the '—set' parameter is changed from -1 to 1 (or the other way around), it might be the case those servos have a Pulse Width ranging from 0.5ms to 2.5ms (default range considered is from 1 to 2ms).

In this case it is necessary to change the minimum and maximum pulse width parameter at the Cubotino_T_servo_settings.txt file (t_min_pulse_with, t_max_pulse_with, b_min_pulse_with, b_max_pulse_with) with values that best fit your servos.
Save the text file and re-launch the Cubotino_T_servo.py (pulse width parameters are uploaded when this script is started)
Re-check the servo angle range: If 180 degrees are now covered set the servos to their mid position.
In case the servo for the Cube_holder makes less than 180° rotation, despite you've tried to enlarge the Pulse Width range, it is necessary to increase the rotation range to slightly more than 180° in order to get the robot working properly.

There are tutorials in internet on how to increase the rotation angle, by adding some resistors in series with the servo potentiometer (servo must be opened for this eventual change).
2k2 Ω resistors were used by Jonesee (reference https://www.thingiverse.com/make:1034575).
Andy (search for G7UHN at https://www.instructables.com/CUBOTino-a-Small-Simple-3D-Printed-Inexpensive-Rub/#discuss) has reported 'my servos had only about 90° range of motion out of the box (despite being advertised as 180° servos) so I had to modify them, adding 3k resistors to both ends of the internal (5k) potentiometer'.

Top_cover (t_**servo) angles**:

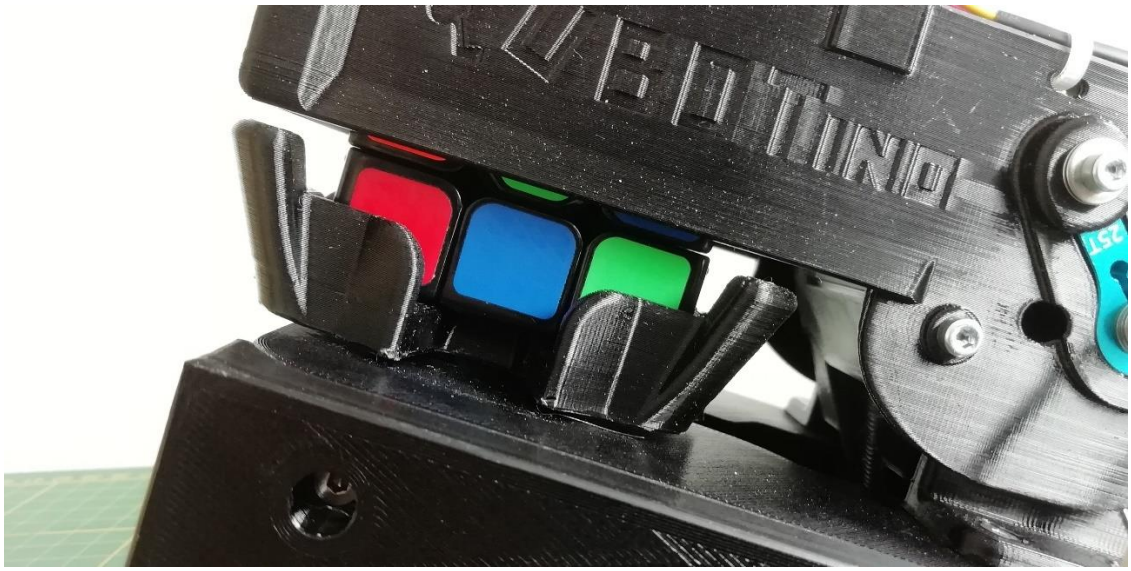
- Working angles for the servos, are set in a (json) text file: Cubotino_T_servo_settings.txt
- There are 5 defined angles (most of time mentioned as positions):
- Close: position to constrain the top and mid cube layers
- Rel: position to release tension, from cube, at Close position

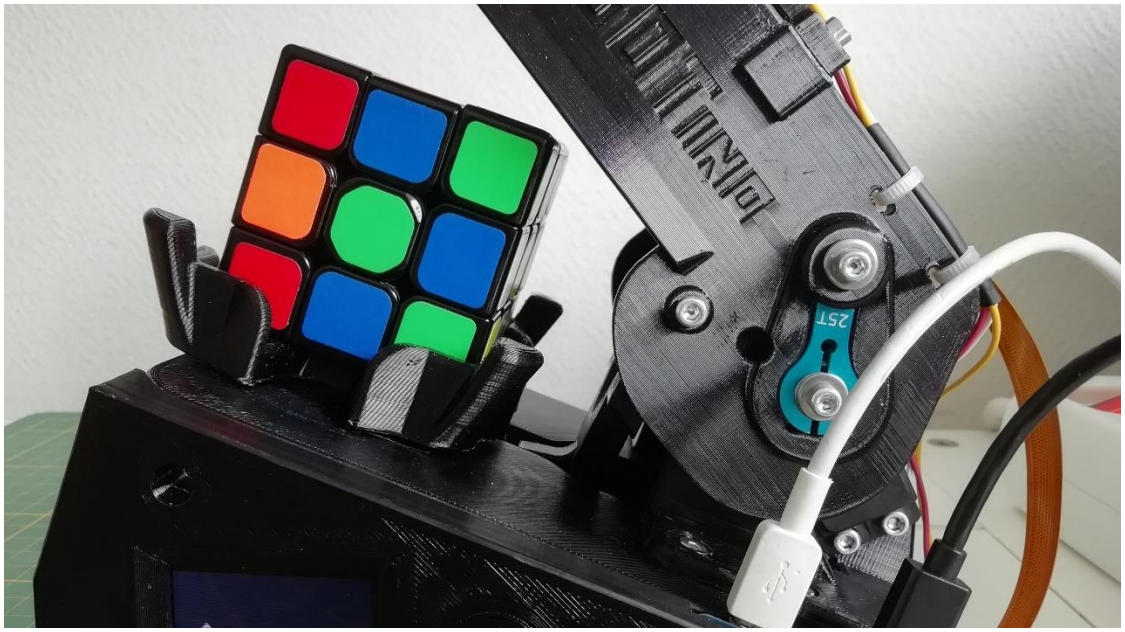
  Open: position without interferences with the cube and Cube_holder

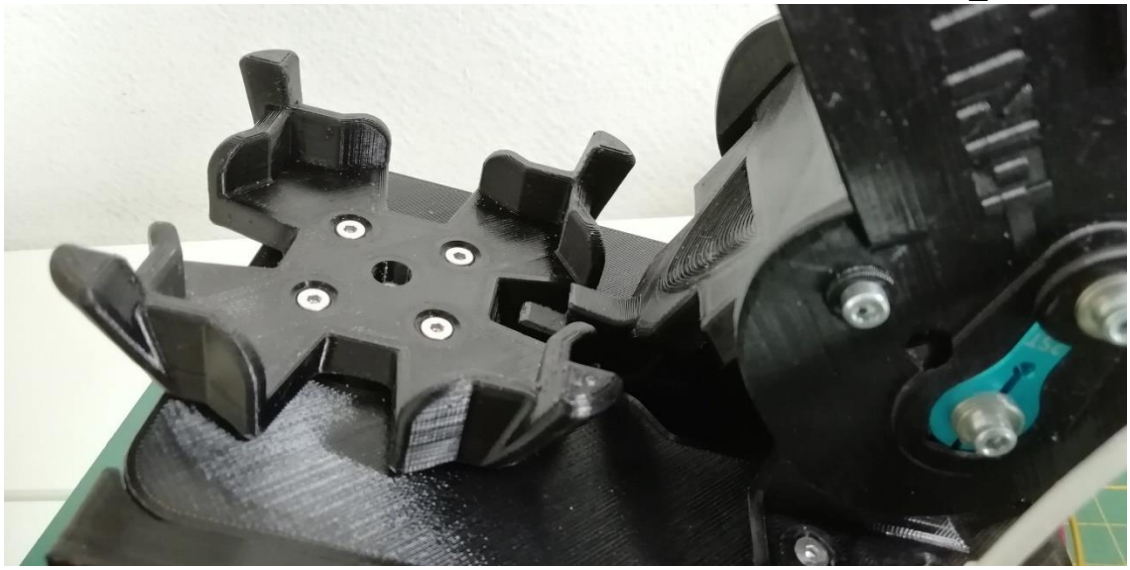
  Read: position for camera reading, with the Lifter almost touching the cube

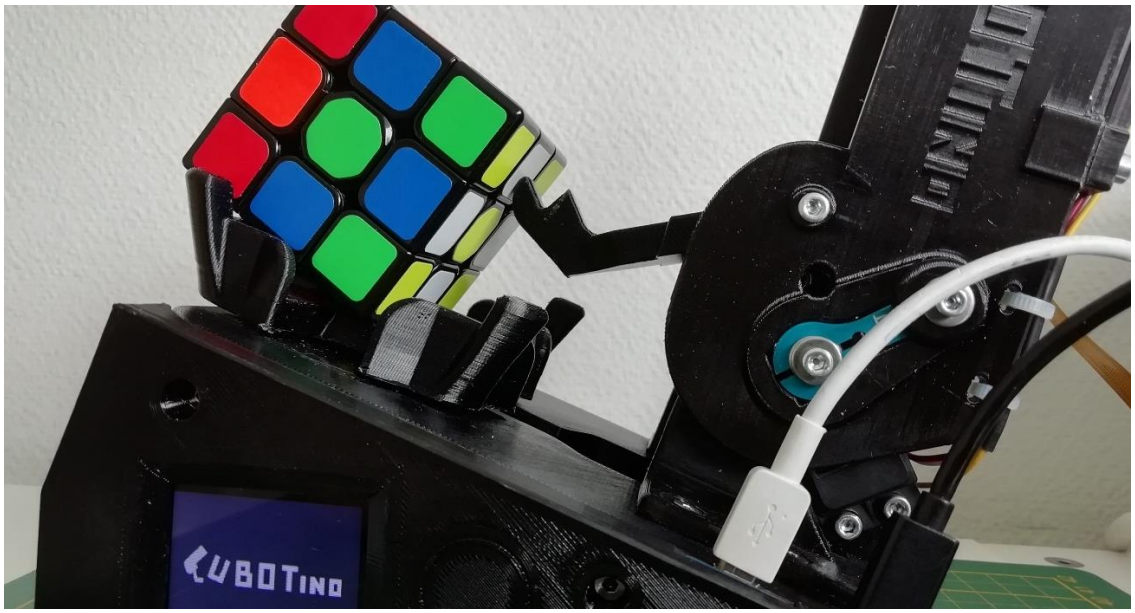
(and unfortunately constraining the Cube_holder)

Flip: position for the Lifter to flip the cube (about 2 cube layers height)








CUBOTino

Cube_holder (b_servo) angles:

There are 3 (+4) defined angles (most of time mentioned as positions):

CCW: position to spin or rotate the Cube_holder >90° CCW from Home

(Direction according to the motor point of view)

CW: position to spin or rotate the Cube_holder >90° CW from Home

(Direction according to the motor point of view)

Home: mid position between CCW and CW

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

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

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

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

The Home position has to be well centered.


FRONT

Fine **tuning servos angles (changed on 25th July 2022):**

set the Servos to the mid angle (see Servos test and set to mid position chapter)

assemble the robot

enter the cube folder (cd *cubotino/src*) and activate the venv (*source .virtualenvs/bin/activate*); Attention to the dot in front of virtualenvs

run the script *python Cubotino_T_servos.py –tune True* ; Attention to the space in between '-'

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

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

After the 'Enter command:' type the below commands to test the servos positions:

t_servo = t_servo_close

t_servo = t_servo_open

t_servo = t_servo_read

t_servo = t_servo_flip

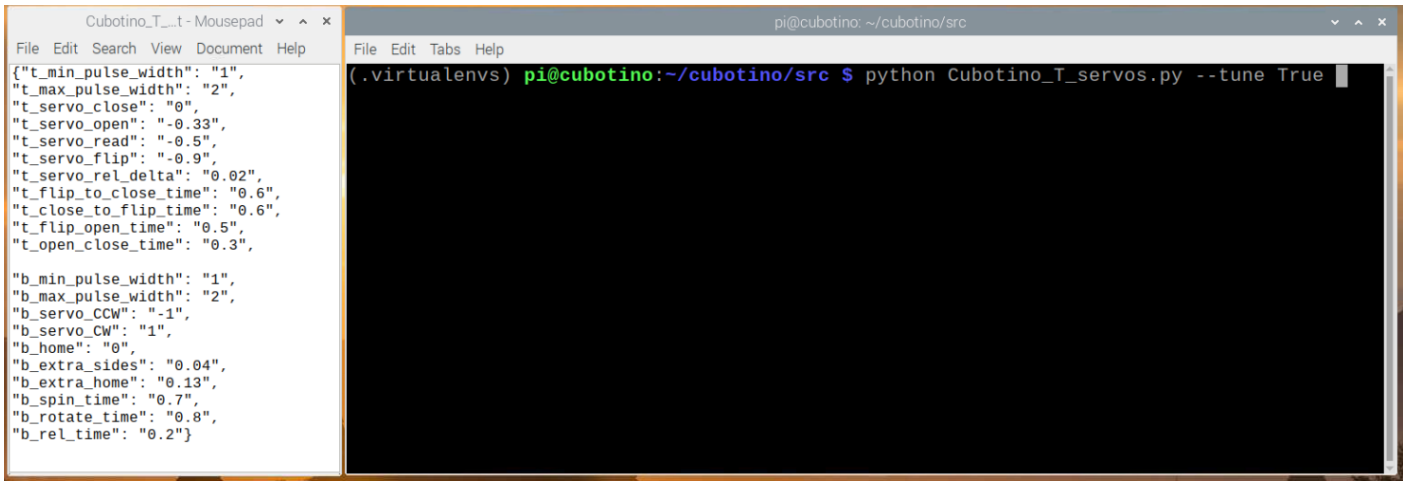
b_servo = b_home

b_servo = b_servo_CCW

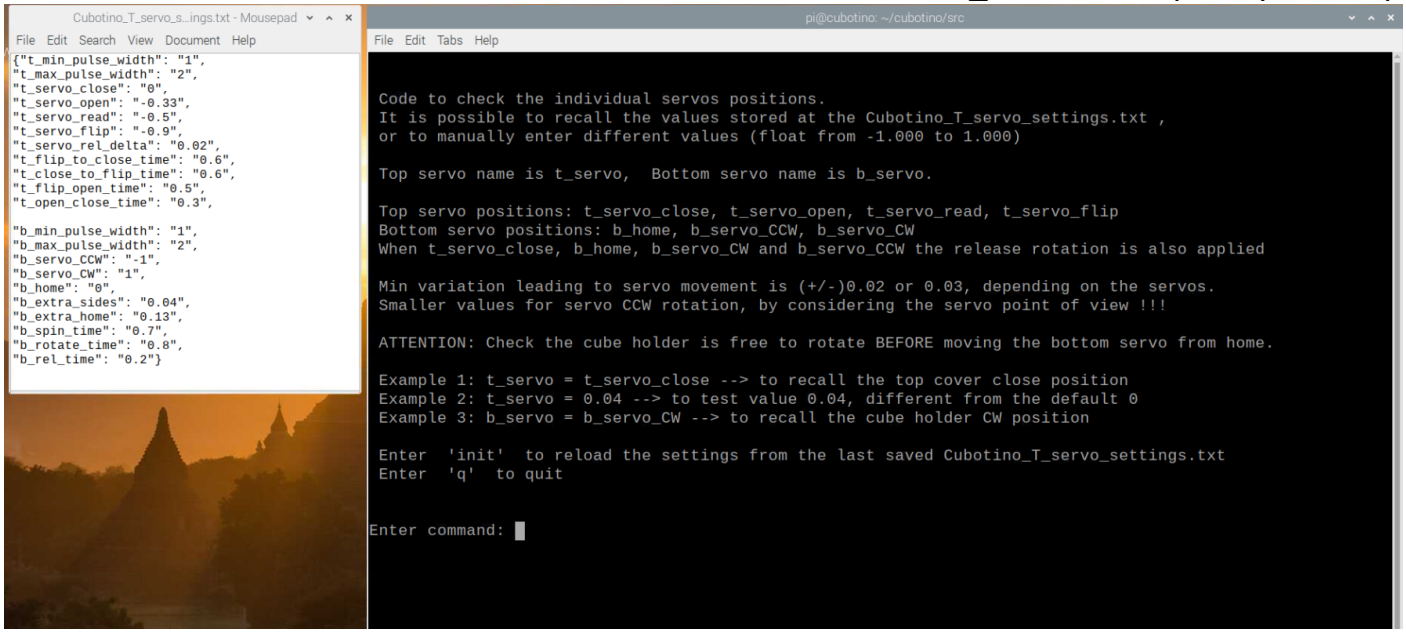
b_servo = b_servo_CW

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.

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



If you type init , instead of a command after the 'Enter command:', the python script reloads the new settings from Cubotino_T_servo_settings.txt, so to verify if everything goes well.



run the script *python Cubotino_T_servos.py* ,without any argument, to test the robot manoeuvring the cube like during a solving process; Take a close look to check if the cube handling is ok.

If the cube layers don't align well, if is suggested to apply some stickers on the cube holder: When the cube holder is home place an 'F' on the cube holder front side, 'L' on the cube holder left side and 'R' on the cube holder right side. Take a movie while the robot manoeuvres a cube and watch it back to see in which position the misalignment is generated.

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 to try 0.02 (0 + 0.02) by typing *'t_servo = 0.02'.*

In case the Top_cover is too close to the cube then the servo position requires to decrease the CW position (CW and CCW are frm the servo point of view).
To decrese 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 to try -0.04 (0 – 0.04) by typing
't_servo = - 04 '.

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

Timers for servos:

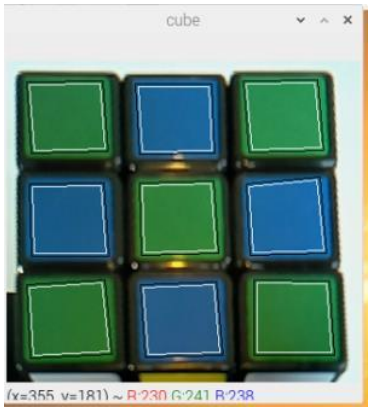
Servos don't provide feedback when they have completed the requested angular rotation; It's necessary to set appropriate waiting time in the script, to allow sufficient time for the servo to complete the action.
It will be convenient to use larger delays at the beginning, and progressively reduce them once the servo angles are adjusted to your system: The default values I've set for the default should be more than sufficient to allow the servos intended rotations.
Once your robot runs smoothly, and in case you' like to reduce the solving time, then you might start reducing those timers. As reference you can check on "Parameters and settings" for the values I'm using on my robot.
Timers for the servos, are set in a (json) text file: Cubotino_T_servo_settings.txt

Frame cropping:

Cropping parameters are set in a (json) text file: Cubotino_T_ settings.txt
PiCamera position, and its FoV, are likely to read both top and back cube faces.
Cubotino_T.py file is delivered with no cropping effect (variables set to zero): this to help the camera assembly angle to be set to get the cube top face centered: First picture below as example.

Image cropping has to 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:


cube

Frameless cube:

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

The cube facelets detection algorithm has been initially developed for the classic cubes, those having the black frame around the facelets.
Starting from 6th August 2022, it is also possible to use the frameless type of cubes
At Cubotino_T_settings.txt there is a “frameless_cube” parameter, with below options:
‘false’ for the classic cube type (default value)

‘true’ for the frameless type

‘auto’ for both types,

The ‘auto’ mode is computationally more demanding, therefore it will take slightly longer (about 1 to two seconds more for the six cube faces).
In the case the Cubotino_T_settings.txt doesn’t have the key “frameless_cube”, because you are re-using an old setting files, then the new Cubotino_T.py file version will simply consider the classic cube type.

You might argue on the reason ‘auto’ is not the default choice for the cube facelets detection method.
The reason is that the used strategy for the frameless cube is less robust.
When facelets_cube is set ‘true’ or ‘auto’, as soon as there are at least 5 detected facelets, without an empty row or column, the remaining facelets will be estimated on their position.

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

Examples of estimated facelets:

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

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

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

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

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

Display initialization and test:

For installation up to 6th August 2022, the display was initialized twice; This wasn't pleasant to be seen, and potentially leading to hardware race by the code
Starting from 6th August 2022, thanks to Yannick solution, the display is initialized once, with better display management.
The new solution requires one more file: Cubotino_T_display.py to the project.
The new file can be used to test the display after its connection:
enter the cube folder (cd ~/cubotino/src)

activate the venv (*source .virtualenvs/bin/activate*); Attention to the dot in front of virtualenvs

run the script *python Cubotino_T_display* ; Attention to the space in between '-'

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

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

PiCamera focus

The V1.3 PiCamera is delivered with fixed focus.
On my first project with PiCamera I did adjust the camera focus, by following this tutorial
https://projects.raspberrypi.org/en/projects/infrared-bird-box/6
It hadn't been easy, yet I managed.
After correcting the focus, I did not glue the lens, as there still was quite some friction, preventing the lens from getting loose.

When I tried to repeat this operation on a second PiCamera I have almost destroyed it.
Both my Cubotinos have PiCamera "as received", one Cubotino has okish focus while the second one is out of focus.
In both cases the robot work properly, the only difference is the sharpness on the images the robot saves.
In case of non-satisfactory results, and if you are good with your hands, you might decide to try … on your own risk.

Parameters and settings

Parameters that are more likely to differ on each system, are into two json files:
Cubotino_T_settings.txt and Cubotino_T_servo_settings.txt

In order to provide a reference, the below json files capture the settings used on my robot:
Cubotino_T_settings_AF.txt and Cubotino_T_servo_settings_AF.txt

On below tables are listed these parameters, with the proposed value to start the tuning, the value that work on my Cubotino, and some information.
Highlighted the default settings that differ from mine

Cubotino_T_settings.**txt (and Cubotino_T_settings_AF.txt), part 1**

Parameter (dict **key)**	Default **value**	AF **value**	**Data type**	**Info**
frameless_cube	false	auto	string	Set the facelets edge detection according to the cube. Options are: 'false', 'true' and 'auto'. If the parameter key is missed (i.e. old setting files) a classic cube is considered, meaning frameless_cube = false. frameless_cube = true should be used when small logos on the cube.
disp_width	130	132	Int	Display width (in pixels); At troubleshooting for Display the explanation to make easy to adjust this parameter this parameter
disp_heigh	160	162	Int	Display height (in pixels); At troubleshooting for Display the explanation to make easy to adjust this parameter this parameter
disp_offsetL	0	-2	Int	Display offset on width **Left** (in pixels); At troubleshooting for Display the explanation to make easy to adjust this parameter this parameter
disp_offsetT	0	-2	int	Display offset on height **Top** (in pixels); At troubleshooting for Display the explanation to make easy to adjust this parameter this parameter

Cubotino_T_settings.txt **(and Cubotino_T_settings_AF.txt),** part 2

Parameter (dict **key)**	Default **value**	AF **value**	**Data type**	**Info**
x_l	0	60	Int	Image cropping at the left, before warping (in pixels). This is meant to removes a slice of the image at the **left** side, external to the cube image. around the bot, and it will increase speed.
x_r	0	80	Int	Image cropping at the right, before warping (in pixels). This is meant to removes a slice of the image at the **right** side, external to the cube image. around the bot, and it will increase speed.
y_u	0	0	Int	Image cropping at the top, before warping (in pixels). This is meant to removes a slice of the image at the **upper** side, external to the cube image. around the bot, and it will increase speed.
y_b	0	110	int	Image cropping at the bottom, before warping (in pixels). This is meant to removes a slice of the image at the **bottom** side, external to the cube image. around the bot, and it will increase speed.
warp_fraction	7	7	float	Image warping index. This parameter is used to alter the perspective from the cube face images. Smaller values increase the effect, meaning it applies a larger variation to the camera image.
151				Image cropping index, that crops the right side of the image after

Cubotino_T_settings.txt **(and Cubotino_T_settings_AF.txt), part 3**

sv_max_moves	20	20	int	Max number of moves requested to the Kociemba solver. When the solver finds a solution matching this movement quantity, that solution is returned even before the timeout expiration (sv_max_time).
sv_max_time	2	2	float	Timeout, in seconds, for the Kociemba solver. The best solution found within the timeout is returned at timeout end, even if it doesn't match the desired max quantity of moves.
collage_w	1024	1024	int	Image width for the unfolded cube file. This parameter determines the image collage realization, and it makes possible to save all images with the same size.
marg_coef	0.1	0.06	float	Defines the margin around the cube faces images. This margin is used to cut the six cube faces with some margin around, for the unfolded cube collage. The margin is calculated by multiplying the detected cube diagonal in pixels to this coefficient. The larger the value the more pixels margin around the cube 0.1 means the margin is 10% of the cube diagonal (calculated on the detected facelets contours).
cam_led_bright	0.1	0.1	float	PWM for the 3W led at Top_cover. Range from 0 (no PWM) to 1 (PWM=100%). At the Cubotino_T_servos.py the max value really delivered to the LED is 30%, therefore values > 0.3 are useless.
detect_timeout	40	40	int	Timeout, in second, for the cube status detection. If the six cube faces aren't detected within this time, the cycle is terminated with a timeout message on the display.

Parameters related to the servos;

Notes:

‘t_’ refers to Top_servo while ‘b_’ refers to Bottom_servo

“Angles’ are in gpiozero range for the Servo class (range from -1 to 1, with 0 as mid angle)

Time is in seconds

Cubotino**_T_servo_settings.txt (and Cubotino_T_servo_settings_AF.txt):**

Parameter (dict **key)**	Default **value**	AF **value**	**Data type**	**Info**
t_min_pulse_width	1	1	float	Min pulse width, in ms of the used top servo. Most of the servos accepts a slightly extended value (<1)
t_max_pulse_width	2	2	float	Max pulse width, in ms, of the used top servo. Most of the servo accepts a slightly extended value (>2)
t_servo_close	0	0	float	“Angle” for Top_cover to constrain the top and mid cube layers
t_servo_open	-0.33	-0.33	float	“Angle” for Top_cover not constraining the cube and Cube_holder
t_servo_read	-0.5	-0.5	float	“Angle” for Top_cover for PiCamera reading. Lifter almost touching the bottom cube face

Note: On Cubotino_T.py and Cubotino_T_servos.py, the string '#(AF ' is placed as comment start, where the above listed parameters are used.

This because those variables weren't initially collected in a json file; Later I decided to comment those rows instead of cancelling them.

Troubleshooting

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

Servos rotate about 180 deg, yet not more than that.

Servos rotation angle of about 90 deg, when you've ordered 180deg

Servos rotation angle of about 270deg

Servos not moving smoothly

Bottom cube layer doesn't align nicely

Top cover usage to flat the cube

Cube status detection error

Robot stuck on reading the same face

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

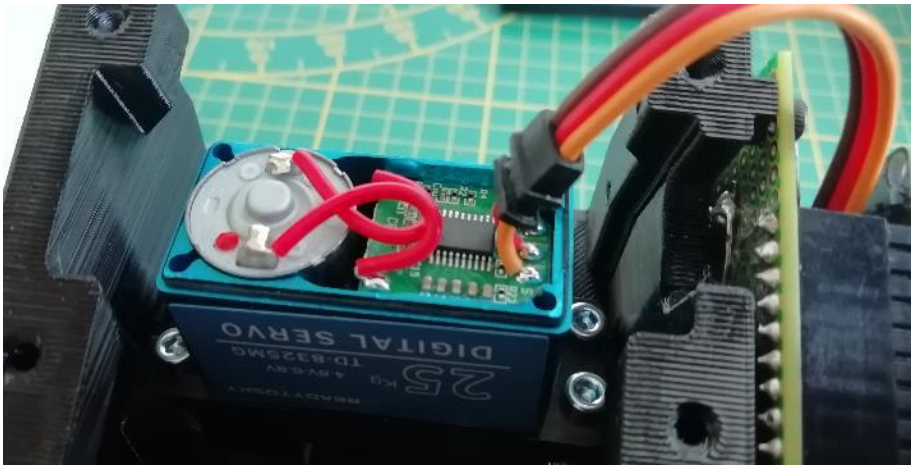
Displayed text and images are un-readable

Program doesn't work as intended

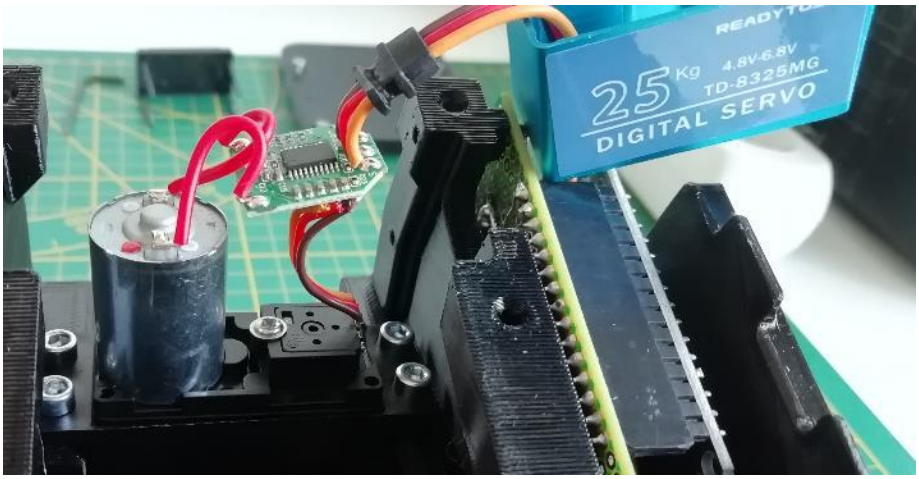
20) PiCamera focus

Increase the servo rotation **range (if needed)**

- It's possible to increase the rotation angle, by adding one or two resistors in series with the servo potentiometer (servo must be opened for this change). In my case I had 1 servo (out of 8 used so far) with insufficient rotation range.
-
- Open the servo (4 very long screws)

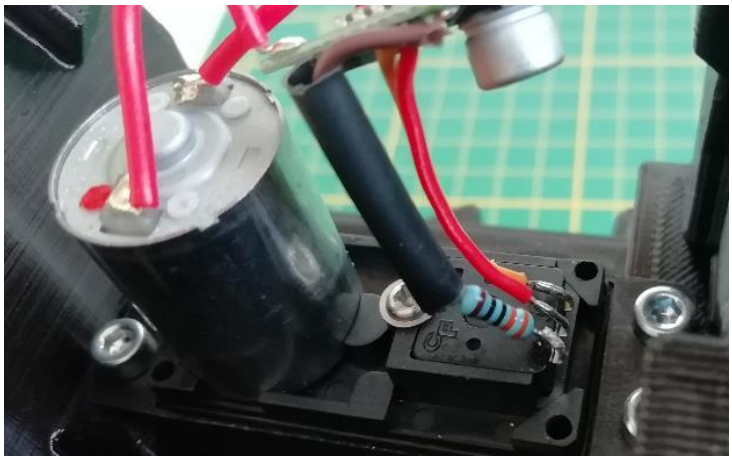


-
-
- Slide out the pcb, and afterward slide the case out of the way.

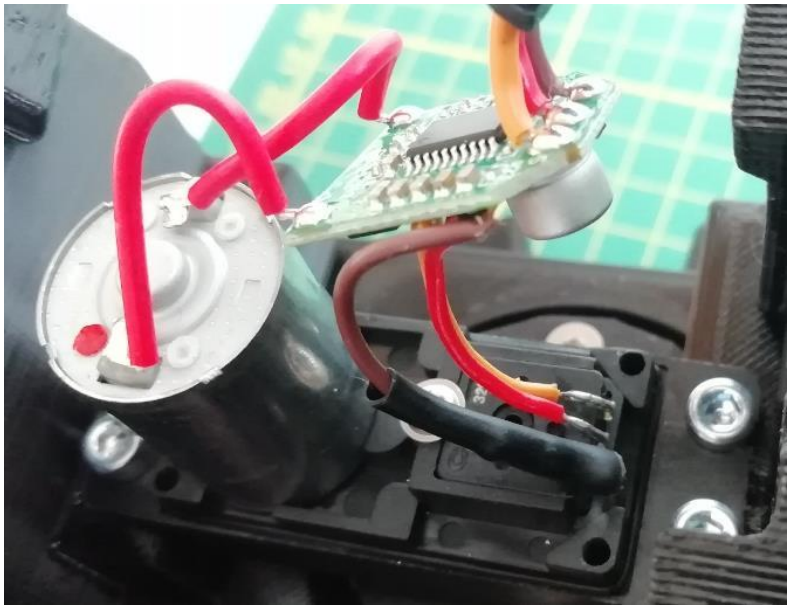


-
-
- Solder one resistor to one external pin of the potentiometer (or one resistor per each of the two external potentiometer pins to keep the same PWM value for the mid position).

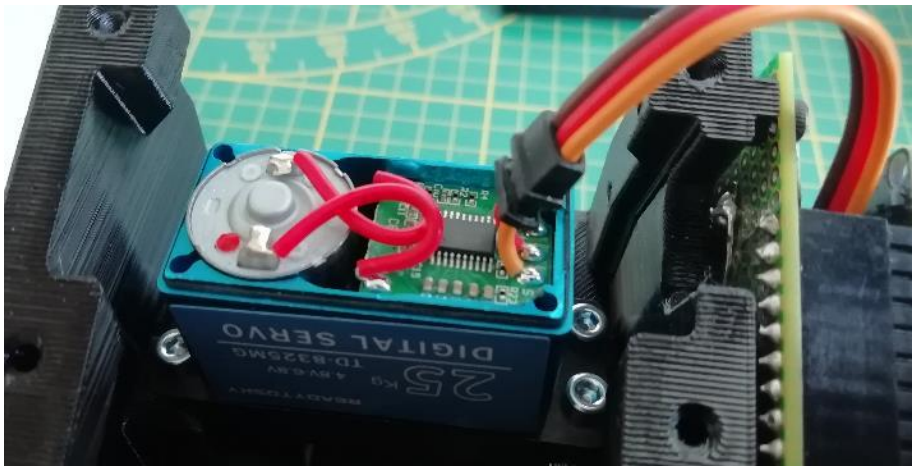
I've added 330ohm to gain about 5 degrees on a servo having pulse width from 1 to 2 ms.



Protect the soldered parts with a sleeve



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



203.
204.
205. Place the cover and close the 4 screws

207.

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

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

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

211. b_min_pulse_width from 1 to 0.5 (meaning the min pulse width reduces from 1ms to 0.5ms)

212. b_max_pulse_width from 2 to 2.5 (meaning the max pulse width increases from 2ms to 2.5ms)

213. t_min_pulse_width from 1 to 0.5 (meaning the min pulse width reduces from 1ms to 0.5ms)

214. t_max_pulse_width from 2 to 2.5 (meaning the max pulse width increases from 2ms to 2.5ms)

215.

216.

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

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

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

220. After adjusting the Pulse Width parameters, in case servos 160ren't from 1ms to 2ms, the default values for servos positions should be scaled by a 0.67 factor, to scale down from 270 to 180 degrees of rotation.

Servos don't move smoothly

Don't use jumper wires, or use quality jumper wires

221. Don't use bread boards, make the Connections board instead.

222. Add the capacitors, to prevent voltage drops when servos are activated.

223. Use an at least 2A power supply for the servos.

224. Use a 20 to 25Kg/cm servo.

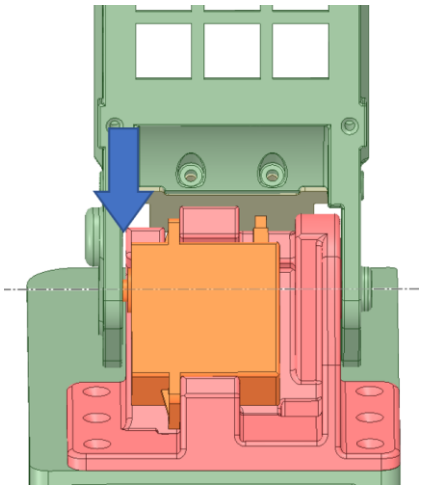
225. Minimize Top_cover rotation friction:

226. Ensure the hole for the M4 screw (pivot) has some gap on the Top_cover hole (Ø4.1 to Ø4.3mm).

227. Rub some candle wax on the screw.

228. Ensure the M4 screw head is not pushing toward the Top_cover; 1mm gas is suggested

229. Ensure gap presence between Top_cover inner surface and the Hinge at the servo gear outlet side, as per below arrow:

230. 

231. Bottom cube layer doesn't align nicely:
232. This is probably the most difficult part of the tuning process.
233. 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.
234. Verify if the cube Holder makes an extra rotation, at both CCW and CW directions, before stopping; If this doesn't happens:
235. 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.
236. Adapt the PWM release CCW/CW value.
237. Place the PWM release CCW/CW at zero, and test if the CCW and CW position have a slightly overstroke from the 90°. If this is not the case, check if the other servo has a larger rotation range. If still not the case:
238. 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)
239. Check in internet how to (slightly) increase the servo rotation angle (additional resistors must be soldered into the servo)
240. Verify if the cube Holder makes an extra rotation, before stopping Home; If this doesn't happen, adapt the PWM release home value.
241.
242.
243.
244. Top cover usage to flatten the cube:
245. The Top_cover isn't intended to keep pushing the cube when it's in the close position; In case the cube layers don't align nicely, by playing with the cube_Holder settings, it's possible to use the Top_cover to level the cube. By lower the Top_cover close position to have a little interference with the cube, will improve the cube layer alignment in particular after flipping the cube. In this case it will be convenient to set one or few units on *PWM release from close setting.* Via this setting is possible to release the tension between the Top_cover and the cube, after pressing it, to allow the cube_Holder to rotate with less effort.

