How to make

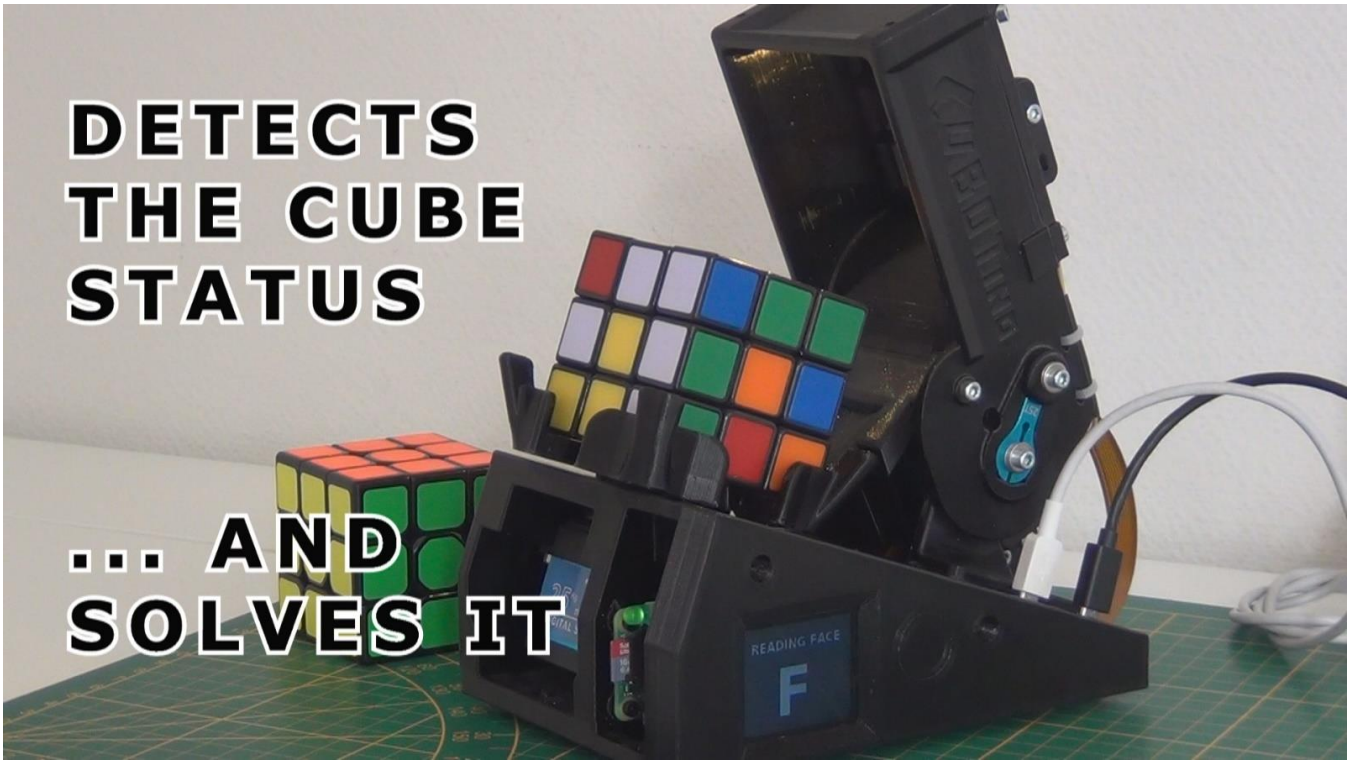
CUBOTino autonomous: A small, 3D printed, Rubik's cube solver robot

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

Andrea Favero, Groningen (NL)

18/06/2022 (check if a newer version is available)

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



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

Summary

1) 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 a clear drawbacks:

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

2) Project scope:

Based on the introduction you probably can guess the project scope 😊
This second Rubik's cube solver robot project wants to be affordable, to attract more people and especially students into robotics and programming.

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

Project targets for this robot:

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

3) Robot name:

I've started this project with the idea to write and share the instructions, and along the way I thought a robot name would make the project more complete.
By combining CUBE, ROBOT and -INO, the Italian suffix for diminutives (to remark the small robot size), the chosen name is CUBOTino
By considering the Top cover, combined with the Lifter, has a "C" profile shape, then CUBOTino become:

4) 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

5) Next steps:

Work out the Medium robot version, yet it won't be before winter 2022 😊.

6) Safety:

Energize the robot only via USB ports having a class 2 insulation from the power supply net; Phone charger normally have this safety feature, yet you're requested to "double" check it.
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.

7) 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

8) 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
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

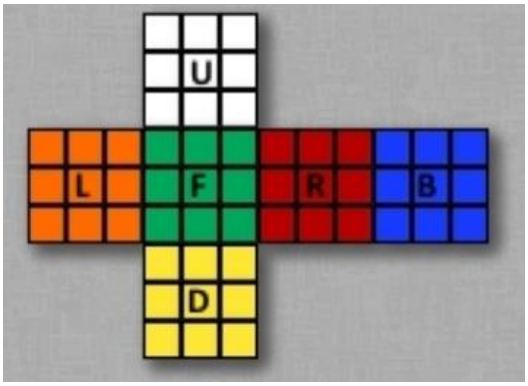
I've spent some time on this method, but the quality of my soldering has proved to don't be sufficient; I'm even doubting if it really fits the overall project scope, as this method involves too many skills.

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

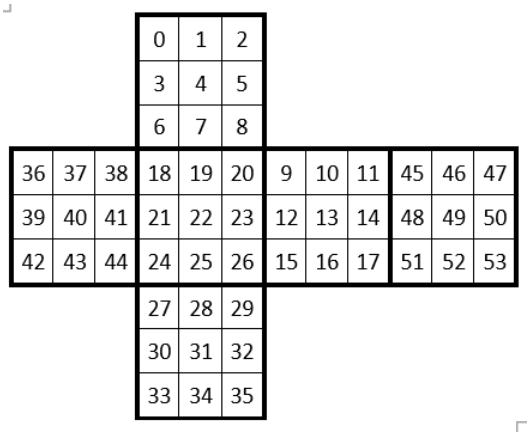
Considering the Base version has been "published" on early April 2022: I'm pretty sure in little time people will design their own version … and probably the Medium version will come to live in this way 😊

9) 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 Cube_holder 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.



10) 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. All parts are made by 3D printing:
   - This makes possible to pursue the needed geometries, also complex shapes.
   - The biggest parts can still be printed on a relatively small plate (min plate 200x200 mm).
   - Some of the parts are split, mainly for easier, and better, 3D printing; Others are split for assembly reasons.
   - All the overhangs have been designed to enable 3D printing without support.

There are many different examples of Rubik's cube solver robot, based on two servos; I think most of them are variations from the one made by Matt's on 2014: https://hackaday.com/2014/06/28/rubiks-cube-solver-made-out-of-popsicle-sticks-and-an-arduino/
In these executions, the solution used to flip the cube on one of the horizontal axes, requires the main pivot (servo) to be placed rather far from the cube; This obviously increases a lot the overall dimensions.

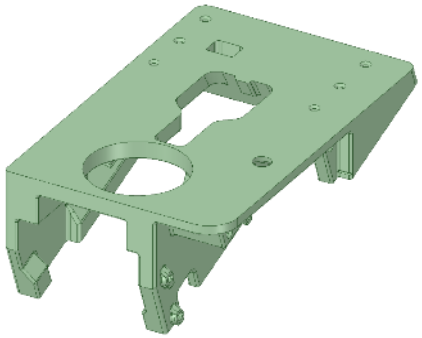
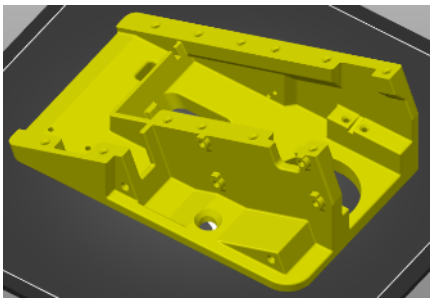
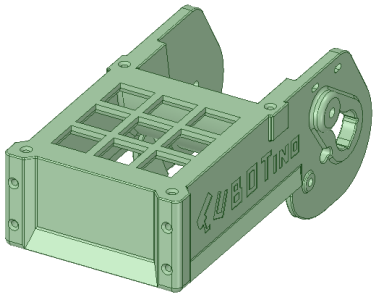
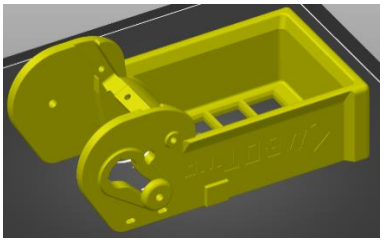
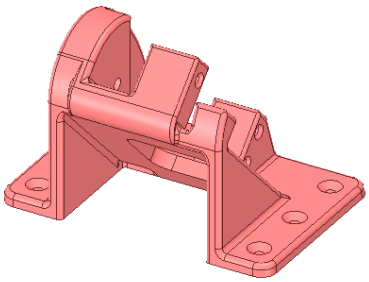
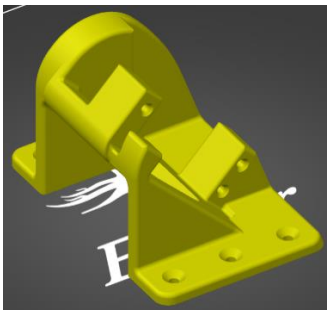
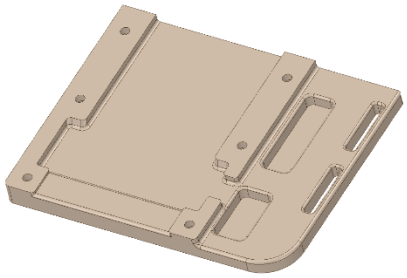
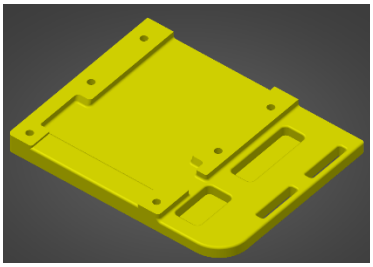
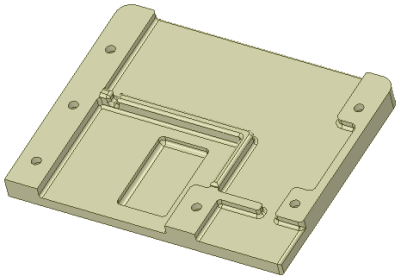
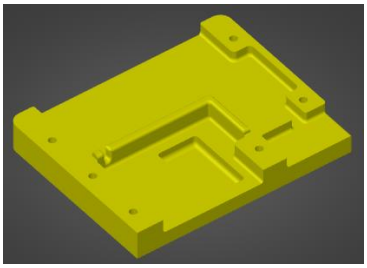
11) 3D printed parts:

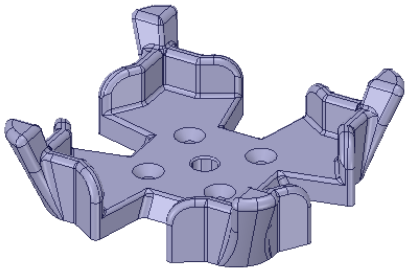
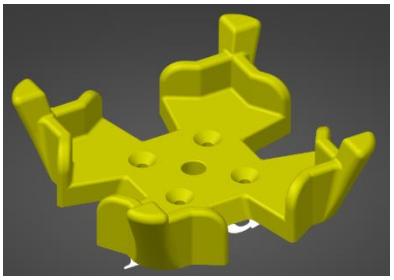
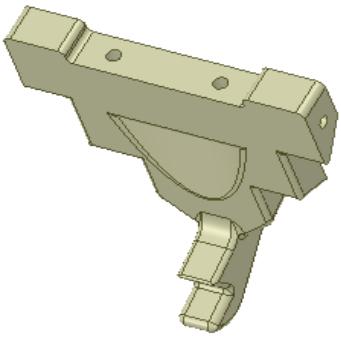
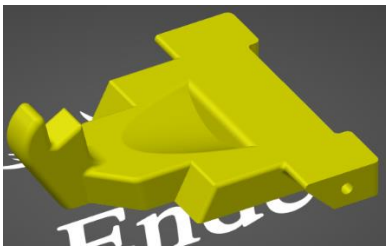
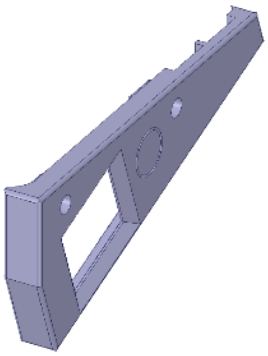
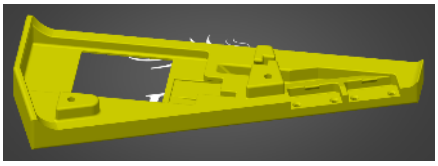
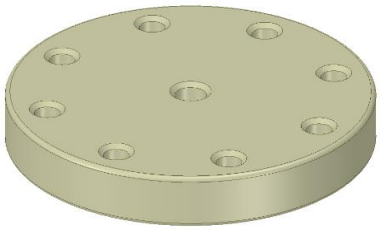
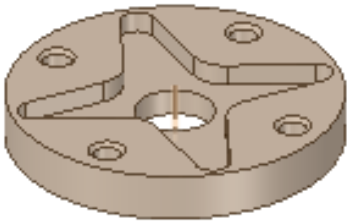
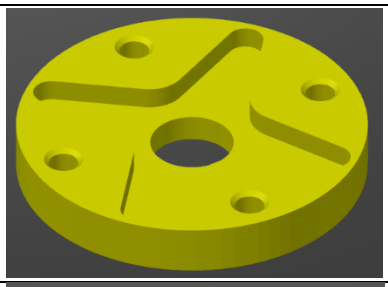
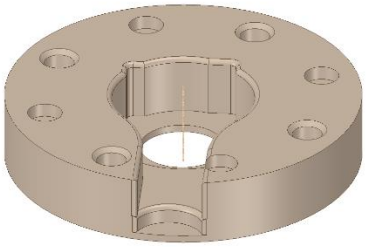
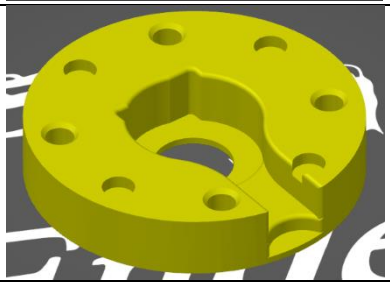
See notes below for filament quantity, and printing time …

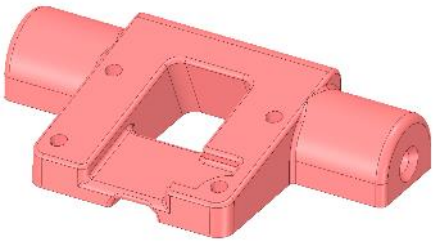
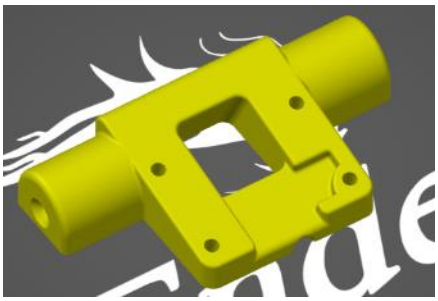
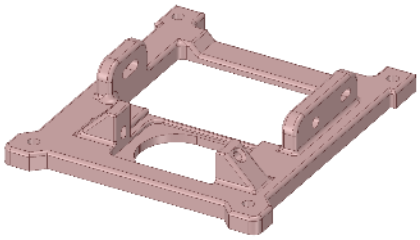
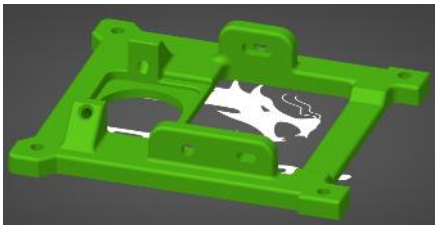
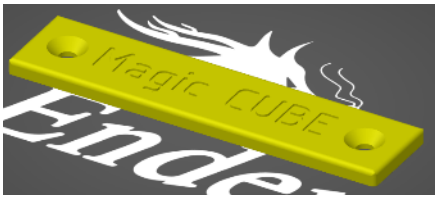
Ref	Part	Filament		Printing time	Version specific parts
		Meters	Grams		
1	Structure	49	145	14h33m	
2	Top_cover	39,5	117	12h26m	
3	Hinge	17,2	51	5h37m	
4	Baseplate front	12,2	36	3h33m	
5	Baseplate back	12,6	37,3	3h36m	
6	Cube_holder	11,6	34,4	3h36m	
7	Cube_lifter	5,5	16,2	1h48m	
8	Servo_axis_sup (or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

Notes:

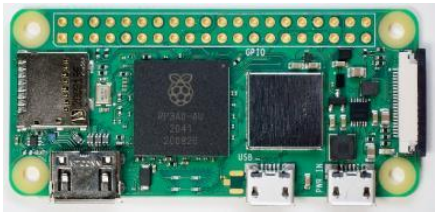
1. The biggest part is the Structure, and it easily fits on printers with plate size of 200x200mm.
2. Filament length is based on Ø1.75mm.
3. Filament weight is based on PETG density (1.23g/mm³), and printing settings I've use on my Ender 3 printer, for accurate result:
    - 0.2mm layers
    - Low speed (between 25 to 40mm/s for the external parts and 1st layer)
    - 4 layers on vertical walls
    - 5 layers on horizontal walls
    - 30% filling
    - 8mm brim
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

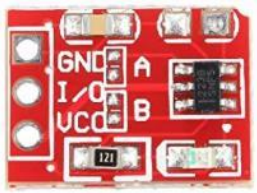
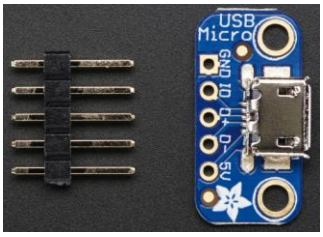
Part name	image	3D print orientation
Structure		
Top_cover		
Hinge		
Baseplate front		
Baseplate rear		

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

PiCamera holder		
PiCamera frame		
Personaliz_plate Or Personaliz_plate01	 	

12) Bought parts:

Q.ty	Part	link to the shop I used	Cost (euro)	Notes
2	Servo TD-8325MG (180deg 25Kg metal) and metal arm “25T”	https://www.aliexpress.com/item/32298149426.html?gatewayAdapt=glo2ita&spm=a2g0o.9042311.0.0.5d1e4c4d14Qjaz	25 (2 servos + 2 arms)	
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	
~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	

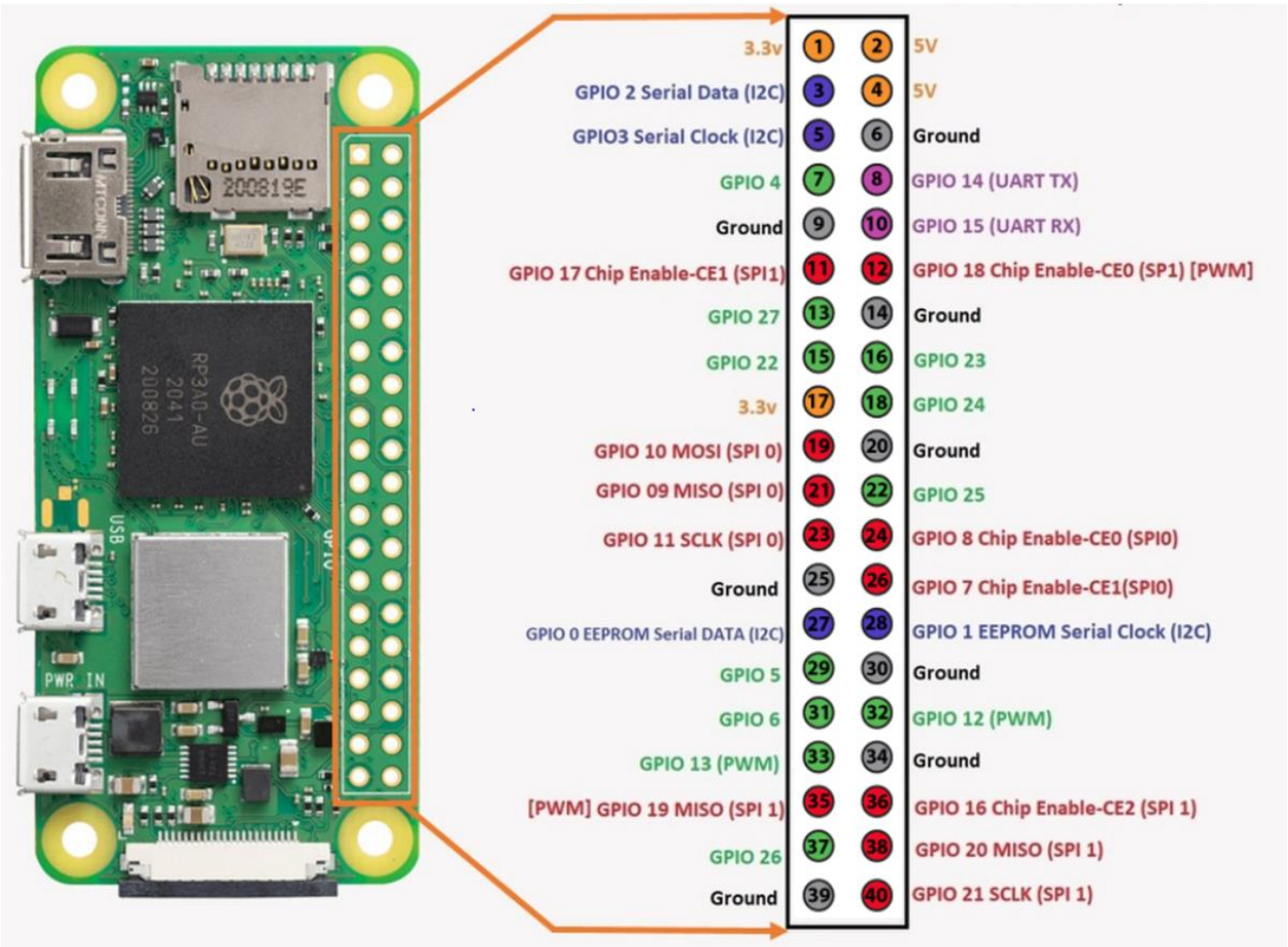
Screws:

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: 2x 2A phone charger with micro-USB cable, nowadays into some drawers at home….

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

13) Rasperry pi Zero 2 GPIO pinout:

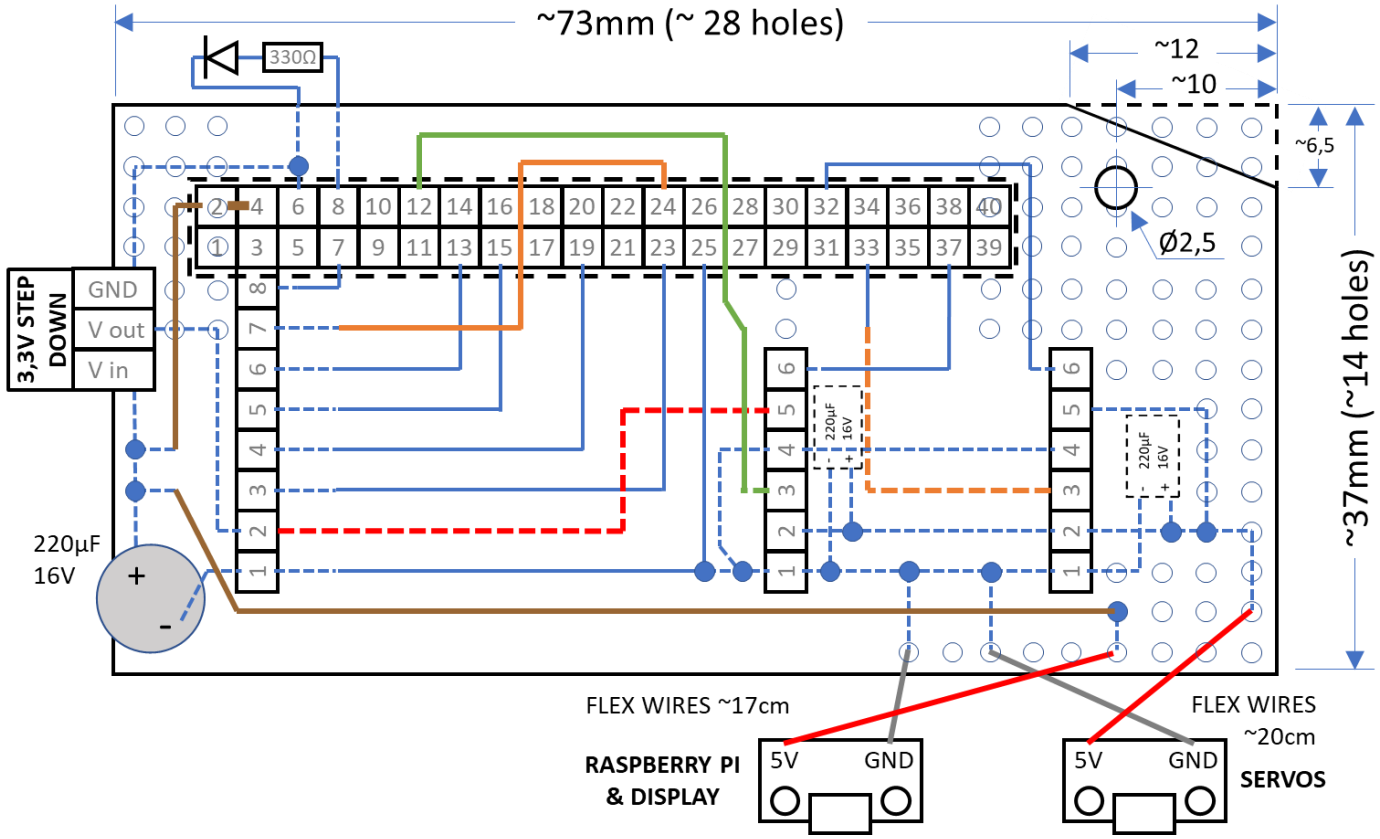


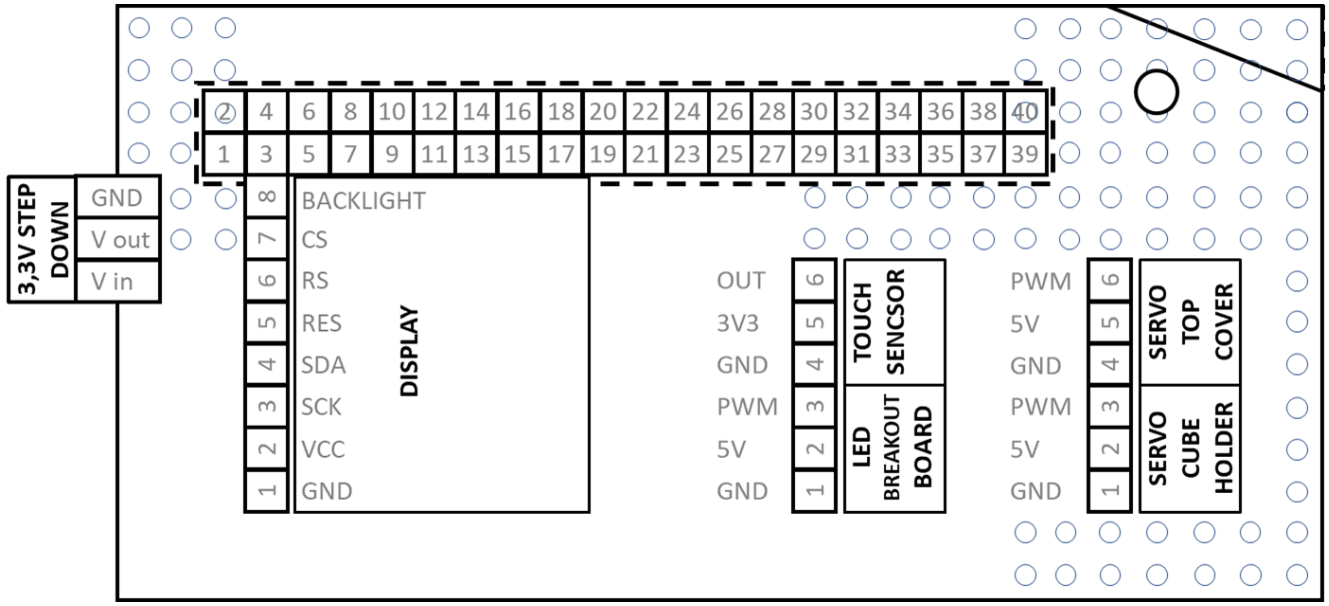
Used pins:

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

14) Connections board:

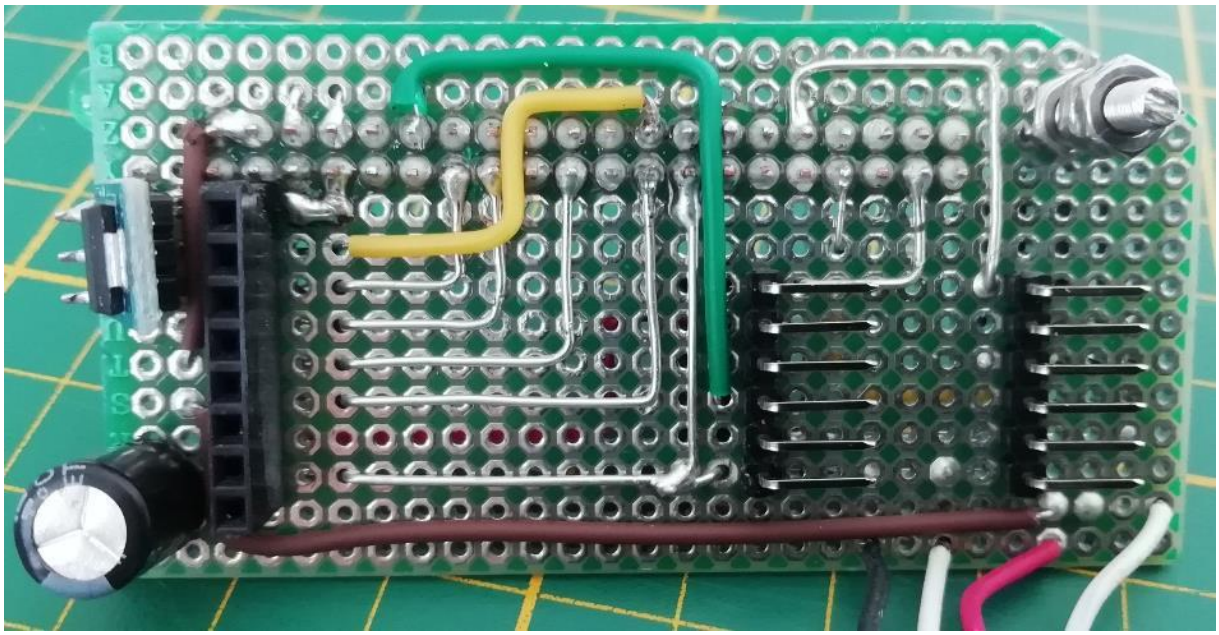
Front view (the 2x20 header is on the opposite side):

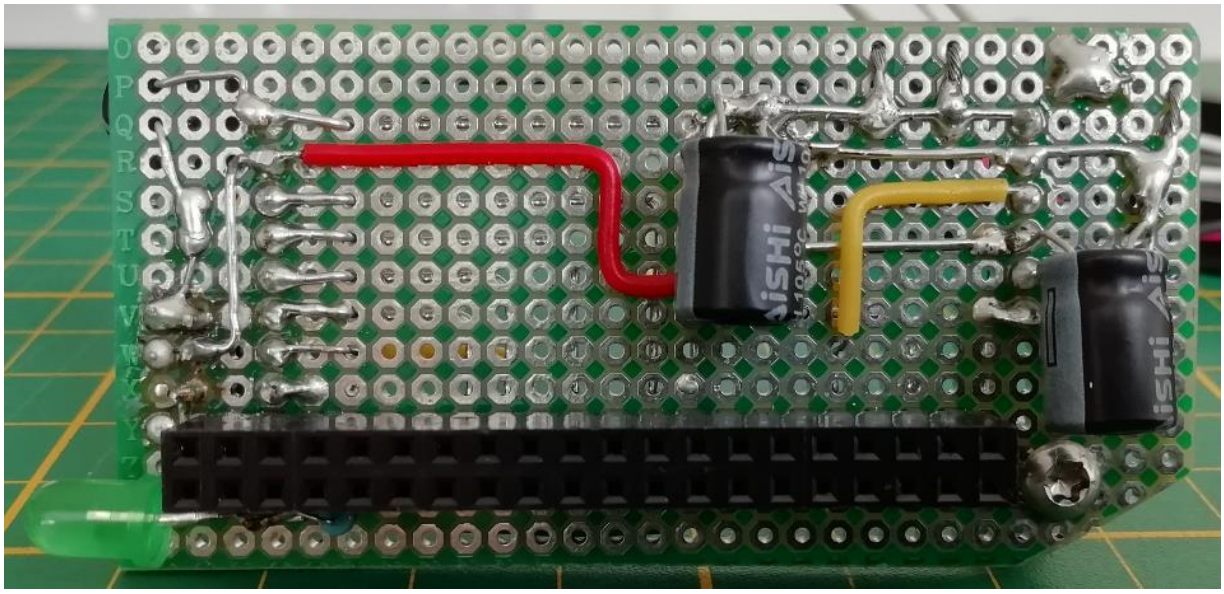


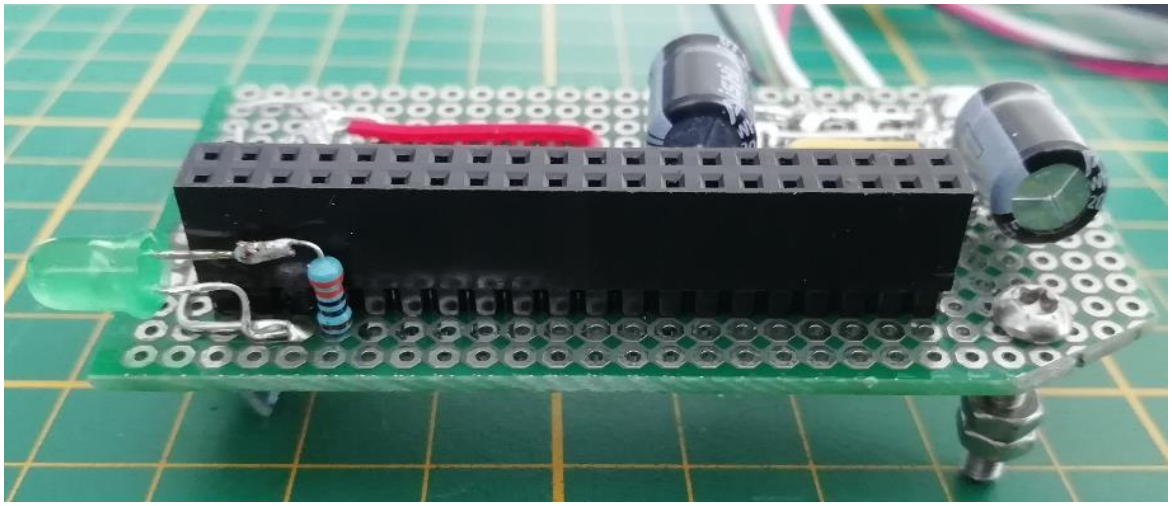


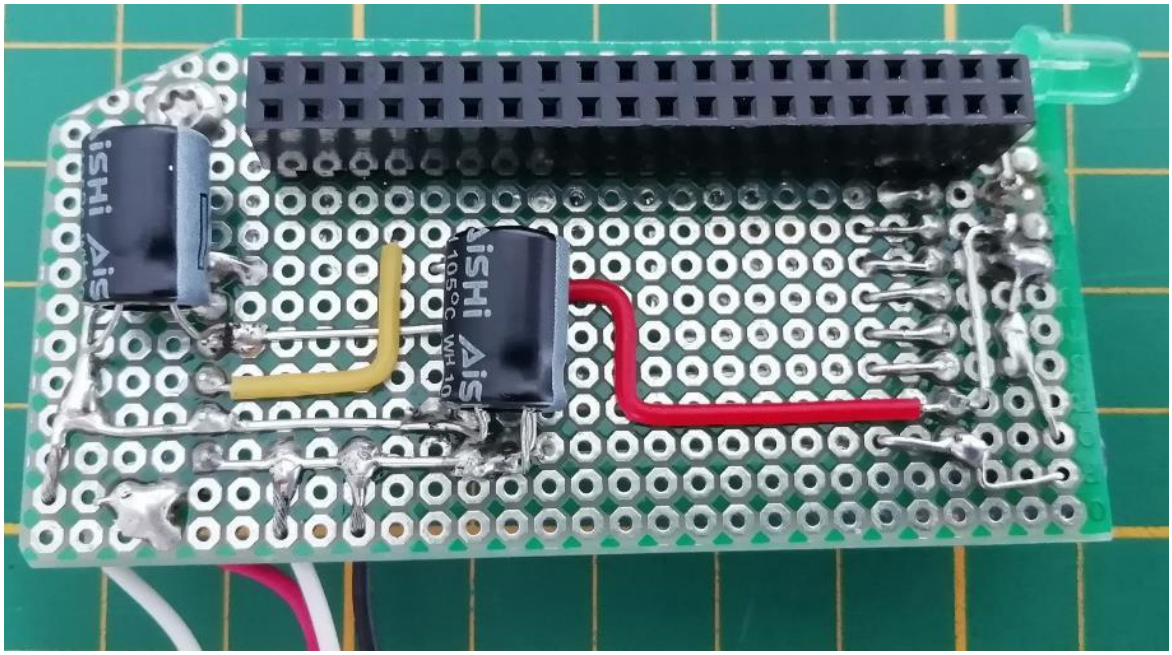
Notes:

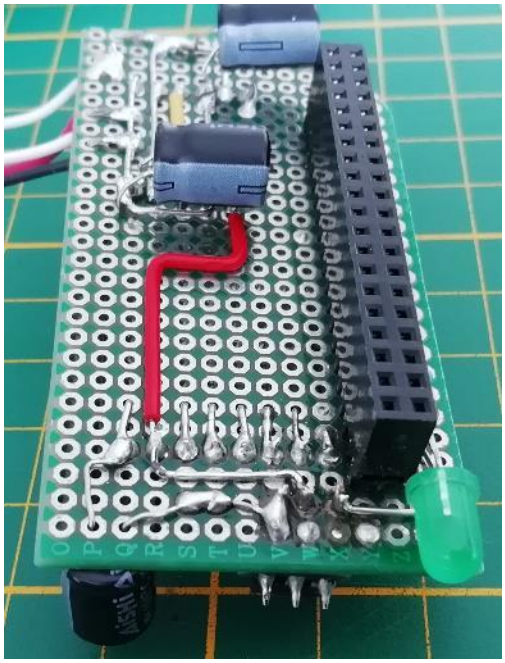
1. The prototype board should be of a double face type.
2. Start by positioning the 2x20 header, at 3rd hole from the corner.
3. Count the holes for the other parts positioning.
4. Dashed lines for parts/wires on the opposite side.
5. Blue lines are wires without insulation.
6. Red, Brown, Orange and 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 ~15.5mm from the board surface.
11. Add the last 2 capacitors, close to the servo connectors, once the board has been tested.



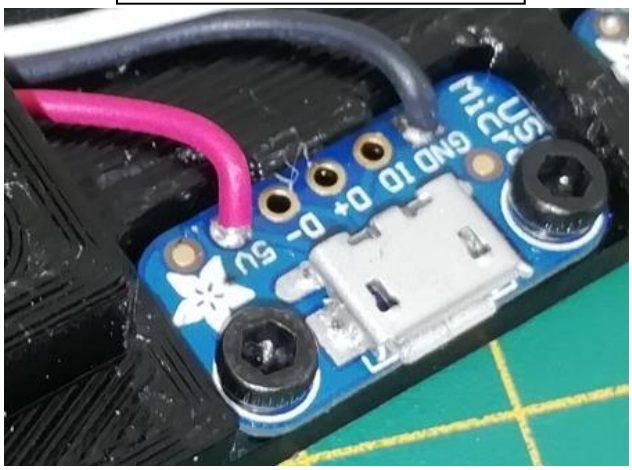








microUSB breakout board:
Wire soldering orientation



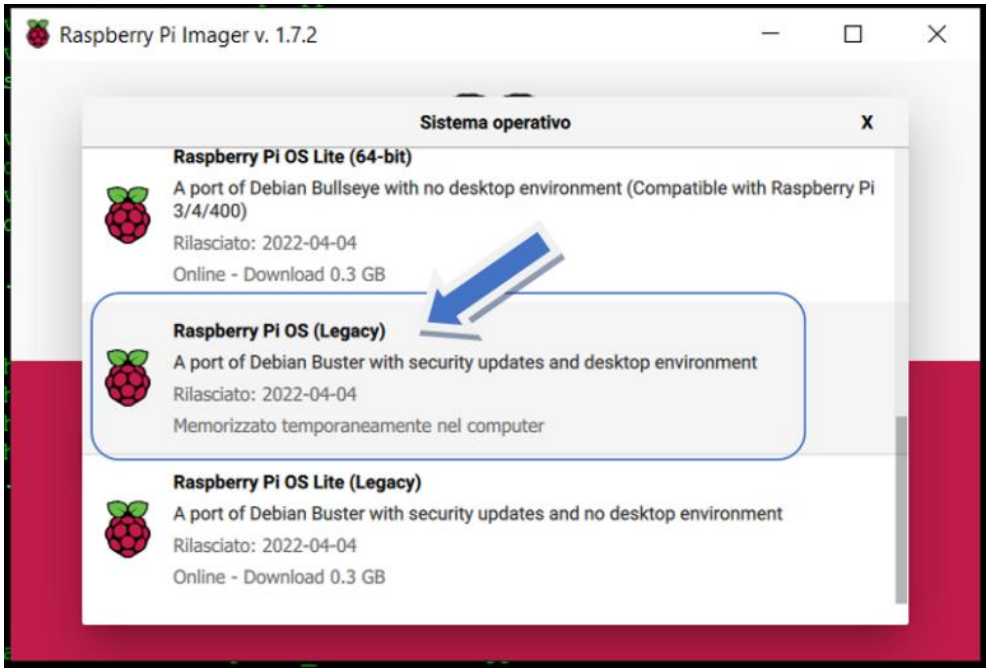
15) Setting up Raspberry Pi Zero 2:

This is a rather long process…
I've repeated the full process few times, to get it done without error messages, below what has worked for me:

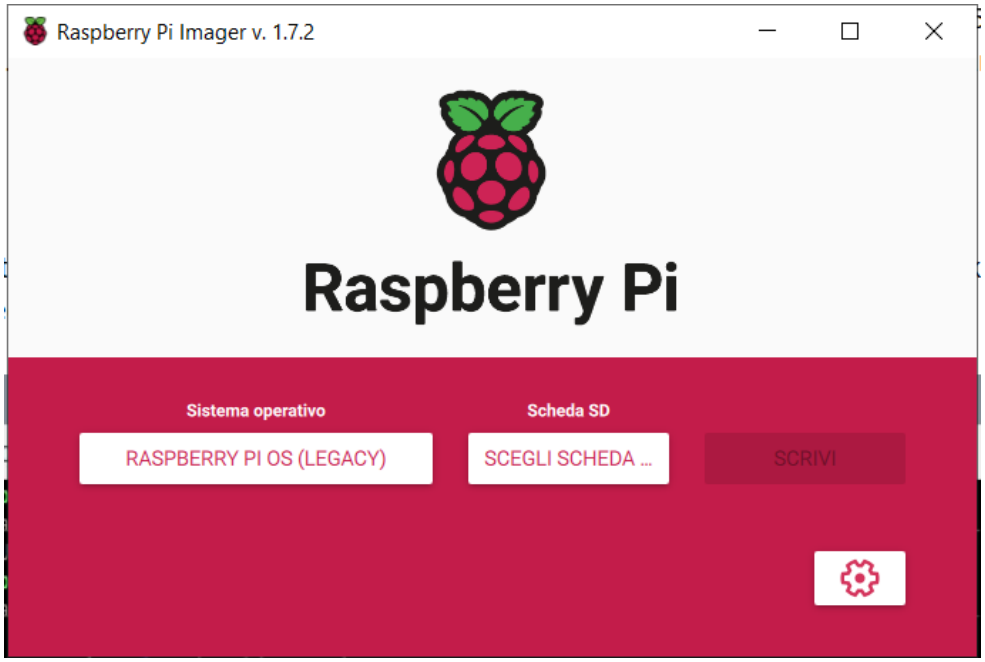
Step1: Download the Raspberry Pi Imager, from https://www.raspberrypi.com/software/

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

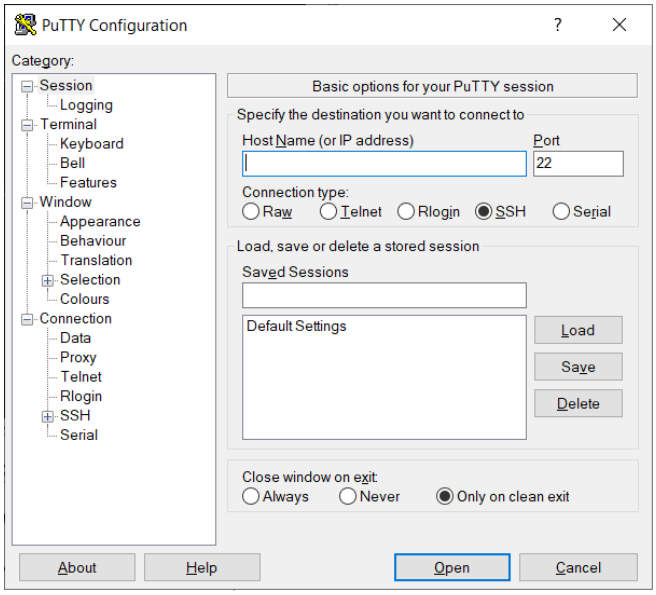


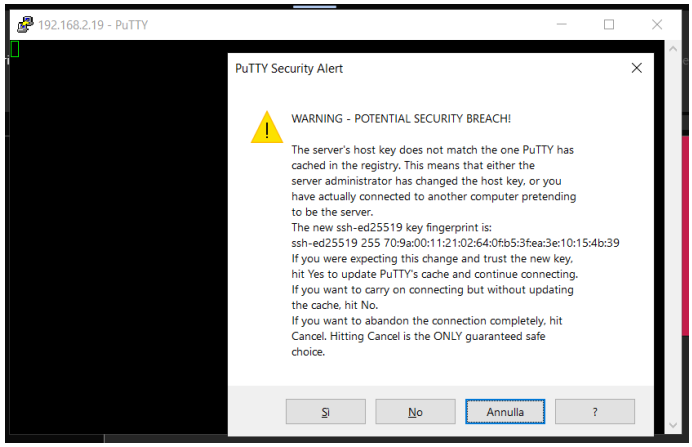
c. On settings, enter:
    a. raspberry.local
    b. check SSH
    c. enter *pi* as username
    d. enter a password (my choice has been *raspberry*, as no sensitive data)
    e. check set WiFi
    f. enter your SSID
    g. enter your network password
    h. enter your Country code
    i. set your local 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: 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*

pi@xxx.xxx.x.xx's password: yyyyyyyy (password *raspberry* in my case, and above suggestion)

Note, in particular for the following steps: if you copy the commands from this doc, and you're using CLI, press Shift + Enter to paste

Step 6: Update and upgrade the SO:

sudo apt-get update
sudo apt-get upgrade (confirm with y when requested)
sudo apt autoremove

Step 7: Few checks (for eventual future reference):

memory check: *df -h*

```
pi@raspberry:~ $ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/root        15G  2.9G   11G  21% /
devtmpfs        182M     0  182M   0% /dev
tmpfs           214M     0  214M   0% /dev/shm
tmpfs           214M  3.2M  211M   2% /run
tmpfs           5.0M     0  5.0M   0% /run/lock
tmpfs           214M     0  214M   0% /sys/fs/cgroup
/dev/mmcblk0p1  253M   49M  204M  20% /boot
tmpfs            43M  4.0K   43M   1% /run/user/1000
```

Ram check: *free -h*

```
pi@raspberry:~ $ free -h
              total        used        free      shared  buff/cache   available
Mem:          427Mi       121Mi       128Mi       2.0Mi       178Mi       254Mi
Swap:          99Mi       4.0Mi        95Mi
```

Python2 version: *python -V*

```
pi@raspberry:~ $ python -V
Python 2.7.16
```

Python3 version: *python3 -V*

```
pi@raspberry:~ $ python3 -V
Python 3.7.3
```

Check if the micro-processor is a 32bit: *uname -a*

```
pi@raspberrypi:~ $ uname -a
Linux raspberrypi 5.15.32-v7+ #1538 SMP Thu Mar 31 19:38:48 BST 2022 armv7l GNU/
Linux
pi@raspberrypi:~ $
```

armv71 is a 32 bit version

Install CV2 library.