Cube detection error:

It is returned when the interpreted cube status isn't coherent, meaning not having 9 facelets per colour or other inconsistencies. Possible causes:
Objects **on** the table (background); Objects on the table can form square like contour, interpreted as facelets by the cv. This can be solved by positioning the robot to a uniform-coloured surface, without cables and objects in for 30cm around the robot. Another good way to solve this problem is to tune the cropping parameters.

21) 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.

Too little light conditions cannot be compensated by the LED light source.

In case the cube has some prints (i.e. brand) , typically on the white center, it is suggested to carefully scratch out.

In case a frameless cube type is used (facelets without the black frame around the facelets), while at Cubotino_T_settings.txt the frameless_cube parameter is not 'true' or 'auto'.

The setting ‘auto’ to detect the status on cubes with and without the frame works better with good light conditions; If this isn’t possible, set the parameter to the specific type of cube associated to the robot.

Robot stuck on reading the same face, until timeout:

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)

When the frameless_cube is set 'true' or 'auto', the cube status detection algorithm must find 7 facelets with given characteristics before changing face; The remaining two are estimated for the position, not the colour.

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.

When the ambient light is rather hight 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.

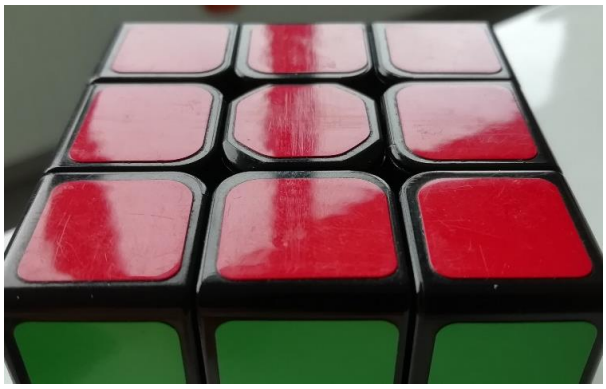
To troubleshooting is important to visualize what the camera sees; This is possible via below steps:
1) Connect to the Rpi via VNC Viewer.
2) If the robot has automatically started at the boot, two processes need to be killed as per "How to operate the robot" Step8.
3) Resize the terminal to no more than half screen, and move it to the right part of the screen.
4) Run the script from the terminal
4_1) *cd ~cubotino/src*
4_2) *source .virtualenvs/bin/activate*
4_3) *python Cubotino_T.py*
5) Press start to let the robot working, and a windows will show what the camera sees.

A contour will be drawn, over the camera image, on every location interpreted as 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 camera_support, to have margin around the top cube face.
B) facelets on the back cube side are also detected (detected means that on the image contours are drawn on the back cube facelets): Apply the cropping as explained for "frame cropping" in the "Tuning" chapter
C) the critic face has a logo on the central 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.

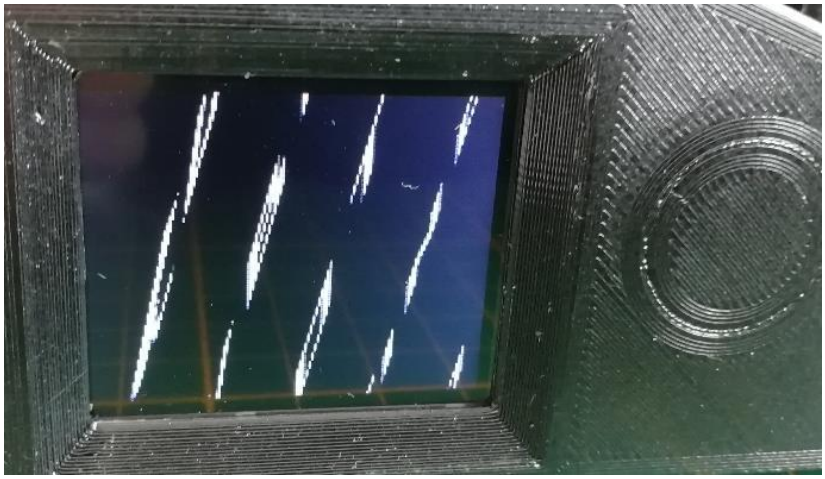
Cube's facelet **and light reflection (cube status detection):**

- Detection of edges, as well as colours, can be largely affected by light reflection made by the facelets.
- I have two cubes available, one with in-moulded coloured facelets, and the other with glossy stickers.
- On the cube with plastic facelet, I made the surface matt by using a fine grit sandpaper (grit 1000); This makes the cube status detection much more unsensitive to the light situations.



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

Display parameters are set in a (json) text file: Cubotino_T_settings.
The display I've received, from the supplies list, wasn't working as intended:



I took me some time to realize a sort of drifting…. And to find the fix.
By slightly changing the dimensions of just a couple of pixels on the display width parameter (in my case I had to set 128 instead of 130 pixels), the display was working.
Once that problem was solved, a second one became apparent: On two sides there were some 'tiny death bandwith', again for just a couple pixels.
By checking the ST7735.py code, I discovered it provides offset parameters, for the two display directions, suggesting these issues to be well known…..
By playing with both the display dimensions and the offsets, the display start working simply fine.
I don't expect all the display to have the same issue, yet in case the parameters variables are already in the code, and ready to be properly tuned.

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 feedbacks from the script:
Edit Cubotino_T.py

At __main__ change the Boolean "debug" to True. This variable is used by many functions to print out info to the terminal.

Run Cubotino_T.py

Check the prints

If the print out doesn't suggest much to you, share it at the Instructable chat

When the problem seems more related to the servo program:
Edit Cubotino_T_servos.py

At about row 85 change the Boolean "s_debug" to True. This variable is used by many functions to print out info to the terminal.

Run Cubotino_T_servos.py (activate the venv first, and recall to type python in front).This code activates the servos like solving a predefined scrambled cube.

Check the prints.

Run Cubotino_T.py and let it call the Cubotino_T_servos.py.

Check the prints

If the prints out don't suggest much to you, share it at the Instructable chat

PiCamera **focus**

The V1.3 PiCamera is delivered with fixed focus.
Until now, end of October 2022, I haven't received negative feedback from other Makers because of non-working Cubotinos caused by of focus PiCameras.

On my first project with PiCamera I did adjust the camera focus, by following this tutorial
https://projects.raspberrypi.org/en/projects/infrared-bird-box/6
It hadn't been easy, yet I managed.
After correcting the focus, I did not glue the lens, as there still was quite some friction, preventing the lens from getting loose.

When I tried to repeat this operation on a second PiCamera I have almost destroyed it.
Both my Cubotinos have PiCamera "as received", one Cubotino has okish focus while the second one is out of focus.
In both cases the robot work properly, the only difference is the sharpness on the images the robot saves.
In case of non-satisfactory results, and if you are good with your hands, you might decide to try … on your own risk.

How to operate the robot

Before starting:

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

Use a **uniform-coloured table**:

Part of the table around the robot will be captured on cube face images.
Keep some free space around, and use a uniform-coloured base, to prevent cables or other objects to be eventually detected as facelets.

Power up the **servos**:

This considers an independent power supply for the servos
Connect a 5V power source to the microUSB connector where the servos are connected.
In my case the servos work flawless with a phone charger rated 2A, but I had inconsistent movements when using a phone smart-charger rated 2A, and with a normal phone charger rated 1A.
Notes
Do not energize the servos once the SBC is up and running, if you aren't sure whether the cube holder is positioned to home.

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

Power up Raspberry Pi Zero 2:

Connect a 5V power source to the remoted microUSB connector where Raspberry Pi is connected.
Do not connect phone smart-charger, those that can deliver a voltage higher than 5.1V.
In my case the SBC works flawless with a phone charger rated 1A (note the official documentation suggest higher current).

Start **a solving cycle:**

Position the cube on the cube holder; any cube orientation is accepted.

Cube layers should be reasonably aligned.

Shortly touch the PCB_cover in front to the touch sensor (the circle suggests the touch sensor location).

The robot reacts by energizing the LED, and by indicating CAMERA SETUP on the display.

Raspberry Pi shut down:

Please be noted Raspberry Pi, like normal PC, cannot be unpowered when it is working.
To shut it down, there are few possibilities:

Connect to the Raspberry Pi via SSH, and type *sudo halt -p*

The SBC closes the open applications and files
When the SBC activity is almost done, the led at Connections board will goes off
Wait additional 10 seconds and the power supply can be safely removed.

If the robot has proved to work without errors, un-comment last row at Cubotino_T_bash.sh file (*halt -p*).

This means the SBC will automatically shut down when the Cubotino_T.py file ends.

In order to quit the *Cubotino_T.py* script, touch the Touch_button long enough (ca 6 seconds) until the 'SHUT DOWN' appears on display; The SBC will close the open applications and files.
Differently, iIf the touch sensor is released as soon as the display shows 'SURE TO QUIT?', then the robot will consider it as a stop request instead of a shut-down request.
When the SBC shut-down process is almost done, the led at Connections board will goes off.
Wait additional 10 seconds and 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 pause a risk to your kids if they have their hand on the way while the cover closes!

Un-power the servos:

This considers an independent power supply for the servos.
Servos can be unpowered at any moment.
Un-power the servo before shutting down Rpi, if you experience strange servo behaviour when Rpi goes off; The script takes care to set the servo PWM related GPIO to a fix level (low), at the shut-down, but it does not work on 100% of the times.

Running the robot from VNC Viewer:

Here is explained how to run the robot from VNC Viewer, when the python script has been started via the Bash file at Raspberry Pi boot, and 'halt -p' command is uncommented in the bash file.
Under these circumstances:
it is not an option to quit the script from the robot, by keeping the touch button pressed long, as the Raspberry Pi will shut down

it is not possible to run a 'new' script over the first one , as will conflict with Camera resources; It is necessary to quit the running python scrip first.

Steps to do:
Connect VNC Viewer to the robot

Open a terminal

Folder is not relevant

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
```

Search for python Cubotino process, and note the ID (674 on the above example); This is by far the process that takes more CPU and memory resources, making easier to find it.

Search for bash command, from root user, located above python Cubotino, and note the ID (657 on the above example)

First kill the bash process sudo kill -9 ThePIDNumberForBash (by using the above example the command will be *sudo kill -9 657*)

After kill the python process *sudo kill -9 ThePIDNumberForPythonCubotino* (by using the above example the command will be *sudo kill -9 674*)

```
pi@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 shuts off right after the Cubotino python process is killed: Not the wanted result 😊

Automatic robot start' chapter

Step14: Set Thonny to work with the virtual environment.

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

See 'Set Thonny IDE interpreter' chapter

Step15: multiple WiFi settings:

Adding a second (or more) wifi connections, for instance to add your phone wifi 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 wifi hotspot details, you can use the Real VNC app on the phone to show the image processing part.
On September 2022 I've presented this project to the Eindhoven Maker Faire, and I used the phone hotspot to show on a large screen what the robot's camera sees and related image processing.

Steps:

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

ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1
country=NL (use your Country code)

network={
ssid="your_SSID_name" (use your SSID name In my case this is the home WiFi)
psk="your_PASSWORD" (use your PASSWORD; In my case this is the home WiFi password)
priority=10
}

network={
ssid="your_SSID_name" (use your SSID name; In my case this is the WiFi hotspot of my phone)
psk="your_PASSWORD" (use your PASSWORD; In my case this is the WiFi hotspot password of my phone)
priority=20
}

Note: The priority command is needed when both the wifi 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.

22) Files copied to Raspberry Pi

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

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

File	Purpose	Notes
Cubotino_T.py	Main robot script	
Cubotino_T_moves.py	Translates the cube solution (Singmaster notation) in robot moves	
Cubotino_T_servos.py	Manages the servos	
Cubotino_T_set_picamera_gain.py	Manages the Camera gains settings	
Cubotino_T_Logo_265x212_BW.pdf	Cubotino logo, plot on display	The first time Cubotino_T.py is executed, the logo in the pdf file will be converted and saved as jpg. This is a workaround, because Instructables prevents image sharing as file attachment.
Cubotino_T_display.py	Display management	From 6th August 2022
Cubotino_T_settings.txt	Json file with settings for Cubotino_T.py script	Default values, to start the tuning
Cubotino_T_settings_AF.txt		Optimized values for my robot
Cubotino_T_servo_settings.txt	Json file with settings for Cubotino_T_servos.py script	Default values, to start the tuning
Cubotino_T_servo_settings_AF.txt		Optimized values for my robot
Cubotino_T_bash.sh	Bash file to start-up the robot script automatically after Raspberry Pi boots	

Kociemba solver library is necessary.

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

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

23) Servos test and set to mid position

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

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

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

13. Connect a connections arm to the non-assembled servos.
14. Connect the servos to the Connections_board.

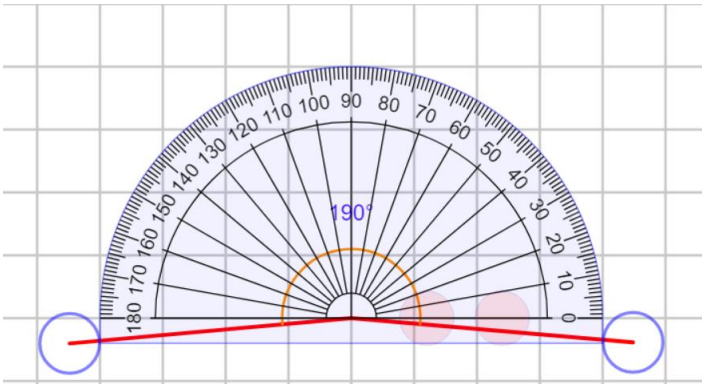
Argument set, and related value, can be passed to Cubotino_T_servos.py, to play with the servos angle and especially to set them to the mid position before the assembling.

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

Position the servo arm as per below picture:

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

The angle can be evaluated by printing a protractor:



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

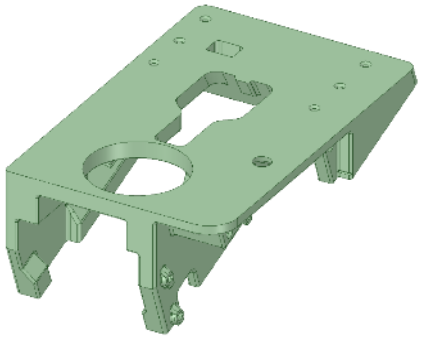
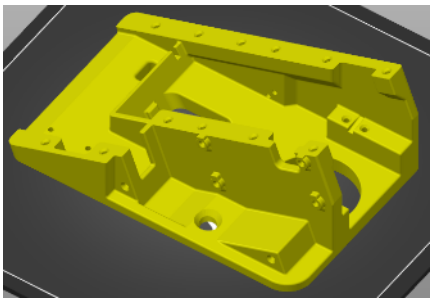
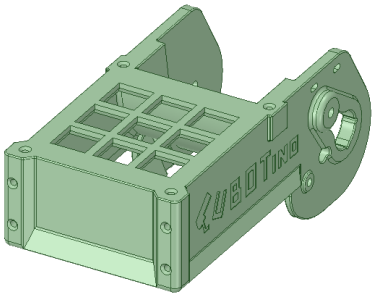
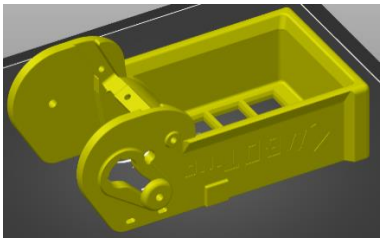
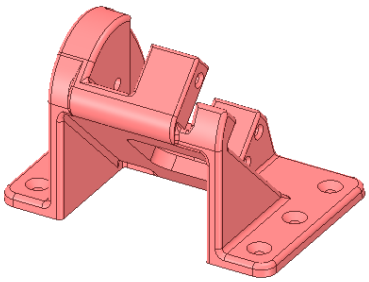
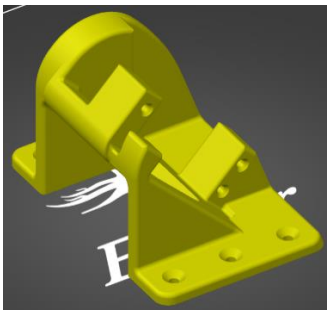
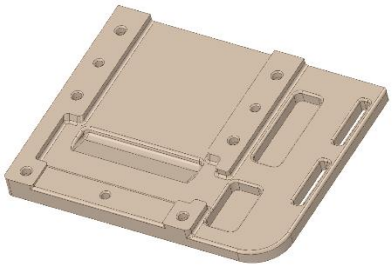
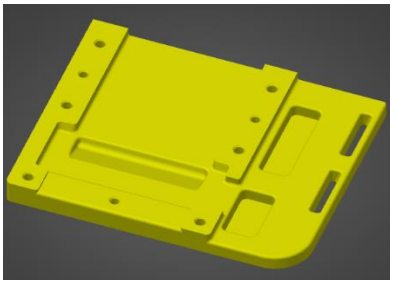
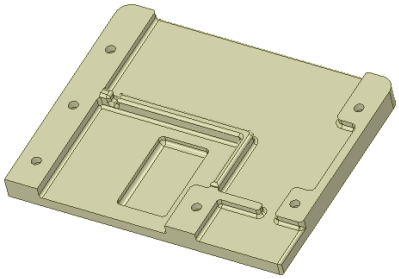
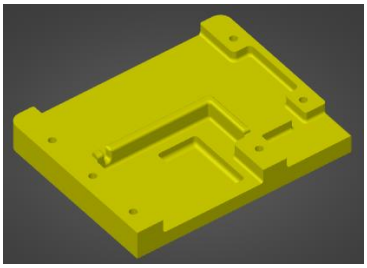
24) 3D printed parts

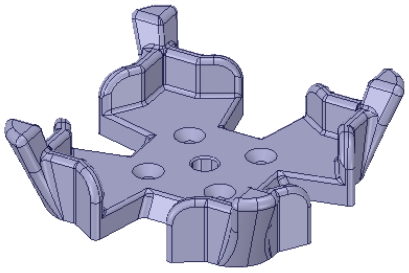
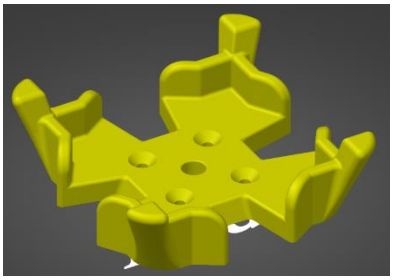
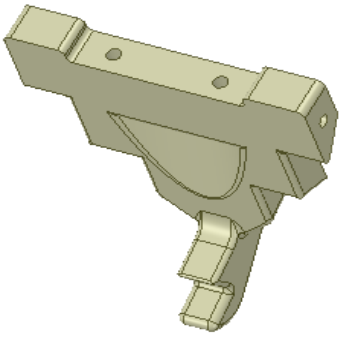
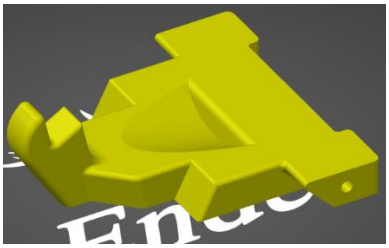
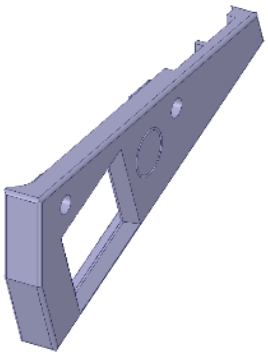
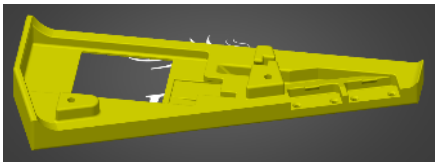
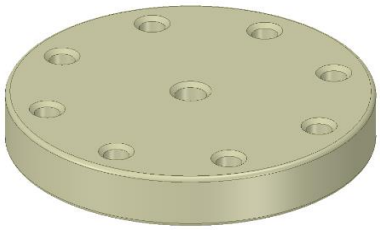
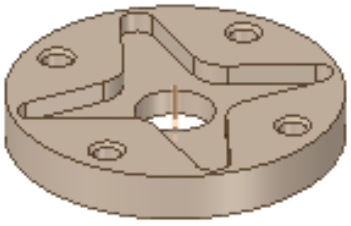
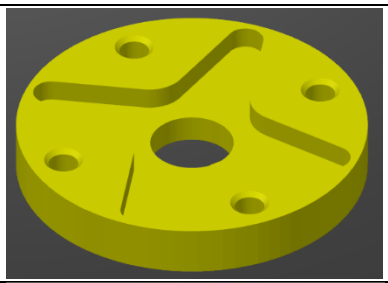
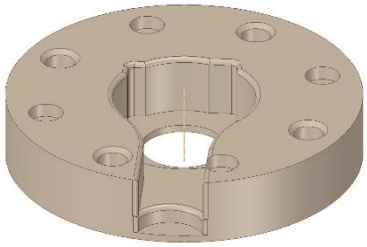
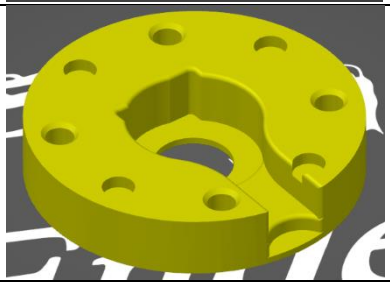
See notes below for filament quantity, and printing time …

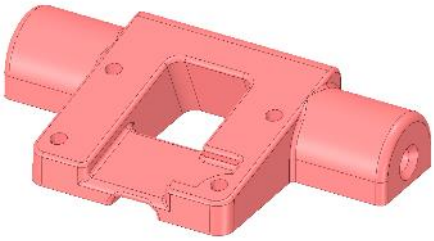
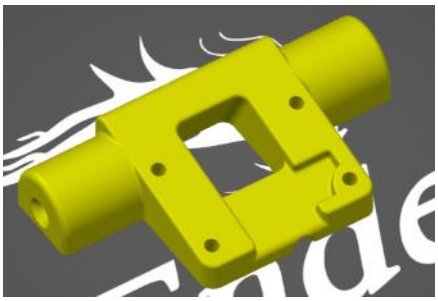
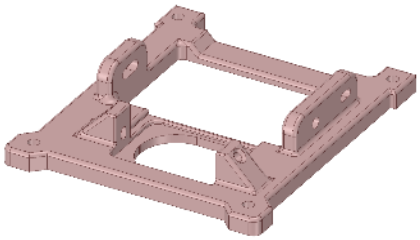
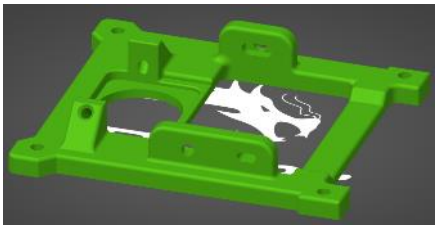
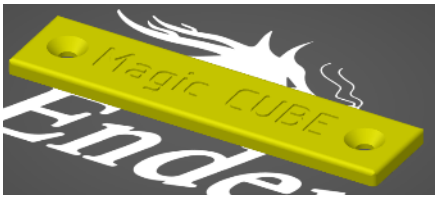
Ref	Part	Filament		Printing time	Version specific parts
		Meters	Grams		
1	Structure	49	145	14h33m	
2	Top_cover	39,5	117	12h26m	
3	Hinge	17,2	51	5h37m	
4	Baseplate front	12,2	36	3h33m	
5	Baseplate back	12,6	37,3	3h36m	
6	Cube_holder	11,6	34,4	3h36m	
7	Cube_lifter	5,5	16,2	1h48m	
8	Servo_axis_sup (or its alternative)	2,4	7,2	0h55m	
9	Servo_axis_inf (or its alternative)	2,4	7	0h52m	
10	PCB_cover_display (or its alternative)	14	41.4	4h13m (4h27m)	Top specific
11	PiCamera holder	2.2	6.5	0h50m	Top specific
12	PiCamera holder frame	6.1	18	2h17m	Top specific
13	Personaliz_plate	1.5	5	0h30	Top specific
	TOTAL	**175m**	**692g**	**51h30m**	

Notes:

21. The biggest part is the Structure, and it easily fits on printers with plate size of 200x200mm.
22. Filament length is based on Ø1.75mm.
23. 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:
    13. 0.2mm layers
    14. Low speed (between 25 to 40mm/s for the external parts and 1st layer)
    15. 4 layers on vertical walls
    16. 5 layers on horizontal walls
    17. 30% filling
    18. 8mm brim
24. The filament quantity, and printing time, in the table should be considered as an upper limit. After printing the parts for the base version, total weight was closer to 400grams than 466grams estimated by the slicer SW.
25. Possible to upgrade the Base version, by printing 3 (or 4) more parts, requiring ~ 8h.
26. All parts have been designed to be printed without supporting the overhangs.
27. Some parts have been split, for easier and better 3D printing.
28. The suggested part orientation for the 3D print is showed on below Table.
29. The **stl** files are available at
    - https://www.instructables.com/CUBOTino-Autonomous-Small-3D-Printed-Rubiks-Cube-R/
    - https://www.thingiverse.com/thing:5407226
    - https://github.com/AndreaFavero71/cubotino/tree/main/stl
30. The **step** files are only available at https://github.com/AndreaFavero71/cubotino/tree/main/stp

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

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

PiCamera holder		
PiCamera frame		
Personaliz_plate Or Personaliz_plate01	 	

25) Assembly steps

Before assembling the robot:

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

Assembly order are initially as per the Cubotino_base_version:

246. Screw the bottom servo to the structure
247. Prepare the sandwich Servo_axis_inf / servo lever / Servo_axis_inf
248. Assemble the Cube_holder to the Servo_axis assembly
249. Assemble the Cube_holder assembly to the bottom servo
250. Assemble the Hinge to the Structure
251. Assemble the “T25” servo arm to the upper servo, only after knowing the servo is on its middle position.
252. Insert the Top_servo assembly into the Top_cover slot
253. Assemble the Top_cover assembly to the Hinge
254. Complete the top servo assembly to the Hinge
255. Assemble the Lifter to the Top_cover

Starting from here, the steps are different from the Cubotino_base_version

256. Assemble the PiCamera holder frame to the Top_cover
257. Assemble the LED breakout board to the Top_cover
258. Position the cables, and assemble the Baseplate_rear
259. Connect the PiCamera flex cable to Raspberry Pi Zero2 board, and fix it to the Structure
260. Fix the Touch_sensor to the PCB cover
261. Assemble the USb breakout borad to the PCB_cover
262. Connect the servos, LED breakout board and Touch sensor
263. dress the cable and connect the display
264. Assemble the PCB_cover
265. Assemble the Baseplate_front to the Structure
266. Stick the PiCamera to its board, via the self-adhesive tape underneath the camera.
267. Assemble the PiCamera module to the PiCamera_holder
268. Assemble the PiCamera holder to the PiCamera holder frame
269. Connect the flex cable to the PiCamera module
270. Personalize the formal Stop plate, transformed to a personalization plate in this robot vesion

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

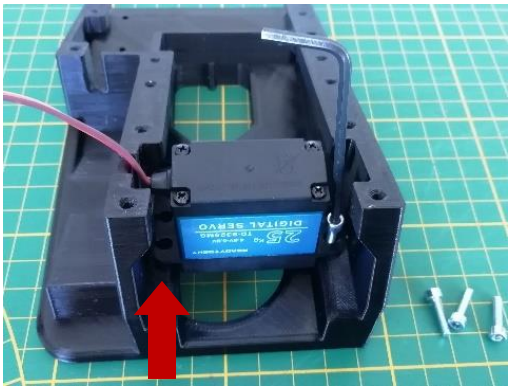
26) Assembly details

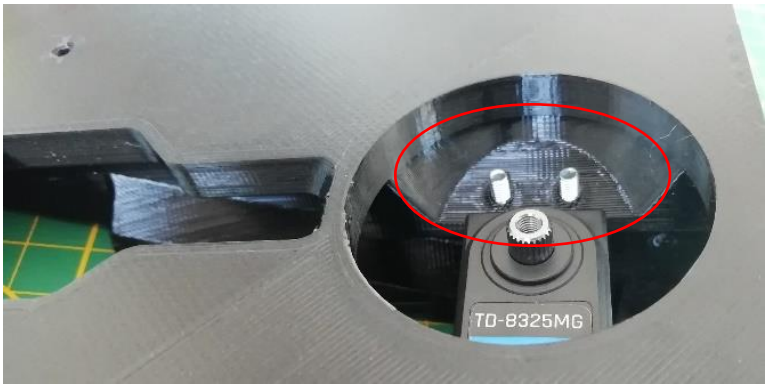
Step4 (Mount the bottom servo to the structure):
4x M3x12mm cylindrical head
Couple of washers

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

Couple of notes:

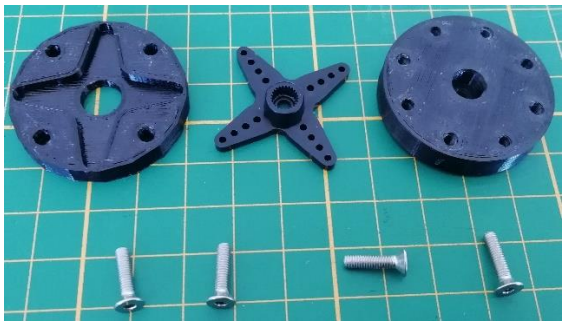
- Before tightening the four screws, checks if the servo output gear is well centred to the Structure hole.
- To limit the protrusion of the below two screws, add a couple of washers to these screws; This to prevent eventual interference with the servo_axis_inf part.



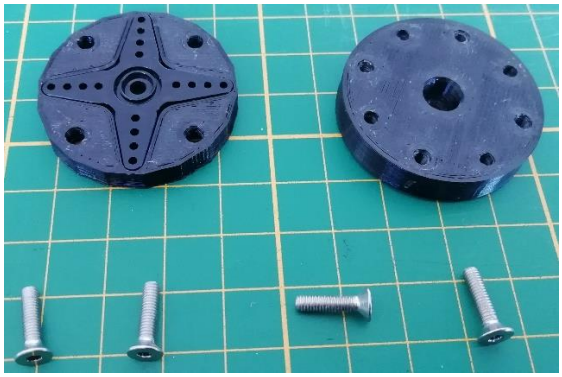


Step5 (sandwich Servo_axis_inf / servo arm / Servo_axis_inf):

4x M3x12mm conical head

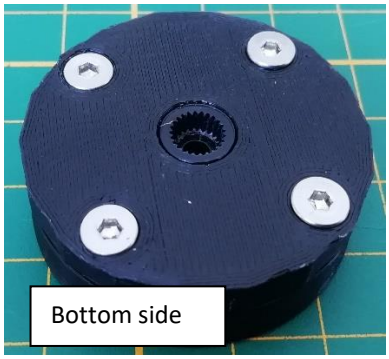


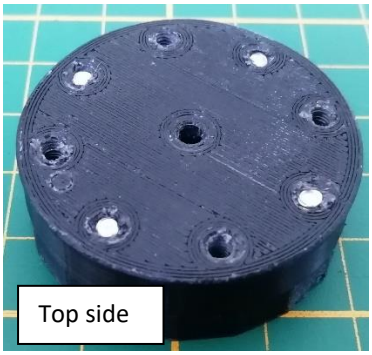
Check if the supplied X arm fits with the indentation of Servo_axis_inf part:



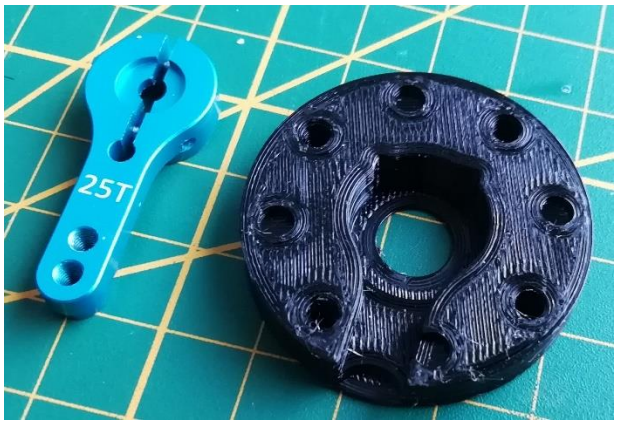
If you don’t have a X arm to make it fit, please print the alternative parts (Servo_axis_inf and Servo_axis_sup) designed to fit the aluminium “T25” arm (arm has to be reduced in length).

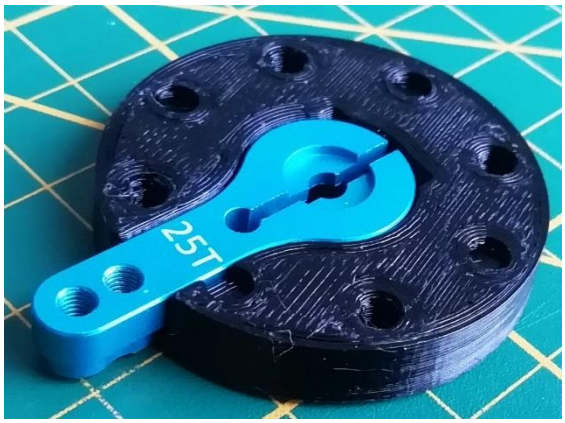
Make the sandwich





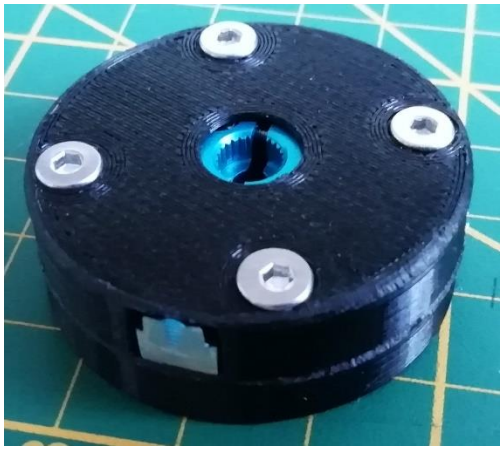
Step5a Alternative servo axis assembly:





Cut the protruding part of the “T25” arm
Adjust its screws to be able to enter the servo outlet gear

Make the sandwich as per Step5
4x M3x12mm conical head



Step6 (Assemble the Cube_holder to Servo_axis assembly):

4x M3x12mm conical head



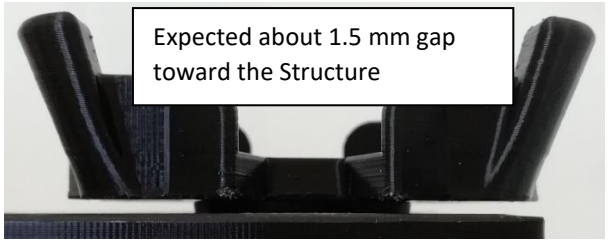
Step7 (Assemble the Cube_holder assembly to the bottom servo):

Try to not rotate the Servo output gear during this step: Gently try to feel the teeth coupling, and if the Cube_holder is not well aligned, retract it, rotate the Cube_holder by about 90 degrees and check again.

Servo output, and Servo arm, have even number of teeth: For sure there is one good coupling

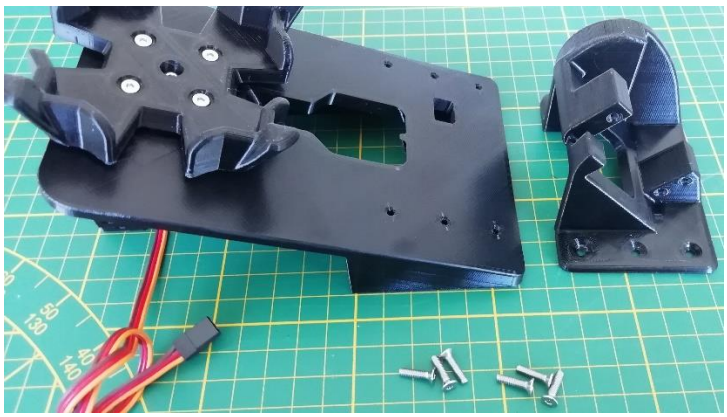

1x M3x12mm cylindrical head

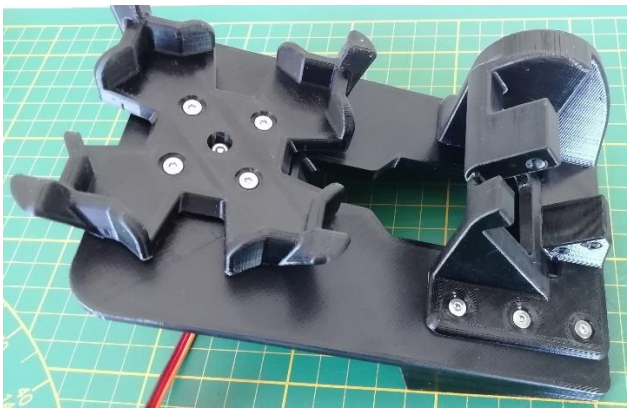




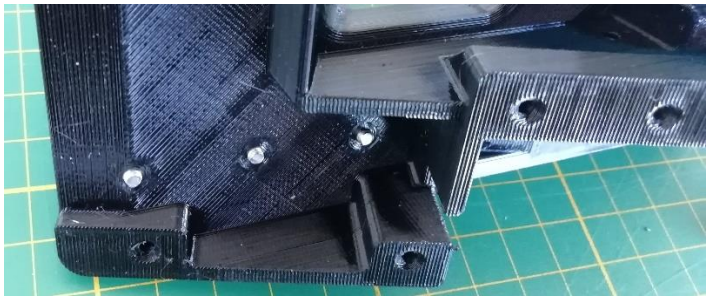
Step8 (Assemble the Hinge to the Structure):

6x M3x12mm conical head





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



Step9 (Assemble the "T25" servo arm to the upper servo):

Place the "T25" arm along the main Servo axis (Servo should be prepared upfront with the gear at middle angle).