(For this installation there are many tutorials in internet, my suggestion is to stick to the same from start to end)

Step 8: Install dependencies for CV2

List of commands to be applied (confirm with Y + Enter when required):

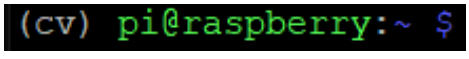
sudo apt-get install build-essential cmake pkg-config
sudo apt-get install libjpeg-dev libtiff5-dev libjasper-dev libpng-dev
sudo apt-get install libavcodec-dev libavformat-dev libswscale-dev libv4l-dev
sudo apt-get install libxvidcore-dev libx264-dev
sudo apt-get install libfontconfig1-dev libcairo2-dev
sudo apt-get install libgdk-pixbuf2.0-dev libpango1.0-dev
sudo apt-get install libgtk2.0-dev libgtk-3-dev
sudo apt-get install libatlas-base-dev gfortran
sudo apt-get install libhdf5-dev libhdf5-serial-dev libhdf5-103
sudo apt-get install libqtgui4 libqtwebkit4 libqt4-test python3-pyqt5

Preparing for the "cv" virtual environment
sudo pip3 install virtualenv virtualenvwrapper
nano ~/.bashrc

append the following lines to the bottom of the file:
 # virtualenv and virtualenvwrapper
export WORKON_HOME=$HOME/.virtualenvs
export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3
source /usr/local/bin/virtualenvwrapper.sh

other commands, to "source" the bashrc and to make the virtual environment:
source ~/.bashrc
mkvirtualenv cv -p python3

Be noted the '(cv)' in front of 'pi@raspberry', indicating the virtual environment is activated:



Step 9: Other libraries (within 'cv' venv), by following the below commands order:

/home/pi/.virtualenvs/cv/bin/python -m pip install --upgrade pip
pip install numpy==1.21.4 (NOTE: the latest 1.21.6 version did not install in Rpi Zero 2 ……)
pip install "picamera[array]"
pip install opencv-contrib-python==4.1.0.25

Check if openCV has been properly installed:

python3	enters python interpreter terminal
import cv2	imports the openCV library, and should return error messages
print(cv2.__version__)	should return 4.1.0
exit()	closes python interpreter terminal

Install GPIO related libraries, into the venv:
pip install Rpi.GPIO
pip install GPIO
pip install gpiozero
pip install pigpio

Install display related libraries, for the display with ST7735 driver
pip install spidev
pip install Pillow
pip install st7735

Install a library to retrieve the mac address (so I can keep using my specific settings)
pip install get-mac

check installed libraries in venv: *pip3 list* (or *pip list* , as only python3 in venv)

```
(cv) pi@raspberry:~/cube $ pip3 list
Package                 Version
----------------------- -----------
colorzero               2.0
get-mac                 0.8.3
gpio                    0.3.0
gpiozero                1.6.2
numpy                   1.21.4
opencv-contrib-python 4.1.0.25
picamera                1.13
pigpio                  1.78
Pillow                  9.1.1
pip                     22.1.2
RPi.GPIO                0.7.1
setuptools              62.1.0
spidev                  3.5
ST7735                  0.0.4.post1
wheel                   0.37.1
(cv) pi@raspberry:~/cube $
```

Step 11: Enable the remote GPIO service, needed for PWM hardware-based timers (PWM !!!); Below command activates this service at every boot

sudo systemctl enable pigpiod

the command should return 'Created symlink /etc/systemd/system/multi-user.target.wants/pigpiod.service → /lib/systemd/system/pigpiod.service.'

Step 12: Make a *cube* folder

Make the cube folder: *mkdir cube*

```
(cv) pi@raspberry:~ $ mkdir cube
```

Enter the cube folder: *cd cube*

```
(cv) pi@raspberry:~ $ cd cube
(cv) pi@raspberry:~/cube $
```

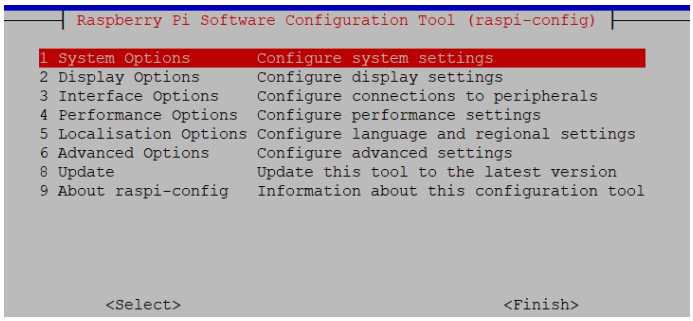
Step 13: setup Raspberry pi interfaces:

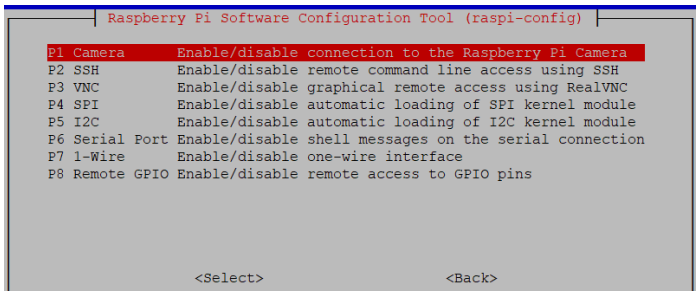
Sudo raspi-config

This opens the below GUI

Use Up and Down arrow to move between options

Use Left or Right arrows to reach <Select>, <Back>, <Finish> with





Activations needed at Interface Options:

Interface	**Info**
Camera	
SSH	Used to virtually connect (screen, tape plate, mouse) for the initial settings
VNC	Also used for virtual connection, but it enable GUI interaction, like VNC Viewer. This is necessary when tuning the vision related part, or simply to enjoy the computer vision part when running
SPI	Used to interact with the little display on the robot
Serial communication	Used to energize the additional LED, as feedback when Rpi is ON

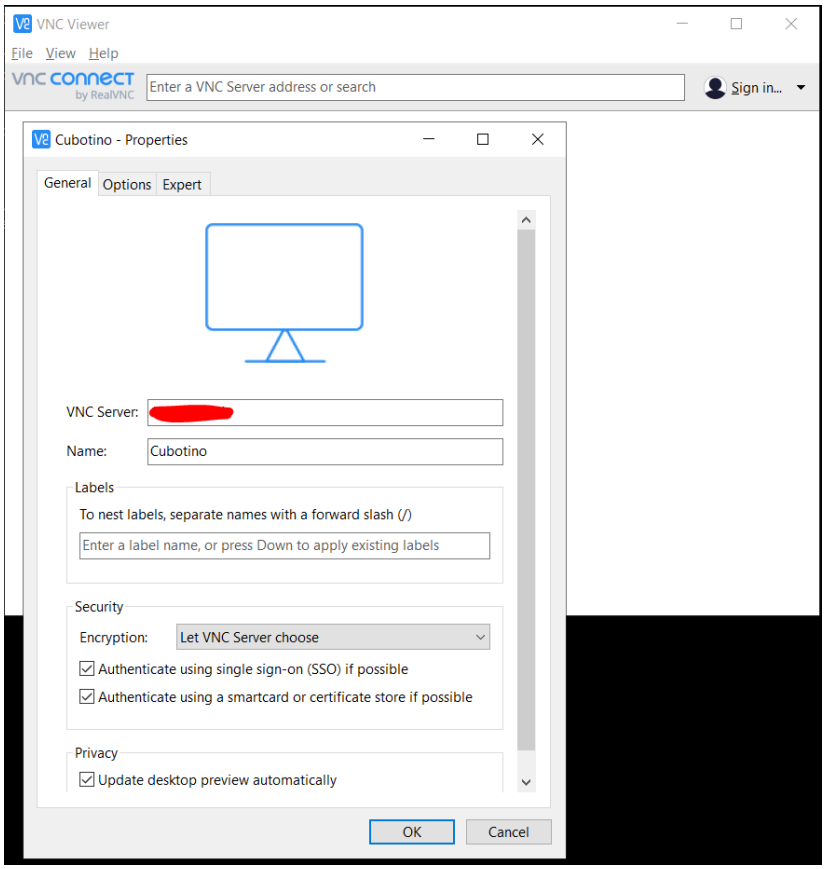
In Display Options, choose a Resolution close to the one of your monitor (in my case DMT mode 85 is the closest, see next pages for the proper Resolution selection)

Reach <Finish> to close the GUI, and confirm the reboot

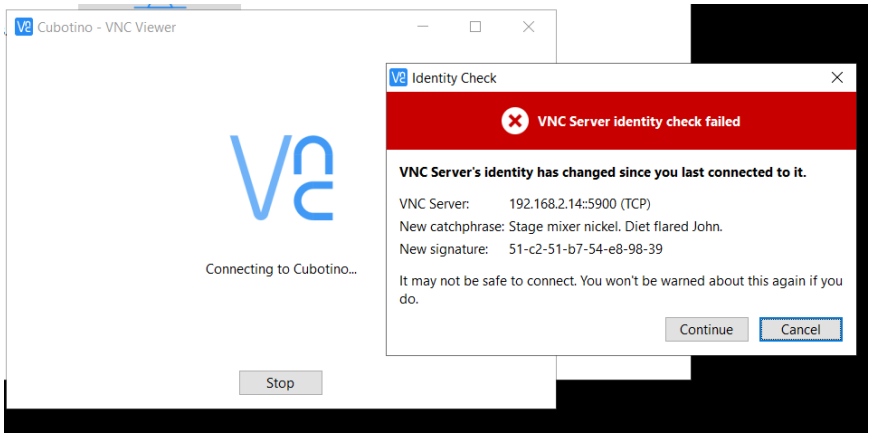
At the next connection with the Raspberry Pi, it will be more convenient using VNC Viever, because of the easy file transfer and later the graphical support.
For VNC Viewer installation: https://www.realvnc.com/en/connect/download/viewer/

Step 14: Make a new connection at VNC Viewer

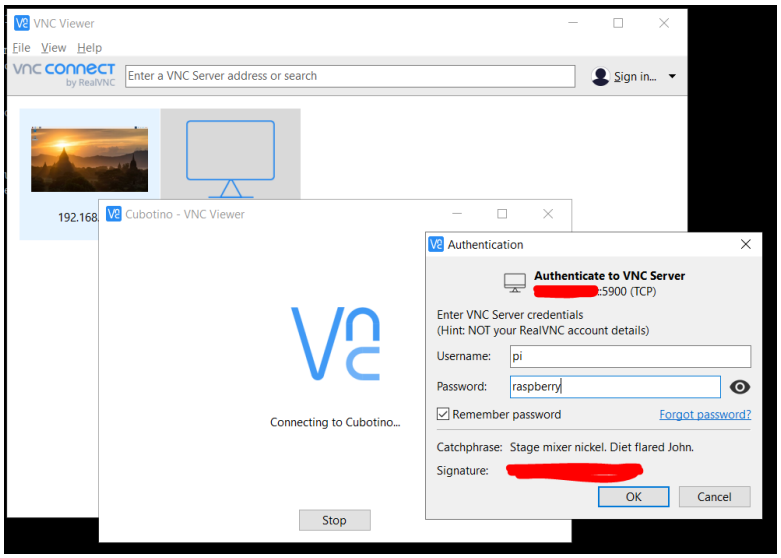
Make a new connection: File, New connection …., insert the IP address at VNC Server, and a Name



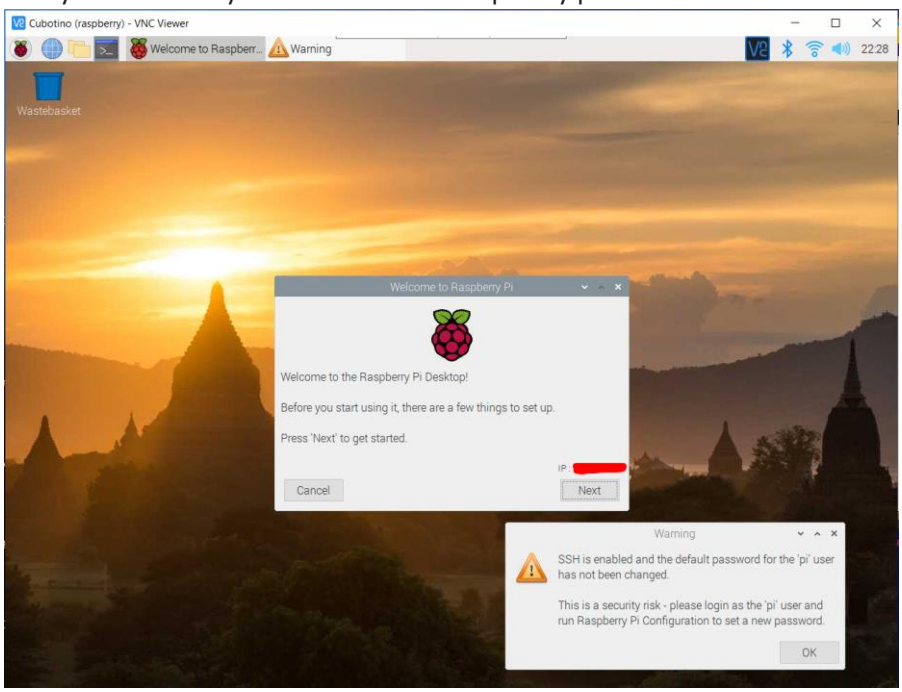
Confirm the warning



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*
- Setup screen, in case necessary to correct the black boarders
- Select Wi-Fi Network
- Update Software

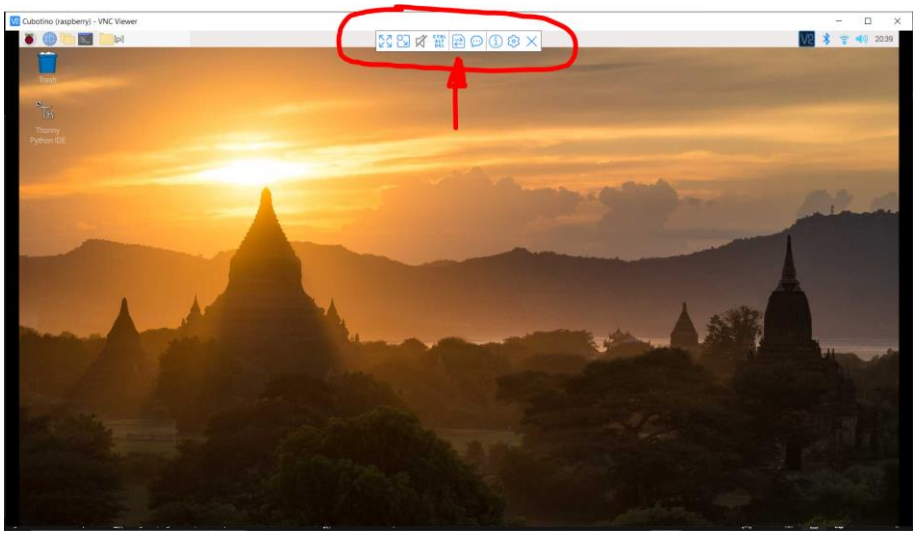
In case the Display Resolution proposed, do not fit well your monitor
Check https://www.raspberrypi.com/documentation/computers/config_txt.html#video-options to find out your hdmi group and your hdmi_mode

From home/pi *sudo nano /boot/config.txt*
Uncomment *hdmi_force_hotplug=1*
hdmi_group=*2* (your hdmi group, 2 in my case)
hdmi_mode=*82* (your hdmi mode, 82 in my case)

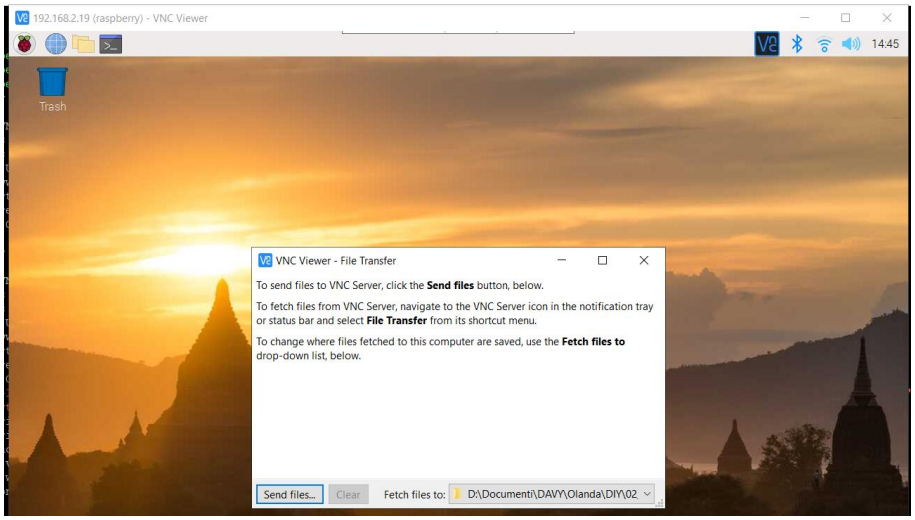
Close the file with Ctrl + X, confirm with Y, and Enter to maintain the same file name

Step 15: Copy files to the Raspberry Pi (in the cube folder):

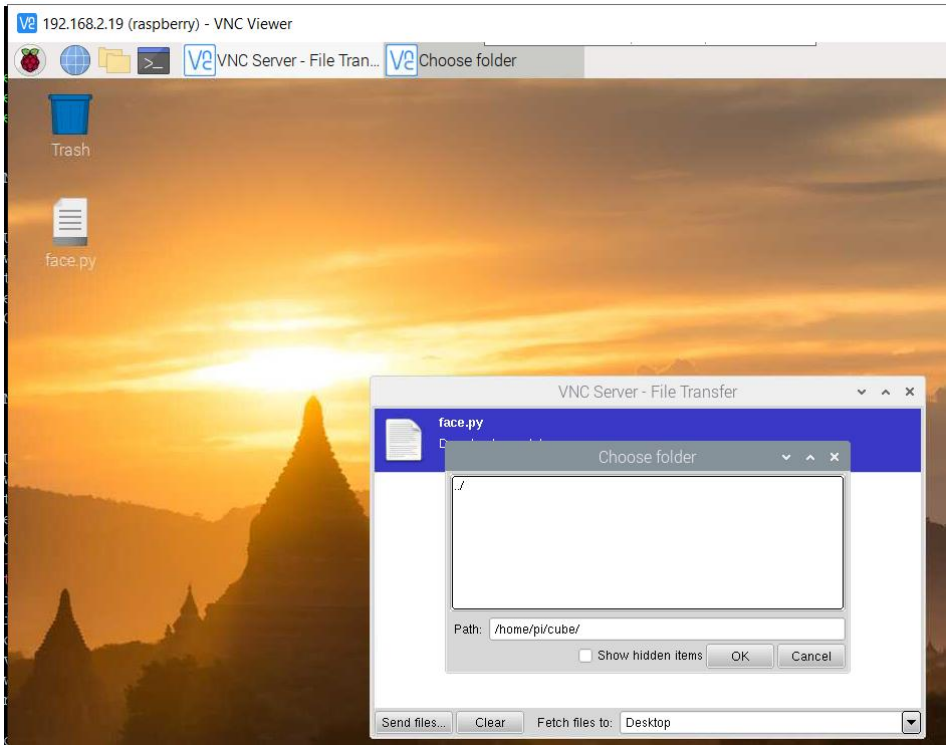
List of the files to be copied is on chapter Files to be copied to raspberry Pi)
To transfer files to Raspberry pi, hoover the mouse on top of the VNC Viewer, and choose the central icon "transfer files"



Follow the description on screen;
TIP: Move this window to the side, as it always hide another one with the destination folders in Rpi



The first time select just one file, to get the chance to select the destination folder:



Step 16: Install the Kociemba solver library

See 'Kociemba solver installation' chapter

Step 17: 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 18: Set things to get the robot starting automatically at the raspberry Pi boot.

See 'Automatic robot start:' chapter

Step19: Set Thonny to work with the venv.

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

See 'Set Thonny IDE interpreter' chapter

16) Files to be copied to Raspberry Pi

Below listed robot specific files, that needs to be copied into */home/pi/cube* 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_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_servo.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:

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 Note: See Kociemba solver installation chapter

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.

17) Kociemba solver installation

RubiksCube-TwophaseSolver is the great solver developed by Mr. Hergbert Kociemba (https://github.com/hkociemba/RubiksCube-TwophaseSolver)

Because the installation can take up5 (five !) hours, you might consider which method is more suitable in your case:

Method	Situation	Description	Difficulty	Estimated time
A	Solver not available in your PCs	Install the solver directly in the Raspberry Pi	Very easy	~ 5 hours
B		Install solver to a PC with Python, and copy the solver files to Raspberry Pi	Some attention is required	~ 1hour
C	Solver already available in your PC	Copy of the solver files to Raspberry Pi	Some attention is required	~ 10 minutes

Method A, Solver not available in your PC and solver installation directly in raspberry Pi:

1) From the terminal enter the cube folder: *cd cube*
2) Activate the Python venv: *workon cv*
3) Install the solver: *pip install RubikTwoPhase* (this part take less than one minute)
4) Activate the python interpreter: *python*
5) From the python prompt, import the solver to start the lookup tables generation: *import twophase.solver as sv* (this takes 5 hours on a raspberry Pi Zero 2, personally verified).
6) Once the last table is done, the python prompt is returned, hopefully without any warning.
7) Exit python interpreter: *exit()*
8) The installation is done; Check for the RubikTwophase library in the venv (cv) installed: *pip3 list*

```
(cv) pi@raspberry:~/cube $ pip3 list
Package                Version
---------------------- -----------
colorzero              2.0
get-mac                0.8.3
gpio                   0.3.0
gpiozero               1.6.2
numpy                  1.21.4
opencv-contrib-python 4.1.0.25
picamera               1.13
pigpio                 1.78
Pillow                 9.1.1
pip                    22.1.2
RPi.GPIO               0.7.1
RubikTwoPhase          1.0.9
setuptools             62.1.0
spidev                 3.5
ST7735                 0.0.4.post1
wheel                  0.37.1
(cv) pi@raspberry:~/cube $
```

9) installation folders when method A):
   - Solver python files are installed in: /home/pi/.virtualenvs/cv/lib/python3.7/site-packages/twophase
   - Solver lookup table files are installed in: /home/pi/cube/twophase

List of python files are located at */home/pi/.virtualenvs/cv/lib/python3.7/site-packages/twophase*:

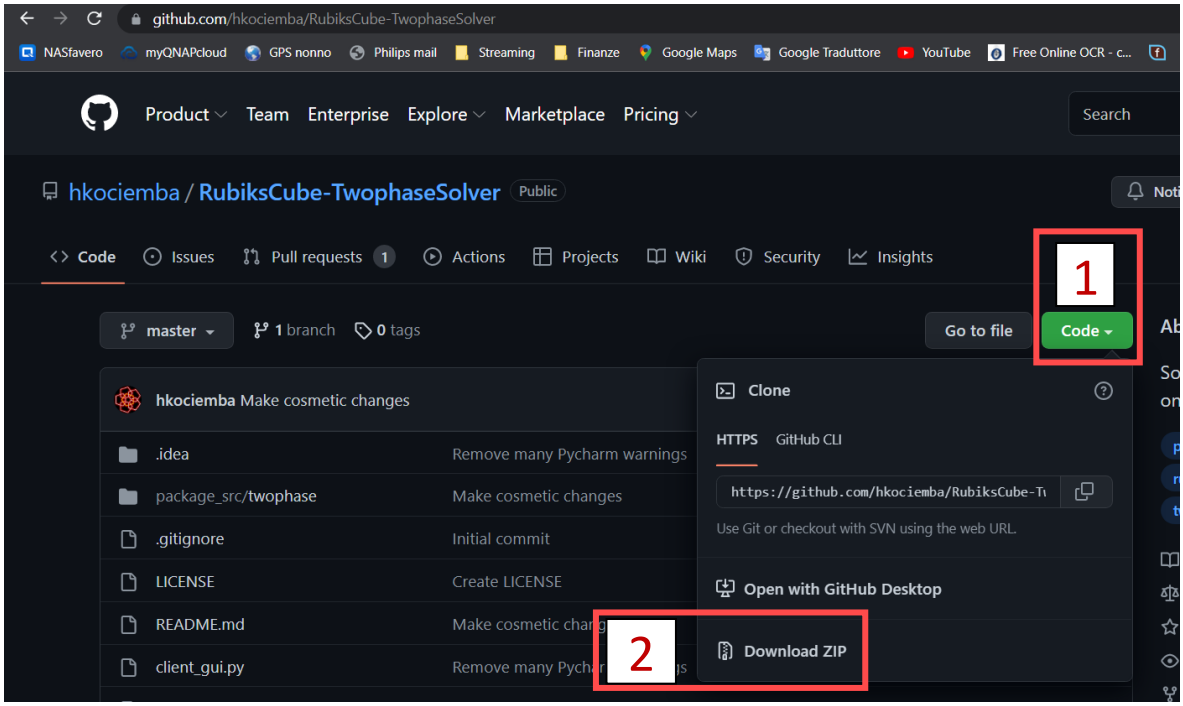
- *client_gui2.py*
- *coord.py*
- *enums.py*
- *misc.py*
- *pruning.py*
- *sockets.py*
- *symmetries.py*
- *client_gui.py*
- *cubie.py*
- *face.py*
- *moves.py*
- *solver.py*
- *vision2.py*
- *computer_vision.py*
- *defs.py*
- *__init__.py*
- *performance.py*
- *server.py*
- *start_server.py*
- *vision_params.py*

List of files locate at */home/pi/twophase*:

- *co_classidx*
- *co_rep*
- *fs_rep*
- *move_d_edges*
- *move_twist*
- *phase1_prun*
- *phase2_prun*
- *conj_twist*
- *co_sym*
- *fs_sym*
- *move_flip*
- *move_ud_edges*
- *phase2_cornsliceprun*
- *conj_ud_edges*
- *fs_classidx*
- *move_corners*
- *move_slice_sorted*
- *move_u_edges*
- *phase2_edgemerge*

Method B, Solver not installed in Raspberry Pi nor in your PC :

1) At your PC make a folder (i.e. cube) in the folder you're used to work with python.
2) Into that folder, make a sub-folder named 'twophase'.
3) Go to https://github.com/hkociemba/RubiksCube-TwophaseSolver
4) Select *Code*, *Download ZIP*



5) Unzip the content into the 'twophase' folder you've made on step 2
   In my case these files would be located at: C:\Users\andre\Python\cube\twophase
   Python file list as per method A)

6) Move back to one level upper than 'twophase': *cd ..*

7) Check the solver functionality at your PC:
   The first time the solver is imported, it generates all the lookup tables; The time needed depends on the PC hardware.
   Launch python interpreter: *python*
   Import the solver: *import twophase.solver as sv*
   The solver will start the tables generation; These files are expected to be in 'twophase' folder, located into the folder you've initially made (cube in my case).

```
(base) C:\Users\andre>cd python\cube

(base) C:\Users\andre\Python\cube>pip install RubikTwophase
Collecting RubikTwophase
  Using cached RubikTwoPhase-1.0.9-py3-none-any.whl (53 kB)
Installing collected packages: RubikTwophase
Successfully installed RubikTwophase-1.0.9

(base) C:\Users\andre\Python\cube>python
Python 3.8.13 (default, Mar 28 2022, 06:59:08) [MSC v.1916 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import twophase.solver  as sv
On the first run, several tables will be created. This takes about 1/2 hour or longer (depending on the hardware).
All tables are stored in C:\Users\andre\Python\cube\twophase

creating conj_twist table...
creating conj_ud_edges table...
...................
```

When all the tables are completed (total process ca 60minutes in my case), if everything went well, the python prompt is presented.