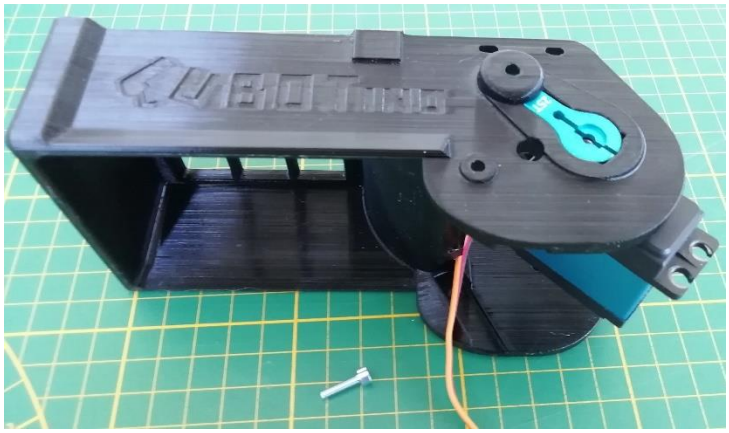
Close the two tiny screws at the arm

Step10 (Insert the Top_servo assembly into the Top_cover slot):

1x M3x12mm cylindrical head

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

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



Note: The screw should not protrude from the Structure plane; Add some washers under the screw head if needed

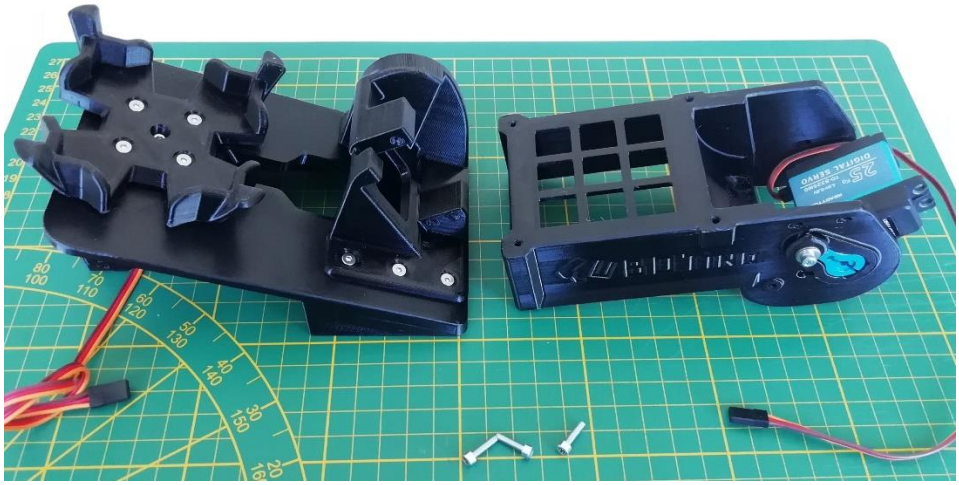
Rotate the servo by about 45deg, to facilitate next steps

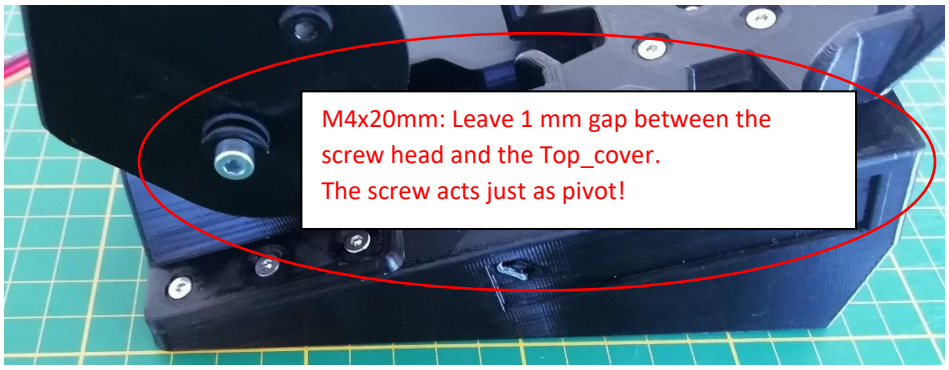
Step11 (Assemble the Top_cover assembly to the Hinge):

1x M4x20mm cylindrical head

3x M3x12mm cylindrical head

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

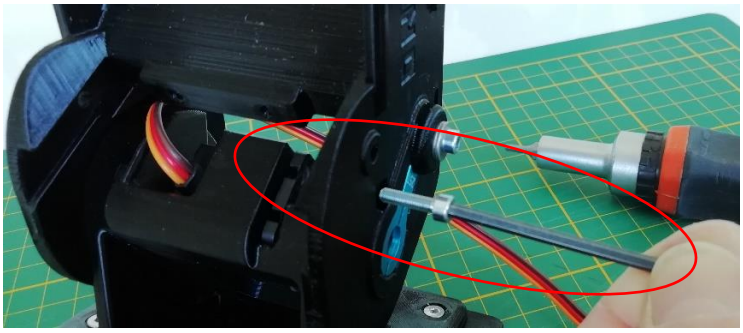


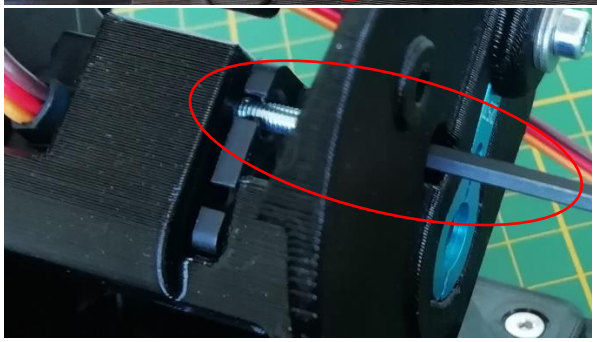


Do not fully tighten the two M3x12mm, until also the third screw is positioned

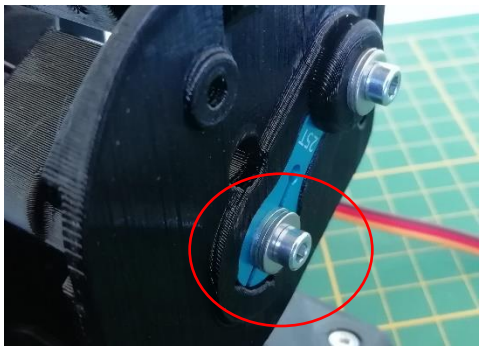
Step12 (Complete the top servo assembly to the Hinge):

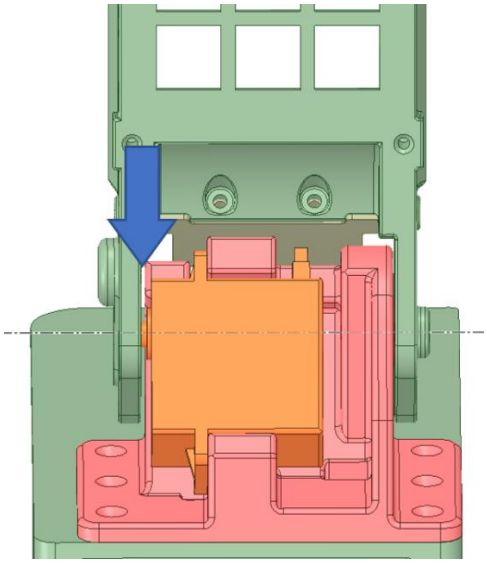
Rotate the Top_cover to have the hole facing the third screw accessible





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

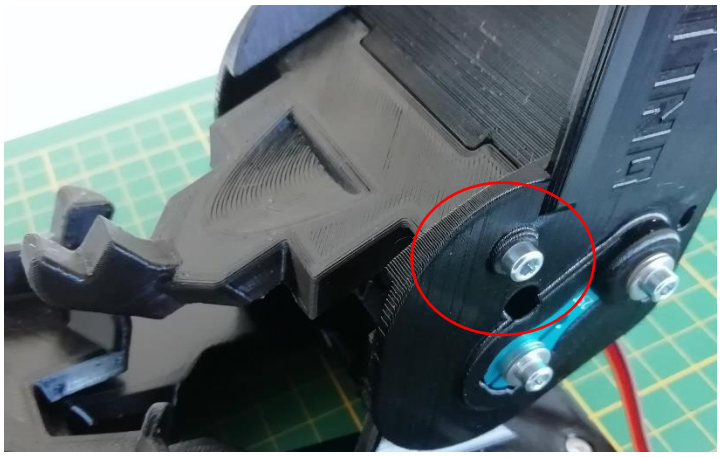




Verify the gap presence in between the Top_cover and the Hinge, at the servo output gear side (arrow at the side).
In case there is little or no gap, unscrew the M3 screws of the servo, and place some little spacers (0.5 to max 1mm) in between the servo and the Hinge (preferably close to the screws locations);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

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

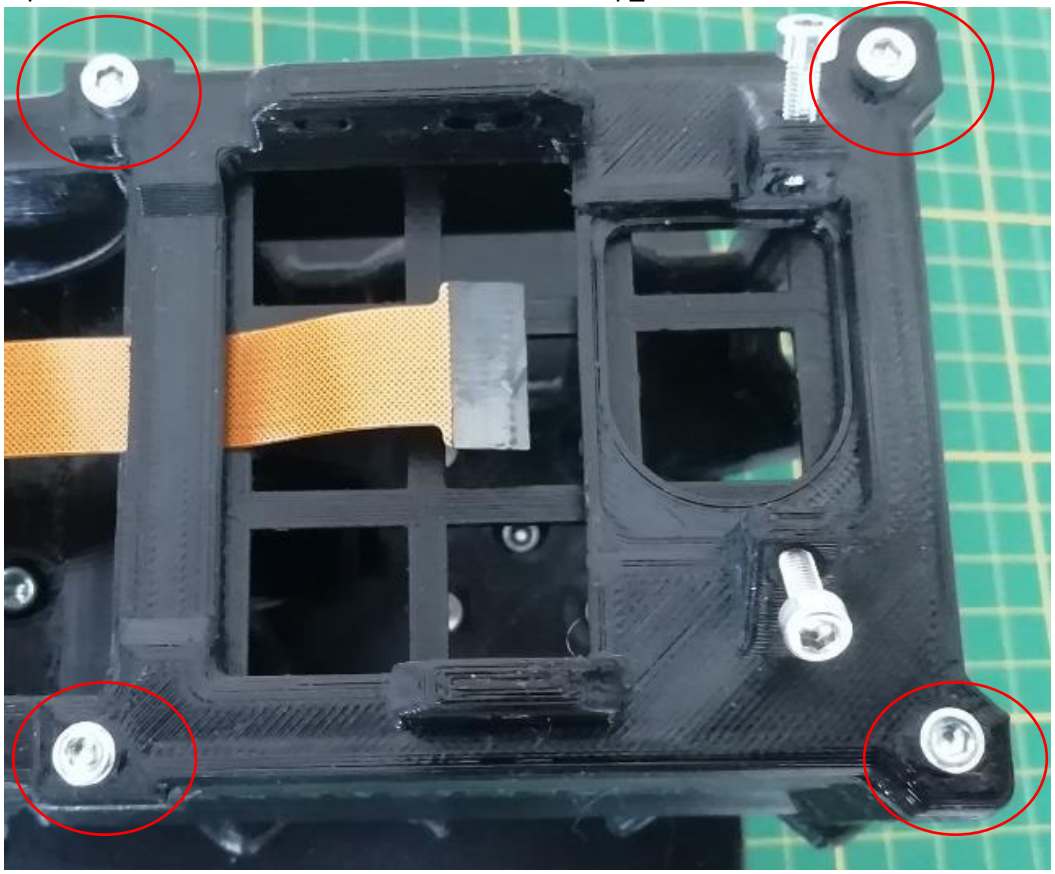
Slide the Lifter into the Top_cover slots



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

4x M3x12mm cylindrical head

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

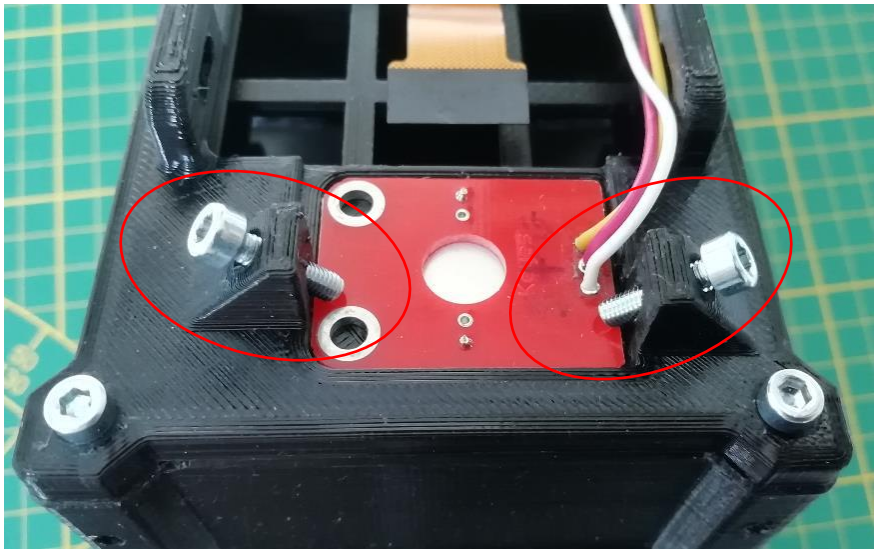


Step15 (Assemble the LED breakout board):

Add 2x M3x12mm cylindrical head

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

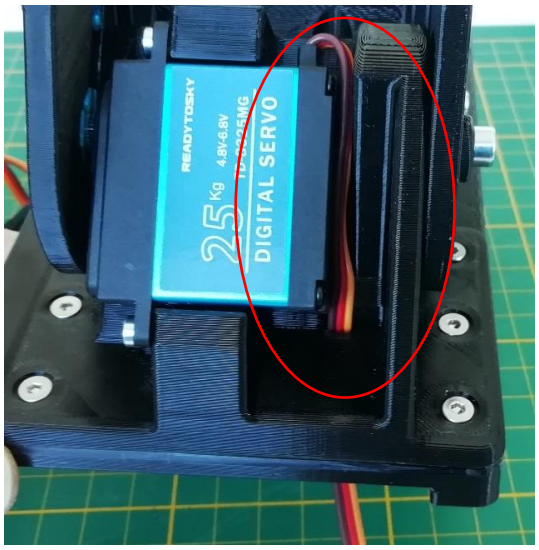
Stop screwing once the screw tip touches the board

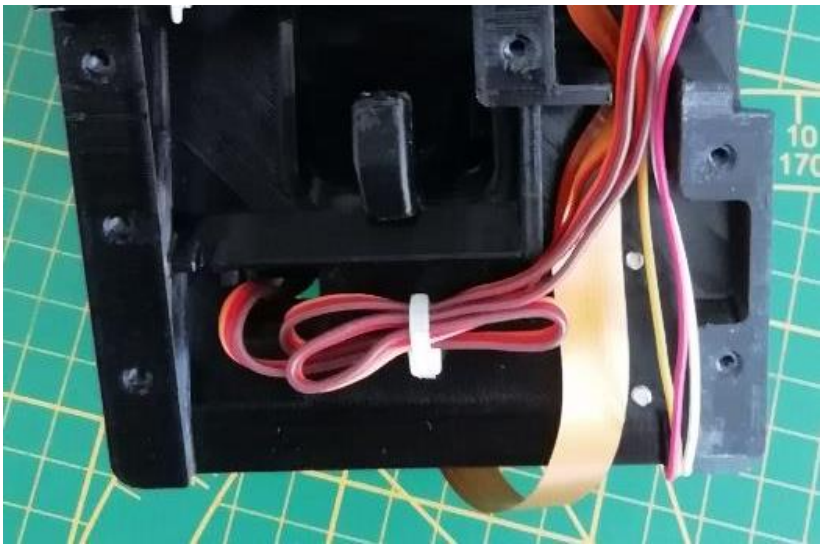


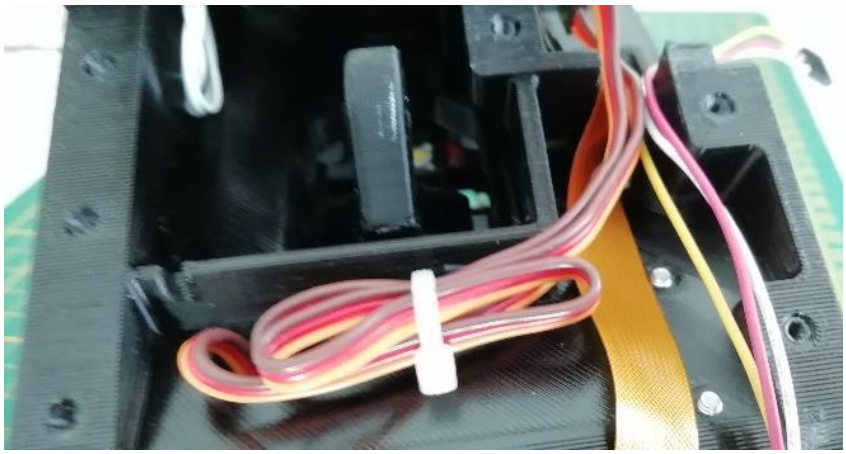
Step17 (Position the cables, and assemble the Baseplate_rear):
6x M3x12mm conical head (4 screws at corners are sufficient)

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

Dress the cables and close the Baseplate_rear.

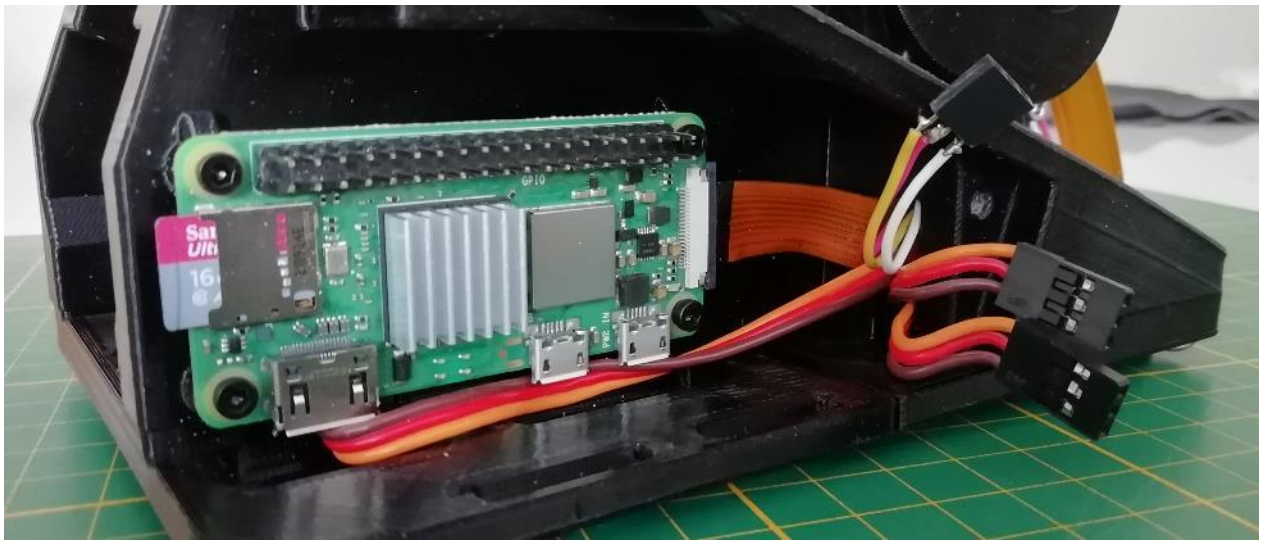






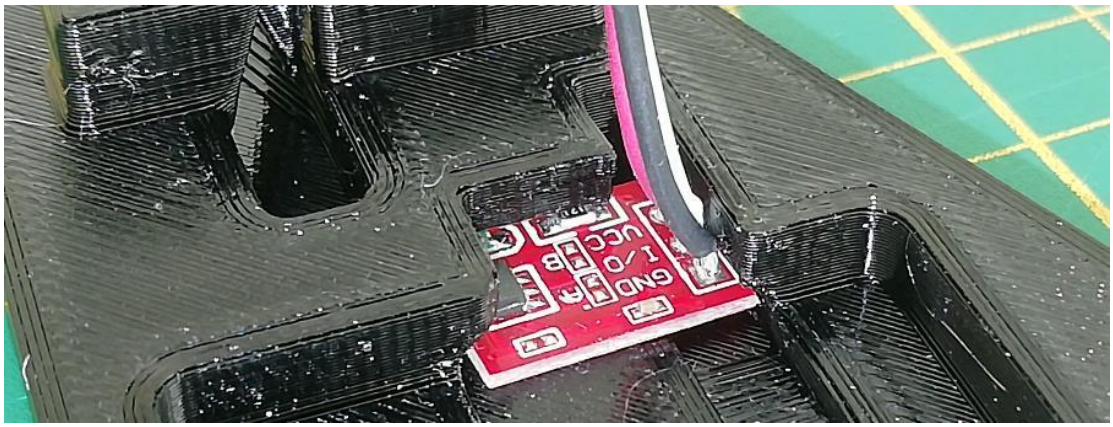
Step17 (Connect the PiCamera flex cable to Raspberry Pi Zero2 board, and fix it to the Structure)

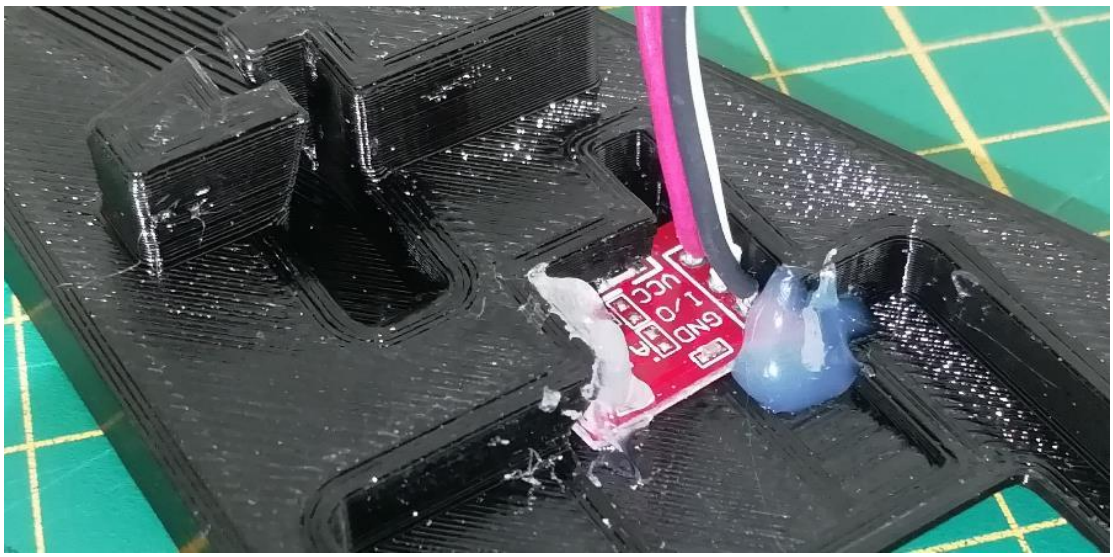
4x M2.5x10mm (or M2.5x8mm) cylindrical head



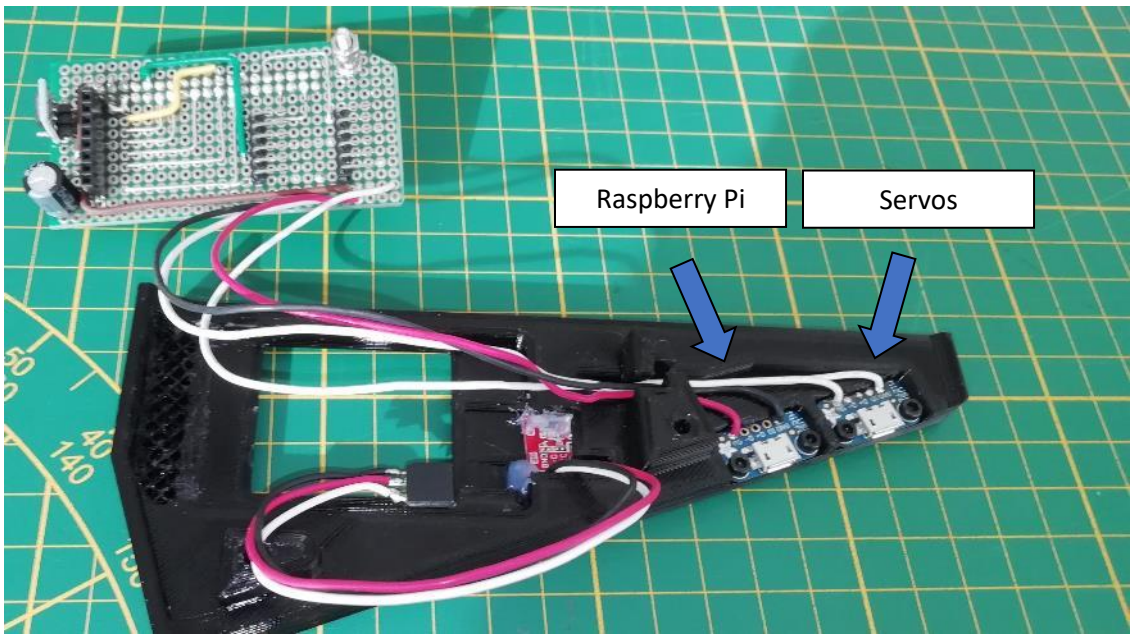
Step18 (Fix the Touch_sensor to the PCB cover):

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

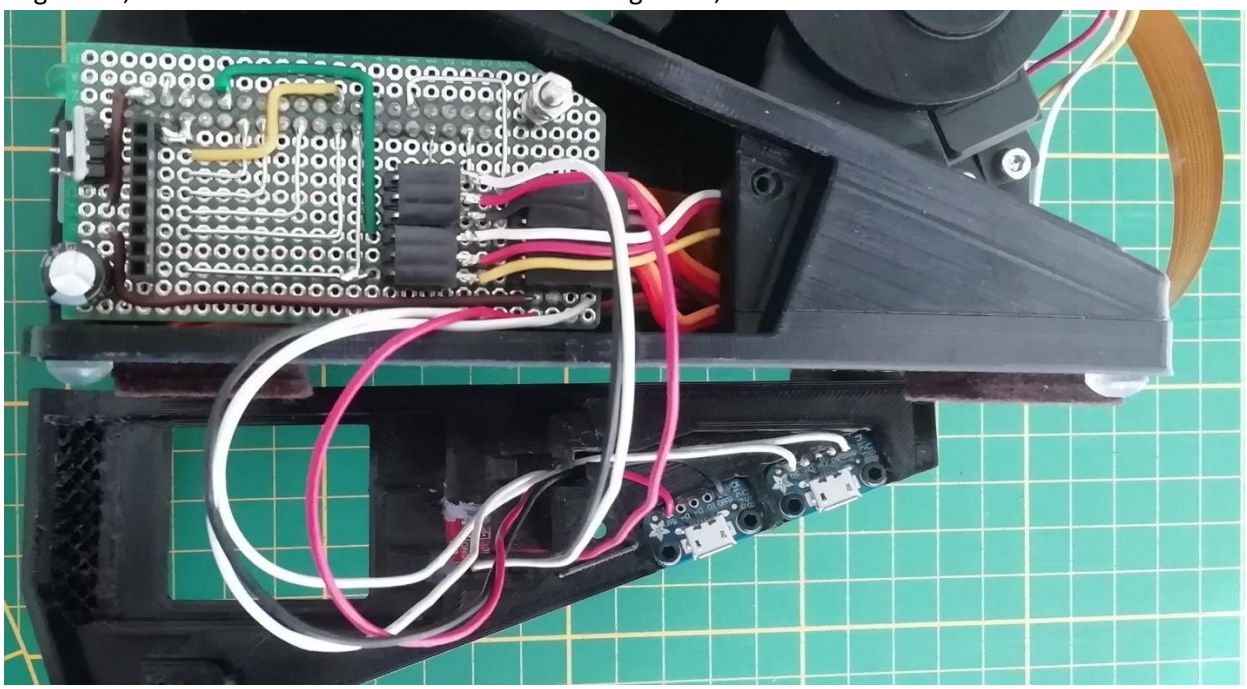




Step19 (Fix the microUSB breakout board to the PCB_cover):
2x M2.5x4mm cylindrical head
The order of the two boards is not critic, below just a proposal

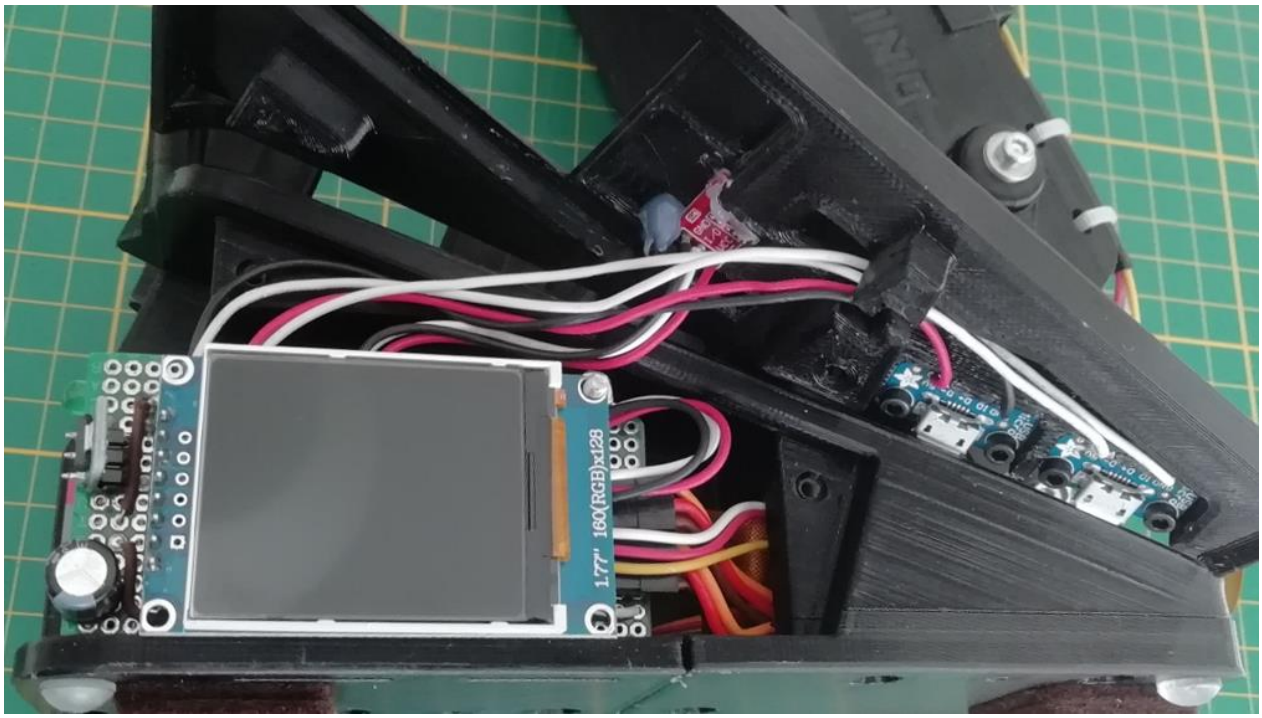


Step20 (Connect the servos, LED breakout board and Touch sensor):
In general, the brown wire of servo connector is the ground, therefore to be oriented toward the bottom



Step21 (dress the cable and connect the display):

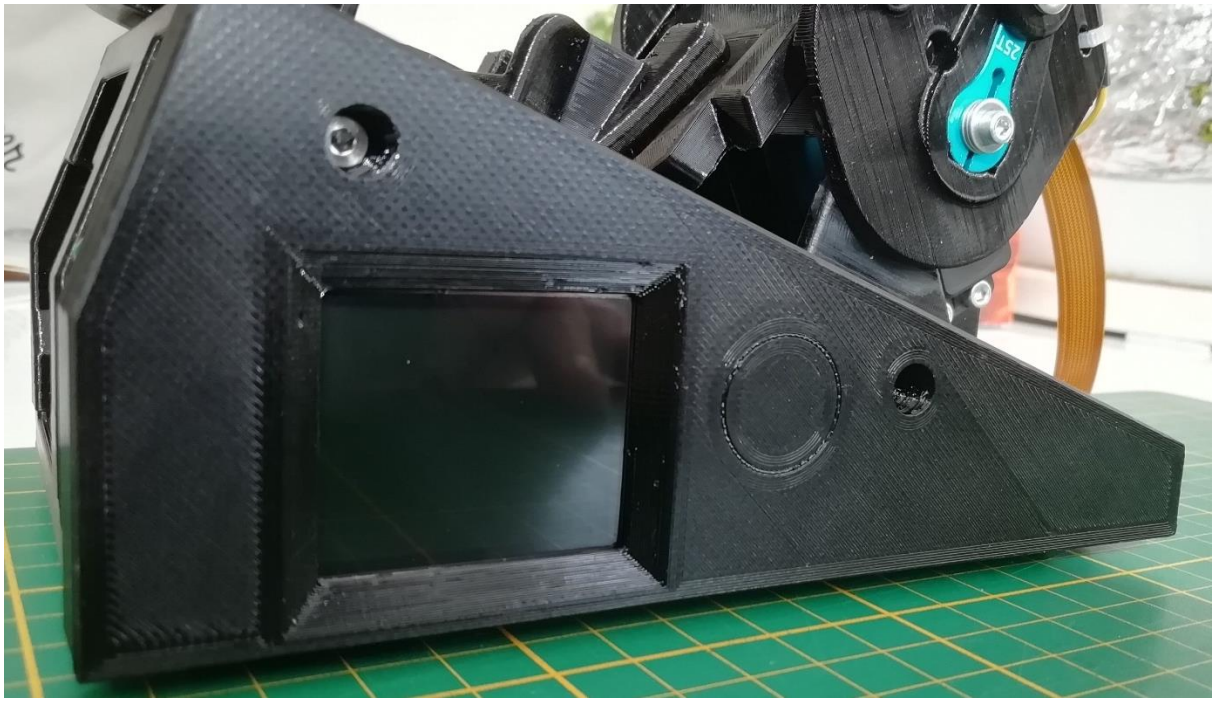
Cables at the microUSB breakout boards have to be well positioned into the groove



Step22 (Assemble the PCB_cover):

2x M3x12mm conical head

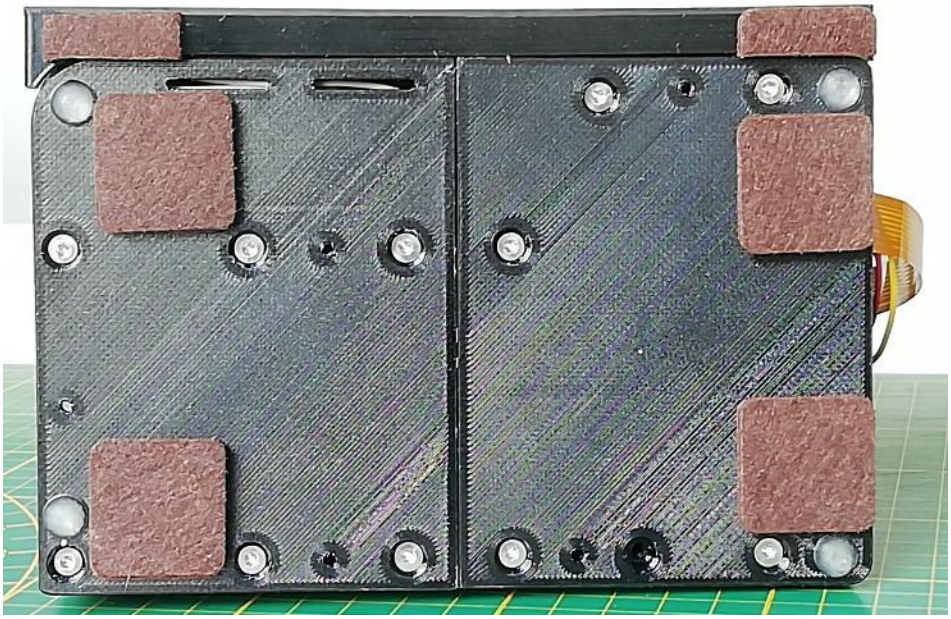
Attention to don't squeeze cables against the Structure



Step23 (Assemble the Baseplate_front to the Structure):

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

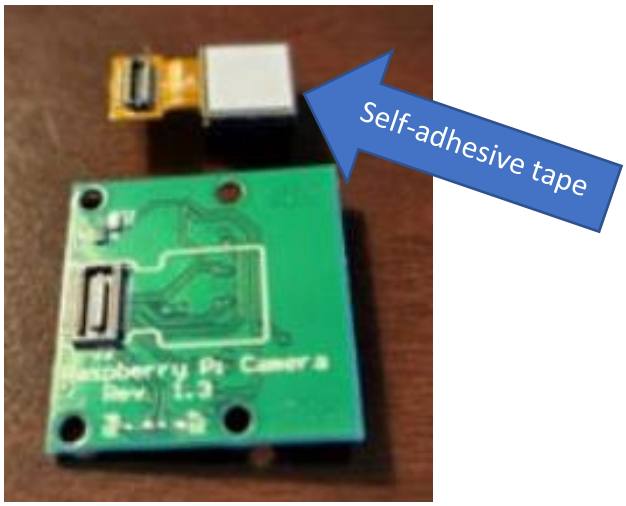
Stick self-adhesive rubber feet (or felt pads) to the baseplate; Those at the back should be placed as close as possible to the corners



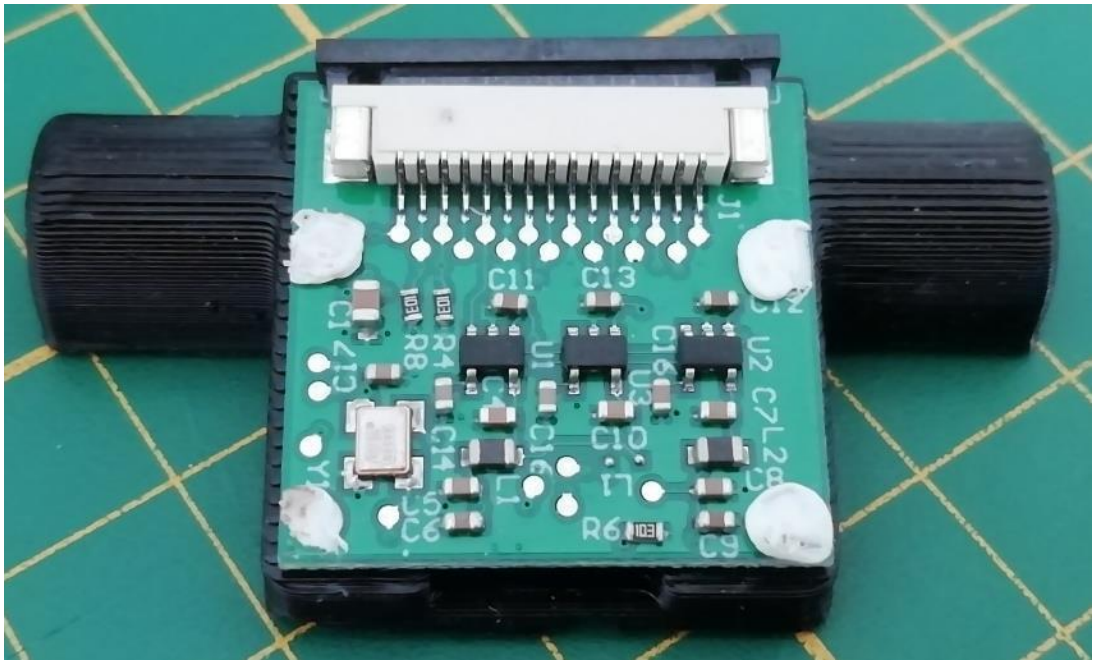
Step24 (Stick the PiCamera to the board):

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

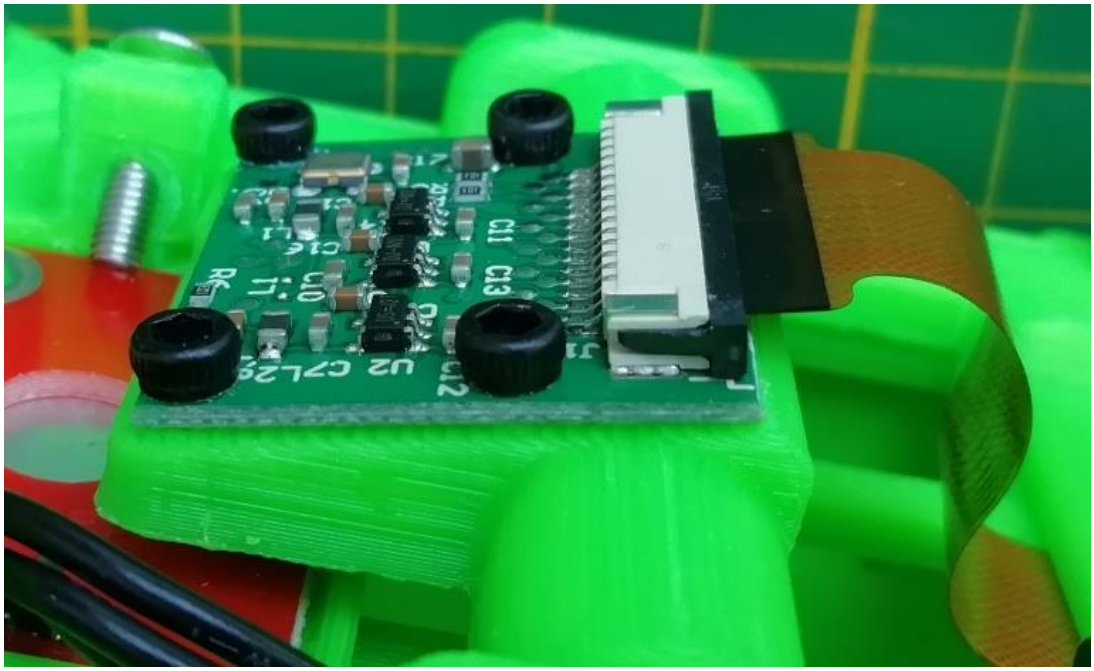
Make us of that tape to secure the camera to the board, to prevent the camera oscillating on its flexible cable (blurred images)



Step25 (Assemble the PiCamera module to the PiCamera holder):
Necessary some little pieces of filament Ø1.75mm.
Force 4 pieces into the holder, slide the board over, deform the filament with hot blade.

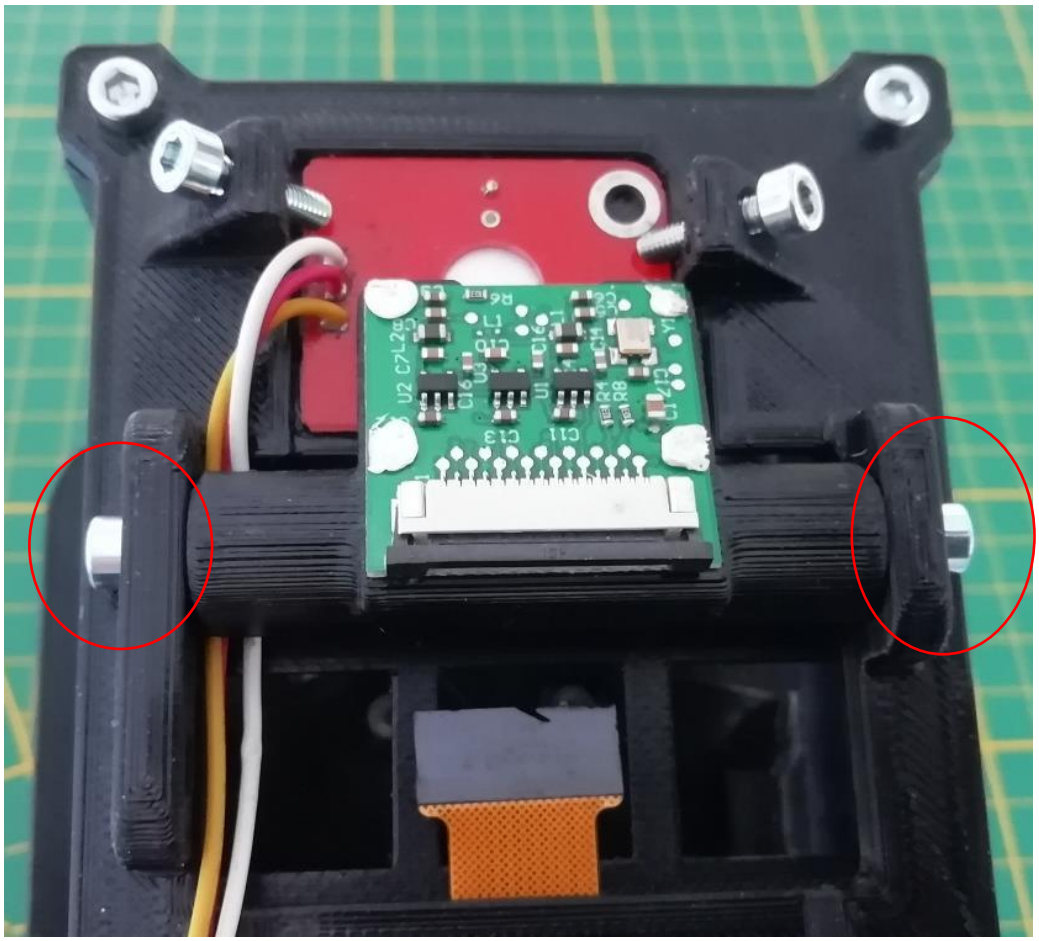


Alternatively, 4x M2x3mm screws can be used or 4x M2.5mm screws can be used.
In this case, first self-thread the slots via the screw without the camera.



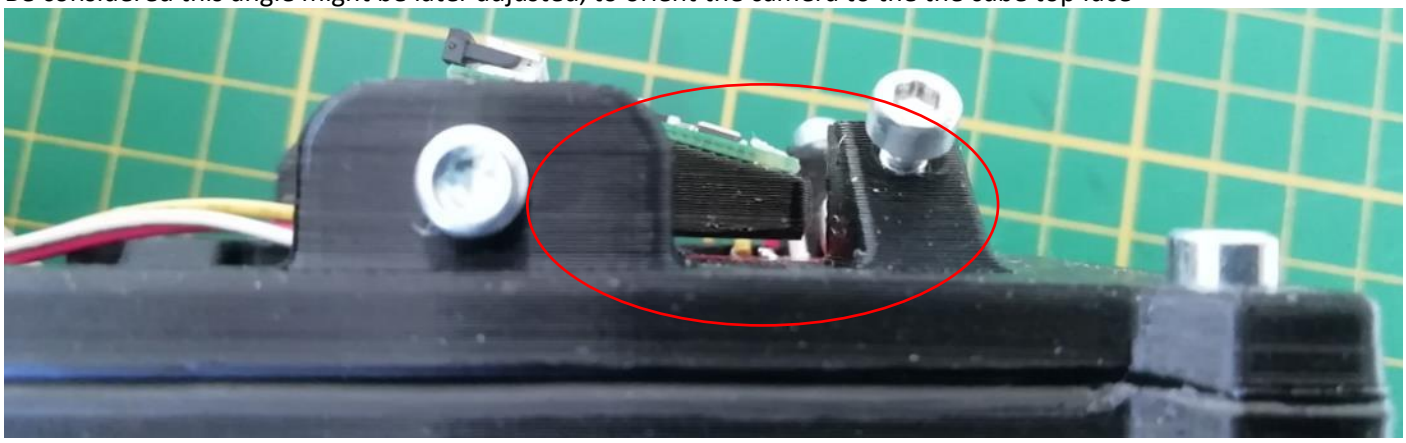
Step26 (Assemble the PiCamera holder to the PiCamera holder frame):

2x M3x12mm cylindrical head

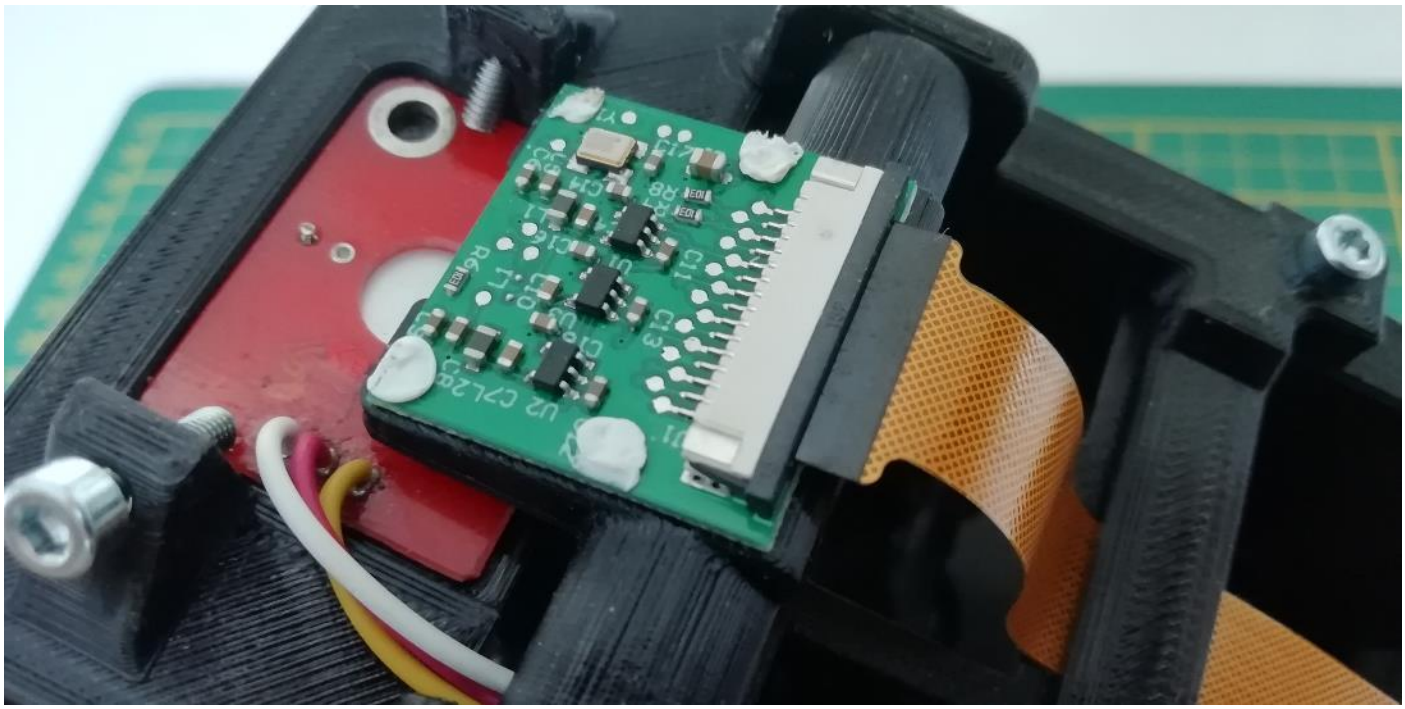


Keep the PiCamera holder parallel lo the Top_cover, while thightening the screws:

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



Step27 (Connect the flex cable to the PiCamera module):



Step28 (Personalize the formal Stop plate):

The Structure has a recess, meant to hold a metal plate working as a Stop touch sensor on the Base version.
On this version, the stop function is available on the PCB_cover, making that recess quite useless and unesthetic.
I made available two alternative stl files of a plate to 3D printed as cover such a recess; One plate is neutral, the second one has ‘Magic CUBE’ words engraved.
Of course, you might consider using the neutral one as baseline to personalize your own Cubotino 😊.

27) Raspberry Pi 3 or 4

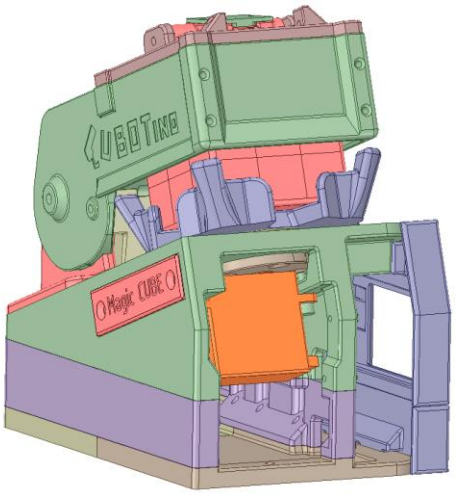
Cubotino_Top_version has been designed in late winter 2021, by considering the Raspberry Pi Zero 2 an easy to source and relatively cheap (ca 15$) component ….

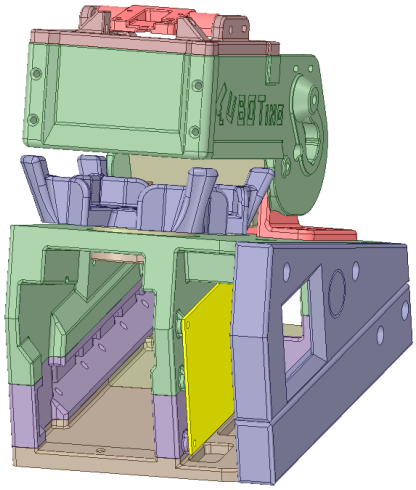
Unfortunately, the market situation has changed: Raspberry Pi Zero 2 boards are scarcely available and are gold priced.

If you have a Raspberry Pi 3 or 4, a possible solution is to increase the robot height by 26mm to embed that larger SBC.
Some notes:

- The connections_boards need to be slightly smaller, to prevent interference with some of the raspberry Pi 3 or 4connectors.
- The height increment can be gained via 3 extension parts to be 3D printed and screwed to the other parts; This choice allows to shrink the robot in future, in case the smaller boards will be back reasonably priced again.
- This approach has been already suggested with Rev. 2.2 on 18/06/2022, yet at that moment I hadn't any "large" Raspberry Pi board available, and I did not realize the interferences with the boards; On that period, I had very little time to work things out, and decided to remove the proposal in less than a week.

Below a few images of how the robot will look like:



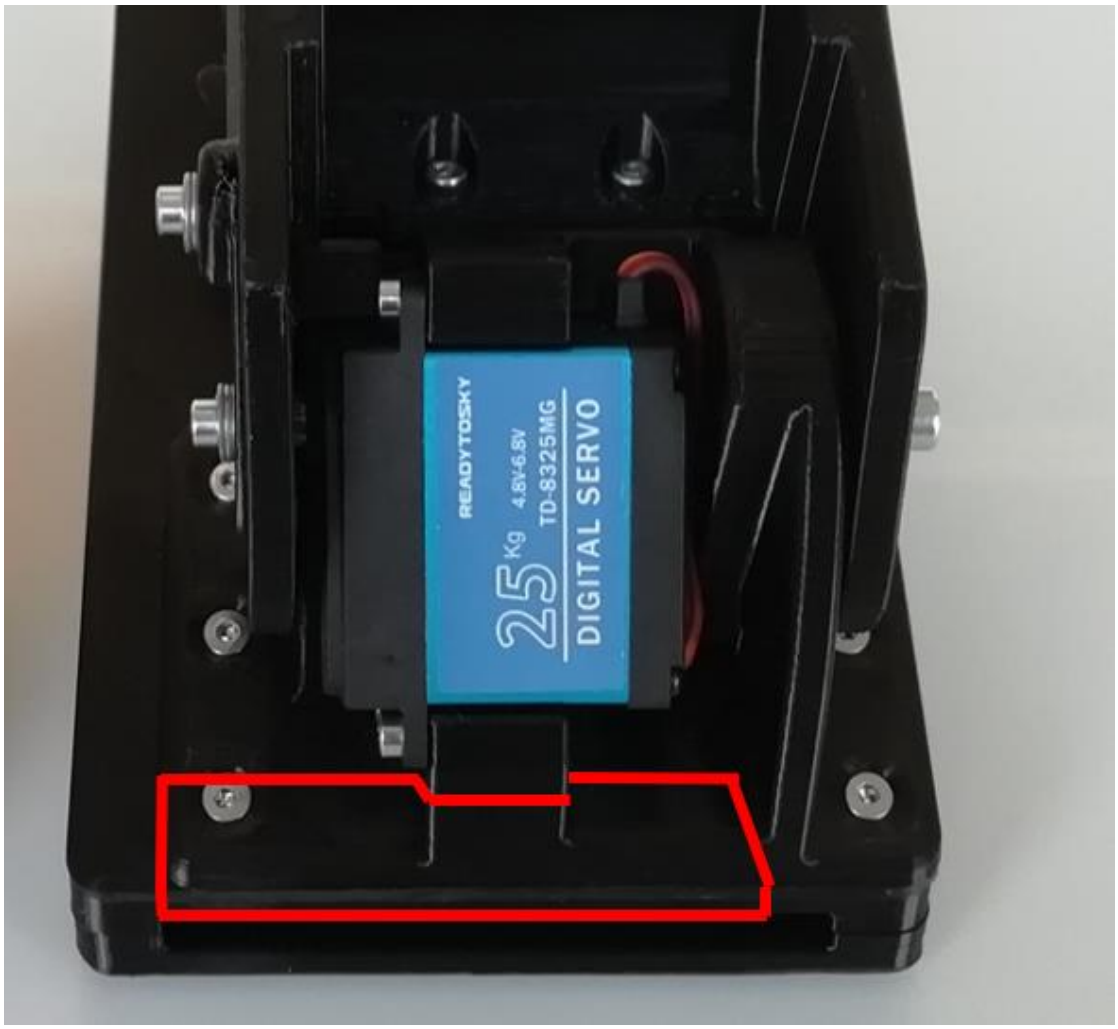


And a picture from ***nrdlrd*** Maker:

Raspberry Pi 3 and 4 have “standard” CSI camera, meaning the flex cable differs from the one at supply list for Raspberry Pi Zero (and Zero2).

Raspberry Pi 3 and 4 have the CSI camera port in a less convenient position than Raspberry Pi Zero: This makes the 30cm flex cable just enough …. meaning it might be just not enough.

The obvious option is to buy a longer cable, meaning the commercial 450mm or 500mm
In case you already have a 30cm cable that is just enough, and you’d like to reduce cable tension, you might consider modifying a couple of parts: Structure and Hinge



This is quite labour intensive if you file/cut the parts, or you revise the files and re-print.
The Hinge currently has 6 fixing screws, considering some screws might be less effective; The removal of one screw is clearly not a problem.
As said, the suggested solution is to buy a longer cable (400 or 450mm), yet this might be a more creative solution.

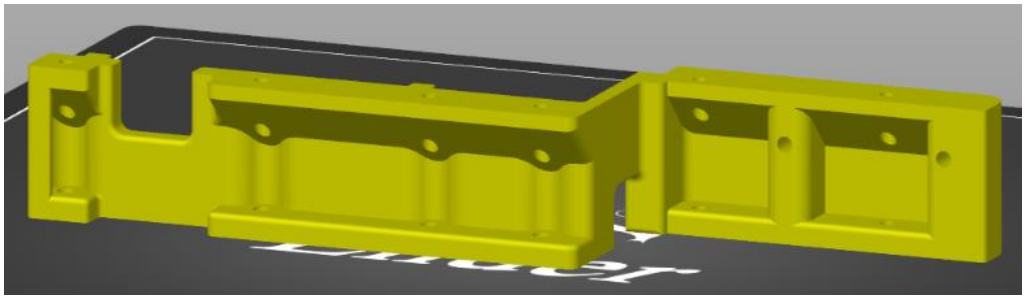
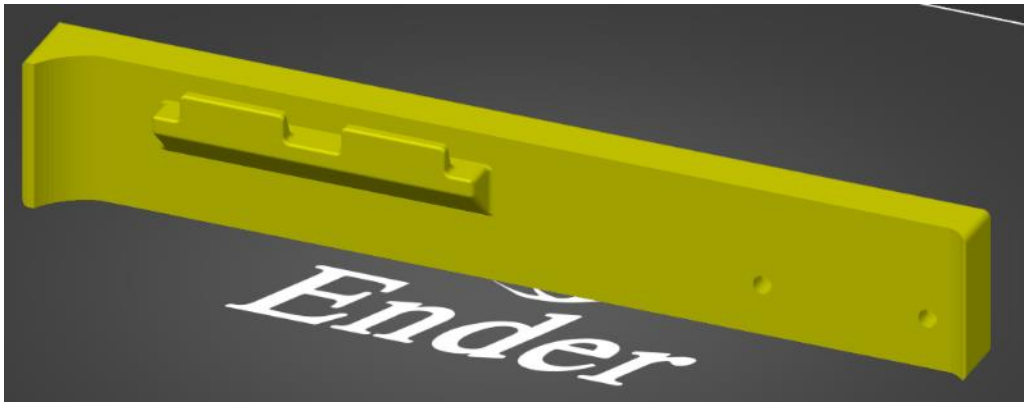
Stl files for the Extensions are (back) available since 22/10/2022 at Instructables and GitHub:

Filament quantity, and printing time …

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

Notes:

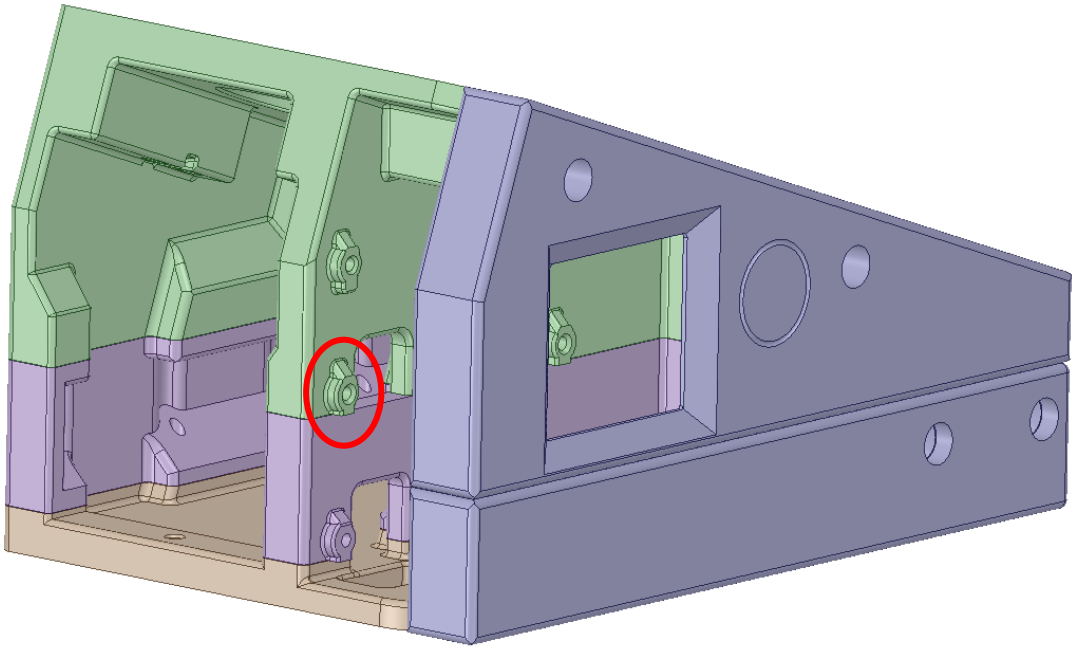
1. Still valid the notes on chapter 3D printed parts (i.e. no support needed).
2. Above Filament consumption and printing time are based on a quality printing setting; If these parts are really meant to be temporary in your case, a much lower quality could be considered.
3. The suggested part orientation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

Part name	3D print orientation
Extension_left	
Extension_middle	
Extension_right	

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

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

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



Assembly:

- The three additional parts are connected by screws.
- Use M3x12mm cylindrical, to connect the Extension_left and the Extension_middle toward the Structure.

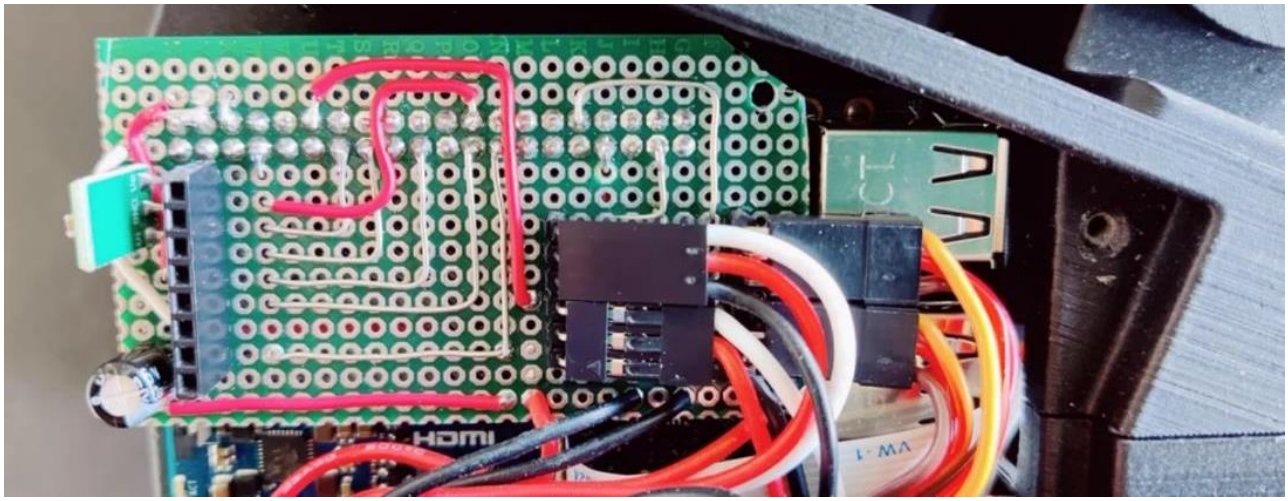
  Note: In case you don't have Allen keys with "spherical" head, to reach the screw under an angle, it might be necessary to use screws with conical head; M3 screws with conical head uses smaller Allen key, that can be used through the holes straight in front.
- Use M3x12mm conical head, to connect the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

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

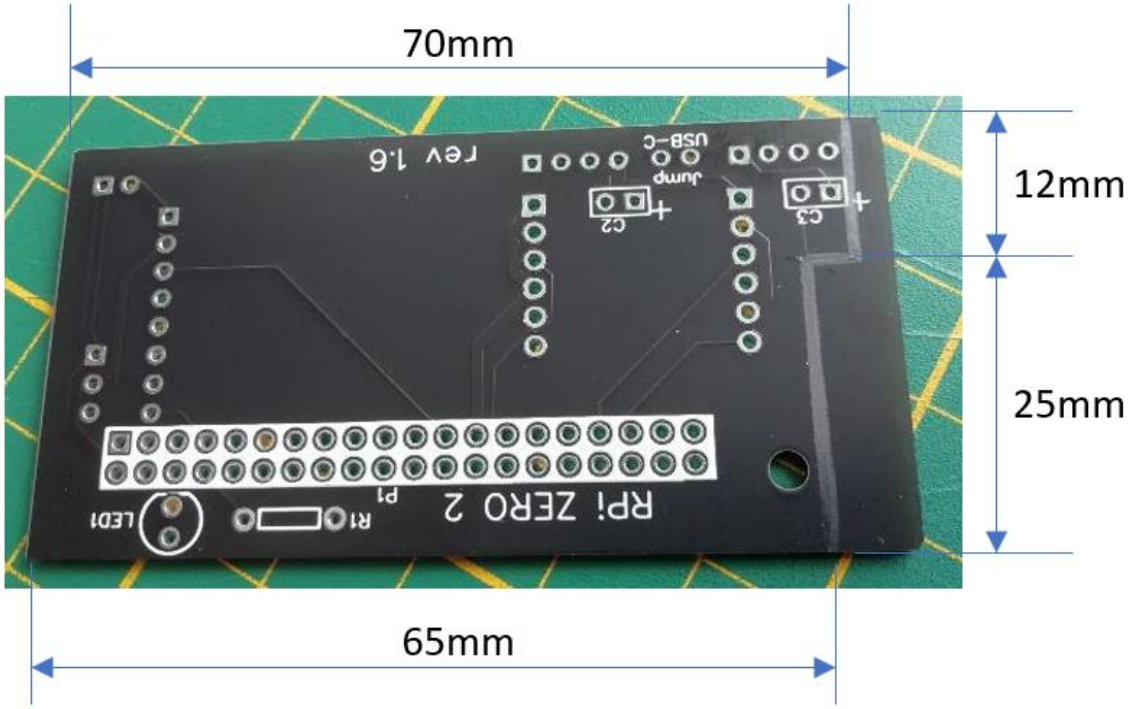
If you make the connections_board, out of a proto-board, the important changes are:

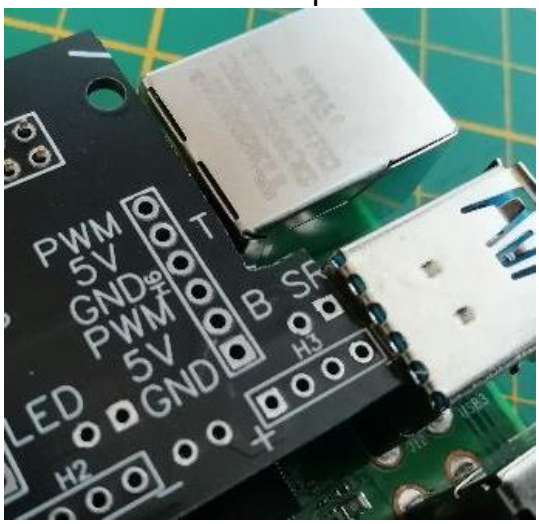
- limit the protrusion of the proto-board, to the right side, to max 4 / 4.5 holes from the last used pin of the Raspberry Pi GPIO connector.
- Solder the C3 capacitor directly to pins 1 and 2 of the (H6) header for the bottom servo.

Below an image from Maker **nrdlrd**, who had used a Rapberry Pi 3:



The V1.6 board can also be used, in combination with Raspberry Pi 3 or 4, by cutting out part of the right side.
In case of **Raspberry Pi 3** the board can have a straight cut, to reduce the board length from 74mm to 70mm.
In case of **Raspberry Pi 4** the board requires a slightly more complex cut:





Notes

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

28) Tuning

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

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

But hey, don't worry …. Other makers have successfully tuned their own Cubotino, and you will too 😊
In case things are getting too difficult, check out the Instructables chats (there are chances other people have asked the same questions); Differently, consider dropping specific questions to that chat.

1. General:

There are parameters that are expected to be differently tuned on each robot.
These parameters are grouped into two (json) text files: See Parameters and settings chapters.
Some of those parameters are quite likely to require tuning, because each robot will slightly differ from others:

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

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

2. Setting servos angles:

The servos at supplies list have 180° of rotation, that is more than sufficient for the lifter and Top_cover angle of this robot, and it should be right sufficient to the Cube_holder; I don't suggest buying 270° servo as this will affect the angle resolution.
Apart from tolerances between different servos, one variation source is the connections between the servo arms, and the servo's outlet gear, having many possible positions (I believe there are 25 teeth).
This means the reference angles set on Cubotino_T_servo_settings.txt working fine on my robot, are not necessarily the best choice on other systems: **These parameters must be tuned on each system!**

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

The servos at supplies list, accept a Pulse Width signal from 1ms to 2ms, wherein 1.5ms is the mid angle.
It is anyhow possible to use servos with different pulse width range, in that case adjust the min_pulse_width and max_pulse_width parameters for the related servo, see Parameters and setting for more info.

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

Detailed info on servo management:

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

    In this case it is necessary to change the minimum and maximum pulse width parameter at the Cubotino_T_servo_settings.txt file (t_min_pulse_with, t_max_pulse_with, b_min_pulse_with, b_max_pulse_with) with values that best fit your servos.
    Save the text file and re-launch the Cubotino_T_servo.py (pulse width parameters are uploaded when this script is started)
    Re-check the servo angle range: If 180 degrees are now covered set the servos to their mid position.
11. In case the servo for the Cube_holder makes less than 180° rotation, despite you've tried to enlarge the Pulse Width range, it is necessary to increase the rotation range to slightly more than 180° in order to get the robot working properly.

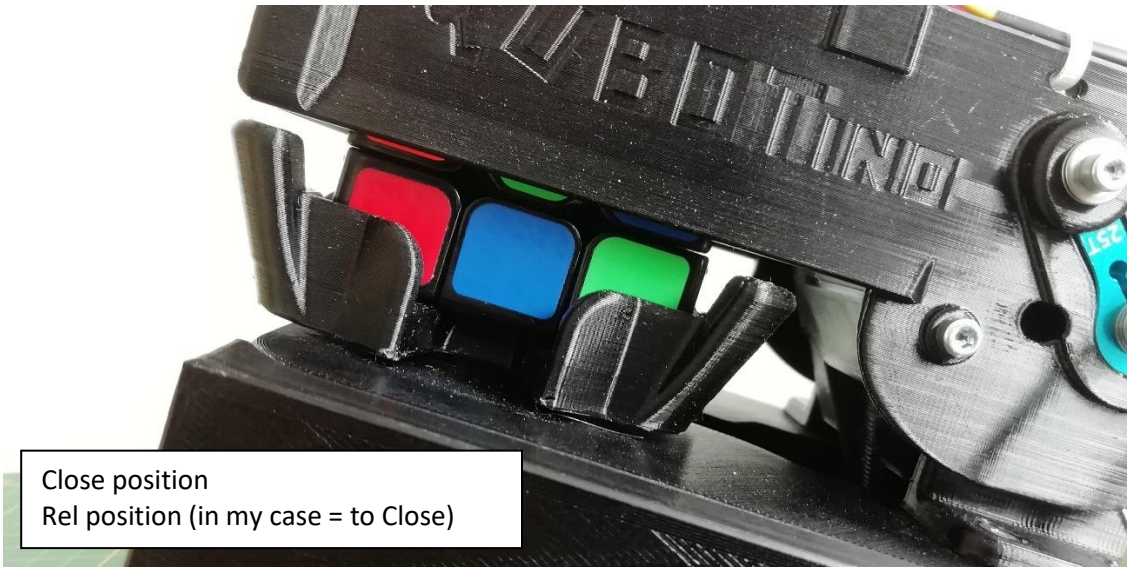
    There are tutorials in internet on how to increase the rotation angle, by adding some resistors in series with the servo potentiometer (servo must be opened for this eventual change).
    2k2 Ω resistors were used by Jonesee (reference https://www.thingiverse.com/make:1034575).
    Andy (search for G7UHN at https://www.instructables.com/CUBOTino-a-Small-Simple-3D-Printed-Inexpensive-Rub/#discuss) has reported 'my servos had only about 90° range of motion out of the box (despite being advertised as 180° servos) so I had to modify them, adding 3k resistors to both ends of the internal (5k) potentiometer'.