8) Exit python: *exit()*

9) Check the of the lookup table files, that should be at the 'twophase' folder; In my case these files would be located at: C:\Users\andre\Python\cube\twophase
Lookup tables files list as per method A)

10) Copy the 20 python files, and the 19 Lookup tables files, to raspberry Pi folder */home/pi/cube/twophase* (to copy files to/from raspberry Pi, check the method in 'Setting up Raspberri Pi' chapter).

11) Check the solver functionality in raspberry Pi:
    a. Enter the cube folder: *cd cube*
    b. Run the python interpreter: *python*
    c. From python prompt import the solver: *import twophase.solver as sv*
    d. The return should be the loading of the tables:

```
(cv) pi@raspberry:~/cube $ python
Python 3.7.3 (default, Jan 22 2021, 20:04:44)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import twophase.solver as sv
loading conj_twist table...
loading conj_ud_edges table...
loading flipslice sym-tables...
loading corner sym-tables...
loading move_twist table...
loading move_flip table...
loading move_slice_sorted table...
loading move_u_edges table...
loading move_d_edges table...
loading move_ud_edges table...
loading move_corners table...
loading phase1_prun table...
loading phase2_prun table...
loading phase2_cornsliceprun table...
>>>
```

    e. Try to solve a scrambled cube:
    sv.solve("DDUDUDUDDFFFLRLBBBLRRFFFLLRUUDUDUDUUBBBRLRFFFRLLBBBRRL", 20 , 2)
    returns: *'F2 R2 F2 L2 U3 D3 R2 B2 L2 B2 (10f)'*

```
>>> sv.solve("DDUDUDUDDFFFLRLBBBLRRFFFLLRUUDUDUDUUBBBRLRFFFRLLBBBRRL", 20 , 2)
'F2 R2 F2 L2 U3 D3 R2 B2 L2 B2 (10f)'
```

12) In this method, all the solver files (python and lookup tables) are into the same folder in */home/pi/cube/twophase*; Files list as per the two lists of files in method A)

Method C, Solver is already installed in your PC but not in the Raspberry Pi:

1) At Raspberry Pi make a 'twophase' folder in */home/pi/cube/*
    a. Enter the cube folder: cd cube
    b. Make the 'twophase' folder: *mkdir twophase*

2) In your PC, locate the solver python files:
    You might search for 'twophase'
    Another possibility is to search on the python interpreter PATH:
        a. run python: *python*
        b. Import a couple of libraries: *Import os, sys*
        c. Check for the PATH: *os.path.dirname(sys.executable)*
        The interpreter PATH is returned
        in my case it returns 'C:\\Users\\andre\\anaconda3', meaning the 'twophase' folder should be under that location: For me 'C:\Users\andre\anaconda3\Lib\site-packages\twophase'

3) Copy the 20 python files (see file list in method A) to Rapsberry /home/pi/cube/twophase
    (to copy files to/from raspberry Pi, check the method in 'Setting up Raspberri Pi' chapter).

4) Locate the solver lookup tables files in your pc:
    You might search for 'twophase' folder again, or for 'phase1_prun'

5) Copy the 19 files (see file list in method A) to Rapsberry /home/pi/cube/twophase
    (to copy files to/from raspberry Pi, check the method in 'Setting up Raspberri Pi' chapter).

6) Check if the solver works in Raspberry Pi, as done in step 11 of method B)

7) In this method, all the solver files (python and lookup tables) are into the same folder in */home/pi/cube/twophase*; Files list as per the two lists of files in method A)

18) Automatic robot start:

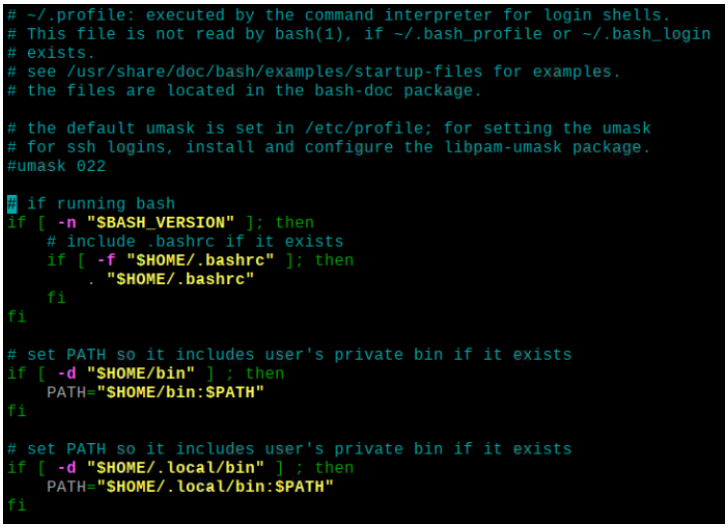
It is possible to have the robot starting-up automatically, when the Raspberry Pi boots.
As reference: https://www.pyimagesearch.com/2016/05/16/running-a-python-opencv-script-on-reboot/#comment-428806

1) From the root (***/home/pi***), edit profile settings (***nano ~/.profile***), and add the below strings at the end:

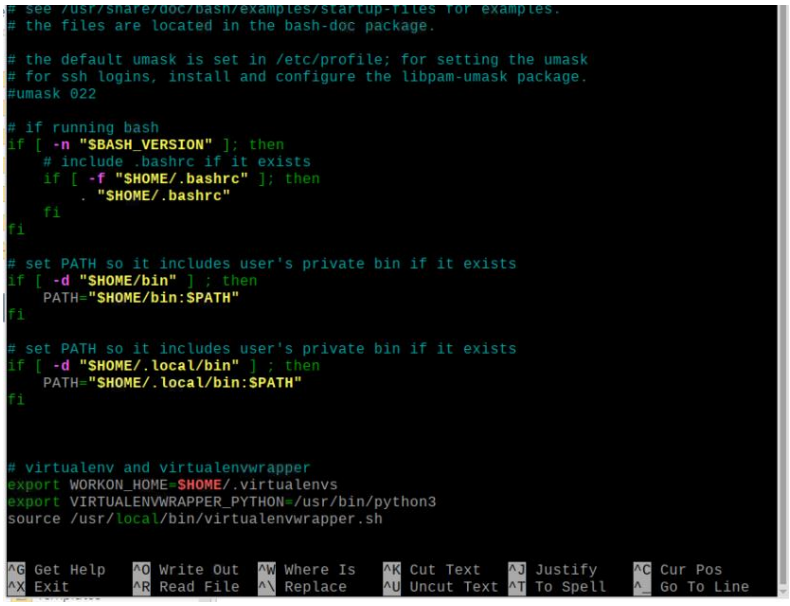
```
# virtualenv and virtualenvwrapper
export WORKON_HOME=$HOME/.virtualenvs
export VIRTUALENVWRAPPER_PYTHON=/usr/bin/python3
source /usr/local/bin/virtualenvwrapper.sh
```

To exit the file press 'Ctrt+X', then 'Y' to confirm the change saving, finally 'Enter' to confirm the same filename.
After edit it has to be activated with: ***. .profile*** (attention to tape **dot** space **dot profile**).

picture on how the .profile file was before the addition:



And after the addition:



2) Edit crontab by typing: ***sudo crontab -e*** (first time choose i.e. nano) and add below commands at the end:

MAILTO=""

@reboot /bin/sleep 5; bash -l /home/pi/cube/Cubotino_T_bash.sh > /home/pi/cube/Cubotino_T_terminal.log 2>&1

Notes:

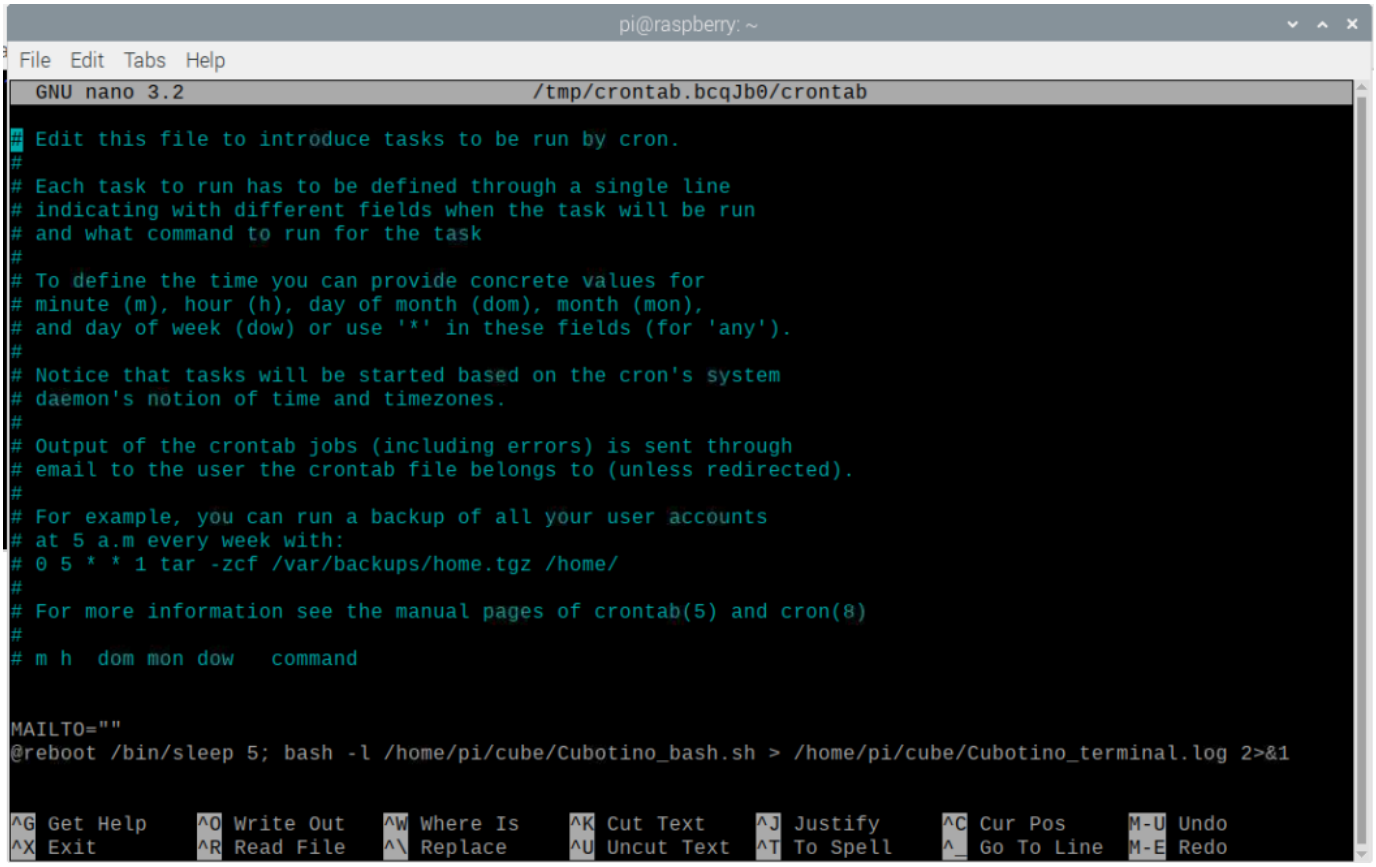
First row's command prevents errors if the email isn't set

Second one imposes 5seconds delay from boot, set user "pi" and sources the bash script; Additionally, it logs eventual error messages to the file *Cubotino_terminal.log*

The *Cubotino_T_bash.sh* bash file can be tested, from the folder where it's located , by typing:

. Cubotino_T_bash.sh (attention to tape **dot** space **Cubotino_T_bash.sh**)

File ***crontab -e*** will result:



3) Create a bash file to manage the commands sequence:

If you haven't copied the *Cubotino _T_bash.sh* file, in */home/pi/cube* folder, it's still possible to create such file; From the root (*/home/pi/cube*), edit the file with *sudo nano Cubotino _T_bash.sh* and add the below content.

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

#######  Andrea Favero,  31 May 2022  ##########################################
#  This bash script activates the venv, and starts the Cubotino_T.py script after the Pi boot
#  When the python script is terminated (GPIO26), the Pi shuts down
################################################################################

source /home/pi/.virtualenvs/cv/bin/activate
cd /home/pi/cube
python Cubotino_T.py

# '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
```

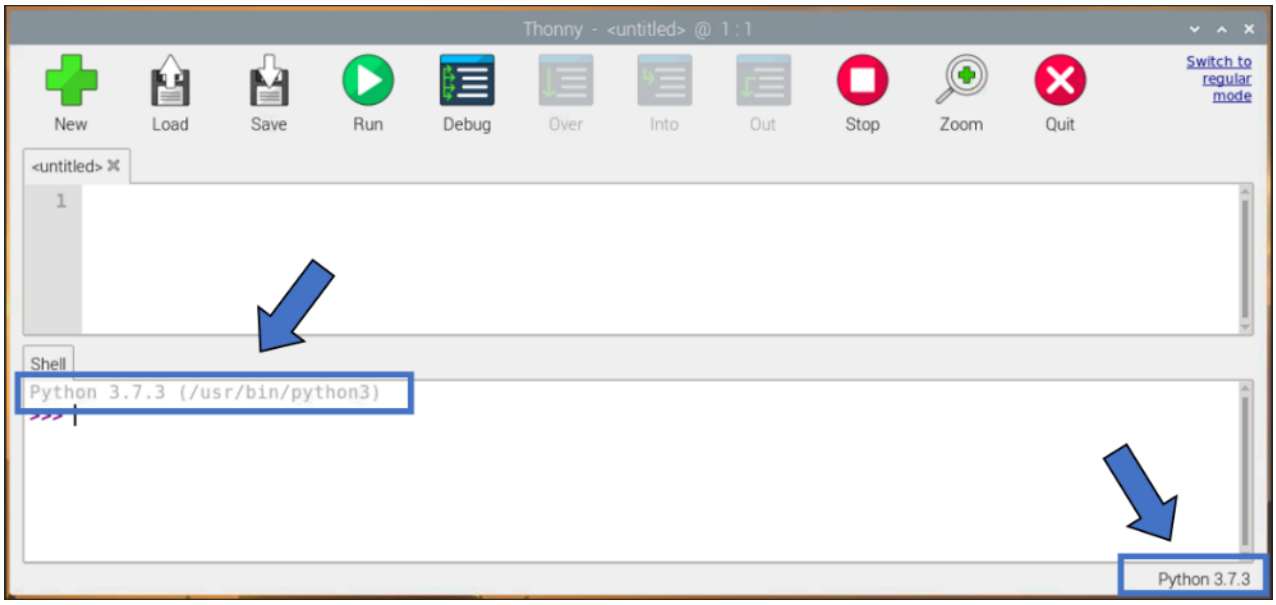
Do not uncomment the last row (halt -p):

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.

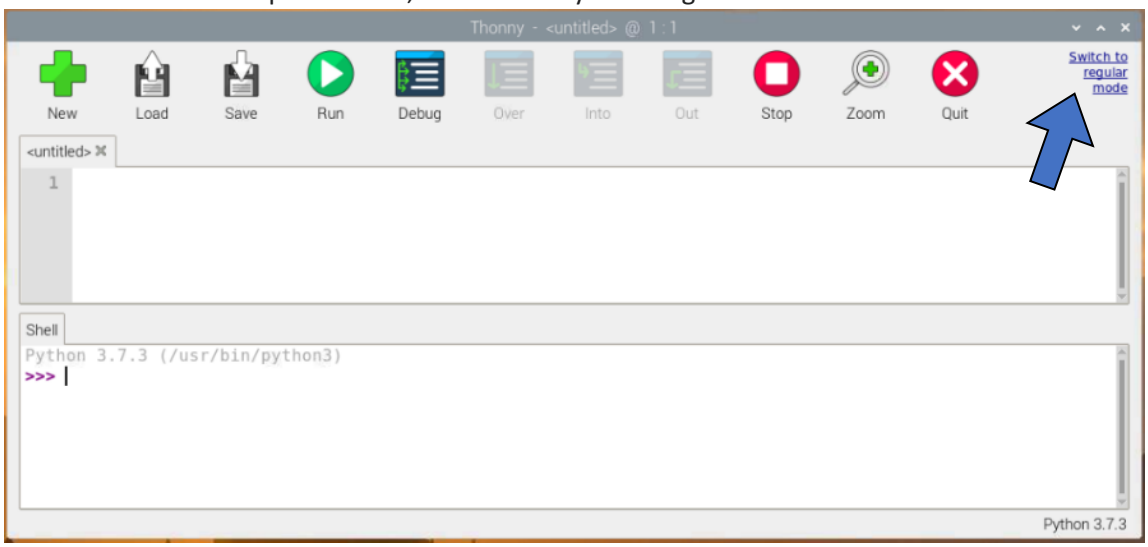
19) Set Thonny IDE interpreter:

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

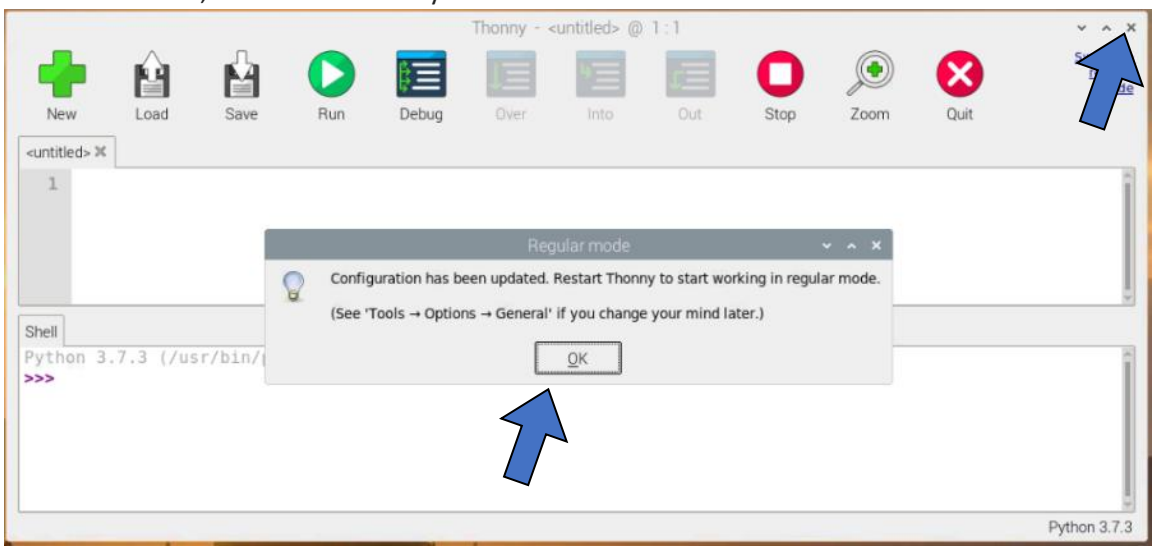
Thonny opens with the standard interpreter (/usr/bin/python3), and if you run Cubotino_T.py it won't find the libraries….



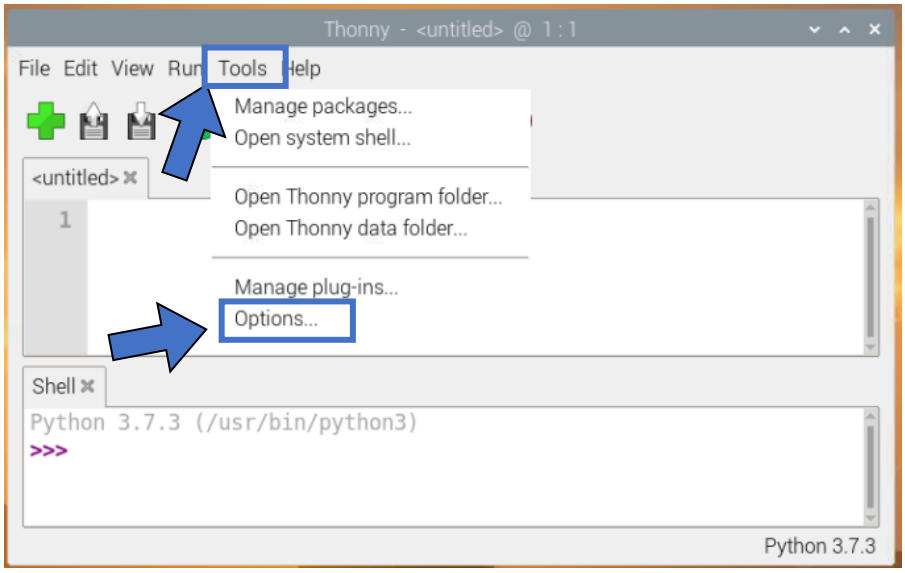
In order to have the Option menu, it is necessary to change the mode:



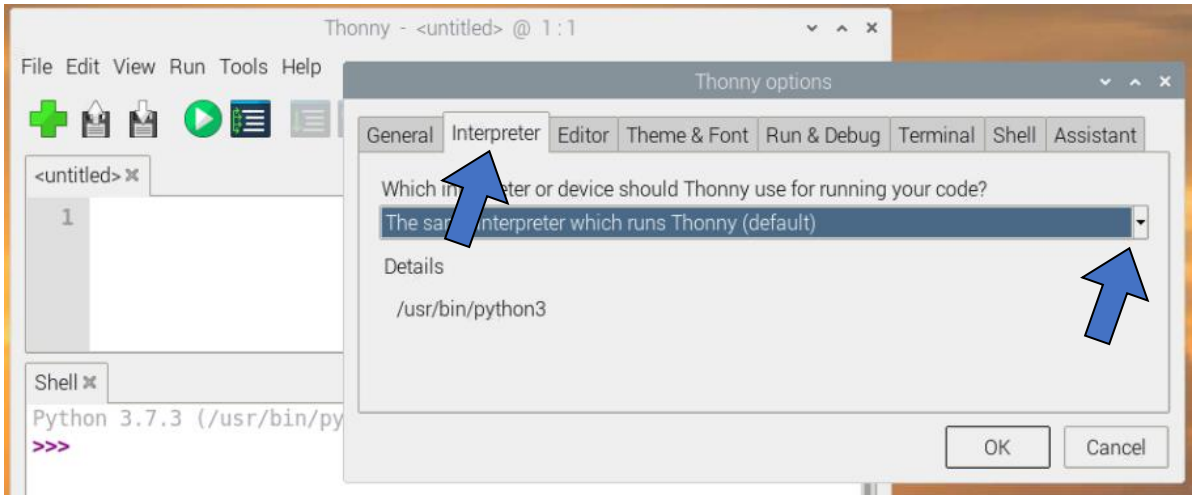
Confirm the info, and restart Thonny



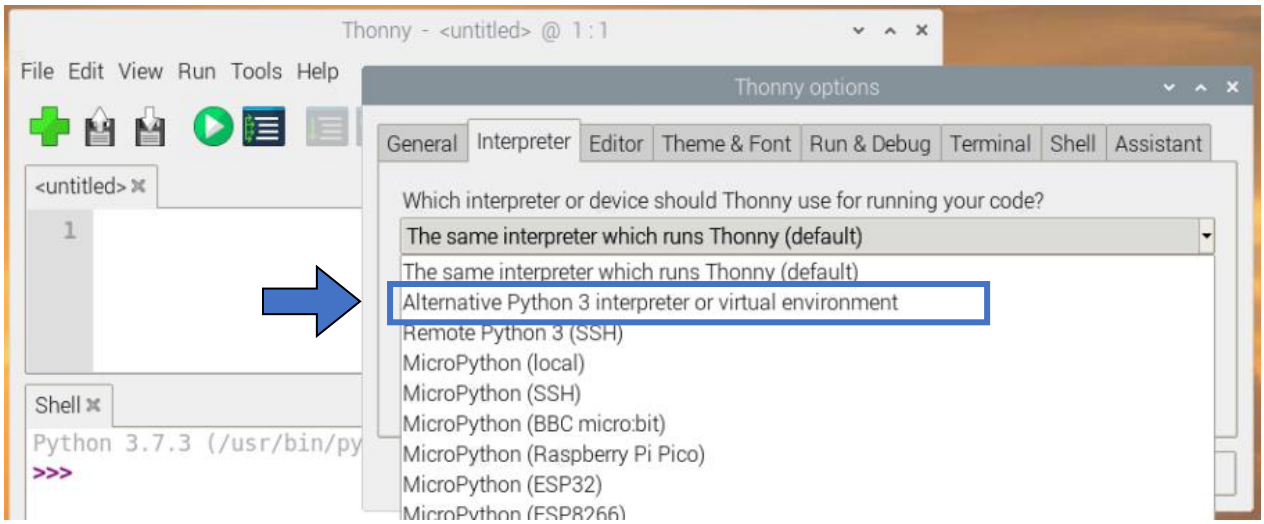
Restart Thonny, and select Tool, Option



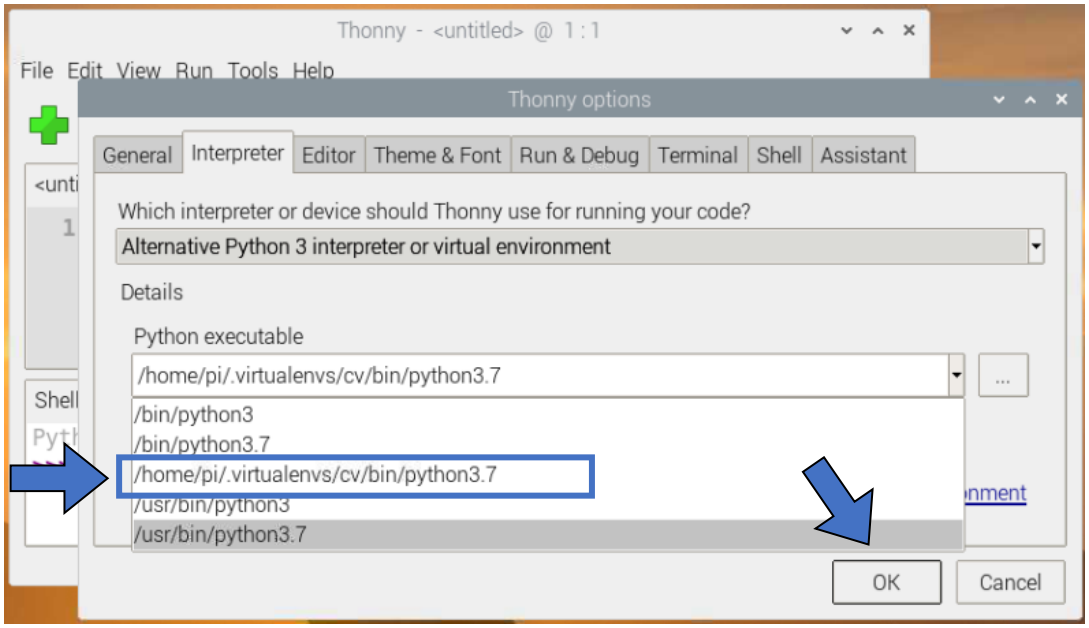
Select Interpreter were it is shown the default setting, pointing to /usr/bin/python3.



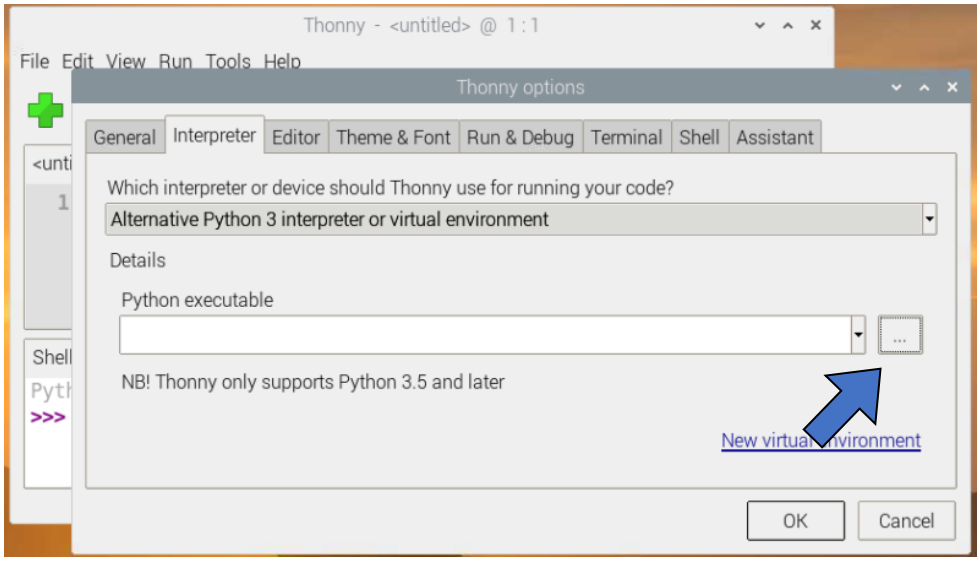
Open the drop-down menu and select ‘Alternative Python 3 interpreter or virtual environment’:



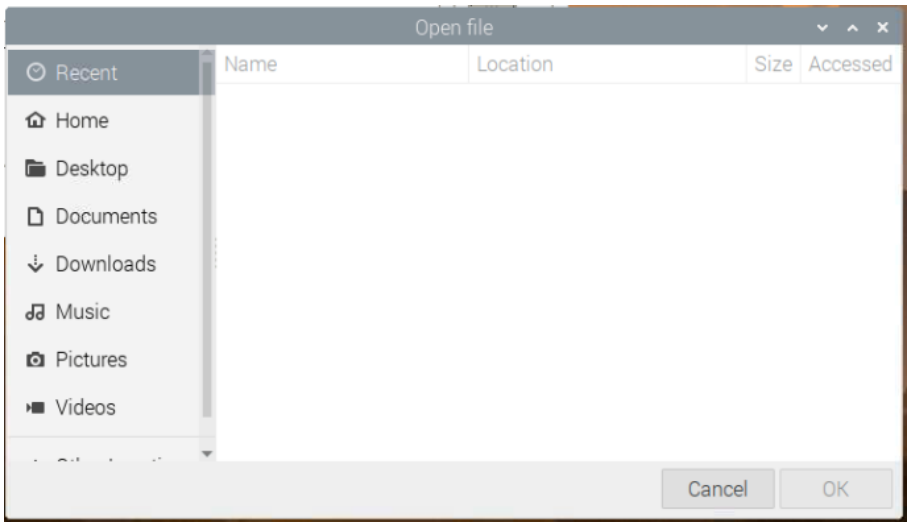
Open the drop-down menu and if ‘/home/pi/.virtualenvs/cv/bin/python3.7’ is listed just select it



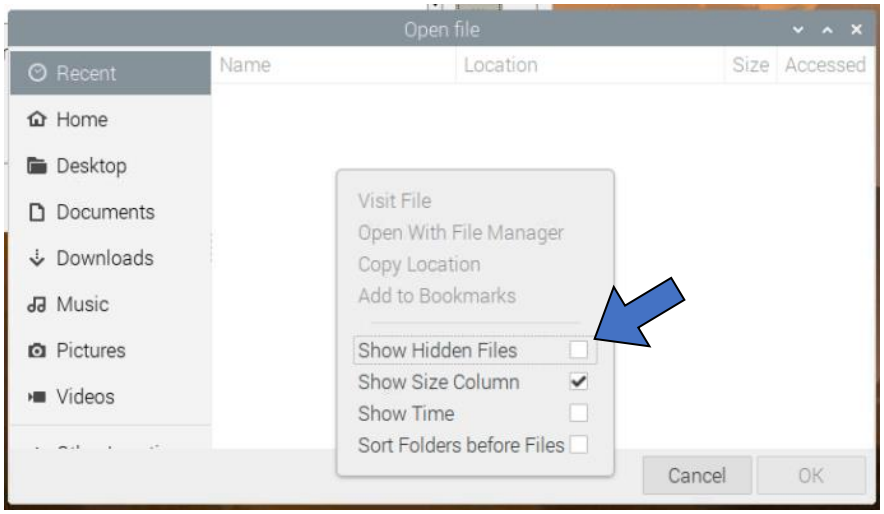
If ‘/home/pi/.virtualenvs/cv/bin/python3.7’ is not listed, select the browse button:



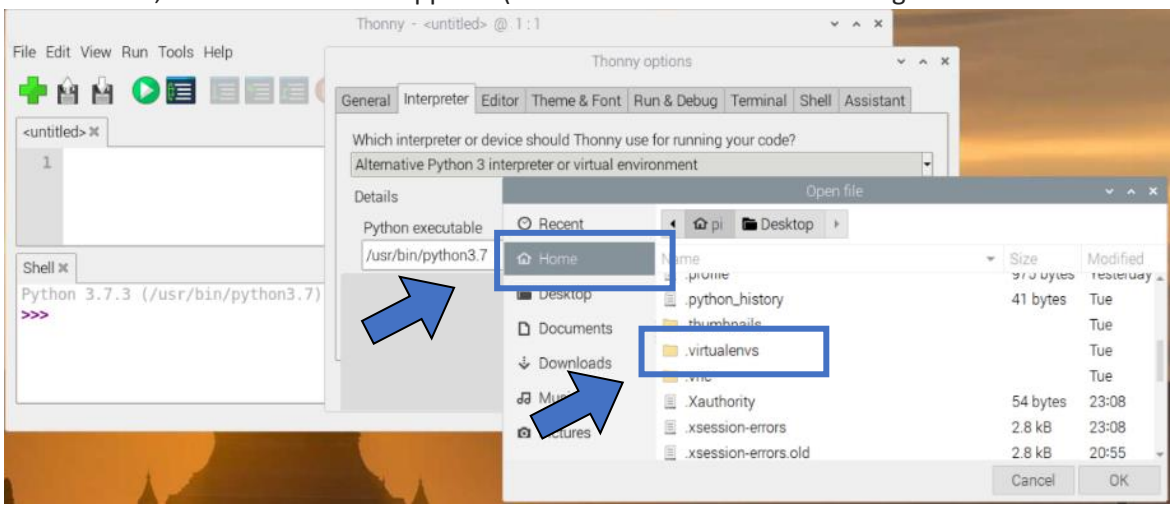
An Open file window opens:



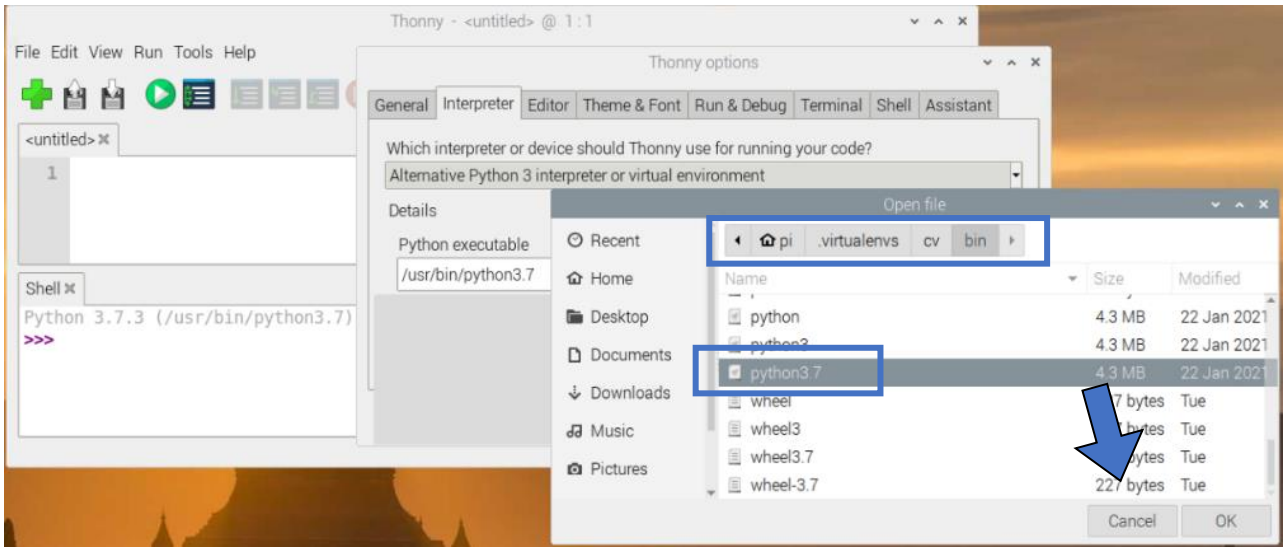
Right click on the empty window part, and check ‘Shows Hidden Files’:



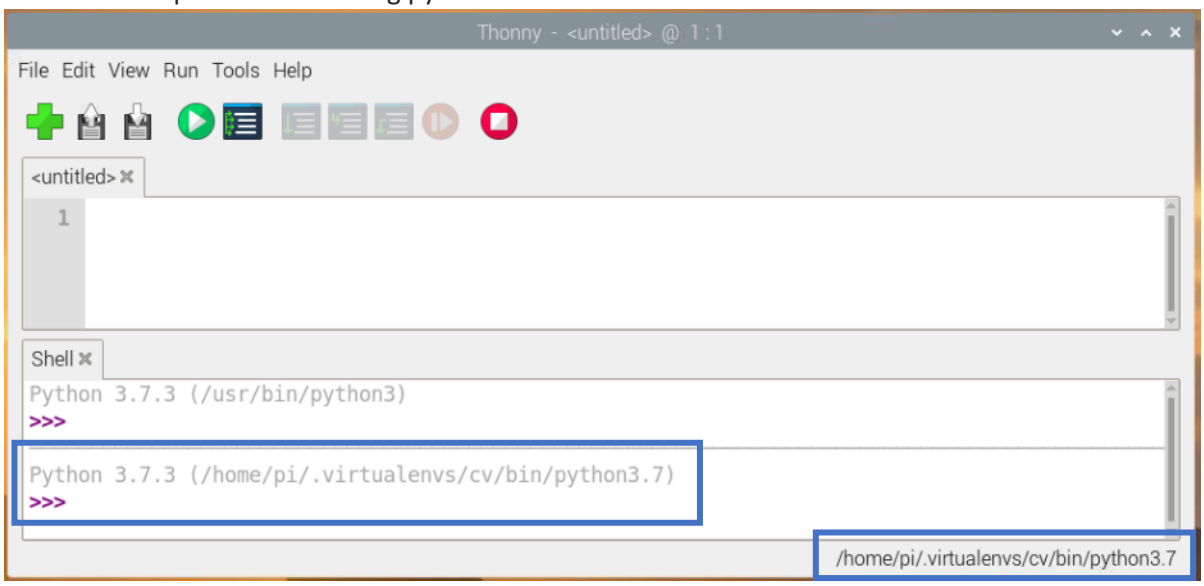
Select Home, .virtualenvs should appears (Note: all folders and files starting with a dot are hidden type)



Select the full path: Home/.virtualenvs/cv/bin/python3.7 and confirm a couple of times



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

20) Assembly steps:

Before assembling the robot:

- Make the connections board
- Setup the Raspberry Pi
- Install all the libraries, and copy the needed files
- Position the two servos output gear to their middle position (see Tuning chapter)

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

3. Screw the bottom servo to the structure
4. Prepare the sandwich Servo_axis_inf / servo lever / Servo_axis_inf
5. Assemble the Cube_holder to the Servo_axis assembly
6. Assemble the Cube_holder assembly to the bottom servo
7. Assemble the Hinge to the Structure
8. Assemble the “T25” servo arm to the upper servo.
9. Insert the Top_servo assembly into the Top_cover slot
10. Assemble the Top_cover assembly to the Hinge
11. Complete the top servo assembly to the Hinge
12. Assemble the Lifter to the Top_cover

Starting from here, the steps are different from the Base version

13. Position the cables, and assemble the Baseplate_rear
14. Connect the PiCamera flex cable to Raspberry Pi Zero2 board, and fix it to the Structure
15. Fix the Touch_sensor to the PCB cover
16. Connect the servos, LED breakout board and Touch sensor
17. dress the cable and connect the display
18. Assemble the PCB_cover
19. Assemble the Baseplate_front to the Structure):
20. Assemble the PiCamera holder frame to the Top_cover
21. Assemble the LED breakout board
22. Assemble the PiCamera holder to the PiCamera holder frame
23. Connect the flex cable
24. Personalize the formal Stop plate

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

21) 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**:
   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).

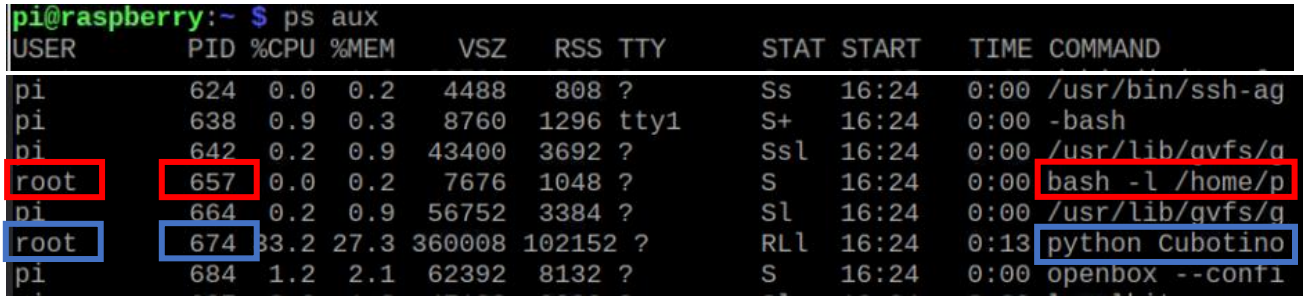
5. **Start a solving cycle:**
   a. Position the cube on the cube holder; any cube orientation is accepted.
   b. Cube layers should be reasonably aligned.
   c. Shortly touch the PCB_cover in front to the touch sensor (the circle suggests the touch sensor location).
   d. The robot reacts by energizing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

6. **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 *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.

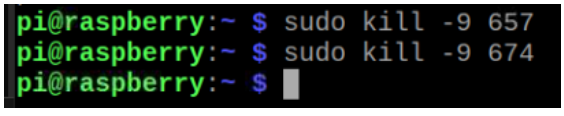
7. **Un-power the servos**:
   Servos can be unpowered at any moment.
   Un-power the servo before shutting down Rpi, if you experience strange servo behaviour when Rpi goes off; The script takes care to set the servo PWM related GPIO to a fix level (low), at the shut-down, but it does not work on 100%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

8. **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. List all the running processes via *ps aux*



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



Note: by reversing the order on steps f and g, the Raspberry Pi will shuts off , right after the Cubotino python process is killed 😊

22) Tuning:

1. **General:**

   There are parameters that are expected to be differently tuned on each robot; These 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 connection 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.