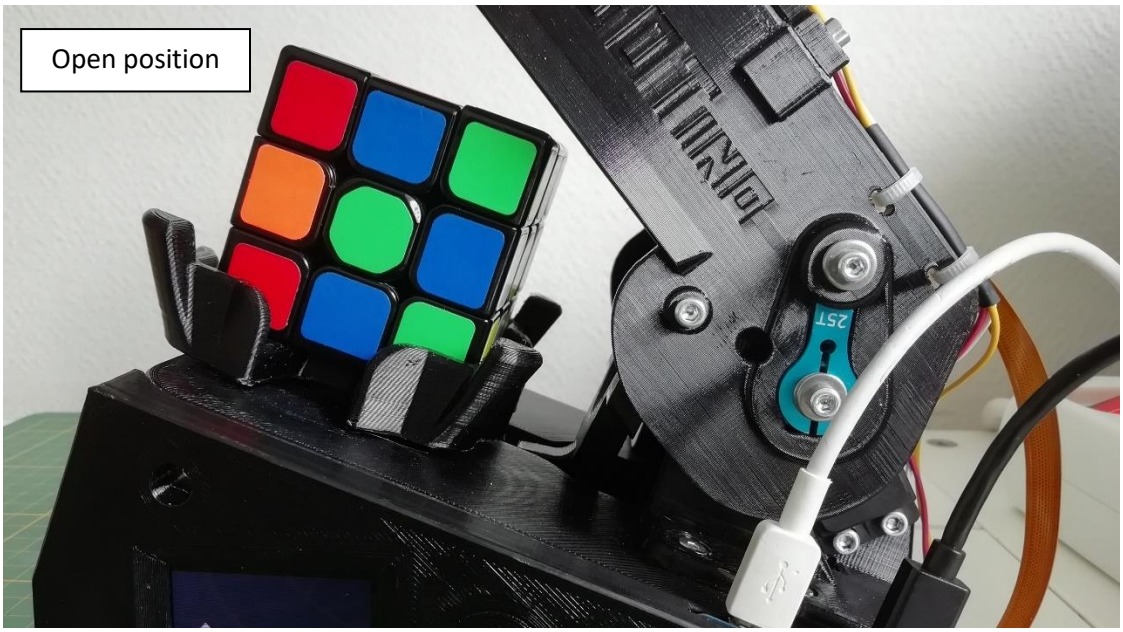
Top_cover (t_servo) angles:

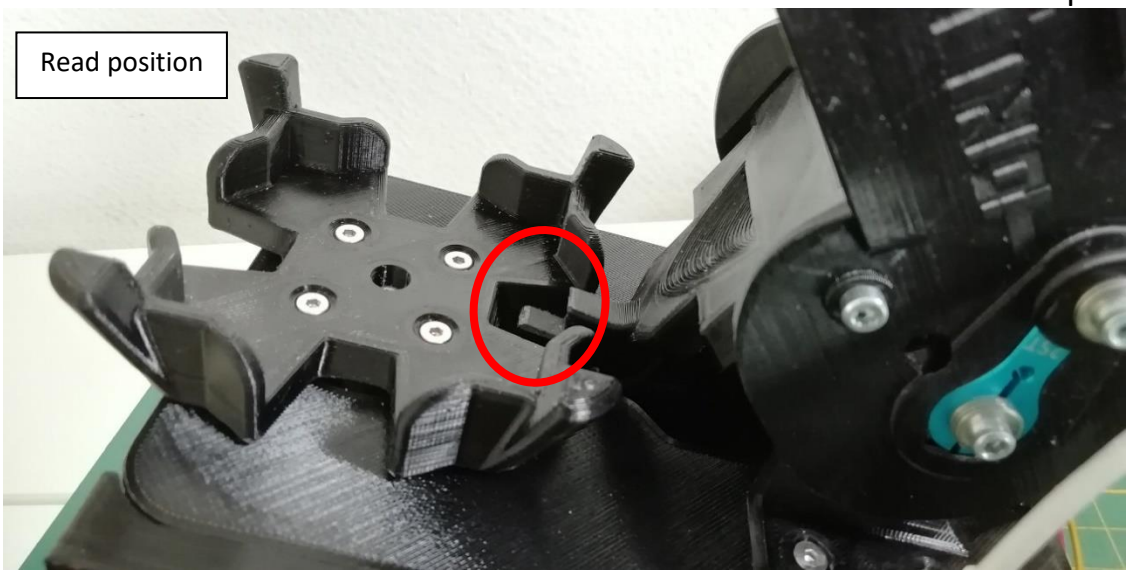
Working angles for the servos, are set in a (json) text file: Cubotino_T_servo_settings.txt

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

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






Read position


Flip position
CUBOTino

Cube_holder (b_servo) angles:

There are 3 (+4) defined angles (most of time mentioned as positions):

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

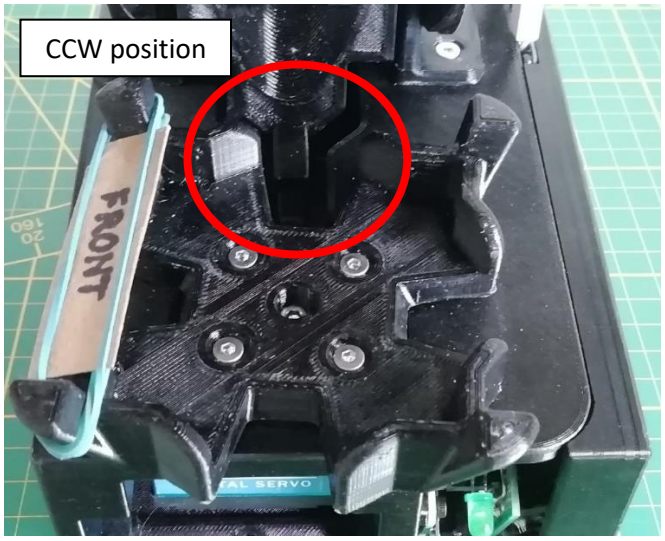
b. CW: position to spin or rotate the Cube_holder >90° CW from Home (Direction according to the motor point of view)

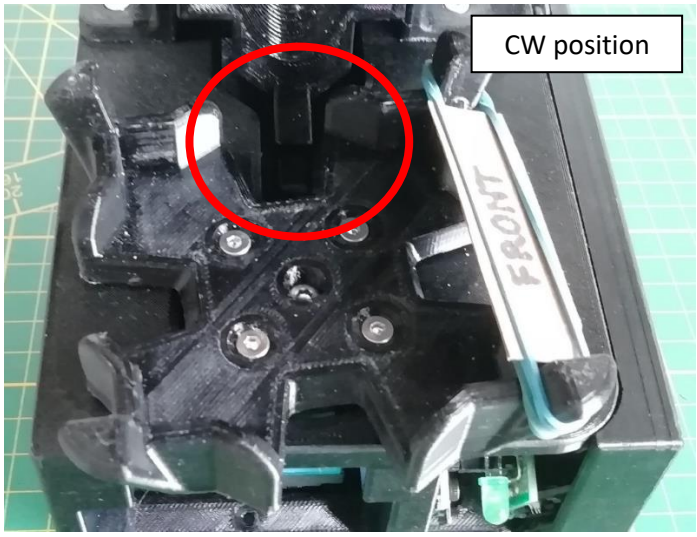
c. Home: mid position between CCW and CW

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

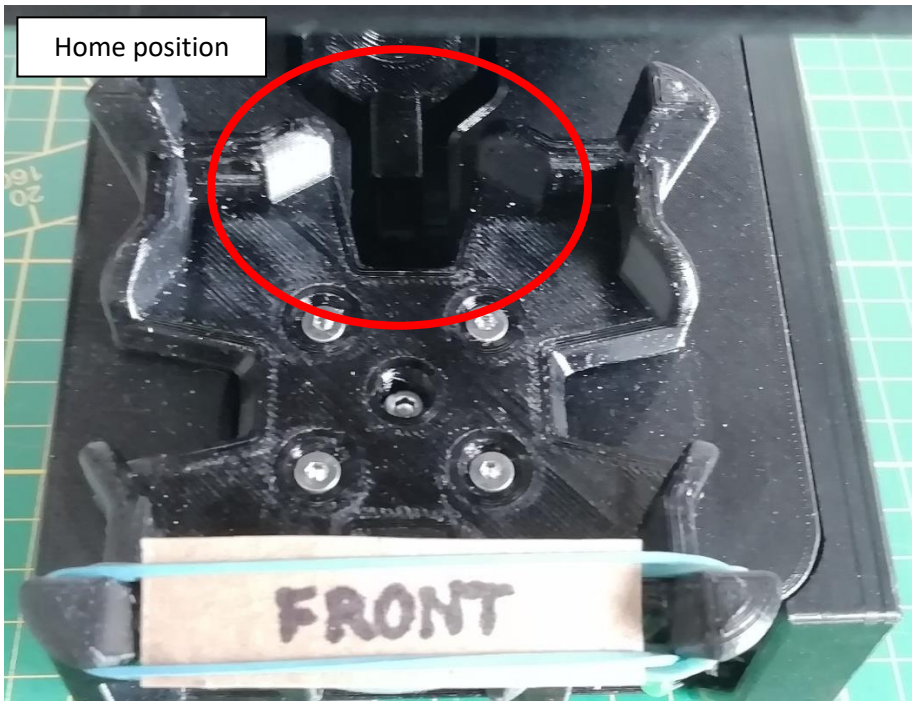
e. Rel from home: position(s) to release cube tension from Top_Cover at Home

Be noted the CCW and CW angles are slightly more than 90° apart from the Home position
This is needed in order to turn ~90° to the cube, after recovering the radial plays: There is play in between the cube and the Cube_holder, and again there is play between the cube and the Top_cover





The Home position has to be well centered.

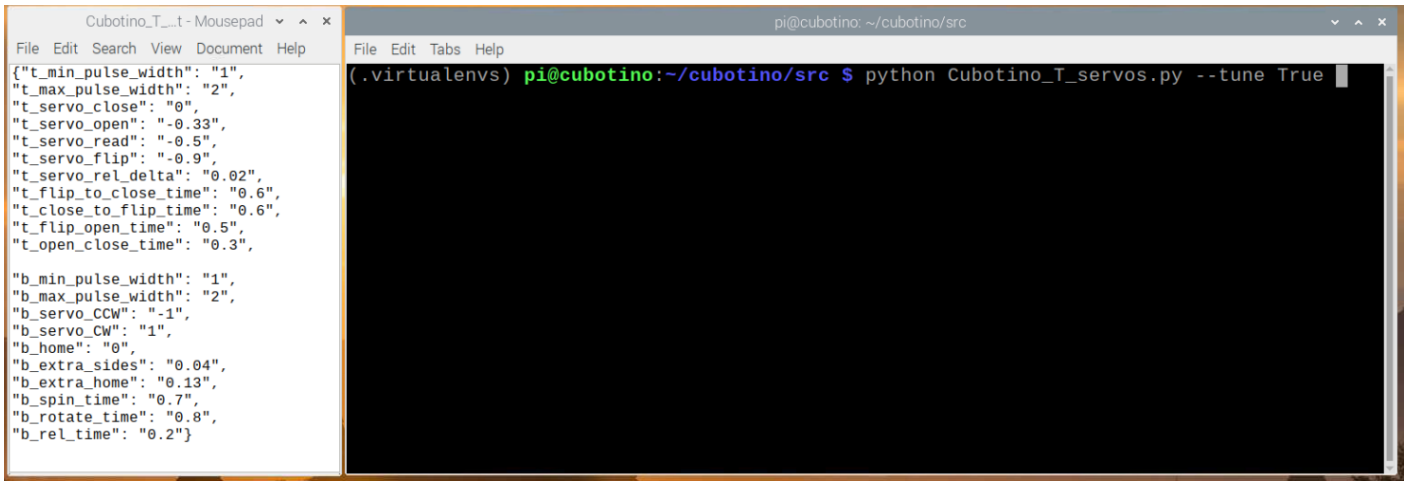


In order to center the Home position, it might be necessary to adjust the below parameters:

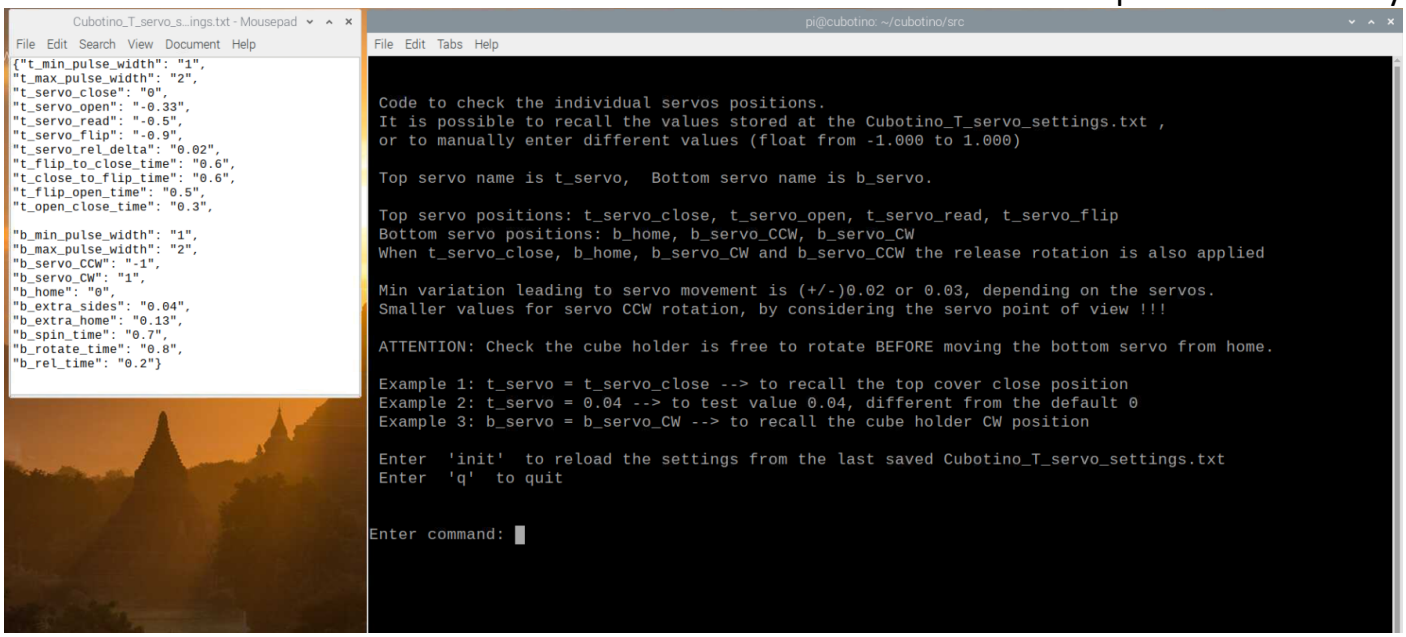
- b_min_pulse_width
- b_max_pulse_width
- b_home

3. Fine tuning servos angles (changed on 25th July 2022):

a. set the Servos to the mid angle (see Servos test and set to mid position chapter)
b. assemble the robot
c. enter the cube folder (*cd cubotino/src*) and activate the venv (*source .virtualenvs/bin/activate*); Attention to the dot in front of virtualenvs
d. run the script *python Cubotino_T_servos.py –tune True* ; Attention to the space in between '-'
e. some info will be printed on the Terminal, to guide this process
f. it is possible to recall the settings stored at Cubotino_T_servo_settings.txt as well as to ender different target values (value should be a float ranging from -1.000 to 1.000)
g. After the 'Enter command:' type the below commands to test the servos positions:
   - t_servo = t_servo_close
   - t_servo = t_servo_open
   - t_servo = t_servo_read
   - t_servo = t_servo_flip
   - b_servo = b_home
   - b_servo = b_servo_CCW
   - b_servo = b_servo_CW
h. 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.
i. Once the position(s) are satisfactory, edit the *Cubotino_T_servo_settings.txt* and save the file to apply the new settings; It is suggested to keep open the Cubotino_T_servo_settings.txt file at the side, to copy paste the command (use shift Ins to paste) and to updare and save the new settings



j. If you type *init* , instead of a command after the 'Enter command:', the python script reloads the new settings from Cubotino_T_servo_settings.txt, so to verify if everything goes well.



k. run the script *python Cubotino_T_servos.py* ,without any argument, to test the robot manoeuvring the cube like during a solving process; Take a close look to check if the cube handling is ok.
l. If the cube layers don't align well, if is suggested to apply some stickers on the cube holder: When the cube holder is home place an 'F' on the cube holder front side, 'L' on the cube holder left side and 'R' on the cube holder right side. Take a movie while the robot manoeuvres a cube and watch it back to see in which position the misalignment is generated.
m. 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 to try 0.02 (0 + 0.02) by typing *'t_servo = 0.02'.*

In case the Top_cover is too close to the cube then the servo position requires to decrease the CW position (CW and CCW are frm the servo point of view).
To decrese 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 to try -0.04 (0 – 0.04) by typing *'t_servo = - 04 '.*

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

4. Timers for servos:

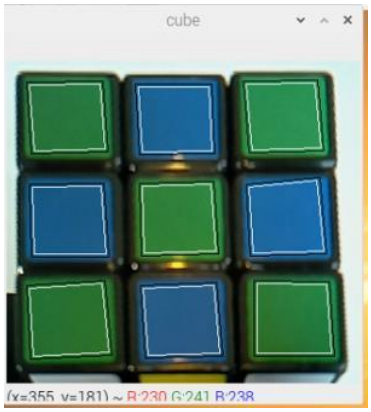
Servos don't provide feedback when they have completed the requested angular rotation; It's necessary to set appropriate waiting time in the script, to allow sufficient time for the servo to complete the action.
It will be convenient to use larger delays at the beginning, and progressively reduce them once the servo angles are adjusted to your system: The default values I've set for the default should be more than sufficient to allow the servos intended rotations.
Once your robot runs smoothly, and in case you' like to reduce the solving time, then you might start reducing those timers. As reference you can check on "Parameters and settings" for the values I'm using on my robot.
Timers for the servos, are set in a (json) text file: Cubotino_T_servo_settings.txt

5. Frame cropping:

Cropping parameters are set in a (json) text file: Cubotino_T_ settings.txt
PiCamera position, and its FoV, are likely to read both top and back cube faces.
Cubotino_T.py file is delivered with no cropping effect (variables set to zero): this to help the camera assembly angle to be set to get the cube top face centered: First picture below as example.

Image cropping has to 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

6. Frameless cube:

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

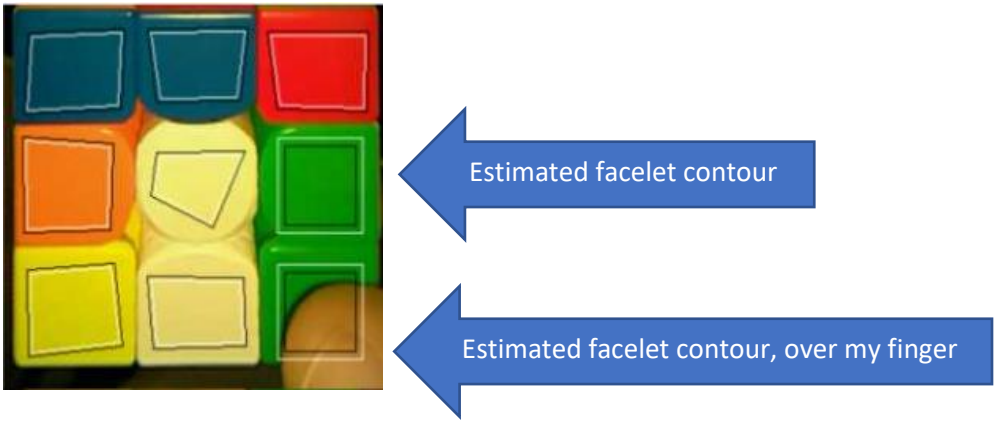
The cube facelets detection algorithm has been initially developed for the classic cubes, those having the black frame around the facelets.
Starting from 6th August 2022, it is also possible to use the frameless type of cubes
At Cubotino_T_settings.txt there is a "frameless_cube" parameter, with below options:

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

The 'auto' mode is computationally more demanding, therefore it will take slightly longer (about 1 to two seconds more for the six cube faces).
In the case the Cubotino_T_settings.txt doesn't have the key "frameless_cube", because you are re-using an old setting files, then the new Cubotino_T.py file version will simply consider the classic cube type.

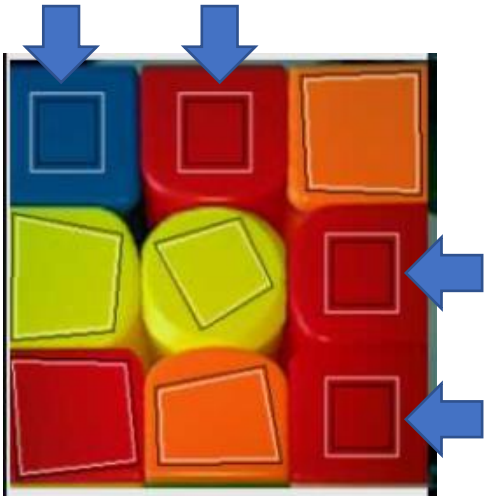
You might argue on the reason 'auto' is not the default choice for the cube facelets detection method.
The reason is that the used strategy for the frameless cube is less robust.
When facelets_cube is set 'true' or 'auto', as soon as there are at least 5 detected facelets, without an empty row or column, the remaining facelets will be estimated on their position.

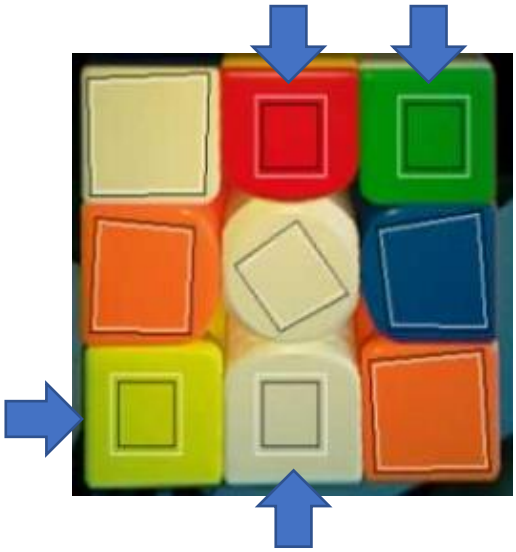
This approach helps when adjacent facelets have the same colour, but it doesn't prevent your finger (or cube logo) to get captured: See below image, with my finger not been evaluated by the algorithm, as it was on the "last two" facelets, and later leading o wrong colour detection from that facelet (likely cube status detection error)
Estimated facelets have the white contour placed outside the black one.



Examples of estimated facelets:

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





Be noted one of the estimated facelets goes to the central white one with the logo.
When the facelets have a printed logo, a defect, or some pollution, it will make more difficult to be detected as facelet, therefore those facelets will end on those estimated.
In this case it will be rather possible to get a detection error, as the average colour retrieved from that facelet isn't very similar to other white facelets.
Apart from the printed logo, below example shows again 4 adjacent estimated facelets.

7. Display initialization and test:

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

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

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

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

a. enter the cube folder (*cd ~/cubotino/src*)

b. activate the venv (*source .virtualenvs/bin/activate*); Attention to the dot in front of virtualenvs

c. run the script *python Cubotino_T_display* ; Attention to the space in between '-'

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

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

8. PiCamera focus

The V1.3 PiCamera is delivered with fixed focus.

On my first project with PiCamera I did adjust the camera focus, by following this tutorial

https://projects.raspberrypi.org/en/projects/infrared-bird-box/6

It hadn't been easy, yet I managed.

After correcting the focus, I did not glue the lens, as there still was quite some friction, preventing the lens from getting loose.

When I tried to repeat this operation on a second PiCamera I have almost destroyed it.

Both my Cubotinos have PiCamera "as received", one Cubotino has okish focus while the second one is out of focus.

In both cases the robot work properly, the only difference is the sharpness on the images the robot saves.

In case of non-satisfactory results, and if you are good with your hands, you might decide to try … on your own risk.

29) Parameters and settings

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

- Cubotino_T_settings.txt and Cubotino_T_servo_settings.txt

In order to provide a reference, the below json files capture the settings used on my robot:

- Cubotino_T_settings_AF.txt and Cubotino_T_servo_settings_AF.txt

On below tables are listed these parameters, with the proposed value to start the tuning, the value that work on my Cubotino, and some information.

Highlighted the default settings that differ from mine

Cubotino_T_settings.txt (and Cubotino_T_settings_AF.txt), part 1

Parameter (dict key)	Default value	AF value	Data type	Info
frameless_cube	false	auto	string	Set the facelets edge detection according to the cube. Options are: 'false', 'true' and 'auto'. If the parameter key is missed (i.e. old setting files) a classic cube is considered, meaning frameless_cube = false. frameless_cube = true should be used when small logos on the cube.
disp_width	130	132	Int	Display width (in pixels); At troubleshooting for Display the explanation to make easy to adjust this parameter this parameter
disp_heigh	160	162	Int	Display height (in pixels); At troubleshooting for Display the explanation to make easy to adjust this parameter this parameter
disp_offsetL	0	-2	Int	Display offset on width **Left** (in pixels); At troubleshooting for Display the explanation to make easy to adjust this parameter this parameter
disp_offsetT	0	-2	int	Display offset on height **Top** (in pixels); At troubleshooting for Display the explanation to make easy to adjust this parameter this parameter
camera_width_res	640	640	Int	Picamera resolution on width. Changes to the Camera resolution has large influence on many functions, and computation time
camera_hight_res	480	480	int	Picamera resolution on height. Changes to the Camera resolution has large influence on many functions, and computation time
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

Cubotino_T_settings.txt (and Cubotino_T_settings_AF.txt), part 2

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

Cubotino_T_settings.txt (and Cubotino_T_settings_AF.txt), part 3

sv_max_moves	20	20	int	Max number of moves requested to the Kociemba solver. When the solver finds a solution matching this movement quantity, that solution is returned even before the timeout expiration (sv_max_time).
sv_max_time	2	2	float	Timeout, in seconds, for the Kociemba solver. The best solution found within the timeout is returned at timeout end, even if it doesn't match the desired max quantity of moves.
collage_w	1024	1024	int	Image width for the unfolded cube file. This parameter determines the image collage realization, and it makes possible to save all images with the same size.
marg_coef	0.1	0.06	float	Defines the margin around the cube faces images. This margin is used to cut the six cube faces with some margin around, for the unfolded cube collage. The margin is calculated by multiplying the detected cube diagonal in pixels to this coefficient. The larger the value the more pixels margin around the cube 0.1 means the margin is 10% of the cube diagonal (calculated on the detected facelets contours).
cam_led_bright	0.1	0.1	float	PWM for the 3W led at Top_cover. Range from 0 (no PWM) to 1 (PWM=100%). At the Cubotino_T_servos.py the max value really delivered to the LED is 30%, therefore values > 0.3 are useless.
detect_timeout	40	40	int	Timeout, in second, for the cube status detection. If the six cube faces aren't detected within this time, the cycle is terminated with a timeout message on the display.
show_time	7	7	int	Time, in seconds, the unfolded cube image is kept on screen. This only applies when a screen (i.e. VNC) is connected.
warn_time	1.5	1.5	float	Time from touching the touch button (after 0.5s filter), after which a warning appears on display. This time only applies when the touch button is pressed while the robot is working (cube status detection or cube solving), in order to allow for the STOP function and preventing unwanted SHUT-OFF (longer press, anticipated by a warning on the screen).
quit_time	4.5	4.5	float	Time from touching the touch button (after 0.5s filter), after which the script starts the quit procedure. This timer starts right after the warn_time elapses.
cover_self_close	false	true	string	Top_cover auto closing at shut down. Options are 'false' and 'true'. If the parameter key is missed (i.e. old setting files) the cover_self_close is set false.

Parameters related to the servos;

Notes:

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

Cubotino_T_servo_settings.txt (and Cubotino_T_servo_settings_AF.txt):

Parameter (dict key)	Default value	AF value	Data type	Info
t_min_pulse_width	1	1	float	Min pulse width, in ms of the used top servo. Most of the servos accepts a slightly extended value (<1)
t_max_pulse_width	2	2	float	Max pulse width, in ms, of the used top servo. Most of the servo accepts a slightly extended value (>2)
t_servo_close	0	0	float	"Angle" for Top_cover to constrain the top and mid cube layers
t_servo_open	-0.33	-0.33	float	"Angle" for Top_cover not constraining the cube and Cube_holder
t_servo_read	-0.5	-0.5	float	"Angle" for Top_cover for PiCamera reading. Lifter almost touching the bottom cube face
t_servo_flip	-0.9	-0.9	float	"Angle" for Top_cover to flip the cube (~2 cube layers)
t_servo_rel_delta	0.02	0.06	float	Delta "angle" for Top_cover to retract after closing
t_flip_to_close_time	0.6	0.38	float	Time for t_servo to move from Flip to Close position
t_close_to_flip_time	0.6	0.42	float	Time for t_servo to move from Flip to Close position
t_flip_open_time	0.5	0.33	float	Time for t_servo to move from Flip to Open position and viceversa
t_open_close_time	0.3	0.1	float	Time for t_servo to move from Close to Open position and viceversa
b_min_pulse_width	1	0.98	float	Min pulse width, in ms, of the used bottom servo. Most of the servos accepts a slightly extended value (<1)
b_max_pulse_width	2	1.98	float	Max pulse width, in ms, of the used bottom servo. Most of the servo accepts a slightly extended value (>2)
b_servo_CCW	-1	-1	float	"Angle" for the Cube_holder at ~90° CCW from Home. CCW is from motor point of view
b_servo_CW	1	1	float	"Angle" for the Cube_holder at ~90° CW from Home. CW is from motor point of view
b_home	0	0	float	"Angle" for the Cube_holder in between CW and CCW positions
b_extra_sides	0.04	0.06	float	"Delta angle" for the Cube_holder to retract from CCW or CW
b_extra_home	0.13	0.1	float	"Delta angle" for the Cube_holder to do before retracting Home
b_spin_time	0.7	0.45	float	Time for the Cube_holder to spin ~90° (cube not constrained)
b_rotate_time	0.8	0.55	float	Time for the Cube_holder to rotate ~90° (cube constrained)
b_rel_time	0.2	0	float	Time for the Cube_holder to rotate back, at CCW, CW and Home

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

30) Troubleshooting

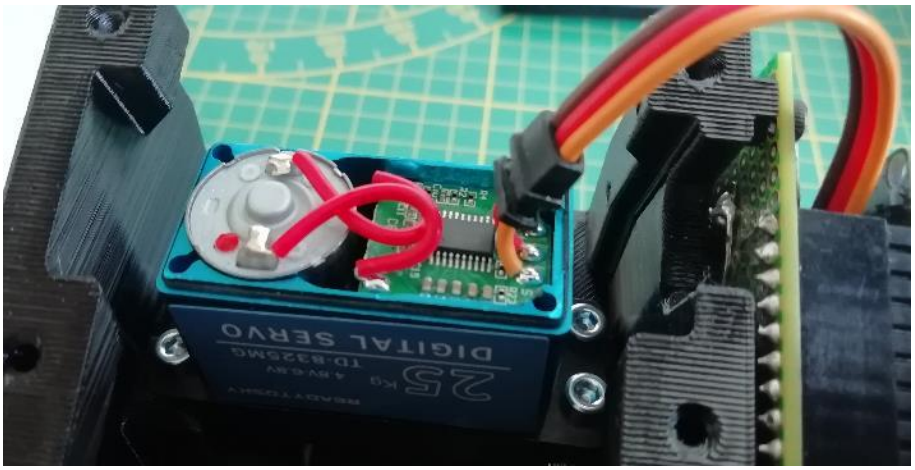
Some of the below aspects were encountered during the robot development, other were posted at Instructables, and remaining are hypothetical:

1. Servos rotate about 180 deg, yet not more than that.
2. Servos rotation angle of about 90 deg, when you've ordered 180deg
3. Servos rotation angle of about 270deg
4. Servos not moving smoothly
5. Bottom cube layer doesn't align nicely
6. Top cover usage to flat the cube
7. Cube status detection error
8. Robot stuck on reading the same face
9. Cube's facelet and light reflection (cube status detection)
10. Displayed text and images are un-readable
11. Program doesn't work as intended
12. PiCamera focus

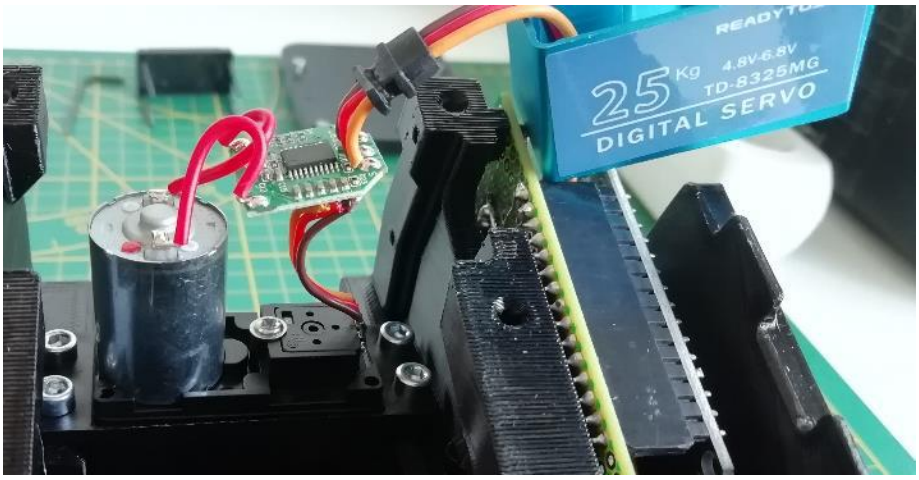
1. Increase the servo rotation range (if needed)

It's possible to increase the rotation angle, by adding one or two resistors in series with the servo potentiometer (servo must be opened for this change). In my case I had 1 servo (out of 8 used so far) with insufficient rotation range.

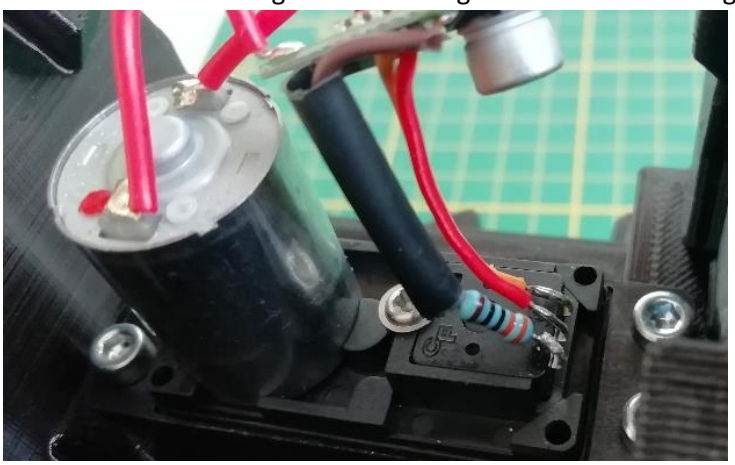
1. Open the servo (4 very long screws)



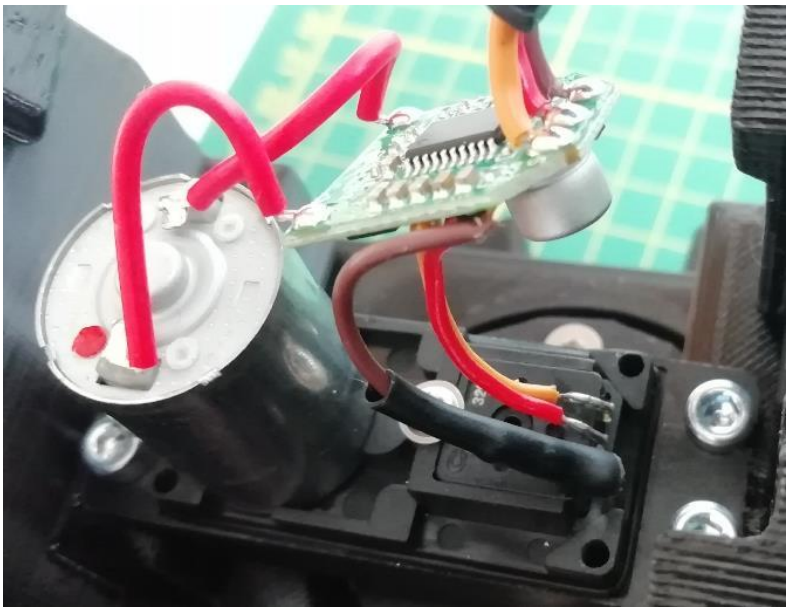
2. Slide out the pcb, and afterward slide the case out of the way.



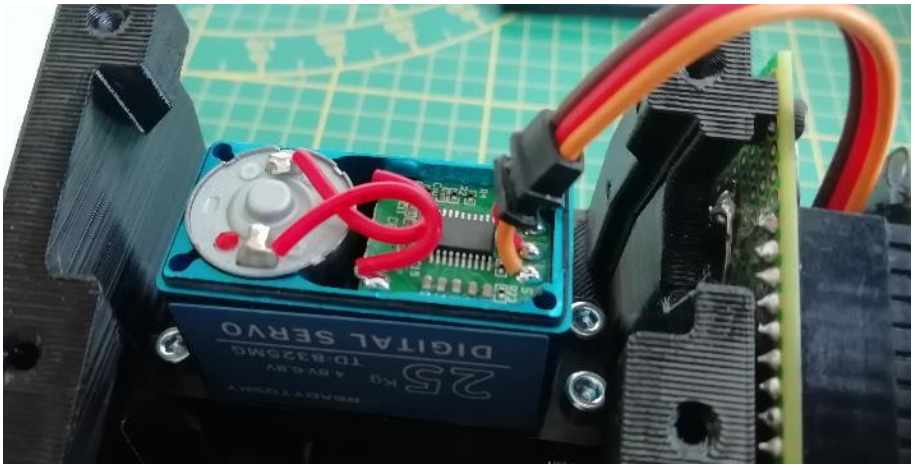
3. Solder one resistor to one external pin of the potentiometer (or one resistor per each of the two external potentiometer pins to keep the same PWM value for the mid position).
   I've added 330ohm to gain about 5 degrees on a servo having pulse width from 1 to 2 ms.