The Cubotino_T_servos.py uses gpiozero library to manage the servo PWM; this library converts a range from -1 to 1 (0 is the mid angle, value is a float), to the PWM signal necessary for the servos.

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

- Check if both servos have 180° rotation; In case only one servo rotates 180° or slightly more, use this servo for the cube holder.
- Set both the servos on their mid angle position, prior to the robot assembly.

Checking the servo rotation angle, and to setting them to their mid position:

1. Connect a connection arm to the non-assembled servos
2. Connect the servos to the Connections board
3. Argument set, and related value, can be passe to *Cubotino_T_servo.py* to play with the servos angle and especially to set them to the mid position before the assembling.
4. Enter *home/pi/cube* forder, activate cv venv (*workon cv*), type *Cubotino_T_servo.py --set 0* to set the servos to the mid position; Attention to the double '-' without space in between, and the space before the 0 (zero).
5. To check the rotation angle of the servos, you can type a different 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.

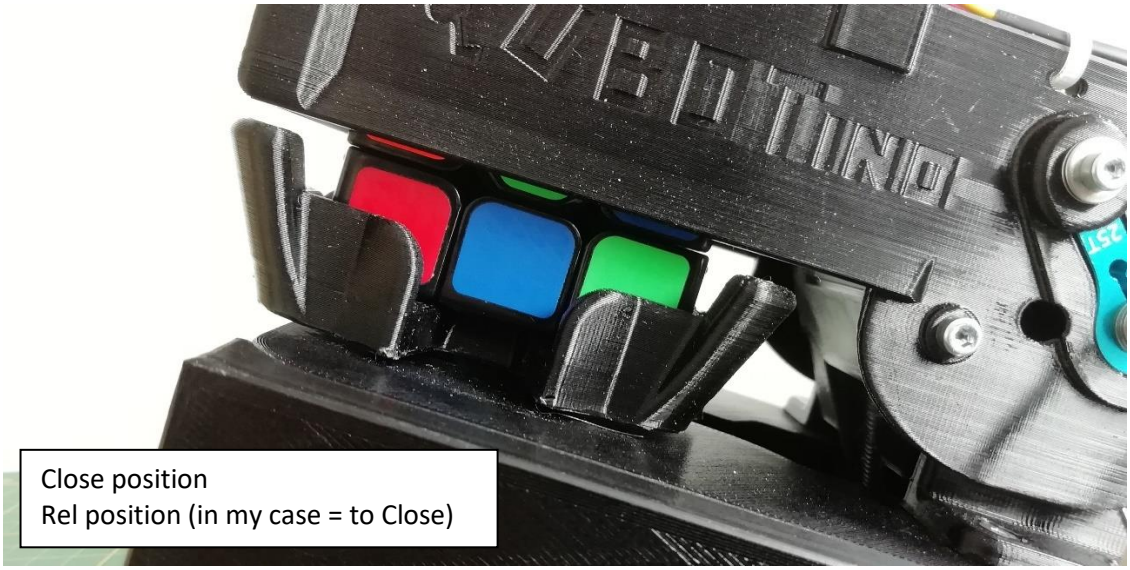
Notes:

1. In case the servo for the cube_holder makes less than 180° rotation, it is necessary to increase its 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'.
2. Working angles for the servos, are set in a (json) text file: Cubotino_T_servo_settings.txt

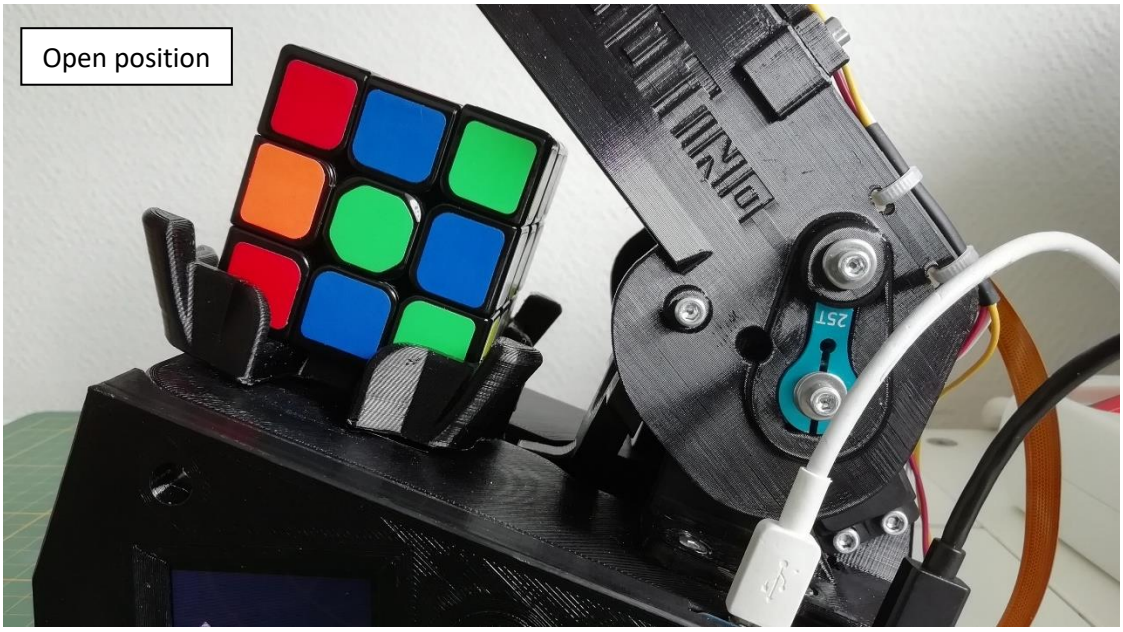
Top_cover (t_servo) angles:

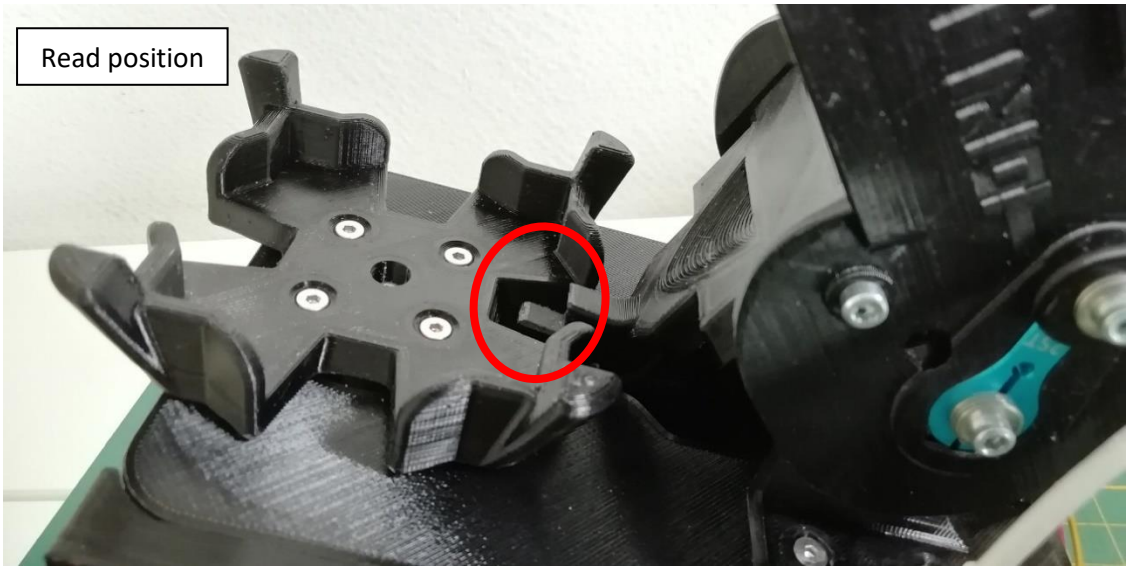
Working angles for the servos, are set in a (json) text file: Cubotino_T_servo_settings.txt
There are 5 defined angles (most of time mentioned as positions):

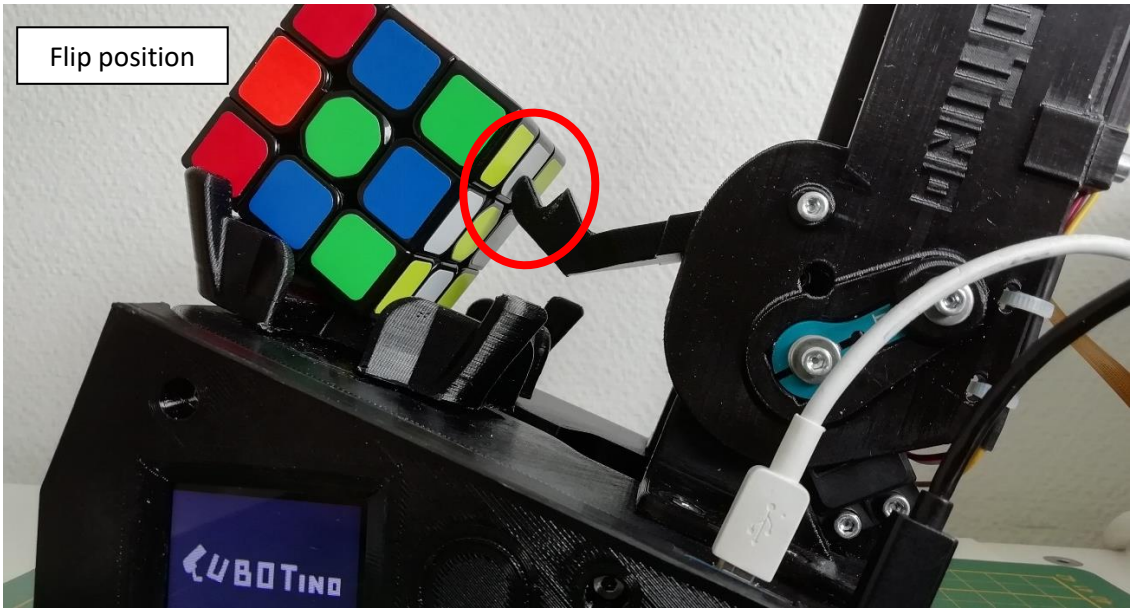
a. Close: position to constrain the top and mid cube layers
b. Rel: position to release tension, from cube, at Close position
c. Open: position without interferences with the cube and Cube_holder
d. Read: position for camera reading, with the Lifter almost touching the cube (and unfortunately constraining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



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


Open position


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)

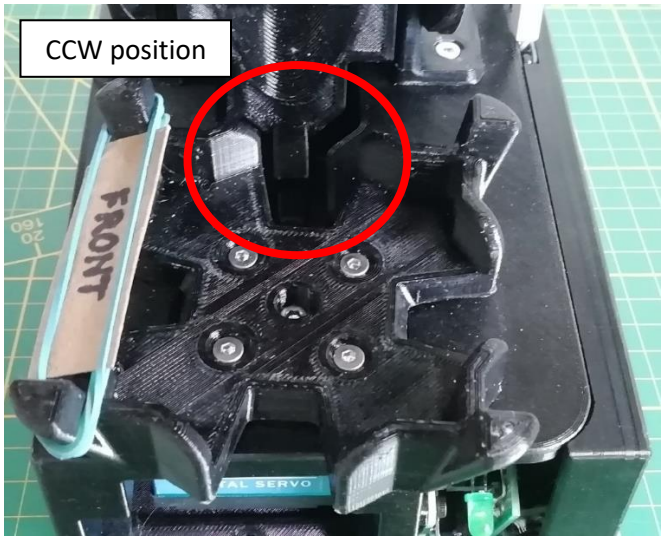
b. CW: position to spin or rotate the Cube_holder >90° CW from Home (Direction according to the motor point of view)

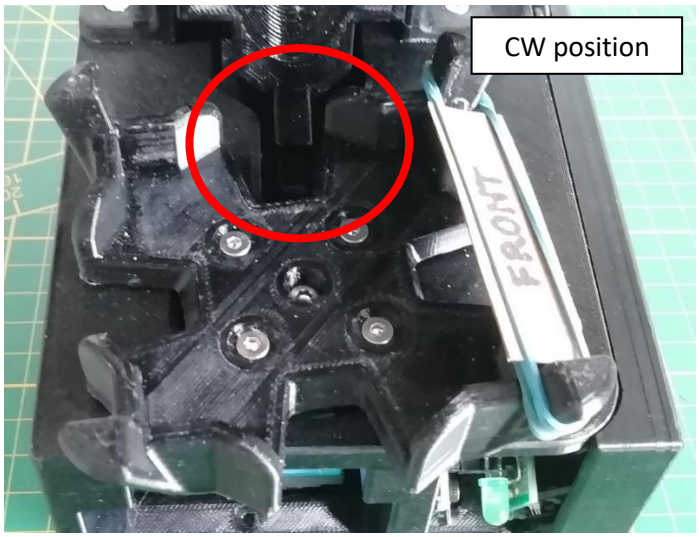
c. Home: mid position between CCW and CW

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

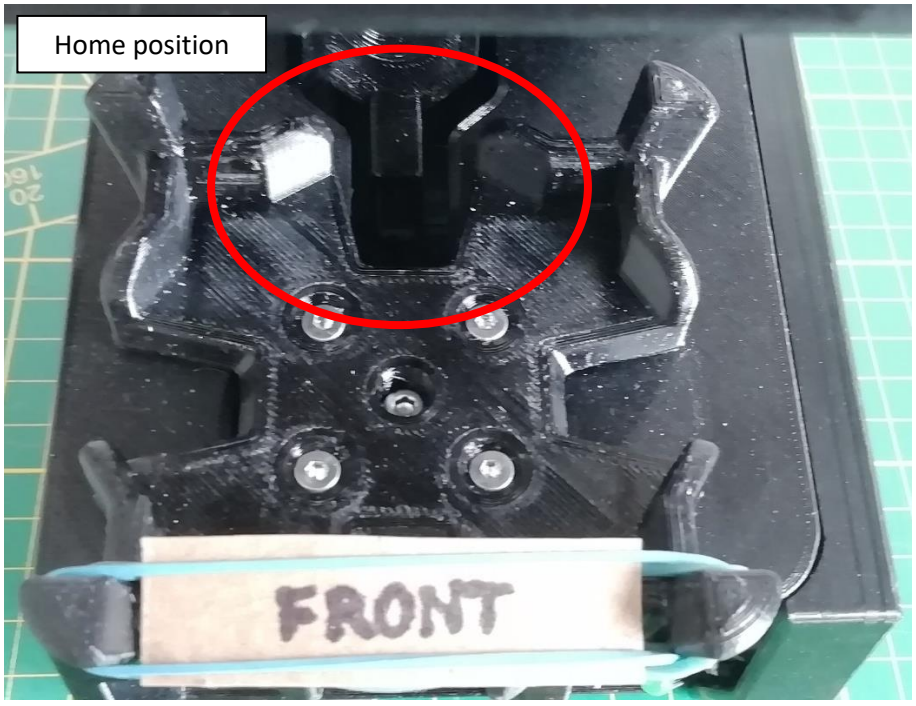
e. Rel from home: position(s) to release cube tension from Top_Cover at Home

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 againa 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:

- b_min_pulse_width
- b_max_pulse_width
- b_home

3. Fine tuning servos angles:

a. set the Servos to the mid angle (see above)
b. assemble the robot
c. enter the cube folder (*cd cube*) and activate the venv (*workon cv*)
d. edit *Cubotino_T_servo.py*: Search for 'demo' (it's at *__main__* fuction) and set the Boolean *demo=False*
e. run the script
f. at REPL (or terminal) enter manually the commands to test the servos positions:
   - t_servo.value = t_servo_close
   - t_servo.value = t_servo_open
   - t_servo.value = t_servo_read
   - t_servo.value = t_flip_up
   - b_servo.value = b_home
   - b_servo.value = b_servo_CCW
   - b_servo.value = b_servo_CW

   To adjust the Top_cover and/or the Cube_holder position, you might enter the value instead of the saved parameters.
g. Once the positions are satisfactory, edit the *Cubotino_T_servo_settings.txt* to apply the new settings
h. edit *Cubotino_T_servo.py*: Search for 'demo' (it's at *__main__* fuction) and set the Boolean *demo=True*
i. run the script: With demo=True the robot will use the servos like during a solving process; Take a close look (or better a movie) to check if the cube handling is ok.

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.
It will be convenient to use larger delays at the beginning, and progressively reduce them once the servo angles are adjusted to your system.
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.

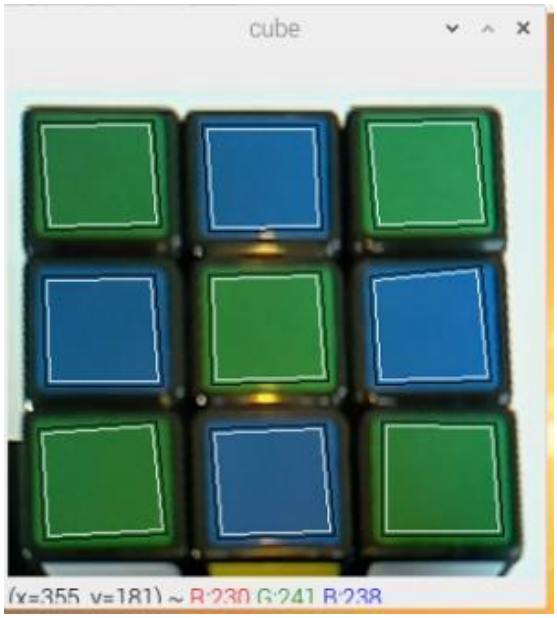
Image cropping has to be tuned, to reduce the image to the region of interest (ROI).

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. Display initialization:

The display is initialized twice at the start-up (once from Cubotino_T.py and once by Cubotino_T_servo.py).
Each time the display is initialized, all the pixels are set white for a second or more.
This is not a big issue, but in case you're a bit picky (like I am 😊…) and you'd like to make it nicer, you might consider to change a few parameters at ST7735 library:
Edit the *__init__.py* file (at */home/pi/.virtualenvs/cv/lib/python3.7/site-packages/ST7735)*

a) Comment out rows 178 and 179, that force the backlight during the display init part

Cubotino.py * | Cubotino_servos.py * | __init__.py

```
173         # Setup backlight as output (if provided).
174         self._backlight = backlight
175         if backlight is not None:
176             GPIO.setup(backlight, GPIO.OUT)
177             GPIO.output(backlight, GPIO.LOW)
178 #           time.sleep(0.1)                     # AF commented out
179 #           GPIO.output(backlight, GPIO.HIGH)   # AF commented out
180
```

b) Reduce the timers for the hardware display reset (rows 227, 229, 231)

Cubotino.py * | Cubotino_servos.py * | __init__.py

```
223     def reset(self):
224         """Reset the display, if reset pin is connected."""
225         if self._rst is not None:
226             GPIO.output(self._rst, 1)
227             time.sleep(0.050)           # AF reduced from 0.5 to 0.05
228             GPIO.output(self._rst, 0)
229             time.sleep(0.050)           # AF reduced from 0.5 to 0.05
230             GPIO.output(self._rst, 1)
231             time.sleep(0.050)           # AF reduced from 0.5 to 0.05
```

c) Reduce the timers for the sw display reset and sleep out (rows 237 and 240)

Cubotino.py | Cubotino_servos.py * | __init__.py

```
233     def _init(self):
234         # Initialize the display.
235
236         self.command(ST7735_SWRESET)    # Software reset
237         time.sleep(0.050)               # delay 150 ms        # AF reduced the delay from 0.15 to 0.05
238
239         self.command(ST7735_SLPOUT)     # Out of sleep mode
240         time.sleep(0.050)               # delay 500 ms        # AF reduced the delay from 0.5 to 0.05
241
```

I've tested the robot with above changes for more than one month, and I couldn't find bad functioning because of these changes.
The change on a) is for sure a very safe one, while the reduction of the timers might not work properly on all the display/display drivers; If you experience issues, after applying these changes, you might come back to the original settings and re-check.

23) Parameters and settings

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

- Cubotino_T_settings.txt
- Cubotino_T_sservo_settings.txt

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

- Cubotino_T_settings_AF.txt
- Cubotino_T_sservo_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 little information.

Cubotino_T_settings.txt (and Cubotino_T_settings_AF.txt):

Parameter (dict key)	**Default value**	**AF value**	**Data type**	**Info**
disp_width	130	132	Int	Display width (in pixels)
disp_heigh	160	162	Int	Display height (in pixels)
disp_offsetL	0	-4	Int	Display offset on width Left (in pixels)
disp_offsetT	0	-2	int	Display offset on height Top (in pixels)
camera_width_res	640	640	Int	Picamera resolution on width
camera_hight_res	480	480	int	Picamera resolution on height
kl	0.95	0.95	float	Coefficient for PiCamera stability acceptance. Lower values are more permissive (range 0 to 1).
x_l	0	60	Int	Image cropping at the left, before warping (in pixels)
x_r	0	80	Int	Image cropping at the right, before warping (in pixels)
y_u	0	0	Int	Image cropping at the top, before warping (in pixels)
y_b	0	110	int	Image cropping at the bottom, before warping (in pixels)
warp_fraction	7	7	float	Image warping index. Larger values increase the warping)
warp_slicing	1.5	1.5	float	Image cropping index at bottom (black bandwidth) after warping. Larger values reduce the cropping
square_ratio	1	1	float	Facelet contour squareness check filter. Larger values are more permissive (0 is perfect square).
rhombus_ratio	0.3	0.3	float	Facelet contour rhombus check filter. Smaller values are more permissive (1 is perfect Rhombus)
delta_area_limit	0.7	0.7	float	Facelet area deviation from median. Larger values are more permissive (0 means no deviation)
sv_max_moves	20	20	int	Max number of moves requested to the Kociemba solver
sv_max_time	2	2	float	Timeout, in seconds, for the Kociemba solver
collage_w	1024	1024	int	Image width for the unfolded cube file
marg_coef	0.1	0.06	float	Cropping margin (%) around the faces images to generate the unfolded cube collage
cam_led_bright	0.1	0.1	float	PWM for the 3W led at Top_cover. Range 0 (no PWM) to 1 (PWM=100%)
detect_timeout	40	40	int	Timeout, in second, for the cube status detection
show_time	7	7	int	Time, in seconds, to keep showing the unfolded cube image on screen
warn_time	1.5	1.5	float	Time from touching the touch button (after 0.5s filter), after which a warning appears on display.
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.

Cubotino_T_servo_settings.txt (and Cubotino_T_servo_settings_AF.txt)

Notes:

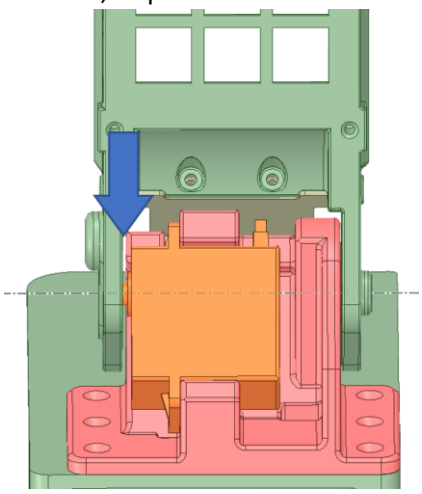
1. 't_' refers to Top_servo while 'b_' refers to Bottom_servo
2. Angles are in gpiozero range for the Servo class (range from -1 to 1, with 0 as mid angle)
3. Time is in seconds

Parameter (dict key)	**Default value**	**AF value**	**Data type**	**Info**
t_min_pulse_width	1	1	float	Min pulse width, in ms
t_max_pulse_width	2	2	float	Max pulse width, in ms
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	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
b_max_pulse_width	2	1.98	float	Max pulse width, in ms
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	float	Delta angle for the Cube_holder to retract from CCW or CW
b_extra_home	0.13	0.13	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 contrained)
b_rotate_time	0.8	0.55	float	Time for the Cube_holder to rotate ~90° (cube contrained)
b_rel_time	0.2	0	float	Time for the Cube_holder rotate back, at CCW, CW and Home

24) Troubleshooting

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

1. Servos don't move smoothly
    1. Don't use jumper wires, or use quality jumper wires
    2. Don't use bread boards, make the 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.

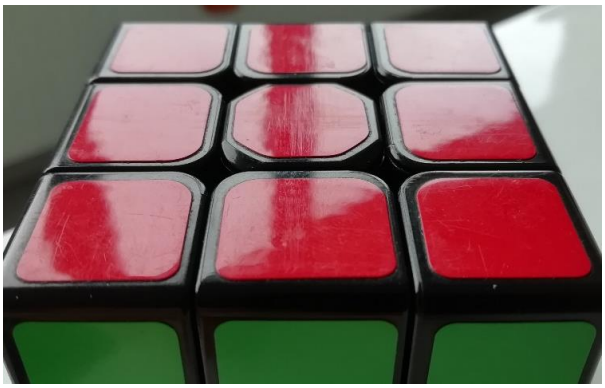
2. Bottom cube layer doesn't align nicely:
    1. Verify if the cube Holder makes an extra rotation, at both CCW and CW directions, before stopping; If this doesn't happens:
        i. Increase the timers, as too small time don't give sufficient time to the servo to make the stroke visible when testing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

3. 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.

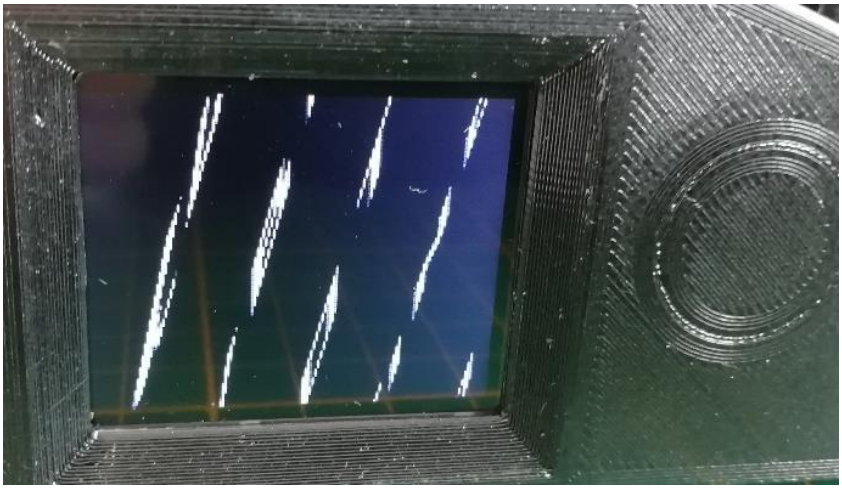
4. Cube detection error:
   It is returned when the interpreted cube status isn't coherent, meaning not having 9 facelets per colour or other inconsistencies. Possible causes:
   1. Table background; Objects on the table can form square like contour, interpreted as facelets by the cv. This can be solved 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. In case the cube has some prints (i.e. brand) , typically on the white center, it is suggested to carefully scratch out.

5. 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)



6. 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.

7. Program doesn't work as intended:
   This is a difficult topic, as my coding skills are rather limited …..
   A good starting point is to get some 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 seem 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_servo.py .This code activate the servos as 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

25) 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)

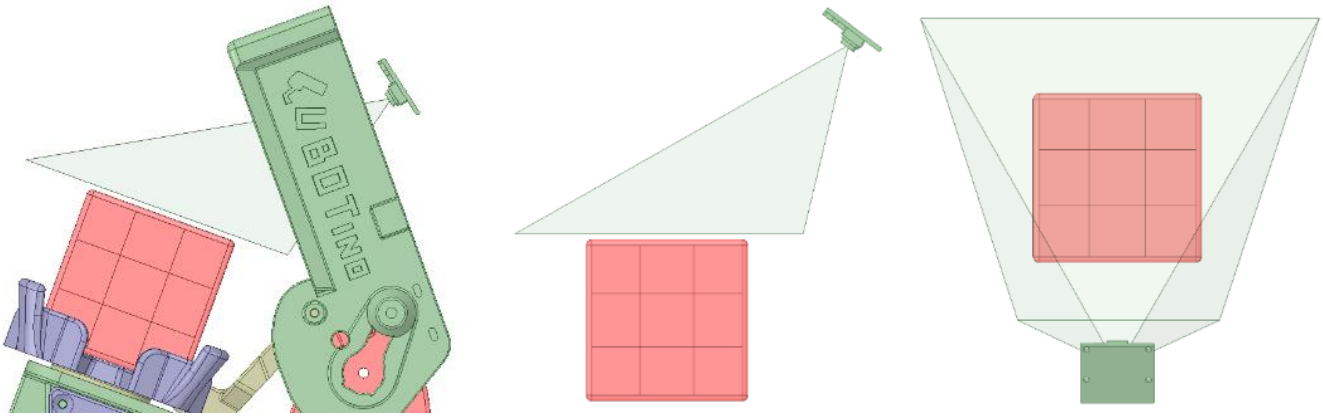
In which the python script '**Cubotino_T.py'** is responsible for the interaction with these elements.

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

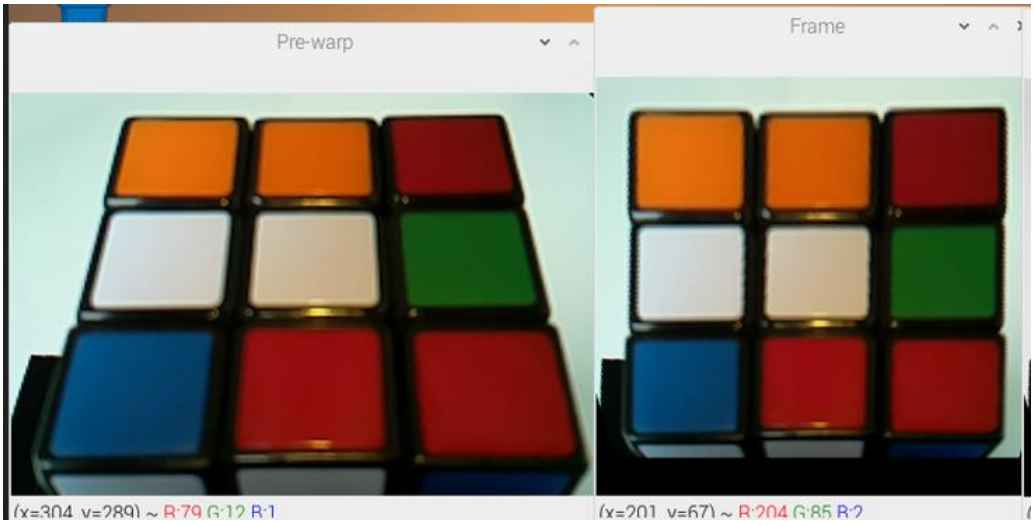


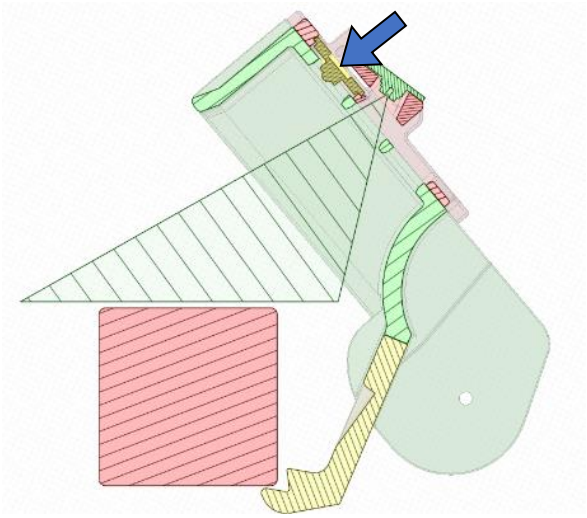
Image warping requires some computation power, but It does not affect the overall timing.

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:

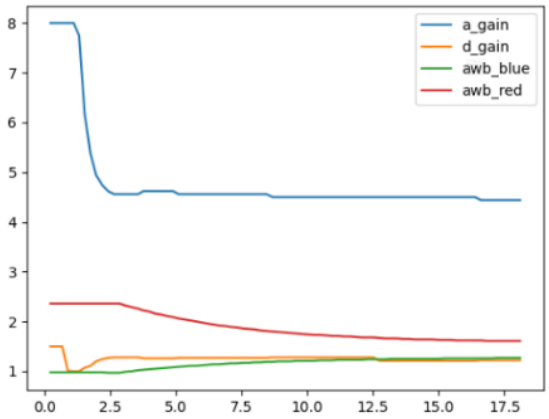


Notes on the chosen LED module:
It has 120-140 deg angle
It is controllable by the MCU
Used a fraction of the 3W

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 Exposition 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 exposition 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 exposition 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 (GaussianBlur), to reduce noise: *blurred = cv2.GaussianBlur(gray, (9, 9), 0)*
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, 10, 30)*
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, kernel, iterations = 4)*

   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, kernel, iterations = 2)*

   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.
6. The 'accepted' contours are plot over the coloured warped image, as visual feedback.

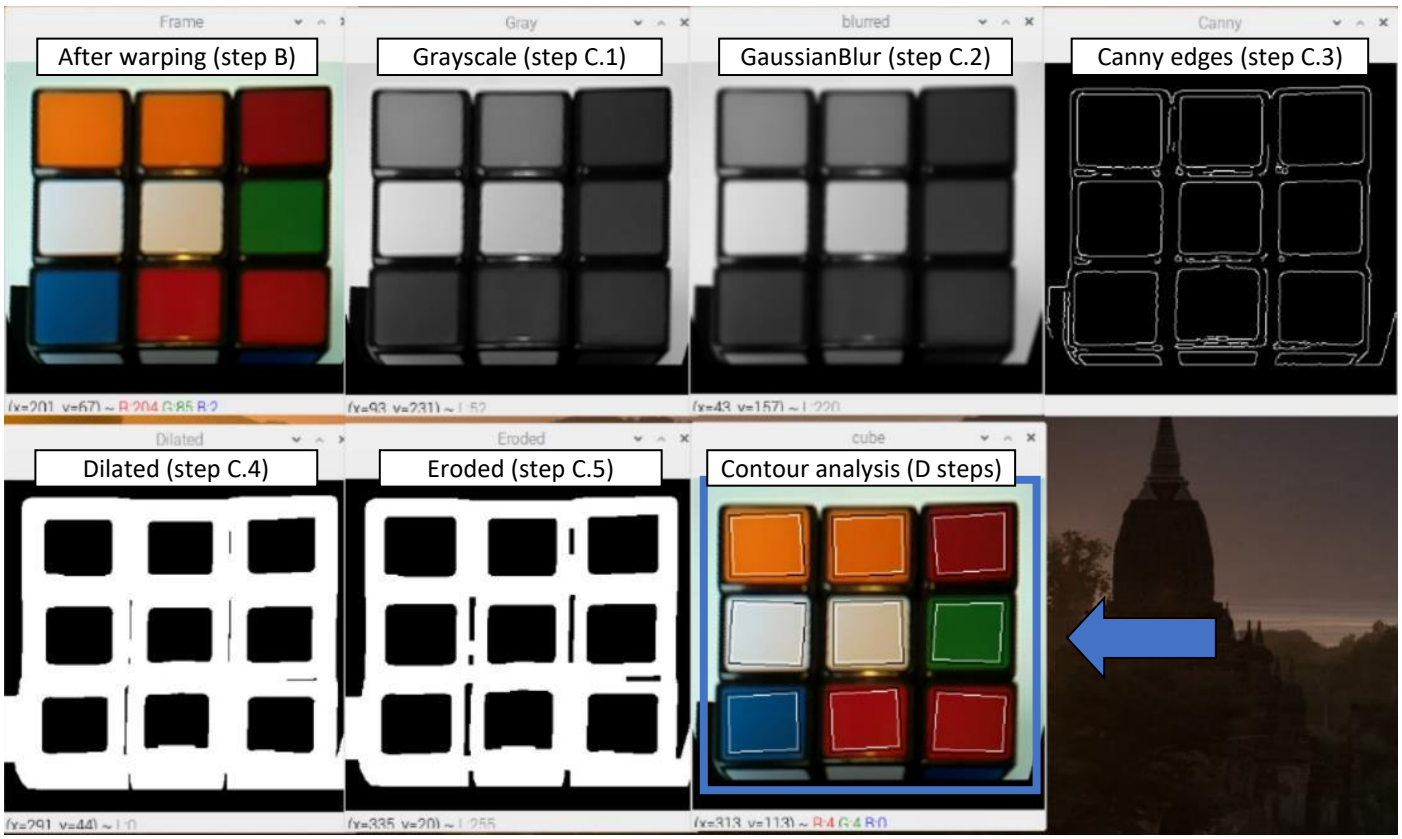
E. Colours retrieved:

On each facelet, are the retrieved 2 main info:

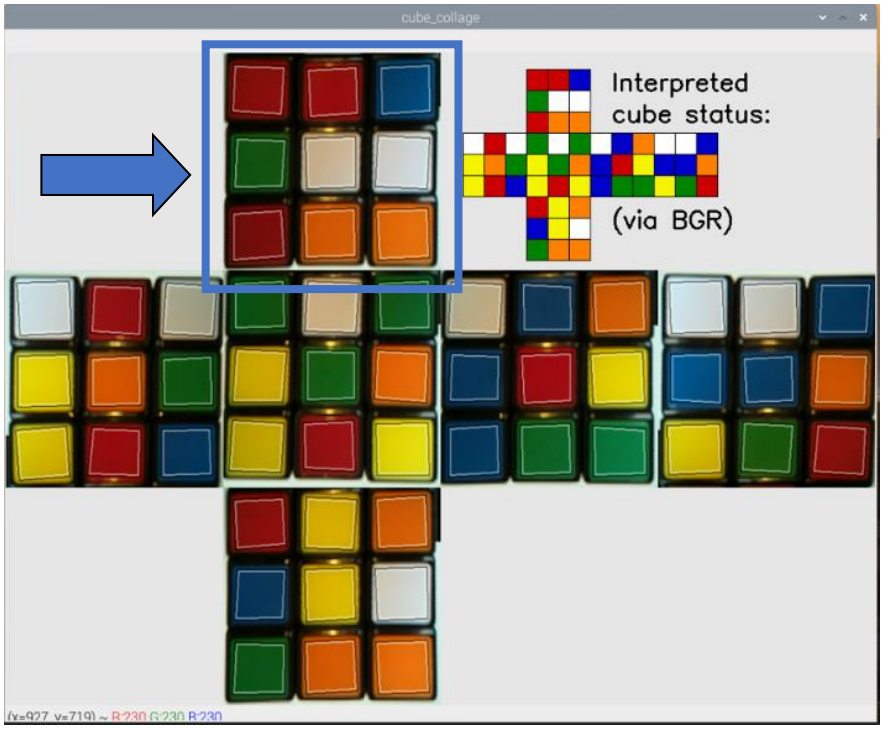
1. Average BGR, for a portion of the facelet around the detected contour center.
2. Average HSV, based on the average BGR.

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
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!

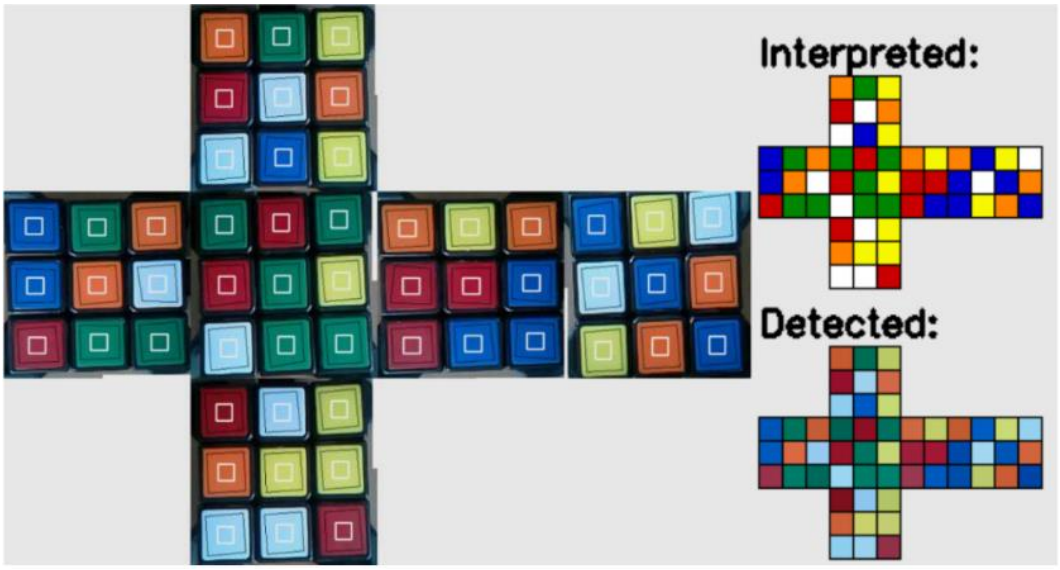
Notes:
Light reflection drastically affect the colour interpretation.
The accepted contour is on the very low contour acceptable area, yet sufficiently large to don't be noise

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

26) 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. Proterties 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.

7. The string '#(AF ' is placed as comment start, on:
   1. Variables requiring to be tuned, because each robot will be slightly different from the others:
      - Servo angles, and servo timers
      - Frame Cropping, as this if Top_cover angle dependent and PiCamera assembly angle dependent
      - On next revision, I'll list all the variable I expect necessary to be tuned
   2. Variables you might want to play with

27) 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.

28) 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_servo.py* scripts use parameters with settings from two json files; This choice to group the parameters 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
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)

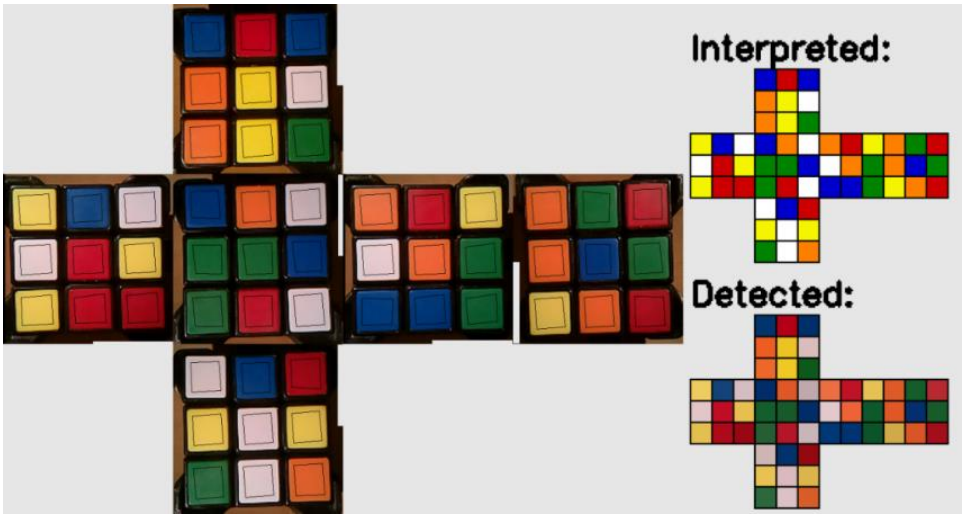
Notes:

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

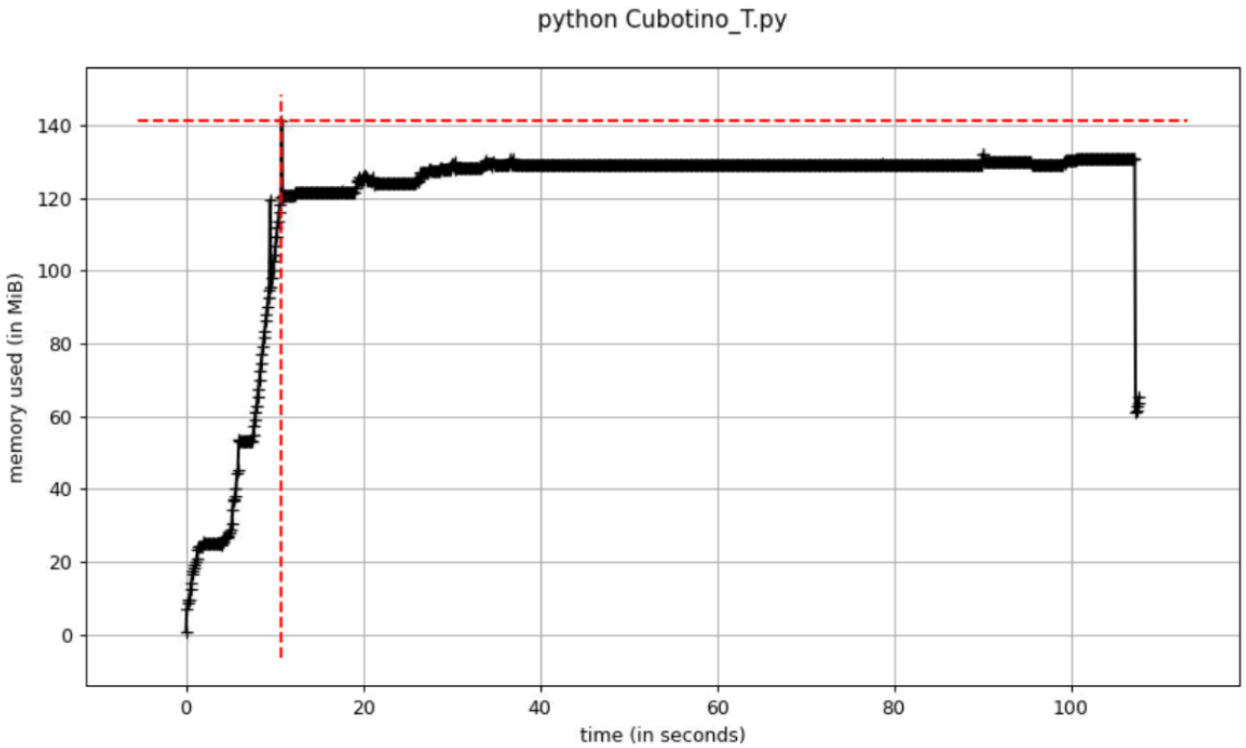


10. Date, and especially time, are used by the robot:
    Raspberry pi doesn't have an integrated RTC, therefore when the robot isn't connected to a PC and/or internet, this info could be inaccurate.
    If the robot establishes a connection to the Wi-Fi, the system time gets updated, yet this will alter the robot time calculation if the update comes when the robot is solving a cube.
    To prevent this problem from happening, the robot script checks at the start-up if there is an internet connection, and in that case, it waits until the system time is updated before proceeding.
    In case there aren't internet connections, the robot simply proceeds with the non-updated system time.
    In my view this approach is sufficient for reliably timing the robot performances.

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 $
```

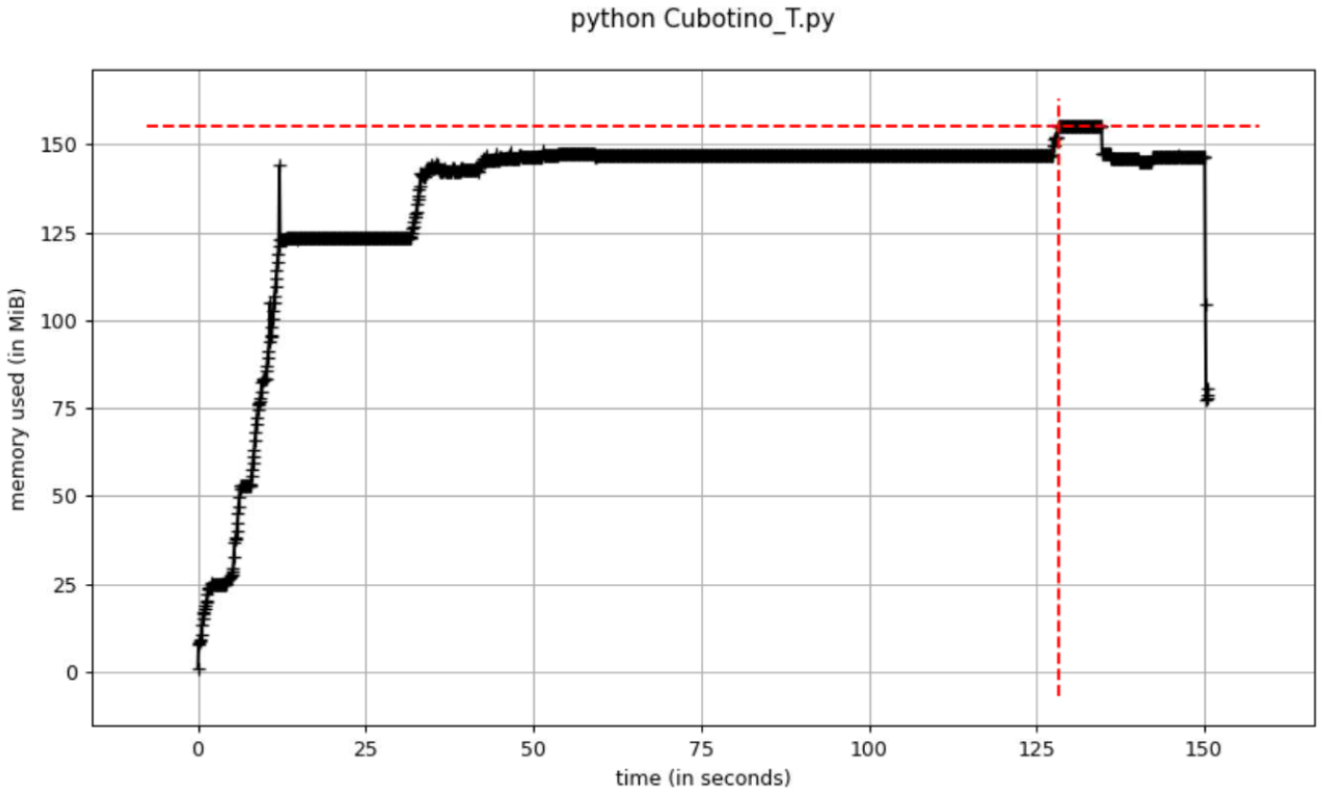
    Without VNC Viewer there is somehow lower memory usage; Below plot includes a full cycle:
    - import libraries
    - read and solve a scrambled cube
    - quit the script



    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.

29) 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.
- to Jacques, who triggered me on the colours sensors direction.
- to Richard, for his keen check of this document and precious feedbacks.
- all the people who have provided feedbacks.

30) Revisions

Rev	Date	Notes
0	01/06/2022	First release. Coming days I'll further complete this document.

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_servo.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 • Doc revision
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

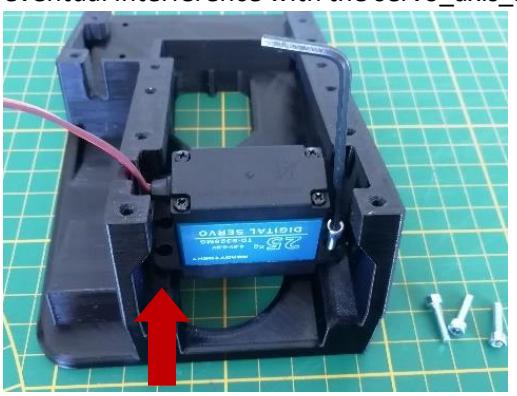
31) Assembly details:

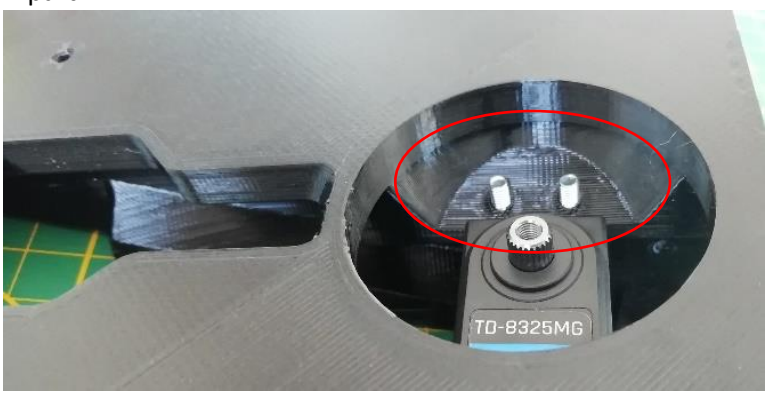
Step4 (Mount the bottom servo to the structure):
4x M3x12mm cylindrical head
Couple of washers

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

Couple of notes:

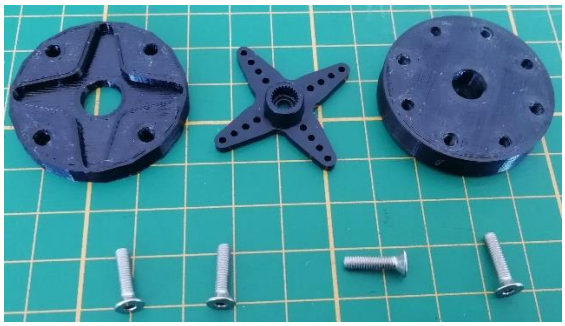
- Before tightening the four screws, checks if the servo output gear is well centred to the Structure hole.
- To limit the protrusion of the below two screws, add a couple of washers to these screws; This to prevent eventual interference with the servo_axis_inf part.



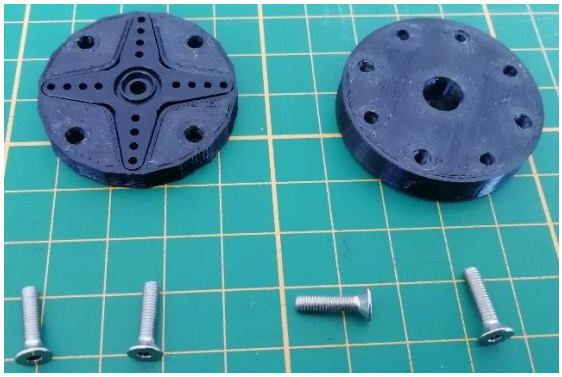


Step5 (sandwich Servo_axis_inf / servo arm / Servo_axis_inf):

4x M3x12mm conical head

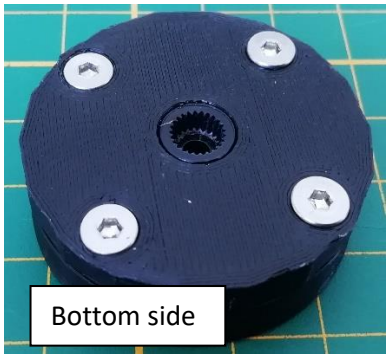


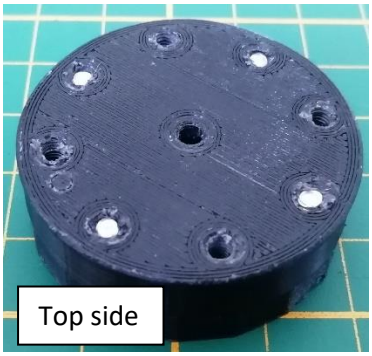
Check if the supplied X arm fits with the indentation of Servo_axis_inf part:



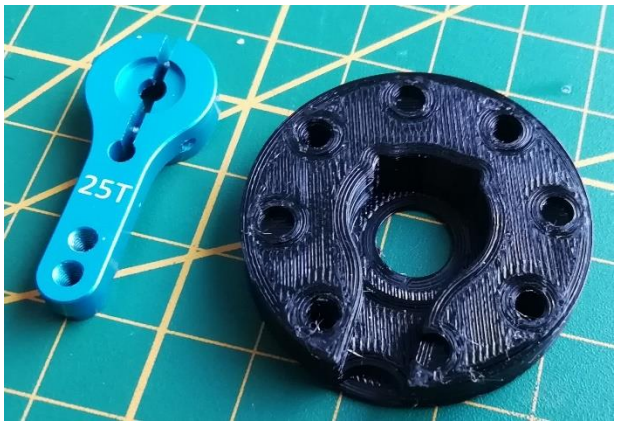
If you don't have a X arm to make it fit, please print the alternative parts (Servo_axis_inf and Servo_axis_sup) designed to fit the aluminium "T25" arm (arm has to be reduced in length).