4. Protect the soldered parts with a sleeve



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

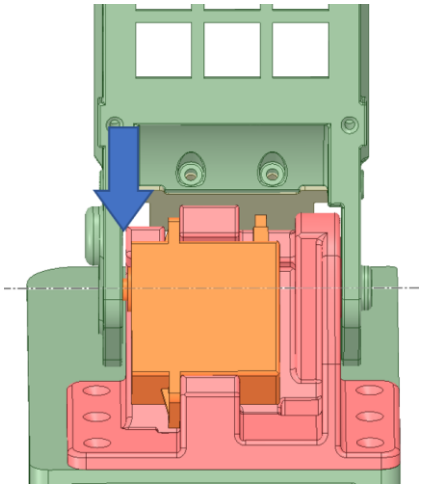


6. Place the cover and close the 4 screws

2. **Servos rotate about 90deg while you've ordered 180deg servos**
   This has probably to do with the Pulse Width of the received servos, ranging from 500µs to 2500µs instead of from 1 to 2 ms.
   This project uses as default a Pulse Width range from 1m to 2ms, yet you can adjust some parameters and get your servos working fine; At Cubotino_T_servo_settings.txt change:
   - b_min_pulse_width from 1 to 0.5 (meaning the min pulse width reduces from 1ms to 0.5ms)
   - b_max_pulse_width from 2 to 2.5 (meaning the max pulse width increases from 2ms to 2.5ms)
   - t_min_pulse_width from 1 to 0.5 (meaning the min pulse width reduces from 1ms to 0.5ms)
   - t_max_pulse_width from 2 to 2.5 (meaning the max pulse width increases from 2ms to 2.5ms)

3. **In case you've got servos with 270deg rotation, it shouldn't be of a problem.**
   If the Pulse Width is from 500µs to 2500µs, then the angle resolution will be acceptable; In this case correct the Pulse Width parameters as per previous Troubleshooting point.
   In case of Pulse Width from 1ms to 2ms I don't know if the angle resolution will be acceptable for this robot.
   After adjusting the Pulse Width parameters, in case servos 222ren't from 1ms to 2ms, the default values for servos positions should be scaled by a 0.67 factor, to scale down from 270 to 180 degrees of rotation.

4. **Servos don't move smoothly**
   1. Don't use jumper wires, or use quality jumper wires
   2. Don't use bread boards, make the Connections board instead.
   3. Add the capacitors, to prevent voltage drops when servos are activated.
   4. Use an at least 2A power supply for the servos.
   5. Use a 20 to 25Kg/cm servo.
   6. Minimize Top_cover rotation friction:
      i. Ensure the hole for the M4 screw (pivot) has some gap on the Top_cover hole (Ø4.1 to Ø4.3mm).
      ii. Rub some candle wax on the screw.
      iii. Ensure the M4 screw head is not pushing toward the Top_cover; 1mm gas is suggested
      iv. Ensure gap presence between Top_cover inner surface and the Hinge at the servo gear outlet side, as per below arrow:



In case your robot has little or no gap:
1) Unscrew the M3 screws holding the top servo, and place some little spacers (0.5 max 1.0mm), in between the servo and the Hinge (possibly close to the screws location); Tighten again the screws.
2) Rub some candle wax on the Hinge surface toward the Top-cover and the Top_cover inner surface toward the Hinge.

5. Bottom cube layer doesn't align nicely:

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

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

1. Verify if the cube Holder makes an extra rotation, at both CCW and CW directions, before stopping; If this doesn't happens:
    i. Increase the timers, as too small time don't give sufficient time to the servo to make the stroke visible when testing the cube holder position.
    ii. Adapt the PWM release CCW/CW value.
    iii. Place the PWM release CCW/CW at zero, and test if the CCW and CW position have a slightly overstroke from the 90°. If this is not the case, check if the other servo has a larger rotation range. If still not the case:
        1. Try to enlarge the Pulse Width range by 0.02 or 0.04 (increase b_max_pulse_width if the Cube_holder doesn't make enough rotation at CW location, decrease b_min_pulse_width if the Cube_holder doesn't make enough rotation at CCW location)
        2. Check in internet how to (slightly) increase the servo rotation angle (additional resistors must be soldered into the servo)
2. Verify if the cube Holder makes an extra rotation, before stopping Home; If this doesn't happen, adapt the PWM release home value.

6. Top cover usage to flatten the cube:

The Top_cover isn't intended to keep pushing the cube when it's in the close position; In case the cube layers don't align nicely, by playing with the cube_Holder settings, it's possible to use the Top_cover to level the cube. By lower the Top_cover close position to have a little interference with the cube, will improve the cube layer alignment in particular after flipping the cube. In this case it will be convenient to set one or few units on *PWM release from close setting.* Via this setting is possible to release the tension between the Top_cover and the cube, after pressing it, to allow the cube_Holder to rotate with less effort.

7. Cube detection error:

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

1. Objects on the table (background); Objects on the table can form square like contour, interpreted as facelets by the cv. This can be solved by positioning the robot to a uniform-coloured surface, without cables and objects in 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_T_settings.txt the frameless_cube parameter is not 'true' or 'auto'.
6. The setting 'auto' to detect the status on cubes with and without the frame works better with good light conditions; If this isn't possible, set the parameter to the specific type of cube associated to the robot.

8. **Robot stuck on reading the same face, until timeout:**
    a. When the frameless_cube is set 'false' (classic cube type), the cube status detection algorithm must find 9 facelets with given characteristics before changing face; If the robot doesn't change the cube face, it is because some of the pre-conditions aren't met (at least 9 facelets, areas of the facelets, distance between the facelets, etc)
    b. When the frameless_cube is set 'true' or 'auto', the cube status detection algorithm must find 7 facelets with given characteristics before changing face; The remaining two are estimated for the position, not the colour.
    c. When the ambient light is rather low, and the U face is rather clear: Increase the ambient light or change the cube orientation to allow the camera to set to a more "balanced" face.
    d. When the ambient light is rather hight 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" Step8.
3) Resize the terminal to no more than half screen, and move it to the right part of the screen.
4) Run the script from the terminal
4_1) *cd ~cubotino/src*
4_2) *source .virtualenvs/bin/activate*
4_3) *python Cubotino_T.py*
5) Press start to let the robot working, and a windows will show what the camera sees.

A contour will be drawn, over the camera image, on every location interpreted as 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.

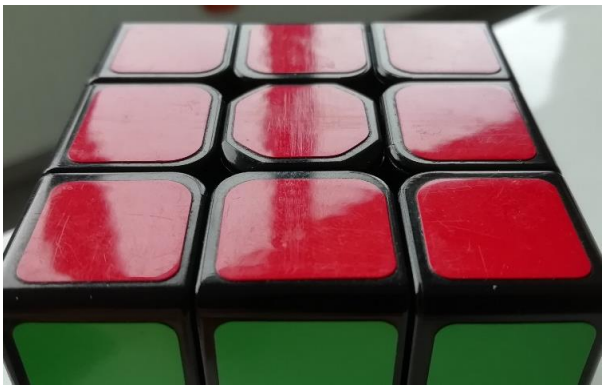
Possible reasons for the facelets detection failure:
A) the camera doesn't see the complete top face of the cube: In this case change the camera orientation angle, via the 2 screws on the camera_support, to have margin around the top cube face.
B) facelets on the back cube side are also detected (detected means that on the image contours are drawn on the back cube facelets): Apply the cropping as explained for "frame cropping" in the "Tuning" chapter
C) the critic face has a logo on the central 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.

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

Detection of edges, as well as colours, can be largely affected by light reflection made by the facelets.
I have two cubes available, one with in-moulded coloured facelets, and the other with glossy stickers.
On the cube with plastic facelet, I made the surface matt by using a fine grit sandpaper (grit 1000); This makes the cube status detection much more unsensitive to the light situations.

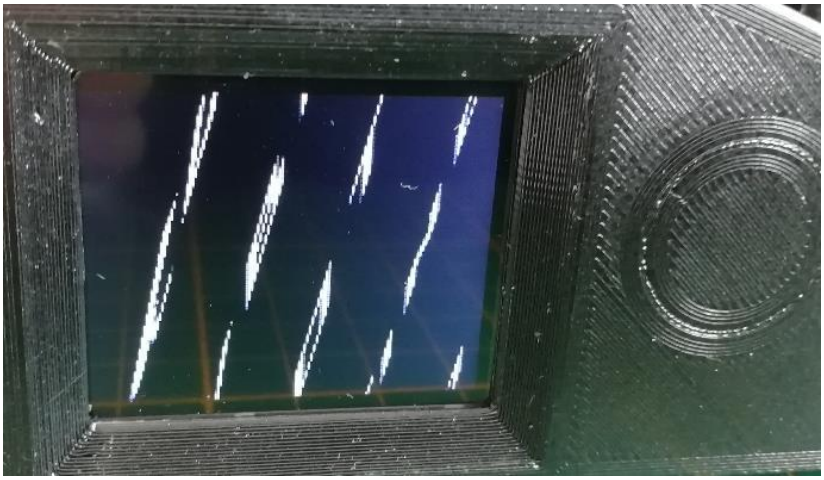
Cube with in-moulded coloured facelet, that I've made matt with sandpaper (grit 1000)

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



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

Display parameters are set in a (json) text file: Cubotino_T_settings.
The display I've received, from the supplies list, wasn't working as intended:



I took me some time to realize a sort of drifting…. And to find the fix.
By slightly changing the dimensions of just a couple of pixels on the display width parameter (in my case I had to set 128 instead of 130 pixels), the display was working.
Once that problem was solved, a second one became apparent: On two sides there were some 'tiny death bandwith', again for just a couple pixels.
By checking the ST7735.py code, I discovered it provides offset parameters, for the two display directions, suggesting these issues to be well known…..
By playing with both the display dimensions and the offsets, the display start working simply fine.
I don't expect all the display to have the same issue, yet in case the parameters variables are already in the code, and ready to be properly tuned.

11. Program doesn't work as intended:

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

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

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

When the problem seems more related to the servo program:

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

12. PiCamera focus

The V1.3 PiCamera is delivered with fixed focus.
Until now, end of October 2022, I haven't received negative feedback from other Makers because of non-working Cubotinos caused by of focus PiCameras.

On my first project with PiCamera I did adjust the camera focus, by following this tutorial
https://projects.raspberrypi.org/en/projects/infrared-bird-box/6
It hadn't been easy, yet I managed.
After correcting the focus, I did not glue the lens, as there still was quite some friction, preventing the lens from getting loose.

When I tried to repeat this operation on a second PiCamera I have almost destroyed it.
Both my Cubotinos have PiCamera "as received", one Cubotino has okish focus while the second one is out of focus.
In both cases the robot work properly, the only difference is the sharpness on the images the robot saves.
In case of non-satisfactory results, and if you are good with your hands, you might decide to try … on your own risk.

31) How to operate the robot

1. **Before starting**:
   At the robot start, the Top_cover will 'suddenly' open: Do not let kids to stick their nose right on top of robot!

2. **Use a uniform-coloured table**:
   Part of the table around the robot will be captured on cube face images.
   Keep some free space around, and use a uniform-coloured base, to prevent cables or other objects to be eventually detected as facelets.

3. **Power up the servos**:
   This considers an independent power supply for the servos
   Connect a 5V power source to the microUSB connector where the servos are connected.
   In my case the servos work flawless with a phone charger rated 2A, but I had inconsistent movements when using a phone smart-charger rated 2A, and with a normal phone charger rated 1A.
   Notes
   a. Do not energize the servos once the SBC is up and running, if you aren't sure whether the cube holder is positioned to home.
   b. If the servos are already connected (or at least energized before the Raspberry Pi boots), *Cubotino_T.py* script takes care to position the servos according a pre-defined order.

4. **Power up Raspberry Pi Zero 2:**
   Connect a 5V power source to the remoted microUSB connector where Raspberry Pi is connected.
   Do not connect phone smart-charger, those that can deliver a voltage higher than 5.1V.
   In my case the SBC works flawless with a phone charger rated 1A (note the official documentation suggest higher current).

5. **Start a solving cycle:**
   a. Position the cube on the cube holder; any cube orientation is accepted.
   b. Cube layers should be reasonably aligned.
   c. Shortly touch the PCB_cover in front to the touch sensor (the circle suggests the touch sensor location).
   d. The robot reacts by energizing the LED, and by indicating CAMERA SETUP on the display.

7. **Raspberry Pi shut down:**
   Please be noted Raspberry Pi, like normal PC, cannot be unpowered when it is working.
   To shut it down, there are few possibilities:
   a. Connect to the Raspberry Pi via SSH, and type *sudo halt -p*
      The SBC closes the open applications and files
      When the SBC activity is almost done, the led at Connections board will goes off
      Wait additional 10 seconds and the power supply can be safely removed.
   b. If the robot has proved to work without errors, un-comment last row at Cubotino_T_bash.sh file (*halt -p*).
      This means the SBC will automatically shut down when the *Cubotino_T.py* file ends.
      In order to quit the *Cubotino_T.py* script, touch the Touch_button long enough (ca 6 seconds) until the 'SHUT DOWN' appears on display; The SBC will close the open applications and files.
      Differently, iIf the touch sensor is released as soon as the display shows 'SURE TO QUIT?', then the robot will consider it as a stop request instead of a shut-down request.
      When the SBC shut-down process is almost done, the led at Connections board will goes off.
      Wait additional 10 seconds and 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 pause a risk to your kids if they have their hand on the way while the cover closes!

8. **Un-power the servos**:
   This considers an independent power supply for the servos.
   Servos can be unpowered at any moment.
   Un-power the servo before shutting down Rpi, if you experience strange servo behaviour when Rpi goes off; The script takes care to set the servo PWM related GPIO to a fix level (low), at the shut-down, but it does not work on 100% of the times.

10. Running the robot from VNC Viewer:

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

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

Steps to do:

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

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

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

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

32) Automatic robot start

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

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

The first time you'll be ask 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 /bin/sleep 5; bash -l /home/pi/src/cubotino/Cubotino_T_bash.sh > /home/pi/cubotino/src/Cubotino_T_terminal.log 2>&1'

33) Rpi shut down via Touch button

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

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

 # 'halt -p' command shuts down the raspberry pi

 # un-comment 'halt -p' command ONLY when the script works without errors

 # un-comment 'halt -p' command ONLY after making an image of the microSD

#halt -p

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

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

34) Introduction

To explain why I've started this project I've to shortly mention my first Rubik's cube solver robot….
That robot is based on a Raspberry Pi 4B (2Gb ram) with a Picamera, it reads the cube status via a camera system, and it solves it: A full autonomous robot.
The complete process takes less than one minute, some references:

- How to make it: https://www.instructables.com/Rubik-Cube-Solver-Robot-With-Raspberry-Pi-and-Pica/
- Youtube: https://youtu.be/oYRXe4NyJqs

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

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

35) Project scope

Based on the introduction you probably can guess the project scope 😊
This second Rubik's cube solver robot project wants to be affordable, to attract more people and especially students into robotics and programming.

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

Project targets for this robot:

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

36) Robot name

I've started this project with the idea to write and share the instructions, and along the way I thought a robot name would make the project more complete.
By combining Cube, 231ren' and -INO, the Italian suffix for diminutives (to remark the small robot size), the chosen name is CUBOTino
By considering the Top cover, combined with the Lifter, has a "C" profile shape, then CUBOTino become:

37) Models

This project considers the robot to be scalable.
The idea is to develop three robot versions, by re-using the same mechanical part to manoeuvre the cube.

Model	Type	Main directions	Status
Base	PC dependent, for cube status and cube solution	• 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)

Notes for the Medium version:

The cube status detection method I've tried for the Medium version, is via 9 colours sensors (LDR+WS2812B leds), as explained at: https://fourboards.co.uk/rubix-cube-solving-robot

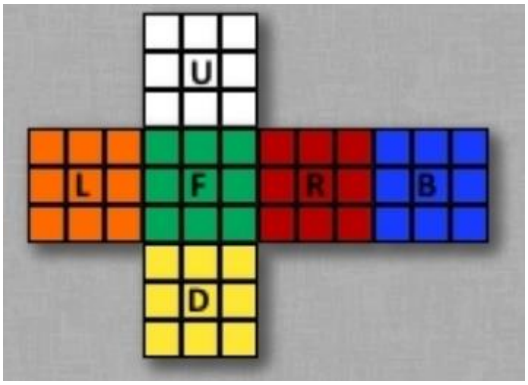
I've spent some time on this method, but the quality of my soldering has proved to don't be sufficient; I'm even doubting if it really fits the overall project scope, as this method involves too many skills.

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

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

38) High level info (Top version)

1. Re-uses most of parts from the base version; Only the PCB_cover has to be re-printed for this Top version.
2. The robot is based on a Raspberry Pi Zero2 (WH) with a 16Gb microSD.
3. A PiCamera (ver 1.3) is used to detect the cube status; Camera is placed at an angle with respect to the cube.
4. Python CV2 library is used for the computer vision part.
5. A led module is used to reduce the influence from the ambient light conditions.
6. A small display provides feedback on the robot task/progress.
7. All coded in Python.
8. This robot works with Rubik's cube size ranging from 56 to 57.5mm (those I've available); It might also work on cubes with slightly smaller size, by adjusting some parameters
9. Cube notations are from David Singmaster, limited to the uppercase (one "external layer rotation" at the time): https://en.wikipedia.org/wiki/Rubik%27s_Cube#Move_notation
10. Cube's orientation considers the Western colour scheme (https://ruwix.com/the-rubiks-cube/japanese-western-color-schemes/):
    The Western color sheme (also known as BOY: blue-orange-yellow) is the most used colour arrangement used not only on Rubik's Cubes but on the majority of cube-shaped twisty puzzles these days.
    Cubers who use this colour scheme usually start solving the Rubik's Cube with the white face and finish with the yellow; This colour scheme is also called **Minus Yellow** because if you add or extract yellow from any side you get its opposite.
    White + yellow = yellow
    red + yellow = orange
    blue + yellow = green



11. Cube solver uses the Hegbert Kociemba, "two-phase algorithm in its fully developed form with symmetry reduction and parallel search for different cube orientations", with an almost optimal target:
    - intro: https://www.speedsolving.com/threads/3x3x3-solver-in-python.64887/
    - Python script: https://github.com/hkociemba/RubiksCube-TwophaseSolver
12. When rotations are express as CW, or CCW, it is meant by facing the related cube face. For the servo the same rule is applied: CW and CCW are meant from the motor point of view.
13. Cube's sides follow the URFDLB order, and facelets are progressively numbered according that order (sketch at side); Facelets numbers are largely used as key of the dictionaries.
14. The robot detects the cube status on cubes with and without the black frame around the facelets.

			0	1	2							
			3	4	5							
			6	7	8							
36	37	38	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							

39) Construction

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

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

Construction principles:

1. The inclined cube-holder is inspired to Hans construction Tilted Twister 2.0 – LEGO Mindstorms Rubik's Cube solver – YouTube;
   This is a clever concept, as it allows to flip the cube around one of the horizontal axes by forcing a relatively small angle change (about 30 degrees, over the 20 degrees of the starting cube holder angle); Once the cube 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 servos axes; I believe Tilted Twister has a total of 8 pivots….
5. All parts are made by 3D printing:

   - This makes possible to pursue the needed geometries, also complex shapes.
   - The biggest parts can still be printed on a relatively small plate (min plate 200x200 mm).
   - Some of the parts are split, mainly for easier, and better, 3D printing; Others are split for assembly reasons.
   - All the overhangs have been designed to enable 3D printing without support.

There are many different examples of Rubik's cube solver robot, based on two servos; I think most of them are variations from the one from Hans on 2008 and the one made by Matt's on 2014:
https://hackaday.com/2014/06/28/rubiks-cube-solver-made-out-of-popsicle-sticks-and-an-arduino/
In these executions, the solution used to flip the cube on one of the horizontal axes, requires the main pivot (servo) to be placed rather far from the cube; This obviously increases a lot the overall dimensions.

40) Computer vision part

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

In this little robot, the computer vision part is achieved by combining the below elements:

- **Raspberry** Zero 2 SBC (the computer part)
- **OpenCV** (an open source library for computer vision; From https://en.wikipedia.org/wiki/OpenCV: OpenCV is a library of programming functions mainly aimed at real-time computer vision).
- **PiCamera** (a camera module, highly integrated with Raspberry Pi)

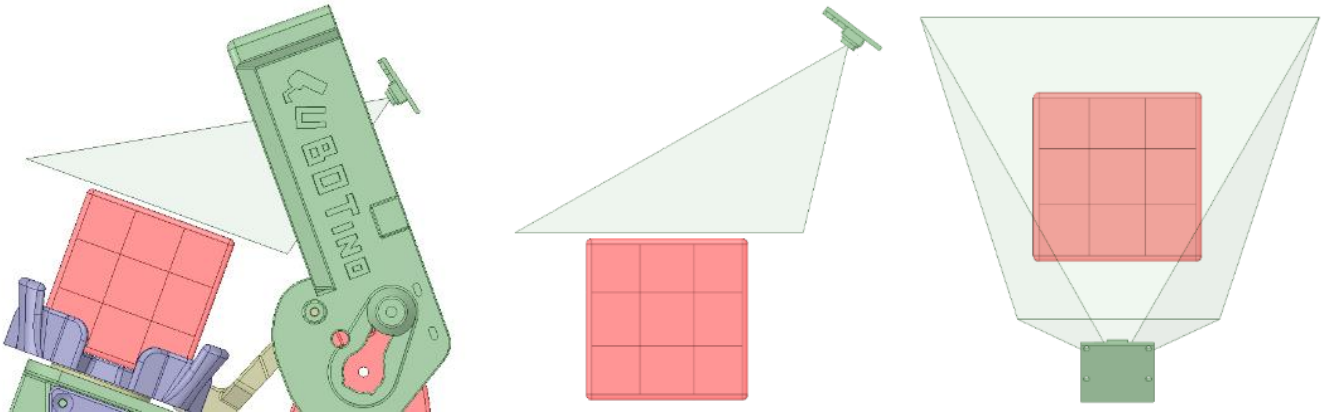
In which the python script '**Cubotino_T.py'** is responsible for the interaction with these elements.

Below listed aspects, are presented on the next pages:

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

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

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



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

1. The flex cable for Raspberry Pi Zero is max 30cm long; This prevented the possibility to mount the camera on an extension of the Top_cover (like I've done on my first robot).
2. Need to move the camera as far as possible, to have sufficient Field of View (FOV); Obviously a complete cube face has to 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 more difficult to evaluate if a detected contour is really a facelet, by evaluating if fitting a square shape.

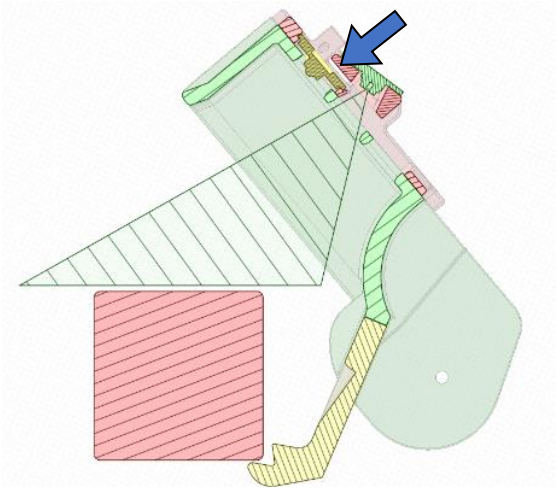
To solve the above problems:

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

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

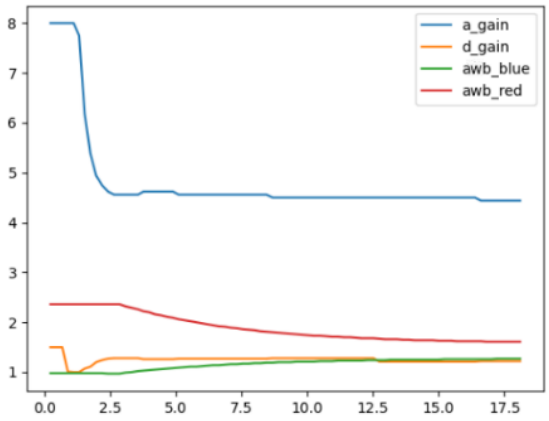
B. Taking consistent images

This is a crucial aspect for proper colour analysis.
The light source addition is a good way to mitigate the environment light conditions, typically out of our control.
On below an image it is visible the position of a 3W LED light source:



Notes on the chosen LED module:
It has 120-140 deg angle
It is controllable by the MCU
Used a fraction of the 3W

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 series of parameters: Analog gain, Digital gain, AWB (Auto White Balance) and Exposure time.
Below the variability of these parameters in time (X axis is in secs), based on measurements made on the robot:



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

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

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

To cover these situations, a so called 'warm-up' function is implemented in Cubotino_T.py script: Once all these parameters are within 2% from the average value, 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.
The camera is then set to fix shutter time, with the average value detected on 4 out of 6 faces; Of course, it will be even better to measure the Exposure time on all 6 cube faces, but only 4 faces are quick to get because of the robot construction.

The camera is now set to take consistent images
Having the LED and camera angled, has turned out to be beneficial to reduce the light reflection

C. Image analysis:

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

1. The warped image is converted to gray scale: *gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)*
2. The grayscale image is filtered with a low pass filter to reduce noise:
   - on classic cube types (frameless_cube set 'false'): *blurred = cv2.GaussianBlur(gray, …*
   - on frameless cube type (parameter_cube set 'true'): *blurred = cv2.bilateralFilter(gray, …,*
3. The de-noised grayscale image is analysed with a Canny filter; This function transform the image to binary, assigning 1 (white) the pixels detected as edges: *canny = cv2.Canny(blurred….*
   - when parameter_cube is set 'auto' the canny filter is applied on both 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 analysed with Dilate, a morphological operation, aimed to join eventual interruptions of the thin edges returned by the Canny filter: *dilated = cv2.dilate(canny,….*

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

   Anyhow I preferred to use a different kernel than Dilate, and still 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 coloured warped image, as guidance for the facelets position.
   In case the frameless_cube is set 'true' or 'auto', as soon as 7 contours are detected the remaining two are estimated for their their position.
6. The 'accepted' contours are plot over the coloured warped image, as visual feedback.

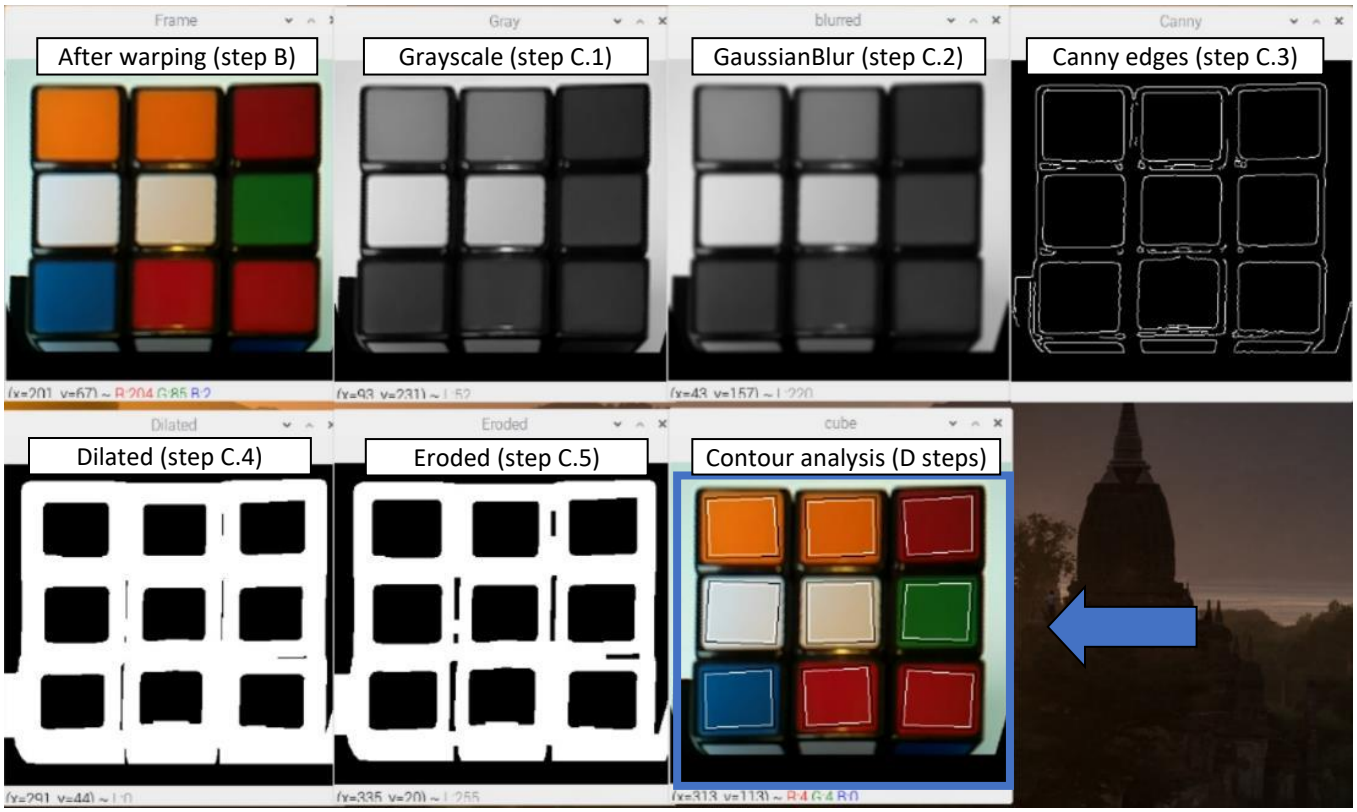
E. Colours retrieved:

On each facelet, are the retrieved 2 main info:

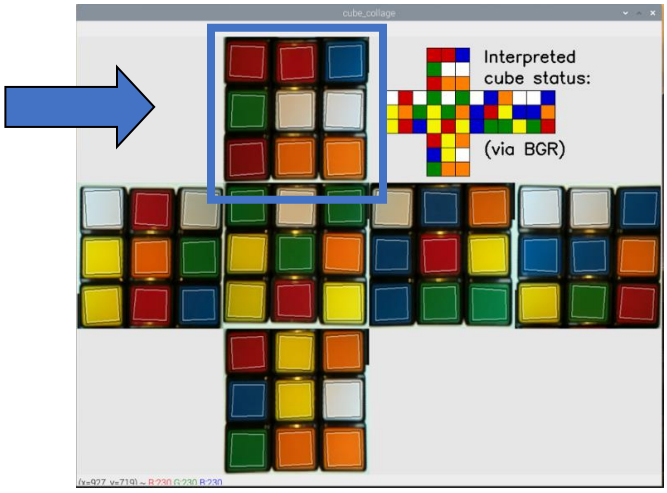
1. Average BGR, for a portion of the facelet around the detected contour center.
2. Average HSV, based on the average BGR.

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

(If you'd like to see these images on screen, change the Boolean "cv_wow" at __main__ to True)



Not all images are oriented as per user point of view: Sides UBDF are 180deg rotated.
The described image analysis process is repeated for the 6 cube faces.
The last processed image of each side, the one with the 'accepted' contours, are stored in RAM.
Once the full cube status is detected, these images are further cropped (based on the detected contours) to generate an unfolded cube image; On this "collage" further info are added, and the whole saved to the microSD.



The U side, highlighted, is the image used above to describe the image analysis

F. Is all of 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 below 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!

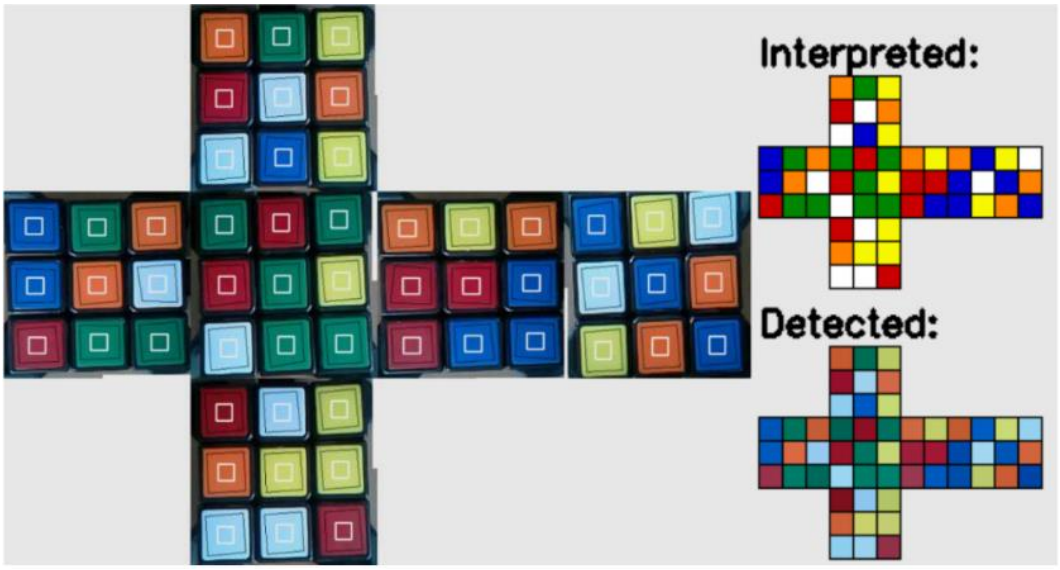
Notes:
Light reflection drastically affect the colour interpretation.
The accepted contour is on the very low contour acceptable area, yet sufficiently large to don't be noise

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

41) Colour'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 centred on the outer contour; This smaller area is used to:
      i. Measure the BGR average value, used for the colour interpretation according to the 1st method (BGR colour distance)
      ii. calculate the HSV average colours, used for colour interpretation according to the 2nd method (Hue value).



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 colours
   c. Opposite faces have known colours couples, white-yellow, red-orange, green-blue (Western colour code).
   This means we can make use of these 6 facelets as colour reference

3. The average HSV, detected on the 6 centers, is used to determine which colour is located on the 6 centers:
   a. White facelet is the one having the largest V-S delta (difference between Value, or Brightness, and Saturation), while the yellow one is located at opposite face.
   b. Remaining 4 centers are evaluated according to their Hue, and the Hue at opposite face.
   c. Orange has very low Hue, and red should be very high (almost 180); Depending on light condition, the red's Hue could "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 colours (at least their centers) have a known average HVS and therefore an average BGR colour; This also informs on the cube orientation (colours) as placed on the cube-holder.

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

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

42) Robot solver algorithm

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

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

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

The robot solver follows instead the below approach:

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

43) Python main scripts, high level info

1. *Cubotino_T.py* is the main python script on the robot; This scripts imports other custom files.
2. *Cubotino_T.py* and *Cubotino_T_servos.py* scripts use parameters with settings from two json files; This choice to group the parameters has been made for easier management, setting, communication.
3. When the script *Cubotino_T.py* is started (eventually automatically at the Raspberry pi boots), the script checks if there are monitors connected. The monitor can also be via VNC, i.e. with VNC Viewer. The presence/absence of a monitor is needed to use/skip commands requiring graphical screen communication. This prevents errors, further than having a better experience.
4. Kociemba solver is tentatively imported from different locations; venv, active folder and 'twophase' sub-folder under active folder.
5. The script uses a "tentative" approach, on a couple of analysis:
    a. (See Colour detection strategy chapter for more info) When the image is analysed, it returns contours of facelets and many unwanted ones; This happens in the function *get_facelets()*. Afterward, consecutive filters are applied to only keep contours having cube 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) When determining the cube status, according to the facelets colour; The analysis starts with a first method determining each (side and corner) facelet colour, based on the colour distance from the colours 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 also the second method doesn't provide a coherent cube status, then an error message is returned, and relevant info logged in a text file.
6. Kociemba solver:
   Kociemba solver is uploaded at the start; In case of multiple cube 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 cube solution to robot movements:
   (see Robot solver algorithm chapter for more info) Cube solutions, in Singmaster notation, sent to Cubotino_T_moves.py that returns a (long) string with the sequence robot movements. Movements are Spin, Rotate, Flip.
8. From cube robot solution to robot movements:
   Robot solution string, in Cubotino notation, is sent to Cubotino_T_servos.py that operates the servos to actuate all the intended movements.

9. Data logged:

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

Column name	**Info**
Date	Date and time (yyyymmdd_hhmmss), i.e. 20220428_213439
FramelessCube	Setting of the parameter frameless-cube at Cubotino_settings.txt
ColorAnalysisWinner	The approach that has returned a coherent cube status; Possible strings are 'BGR', 'HSV' and 'Error' (when both approaches did not provide a coherent cube status)
TotRobotTime(s)	Time, in seconds, from pressing the start button, until the cube is solved or until the robot is stopped
CameraWarmUpTime(s)	Time, in seconds, from pressing the button, until the robot ends all the camera settings
FaceletsDetectionTime(s)	Time, in seconds, from pressing the button
CubeSolutionTime(s)	Time, in seconds, used by the Kociemba solver to return the solution
RobotSolvingTime(s)	Time, in seconds, to solve the cube from when the cube solution is available
CubeStatus(BGR or HSV or BGR,HSV)	Dictionary with the average colours per each facelet, according to the colour space of the winner detecting method; In case both the detecting methods have failed, then the average colour returned by both the colour spaces are reported
CubeStatus	Cube status, in URF notation: i.e. 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)

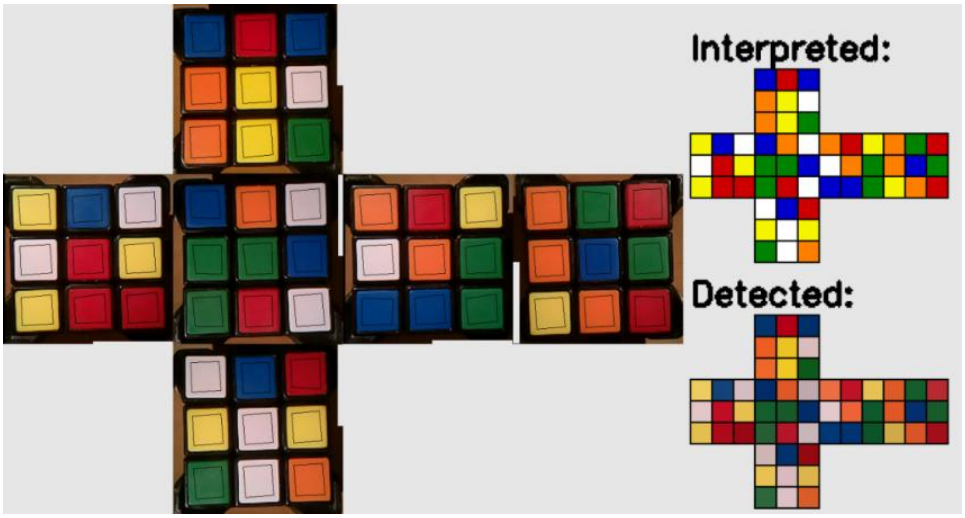
Notes:

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

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

Folder: CubesStatusPictures

Images: cube_collage_date_time.png

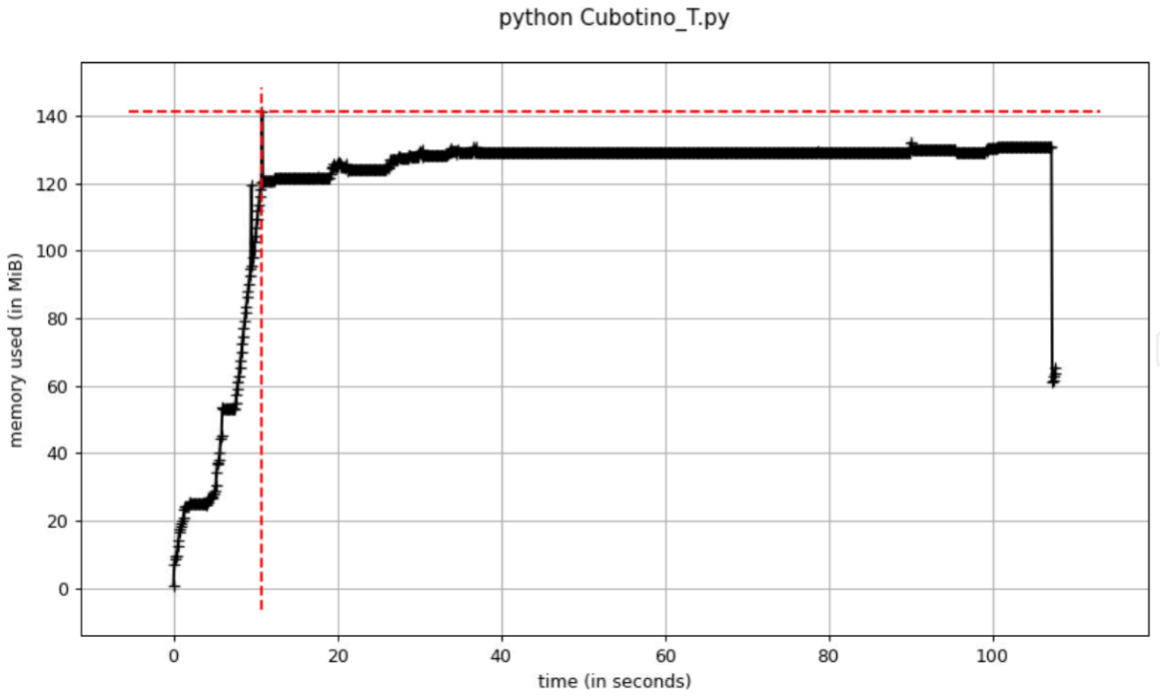


10. Date, and especially time, are used by the robot:
    Raspberry pi doesn't have an integrated RTC, therefore when the robot isn't connected to a PC and/or internet, this info could be inaccurate.
    If the robot establishes a connections to the Wi-Fi, the system time gets updated, yet this will 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.

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

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

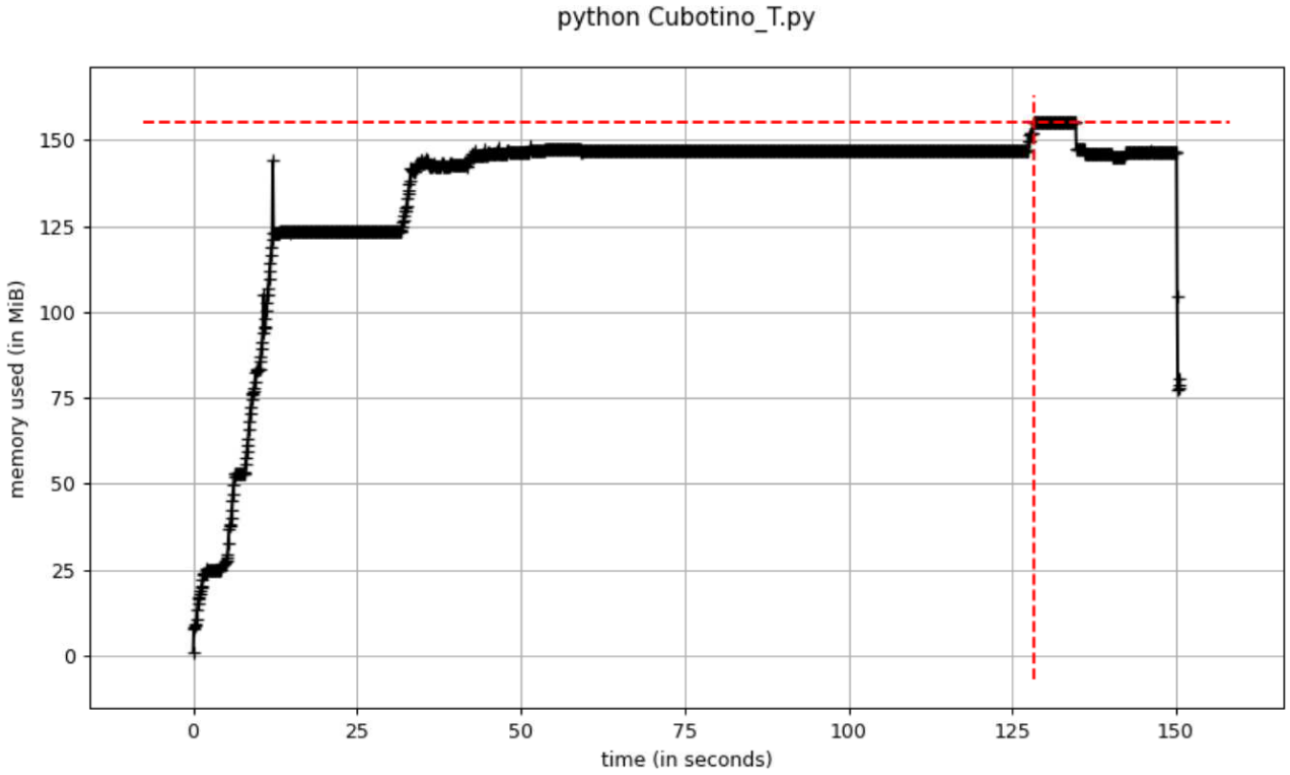
    Without VNC Viewer there is somehow lower memory usage; Below plot includes a full cycle:
    - import libraries
    - read and solve a scrambled cube
    - quit the script



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

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

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



Results:

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

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

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

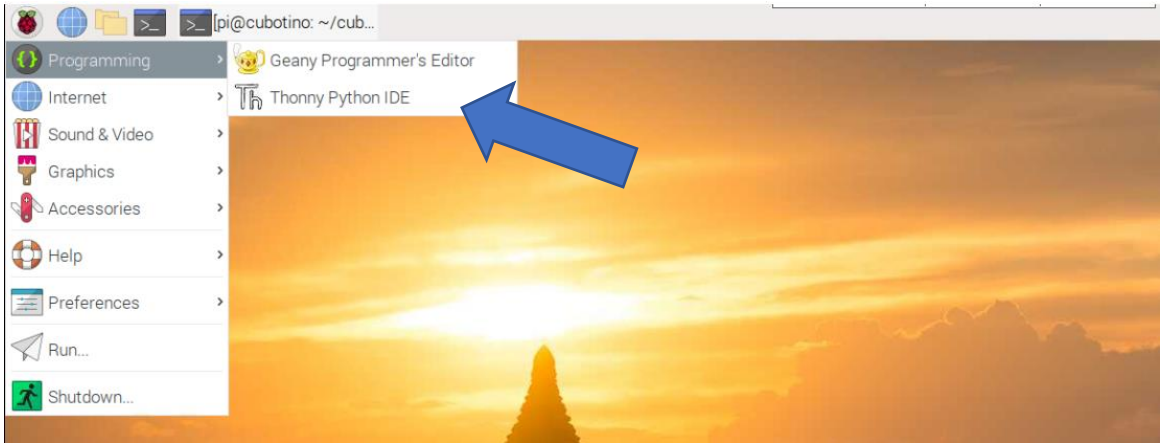
44) Set Thonny IDE interpreter

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

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

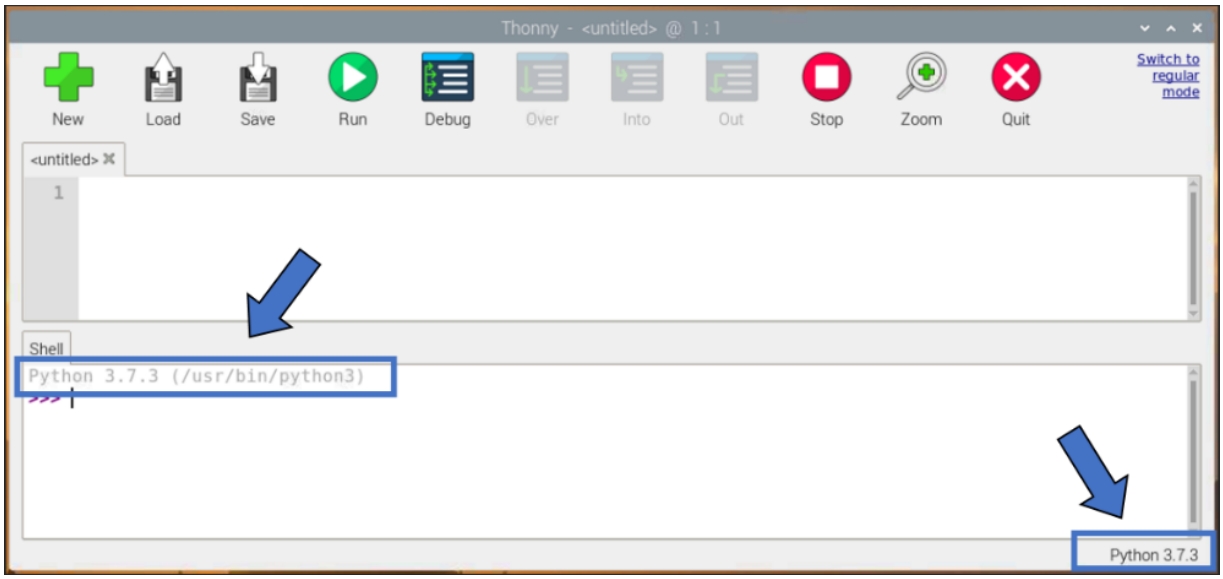
Thonny is part of the Raspberry Pi installation (according to the 'Setting up Raspberry Pi' procedure):

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

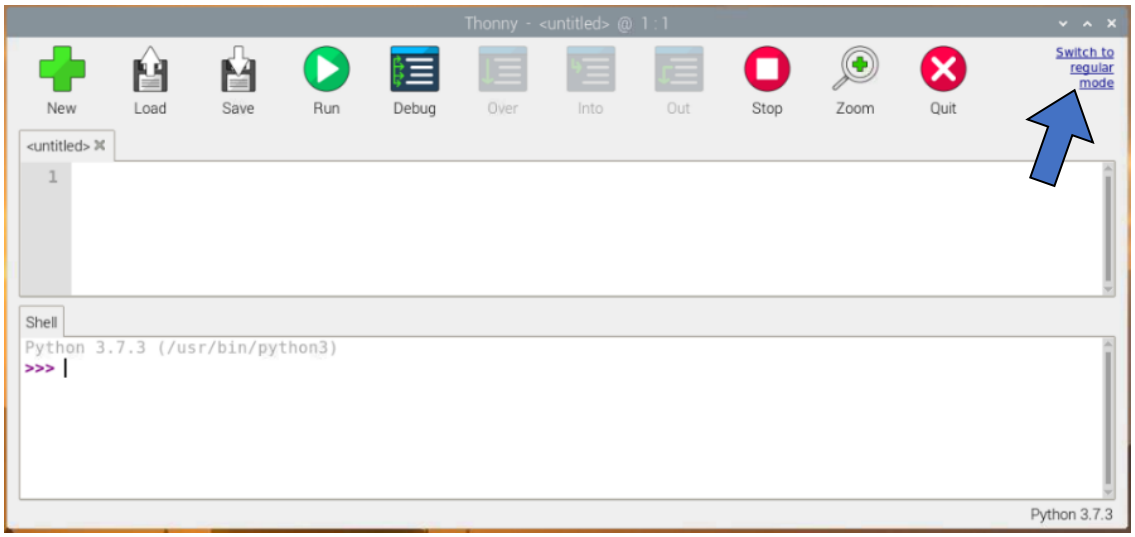


Setting up Thonny IDE interpreter, to work with the venv, it will be handy to tune the parameters hard-coded in the scripts.

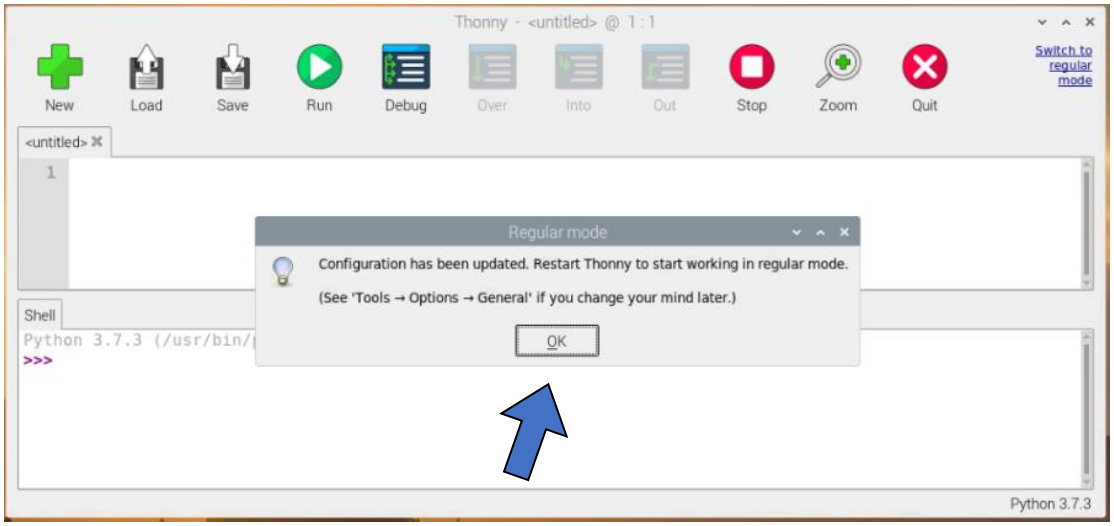
Thonny opens with the standard interpreter (/usr/bin/python3), and if you run Cubotino_T.py it won't find the libraries….



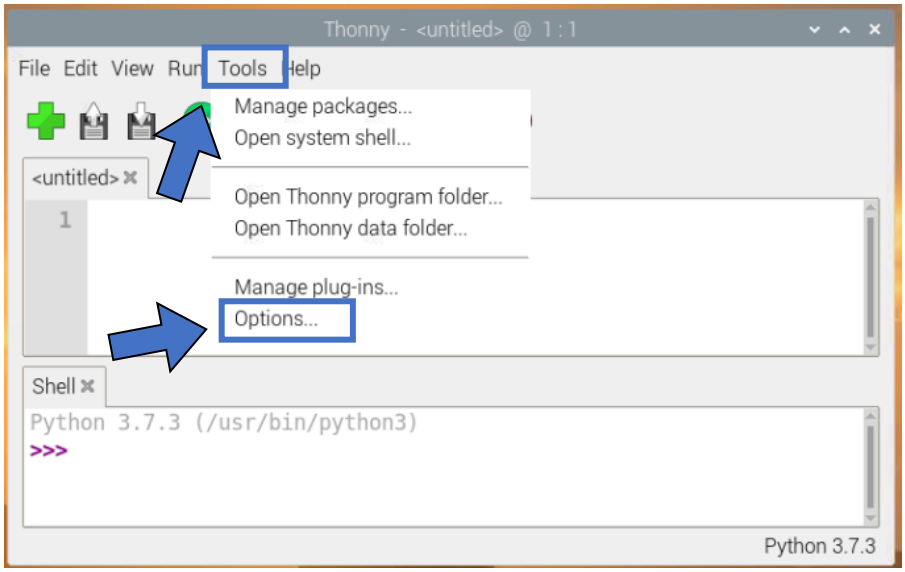
In order to have the Option menu, it is necessary to change the mode:



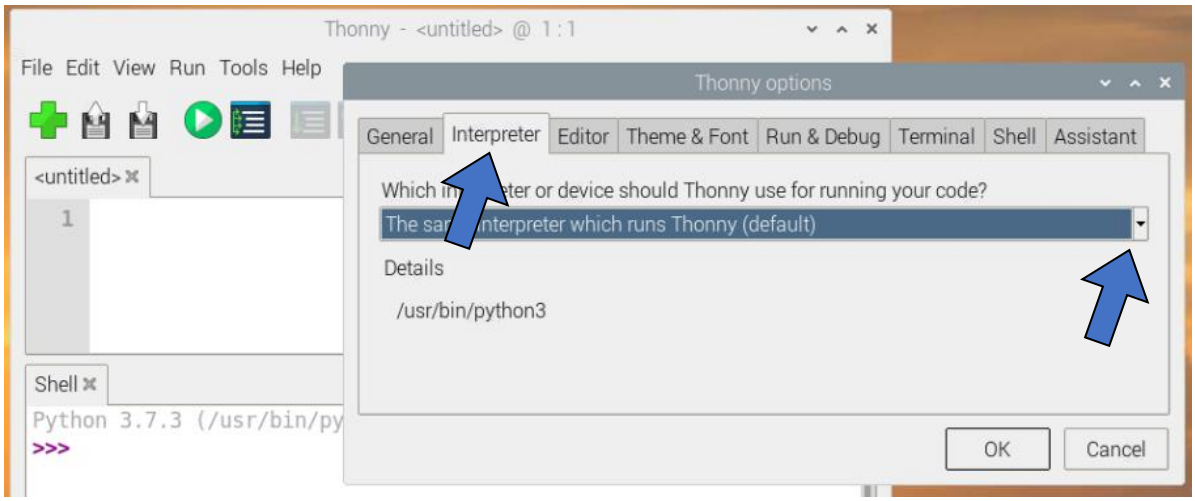
Confirm the info, and restart Thonny



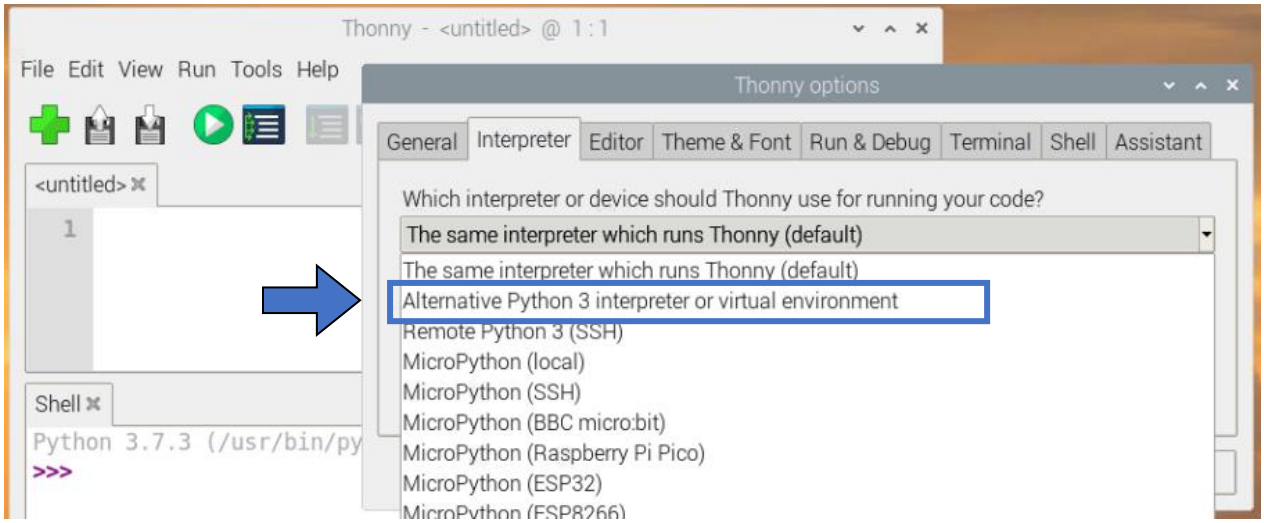
Restart Thonny, and select Tool, Option



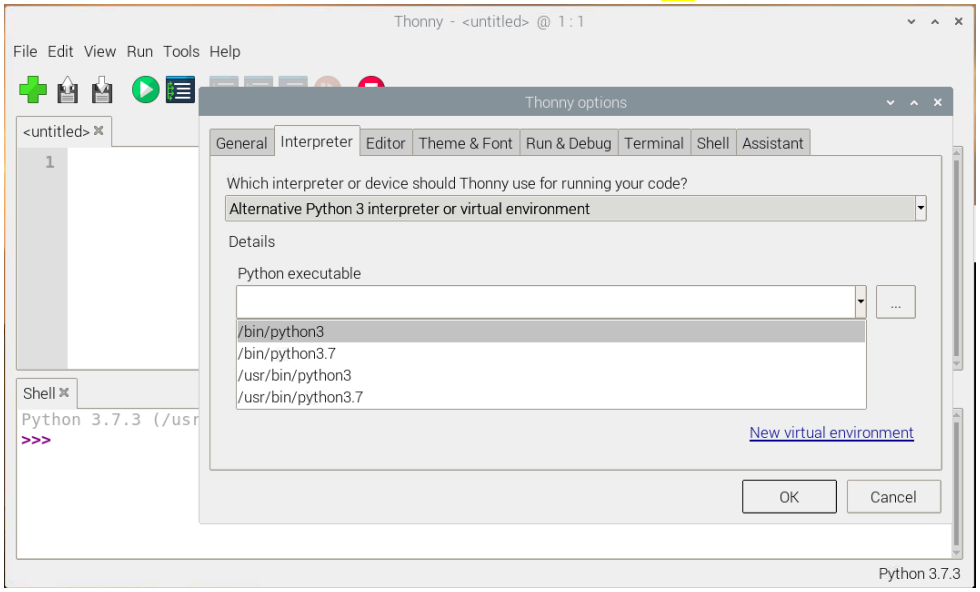
Select Interpreter where it is shown the default setting, pointing to /usr/bin/python3.



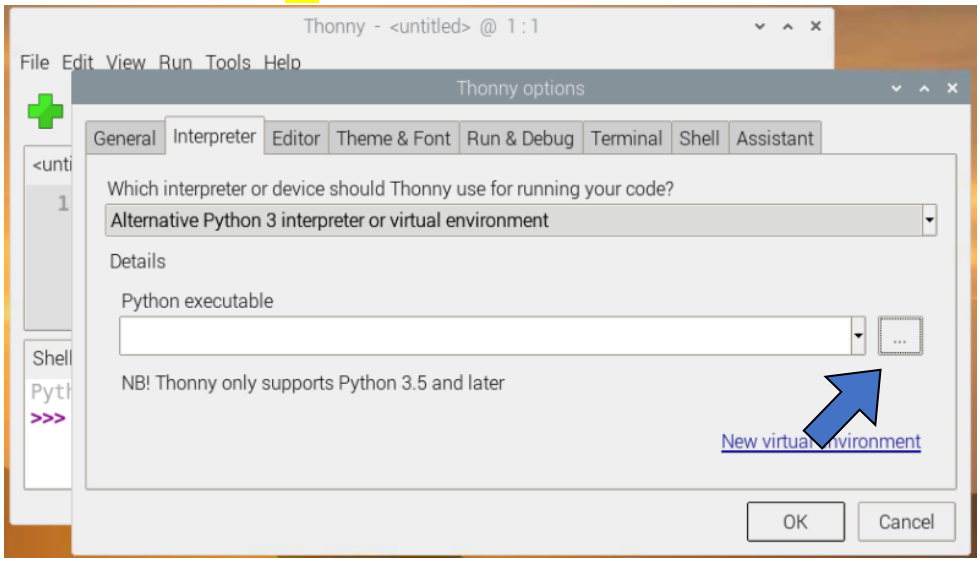
Open the drop-down menu and select 'Alternative Python 3 interpreter or virtual environment':



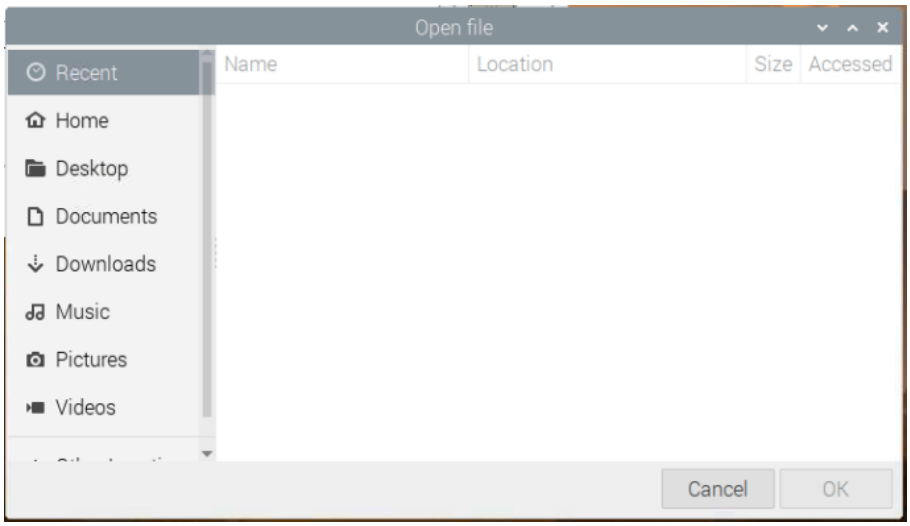
Open the drop-down menu and if '/home/pi/cubotino/src/.virtualenvs/bin/python3' is listed just select it



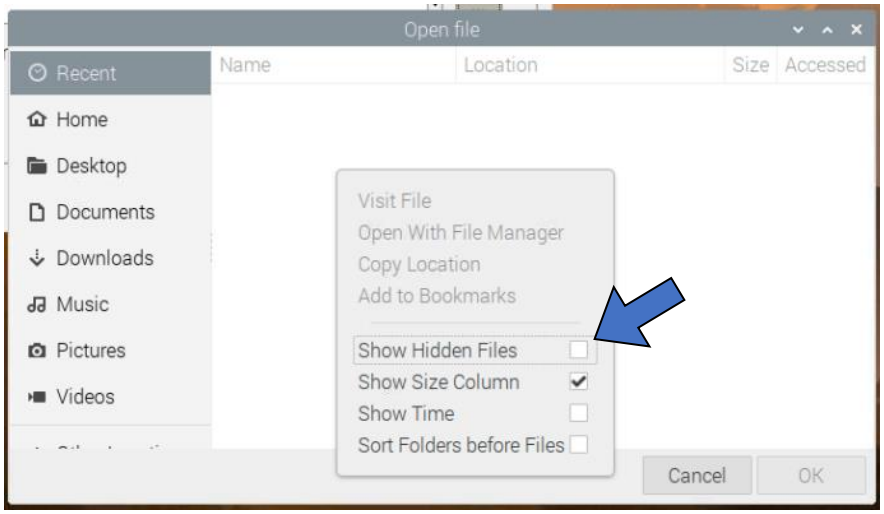
If ‘’/home/pi/cubotino/src/.virtualenvs/bin/python3’ is not listed, select the browse button:



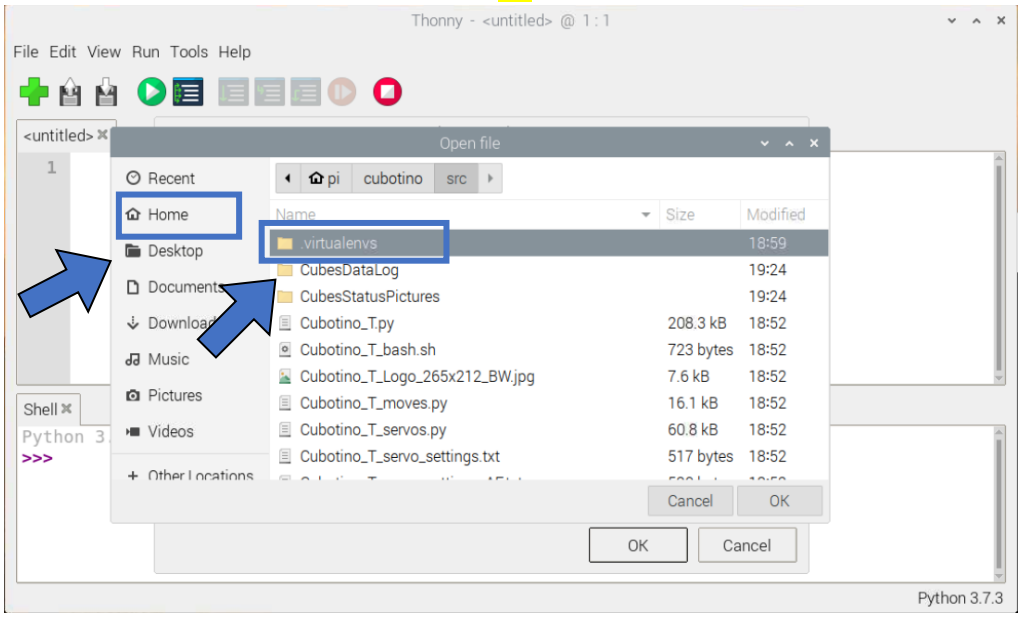
An Open file window opens:



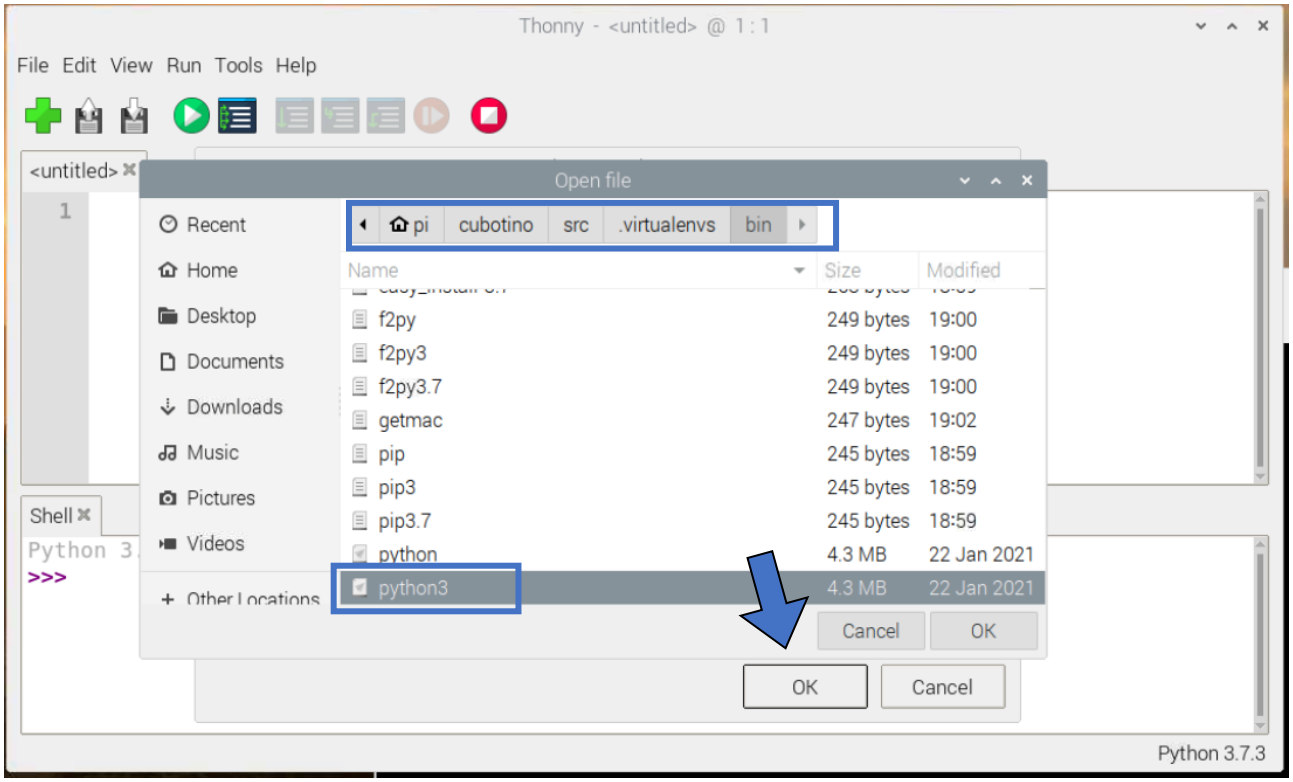
Right click on the empty window part, and check ‘Shows Hidden Files’:



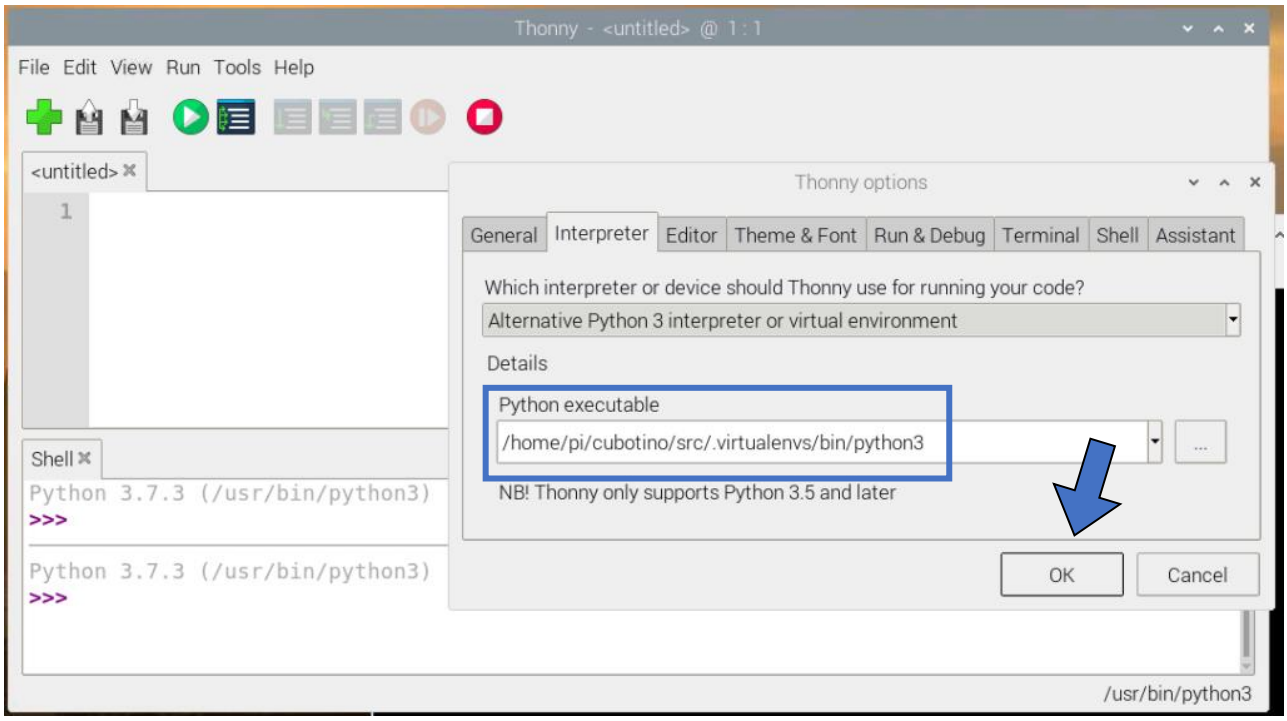
Select Home, .virtualenvs should appears (Note: all folders and files starting with a dot are hidden type)
The path should be ‘’/home/pi/cubotino/src/.virtualenvs’



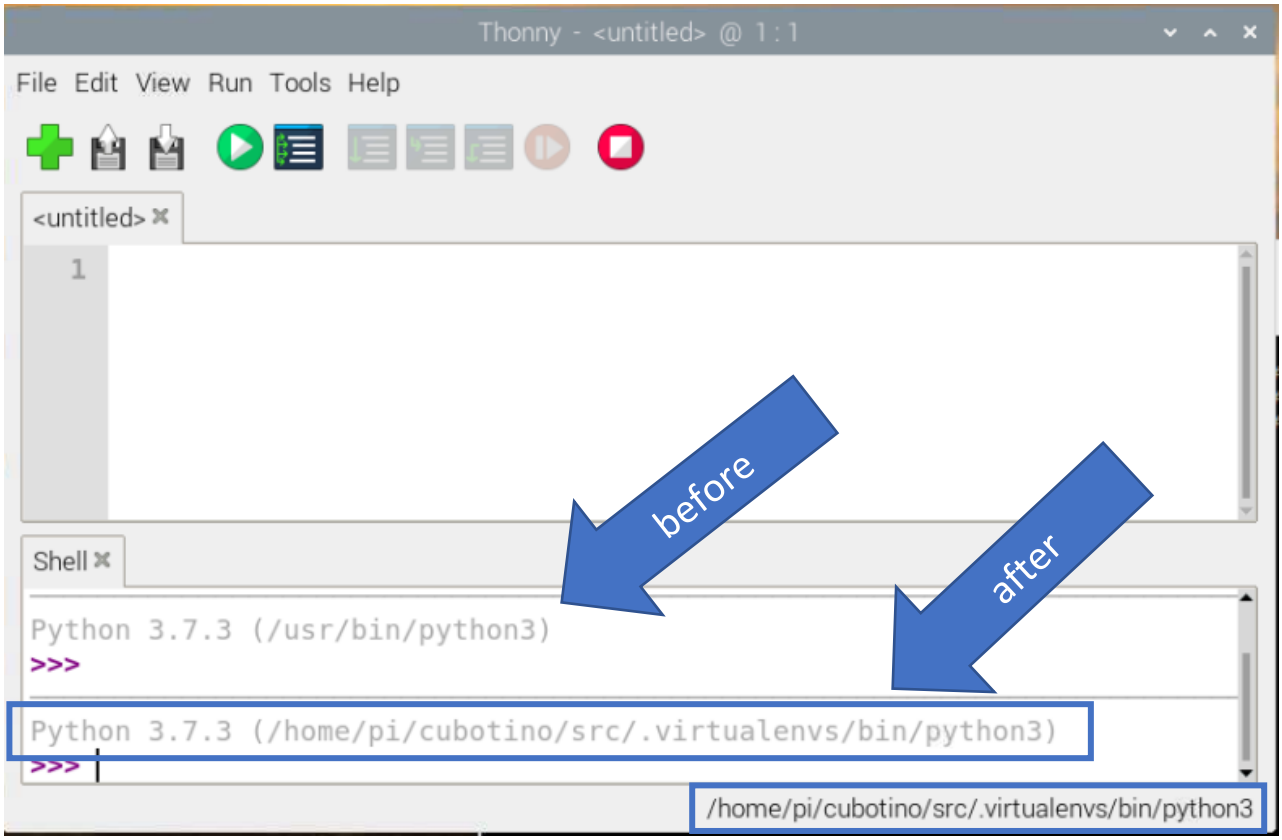
Select python3 from the path: ‘/home/pi/cubotino/src/.virtualenvs/bin/python3’ and confirm a couple of times



Confirm one more time



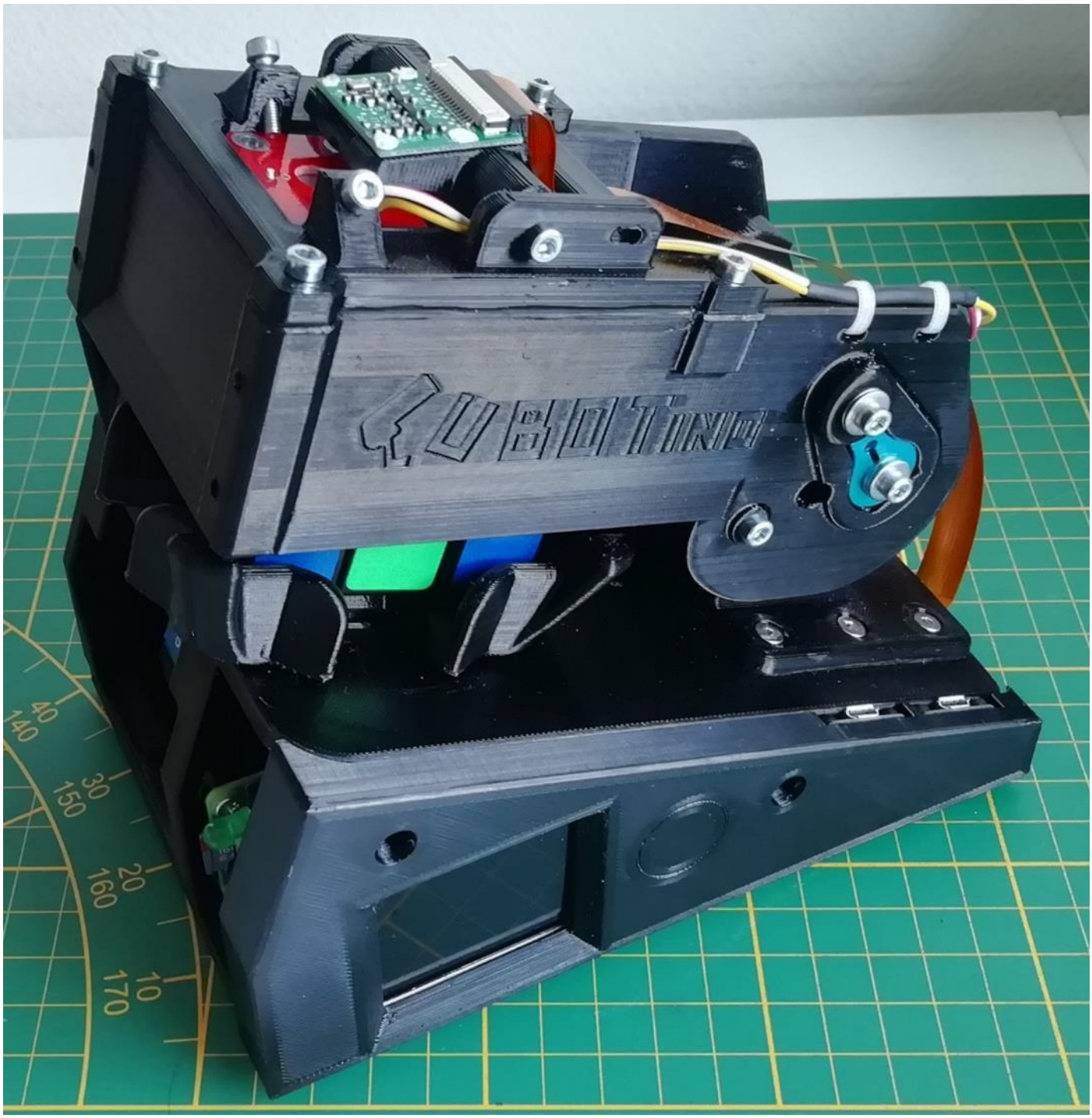
Note the interpreter is now using python3 from the venv

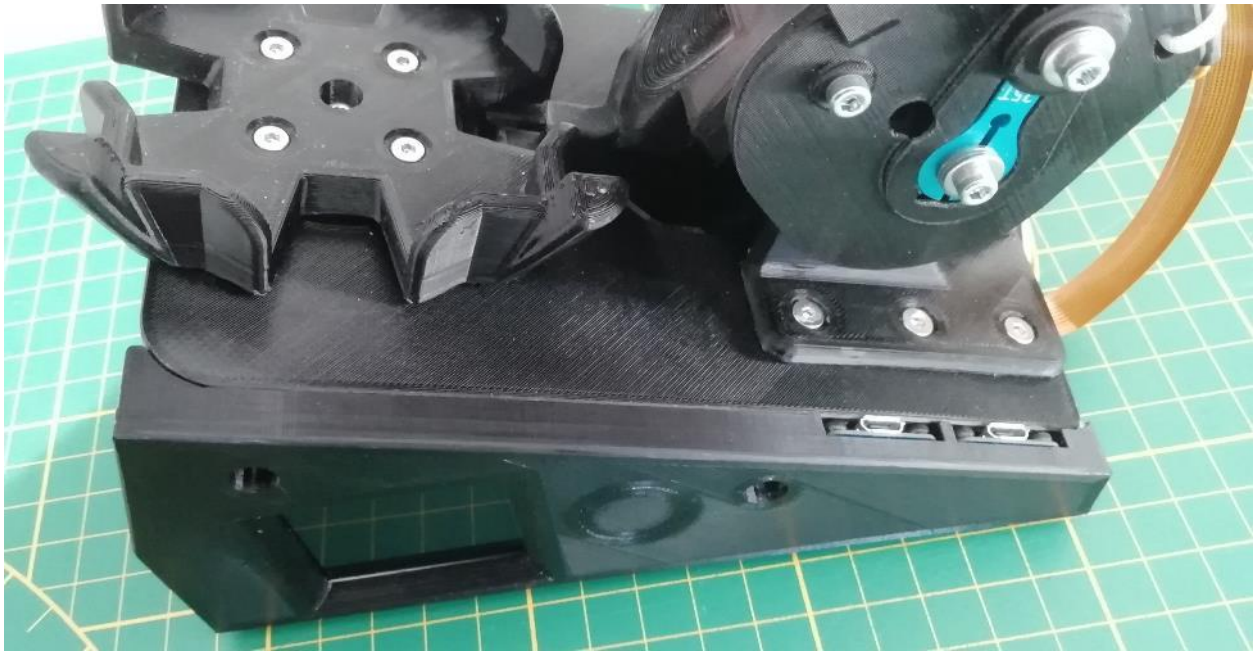


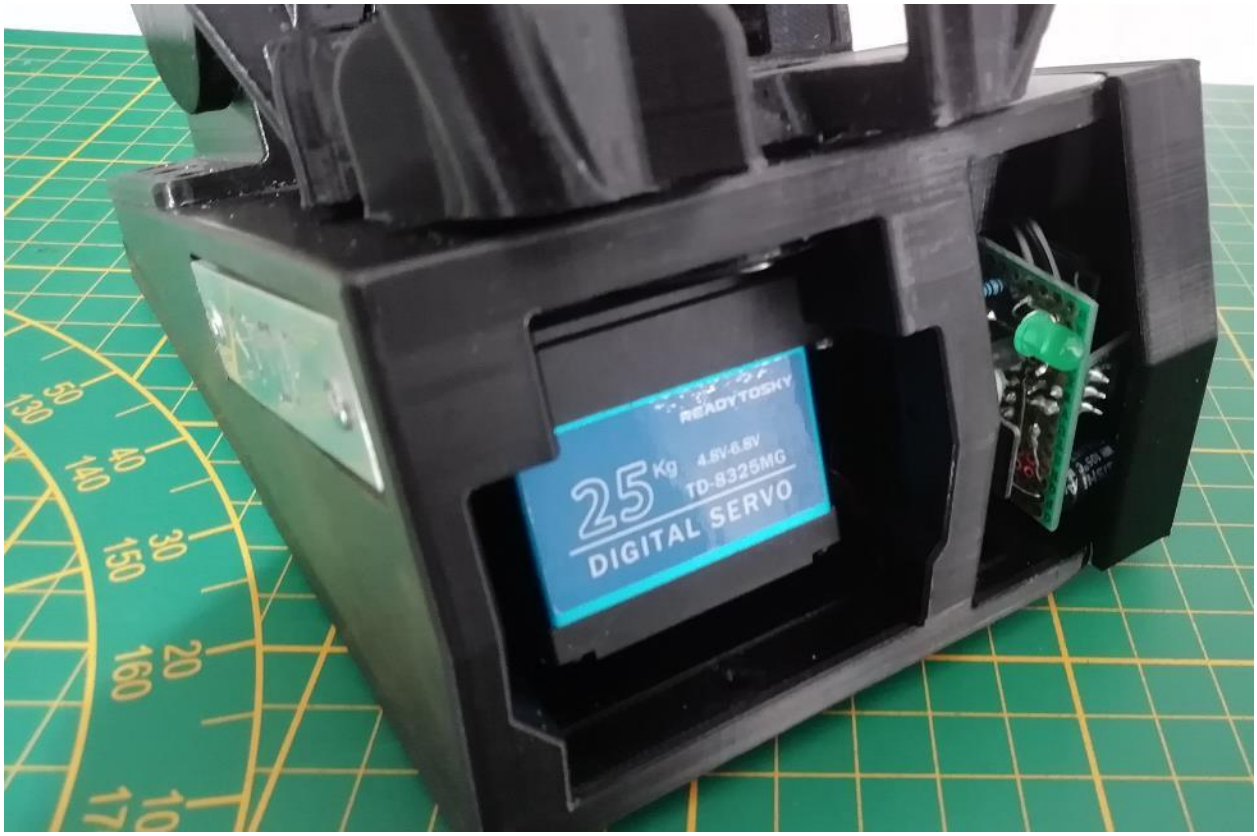
Notes, to get this change proposed as default:

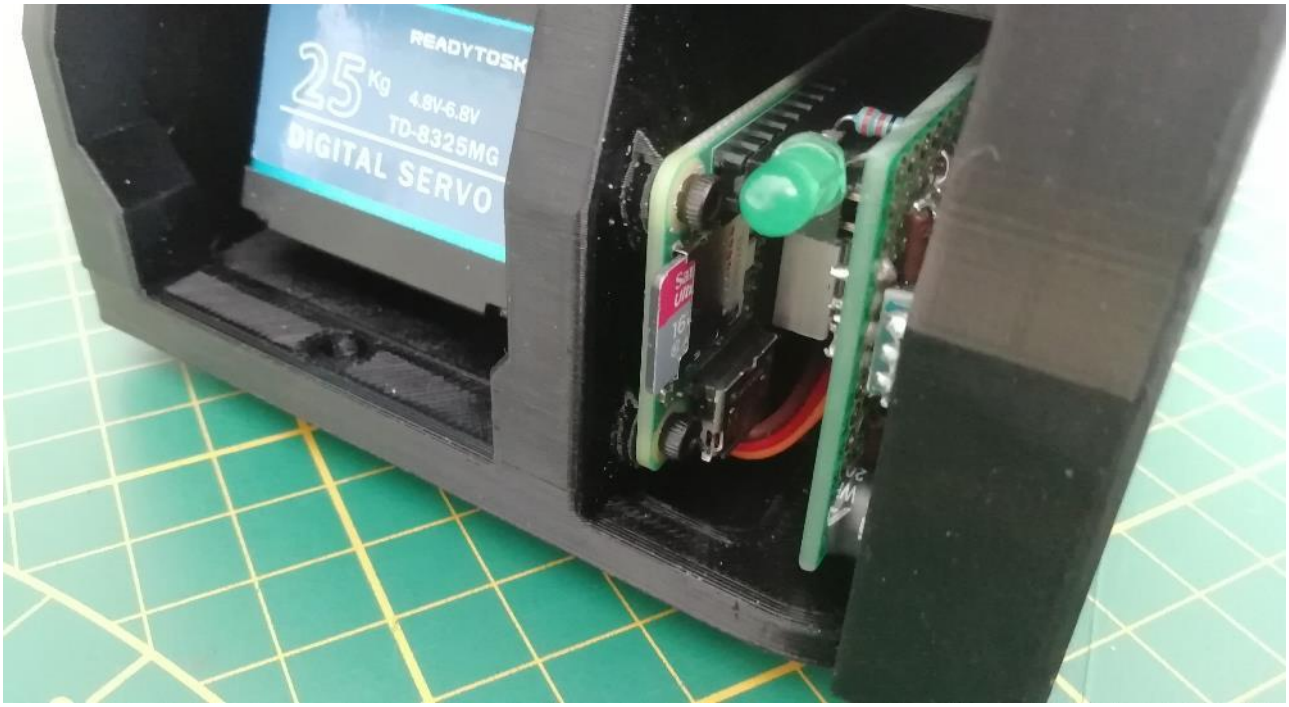
- Do not open any python file
- Close and re-open Thonny

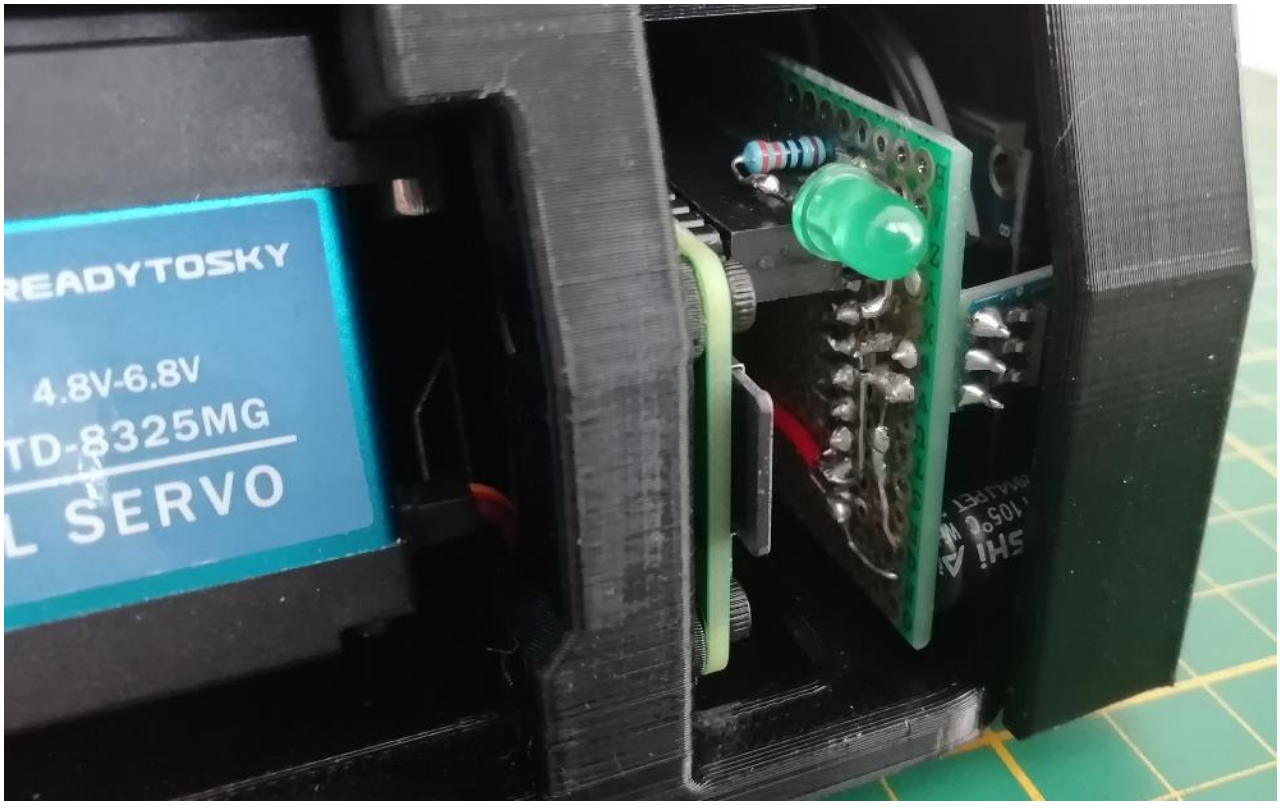
45) Collection of robot's pictures

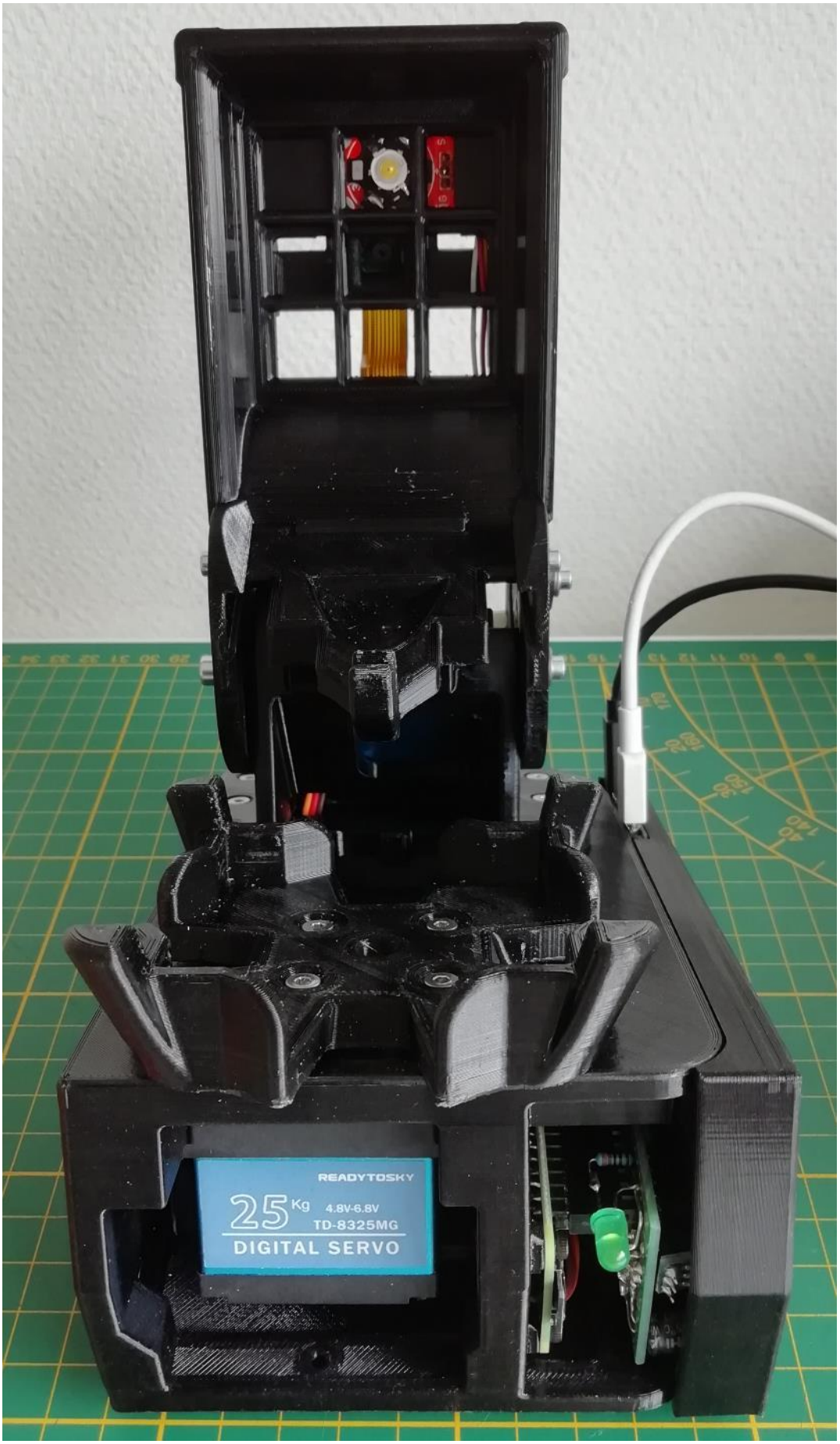


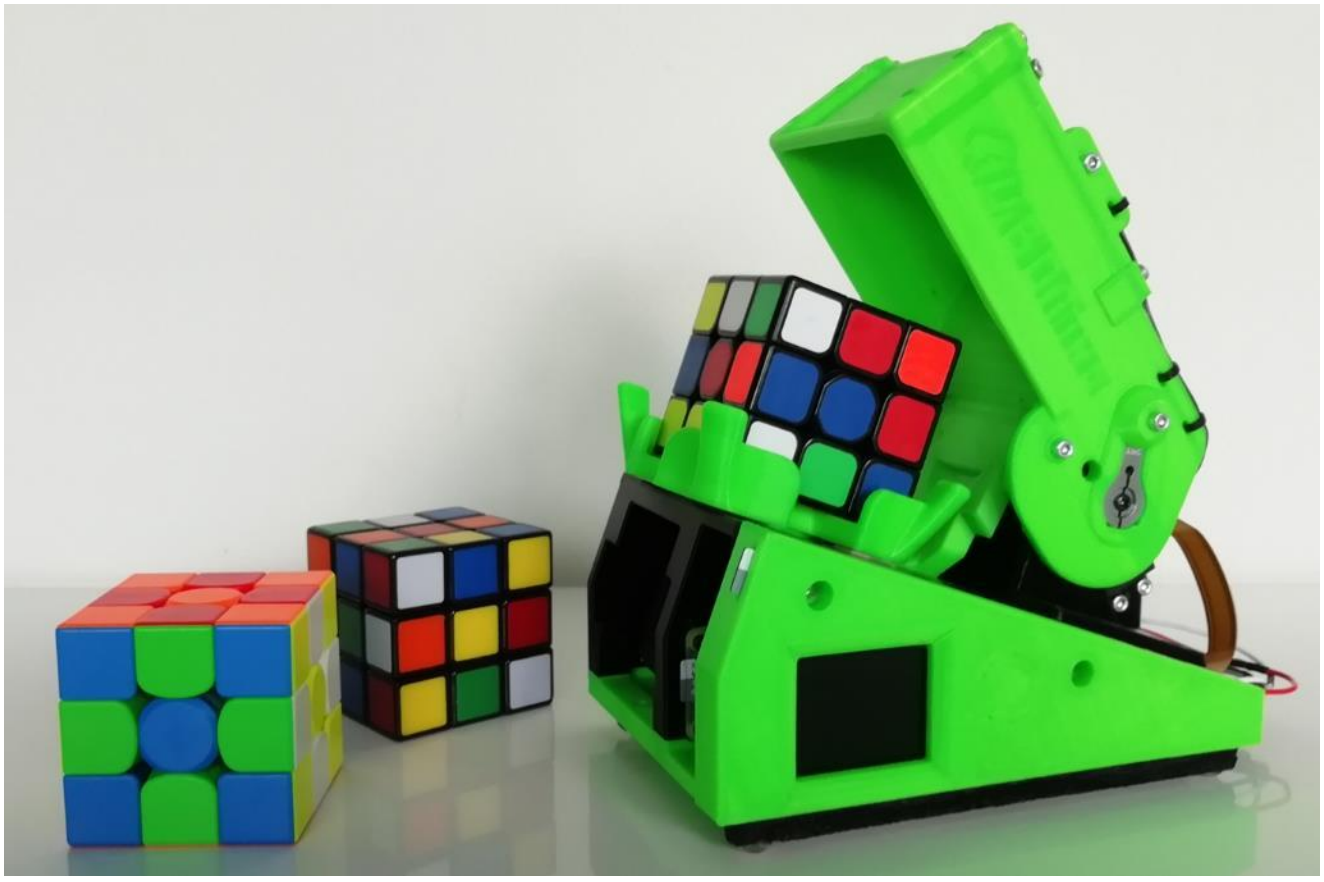


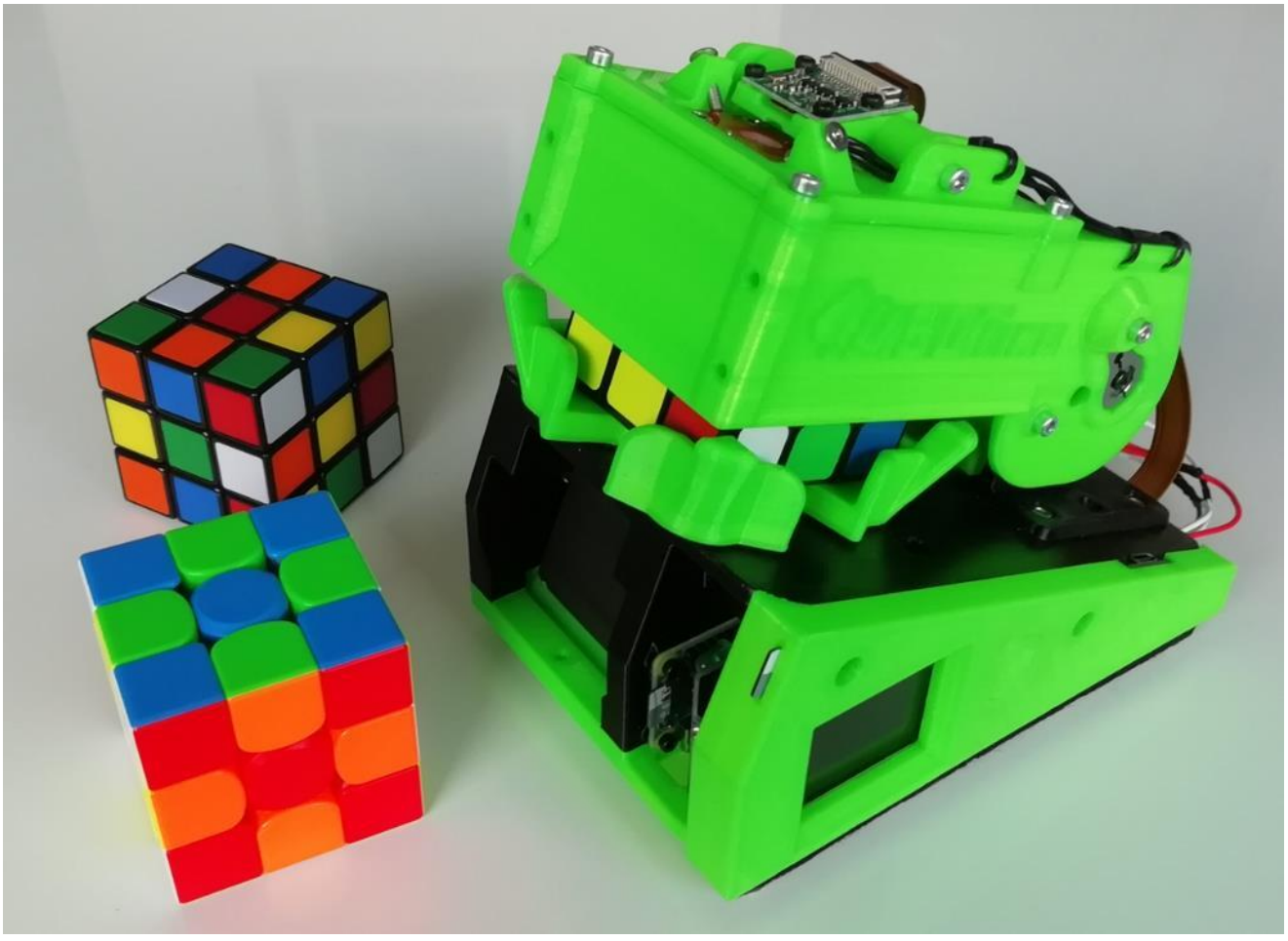

READYTOSKY
25 Kg
4.8V-6.8V
TD-8325MG
DIGITAL SERVO


READYTOSK
25 Kg
4.8V-6.8V
TD-8325MG
DIGITAL SERVO


READYTOSKY
4.8V-6.8V
TD-8325MG
L SERVO


READYTOSKY
25 Kg 4.8V-6.8V
TD-8325MG
DIGITAL SERVO





46) Conclusions

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

Pros:

1. The new way to manoeuvre the cube has proved to be effective.
2. Robot dimensions are very small.
3. The same base mechanic works fine on these two versions.
4. Base version specific:
    a. is relatively cheap (about 30 to 40 euro of material, on early 2022, depending on where you live)
    b. GUI works simply fine
    c. Rather easy to set the robot parameters via the GUI
5. Top version specific:
    a. Raspberry Pi Zero 2 (512Mb ram) has proved to be sufficient for the computer vision, and all the required tasks….also for those not strictly needed, but nice to have.
    b. PiCamera works well despite the short distance from the cube.
    c. Cube status detection rather unsensitive to external light conditions.
    d. Despite the increased complexity, the electronic and wiring is still limited and simple.
    e. Power supply via two common 2A micro-USB phone chargers.

Cons:

1. In general, this is a slow robot; It was known since the start, yet…
2. Noise: Flipping the cube on the horizontal axis generates noise when the cube falls into the seat.
3. Base version specific:
    a. Micropython documentation is somehow limited.
    b. Micropython documentation does not always correspond to real cases, for instance the PWM (on latest two official release) have much lower resolution than reported on the docs.
    c. Single power supply inlet not robust on all the builds
4. Top version specific:
    a. Connections board requires some more skills than the base version
    b. Total material cost is about 100 euro, on early 2022

47) Next steps

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

.

48) Commitment

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

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

In case you cannot find the solution by yourself (part that makes projects fun 😊), please drop a detailed question at the Instructables site (https://www.instructables.com/CUBOTino-Autonomous-Small-3D-Printed-Rubiks-Cube-R/).

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

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

49) Credits

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

Credits to all the people who have provided feedbacks in particular those I've had contact with along this project:

- Jacques, who triggered me on the colour's sensors direction.
- Richard and Scott, for their keen check of this document and precious feedbacks.
- Yannick, and his initiative to simplify the installation via GitHub repo, further than other improvements to the code.
- Chad, and his initiative to draw the connections board in CAD and sharing the Gerber files with us.
- Kevin, and his extensive availability to test the nightly versions.
- Martin, and his cross testing with other servo types and feeding all of us back.

51) Revisions

Rev	Date	Notes
0	01/06/2022	First release.

1	06/06/2022	Rather large revision, in particular:
		• Modified Structure.stl by adding a hole (to see Raspberry Pi integrated LED)
		• Modified Cubotino_T.py and Cubotino_T_servos.py to upload settings from json files
		• Integrated the list of files to be copied to Raspberry Pi
		• Added a few chapters on this document
		  ○ how to set Thonny
		  ○ Parameters and settings
		• Improved the Tuning chapter
		• Corrected a mistake at 'setup Raspberry pi interfaces'
		• Made more complete the 'how to use the robot'
		• Modified Rpi setting to include mac address

1.1	07/06/2022	Correct an error on this doc, for the display pin numbers
2.0	14/6/2022	Modified Cubotino_T.py • to tentatively import Kociemba solver from multiple locations • to visualize/salve the manipulated images to detect the facelets Added: • Kociemba installation chapter • Memory usage info Updated: • Troubleshooting, with more details for the Top_cover < -- > Hinge gap and friction
2.1	17/6/2022	Corrected typo on the Connections_board
2.2	18/6/2022	Added: • 3 stl files, acting as spacers to use Raspberry Pi 3b or 4b instead of Zero 2 • Info related these new (optional) parts
2.3	22/6/2022	• Removed the additions made with rev 2.2 • Modified Cubotino_T.py due to a bug, throwing an error under some rare, yet possible, conditions
3.0	25/6/2022	Since today, is available a simplified setup for the Raspberry Pi With this approach, few other changes were necessary, resulting in: • Modified Cubotino_T.py to better managing permissions on folders and files creation • Updated instruction
3.1	30/06/2022	GitHub: • Moved the robot specific files into a (src) folder; The folder structure is maintained at Raspberry Pi for easy future updates. Updated files: • Cubotino_T.py due to a bug and to make backups of the settings files • Cubotino_T_bash.sh to point the right folder • setup.sh Instructions: • Removed the Manual installation part (to setup the Raspberry Pi) • Further explained the Simplified installation via GitHub
3.2	06/07/2022	Modified Cubotino_T.py to be used with Raspberry Pi Zero (not 2) without crashing. Added some minor improvements

3.3	09/07/2022	Modified: • Cubotino_T_servos.py to upload the settings when used with the argument '—set', to cover servos having pulse width different from the common 1 to 2ms • Baseplate_front.stl for taller servos (up to 31mm protrusion under the flange) Instruction: • Better covered the usage of servos with different pulse width range • Corrected typos on Cubotino_T_servos.py (the 's' was missed) • Better explained how to set the servos 263ositions • Added notes on the max protrusion from the flange, when 'taller' servos
3.4	13/07/2022	Instructions: • Corrected typos • Better explained the servos settings
3.5	25/07/2022	Modified Cubotino_T_servos.py to check and play with the servos positions directly from the command line (additionally from the REPL, as it was so far possible) At the instructions, better explained how to verify the servos positions.
3.6	06/08/2022	Modified Cubotino_T.py: • display is initialized once (thanks Yannick for the nice solution) • frameless cubes can now be detected Modified Cubotino_T_settings.txt and Cubotino_T_settings_AF.txt, by adding • frameless_cubes parameter • cover_self_close parameter Improved or modified the instructions: • Alternative servos that can be used • Gerber files for the connections board (thanks to Chad) • frameless_cube parameter • cover_self_close parameter • Troubleshooting for servos • Troubleshooting for cube status detection • Display init and test • Sticking the PiCamera with its self-adhesive tape to the board • Location for the step files
3.7	11/08/2022	Modified Instructions: • Corrected typo on screw protrusion for display support Modified Cubotino-T.py: • Corrected bug on frameless cube detection • Improved the facelets estimation, and increase up to 4 the estimated ones • Enlarged the permits at text files creation (*_backup.txt, *Log.txt)

3.8	21/08/2022	Modified Cubotino_T.py to import settings for a second Andrea's robot (based on mac address) Modified Cubotino_T_servos.py: • to import settings for a second Andrea's robot (based on mac address) • to include the b_extra_sides setting when spinning toward CW or CCW Single power supply, via USB type-C: Added an alternative PCB_cover stl and stp files, that also includes a slot to store the microSD clone. Modified Instructions: • Additional alternative servos (cheaper) • USB type-C breakout board reference • Connections for a single power supply • Tests of the connections_board, before connecting it to Rpi • Setting for more Wi-Fi connections (i.e. for the phone wifi hotspot) Modified Pcb_support stl file, the one for two micro-USB break out boards, to better couple with the display (reduced by 2.5mm the gap toward the display).
3.9	19/09/2022	Updated instructions to the V1.6 connection_board version
3.10	28/10/2022	Updated instructions: • Re-ordered the content, with initial part fully focused on the robot making process • Added info/pictures on how to increase the servo rotation range. • Added info on how to embed a Raspberry Pi 3 or 4 • Added info related to the PiCamera focus • Better explained the parameters listed in chapter 13 GitHub / Instructbles: • Added the stl files for the "extensions", needed when Rpi 3 or 4 is used • Modified the Hinge.stl (and .stp) to accept 43mm servos Cubotino_T.py: • Changed from the word "exposition" to "exposure" on the display