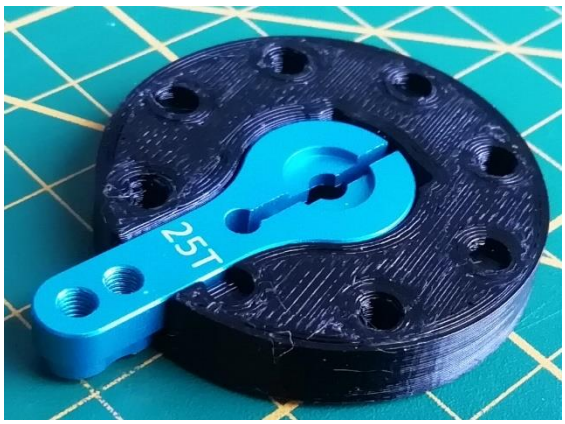
Make the sandwich





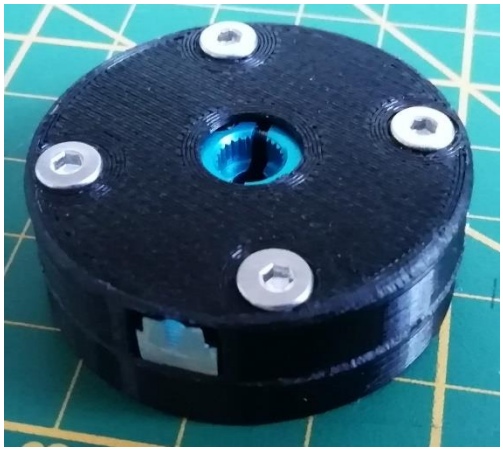
Step5a Alternative servo axis assembly:





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

Make the sandwich as per Step5
4x M3x12mm conical head



Step6 (Assemble the Cube_holder to Servo_axis assembly):

4x M3x12mm conical head



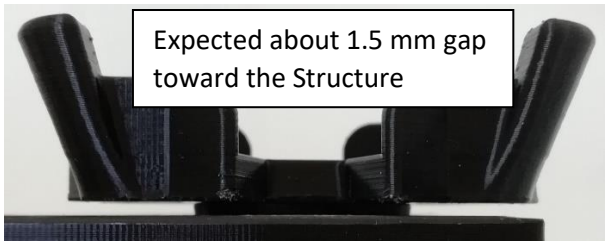
Step7 (Assemble the Cube_holder assembly to the bottom servo):

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

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

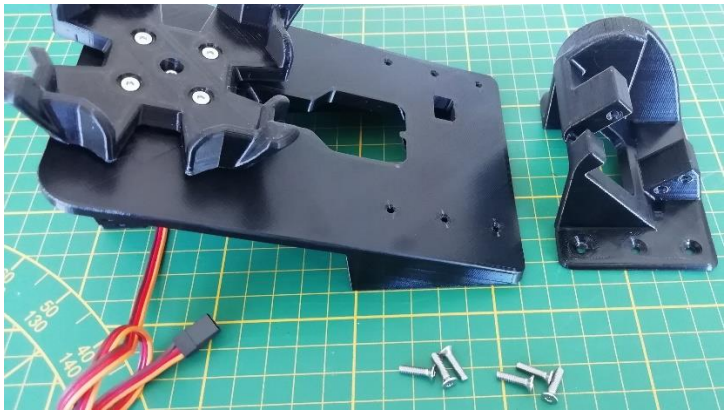

1x M3x12mm cylindrical head

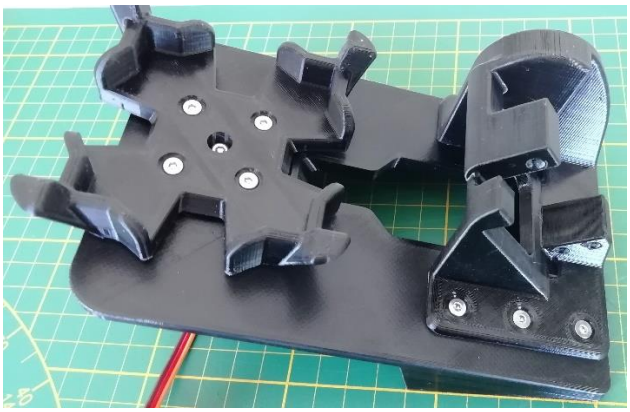




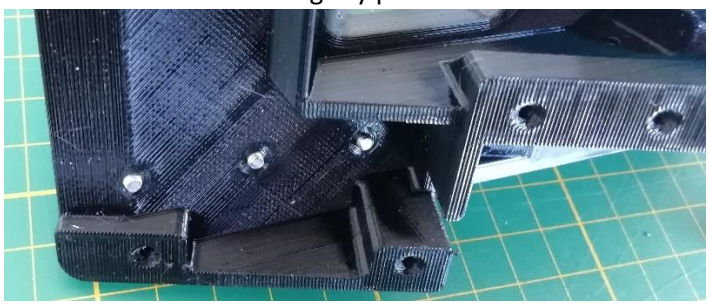
Step8 (Assemble the Hinge to the Structure):

6x M3x12mm conical head



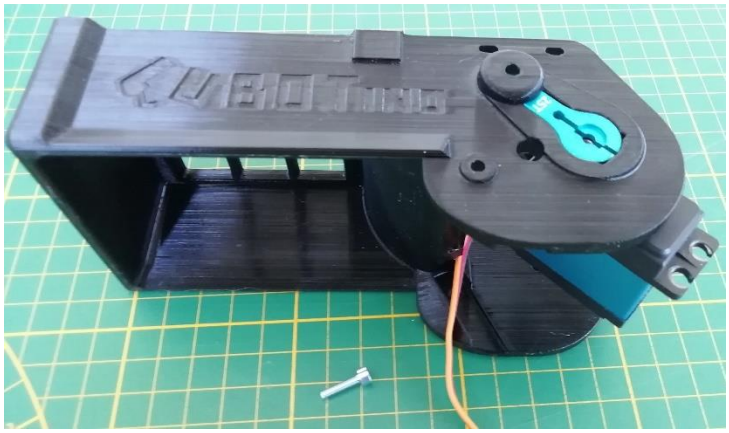


Note: three screws will slightly protrude underneath the Structure; If screws of 12mm then this won't be a problem.



Step9 (Assemble the "T25" servo arm to the upper servo):
Place the "T25" arm along the main Servo axis (Servo should be prepared upfront with the gear at middle angle).
Close the two arm tiny screws

Step10 (Insert the Top_servo assembly into the Top_cover slot):
1x M3x12mm cylindrical head
The slot for the “T25” arm might be tight; Remove eventual excess of material if needed
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

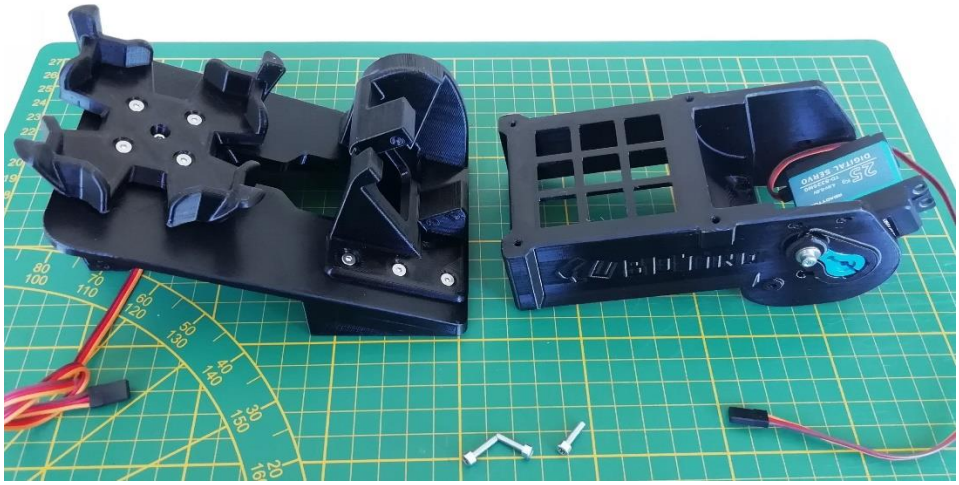
Rotate the servo by about 45deg, to facilitate next steps

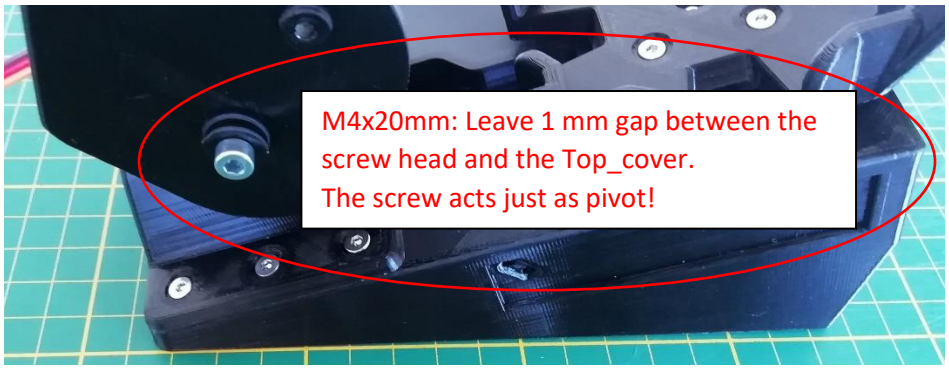
Step11 (Assemble the Top_cover assembly to the Hinge):

1x M4x20mm cylindrical head

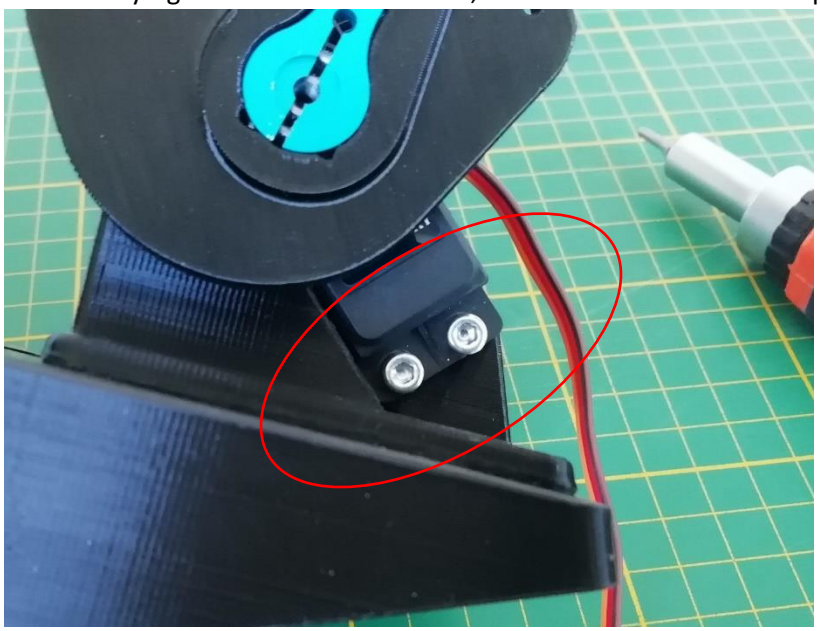
3x M3x12mm cylindrical head

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



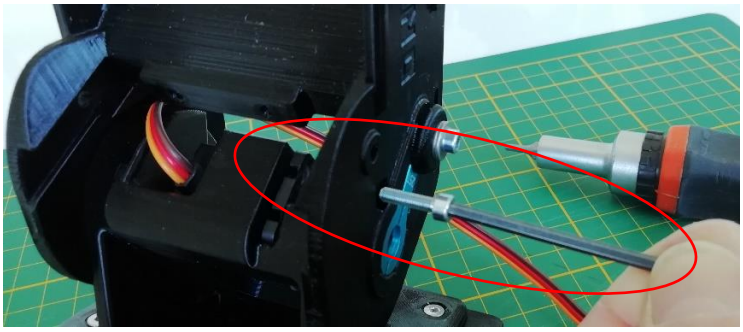


Do not fully tighten the two M3x12mm, until also the third screw is positioned

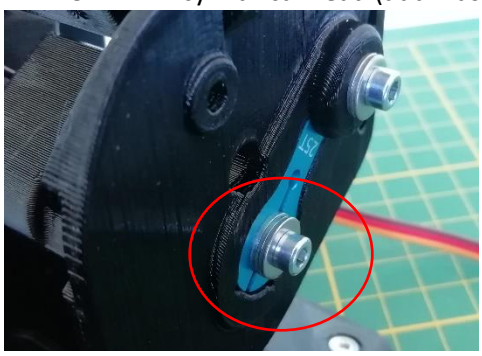


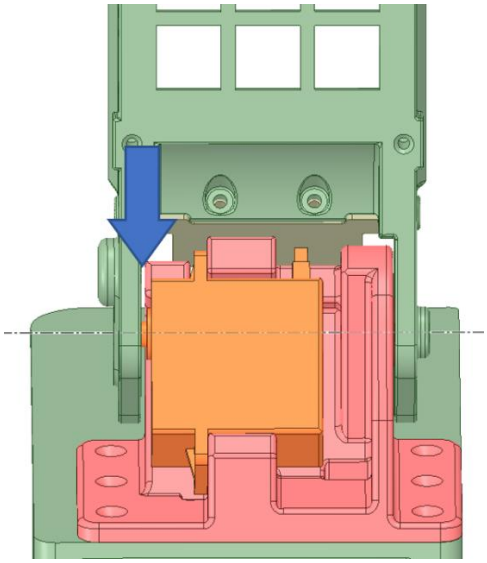
Step12 (Complete the top servo assembly to the Hinge):

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)





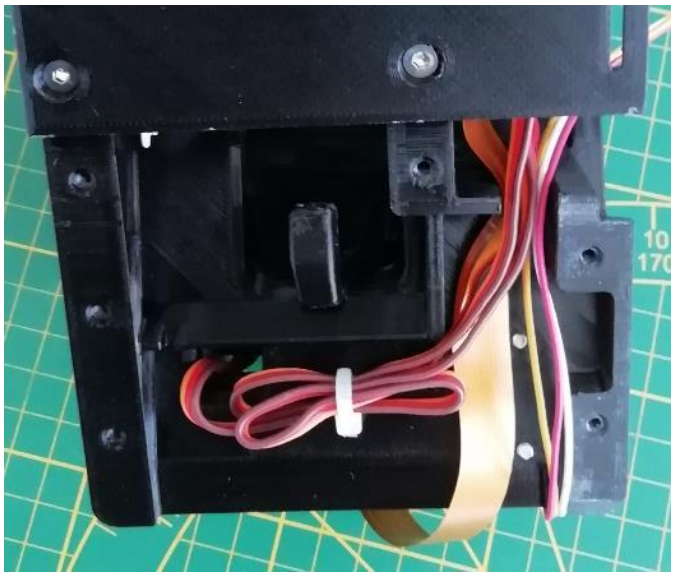
Verify gap presence in between the Top_cover and the Hinge, at the servo output gear side (arrow at the side).
In case there is little or no gap, unscrew the M3 screws of the servo, and place some little spacers (0.5 to max 1mm) in between the servo and the Hinge (preferably close to the screws locations);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

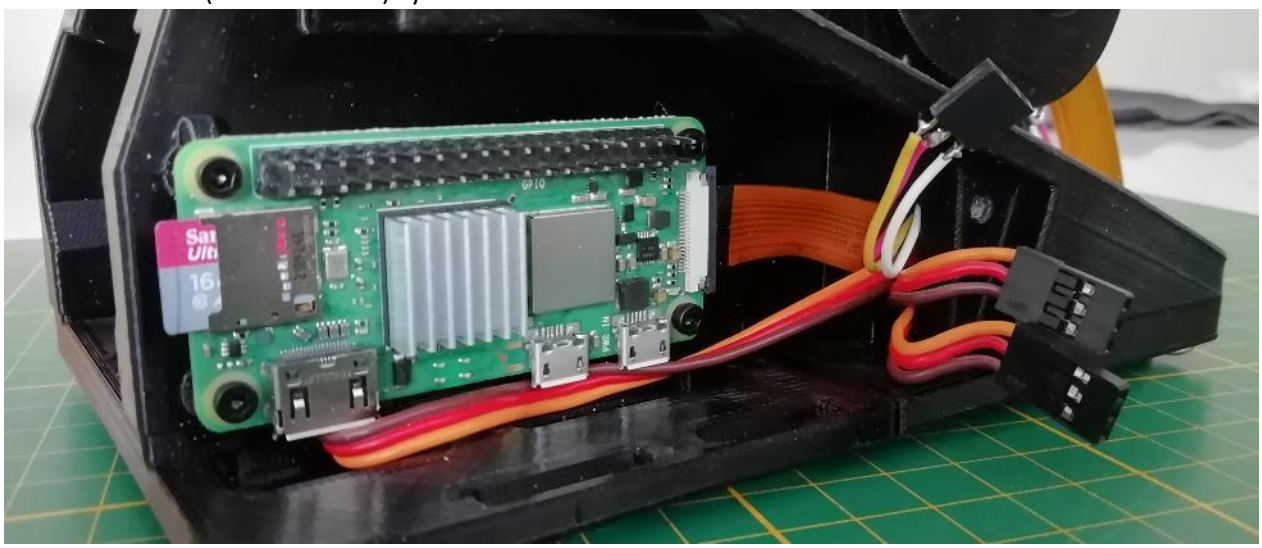
Slide the Lifter into the Top_cover slots

Step14 (Position the cables, and assemble the Baseplate_rear):
6x M3x12mm conical head (4 screws at corners are sufficient)

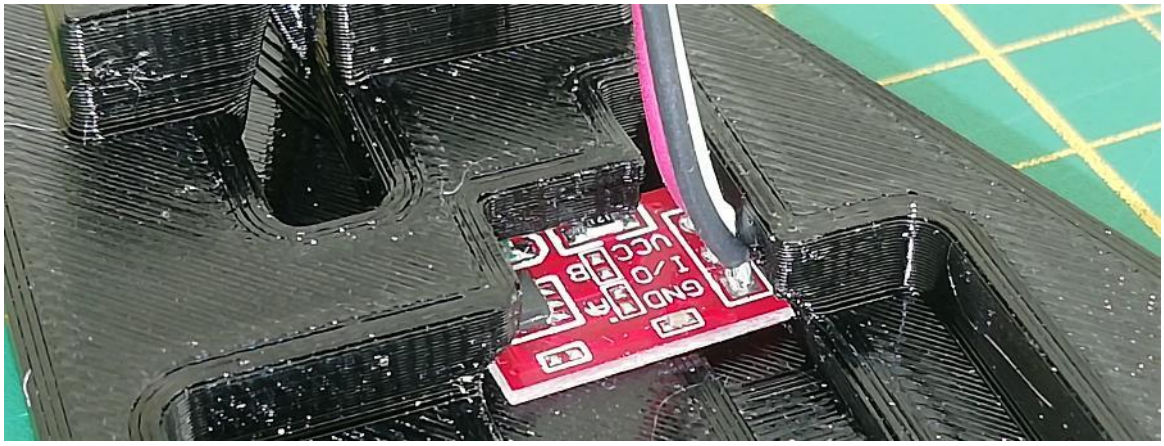
Note: It will be convenient to ‘store’ the excess of cable length as per below picture, as there is very little space left at the board location
Dress the cables and close the Baseplate_rear.

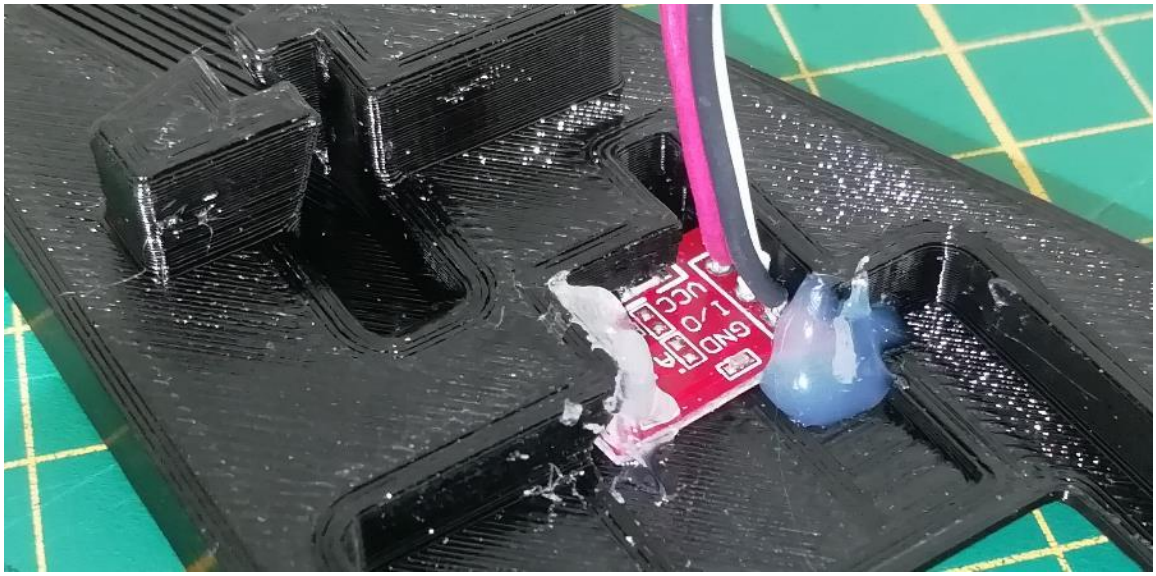


Step15 (Connect the PiCamera flex cable to Raspberry Pi Zero2 board, and fix it to the Structure)
4x M2.5x10mm (or M2.5x8mm) cylindrical head

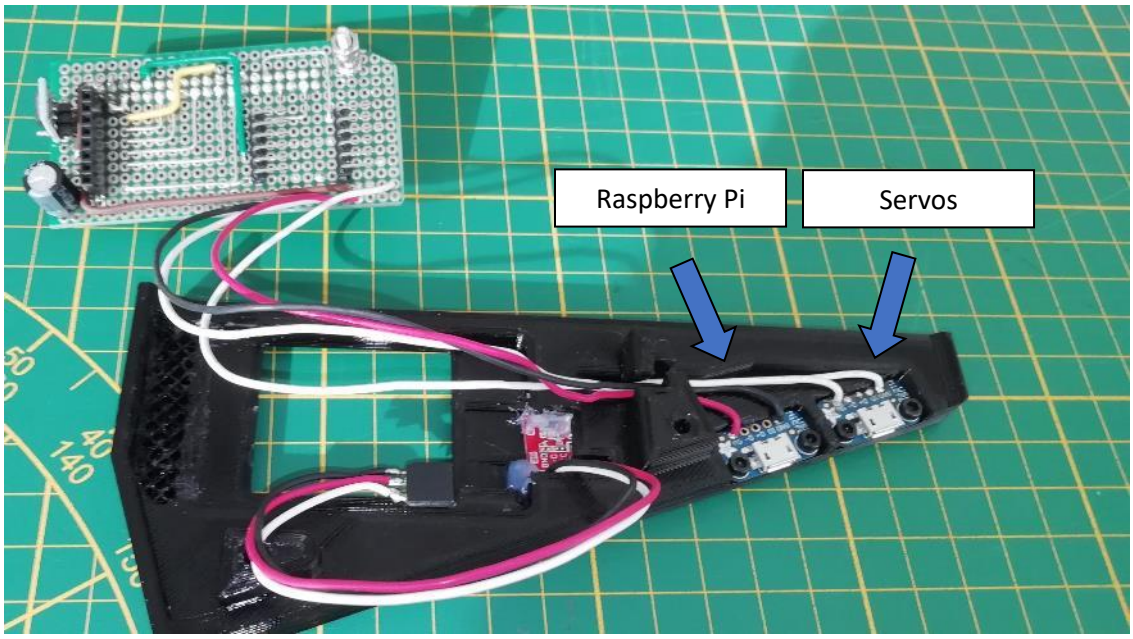


Step16 (Fix the Touch_sensor to the PCB cover):
Inser the board as per picture, and add a couple of hot glue droplets

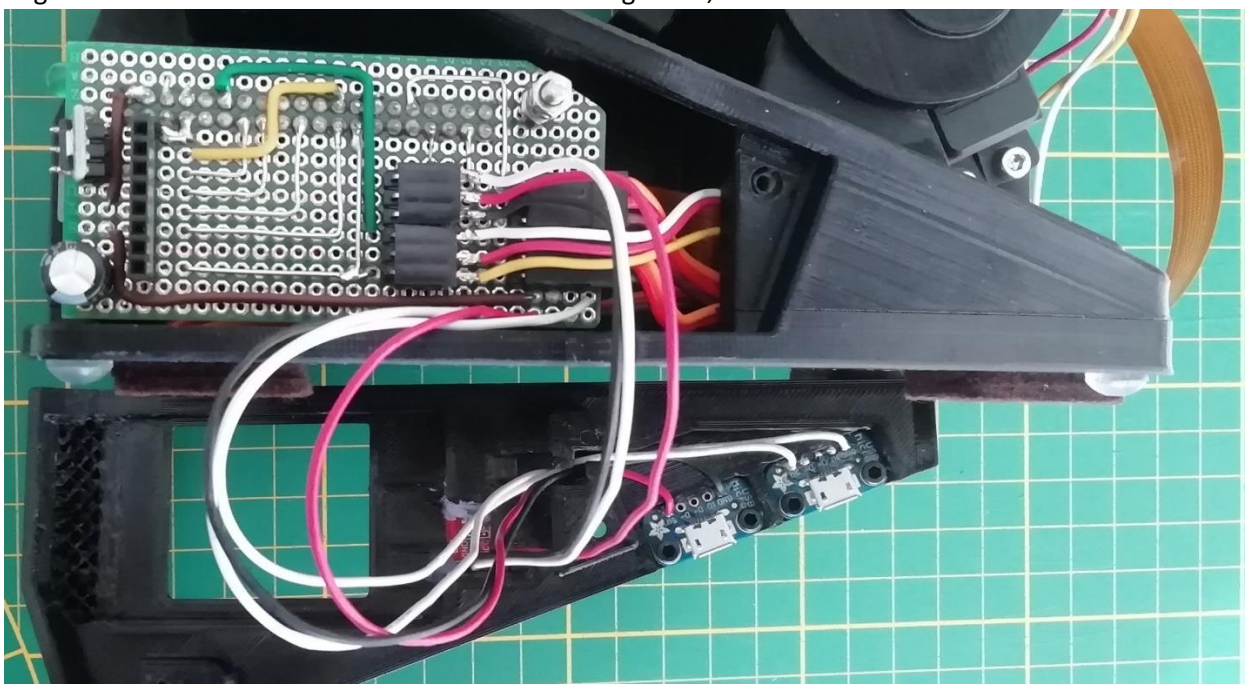




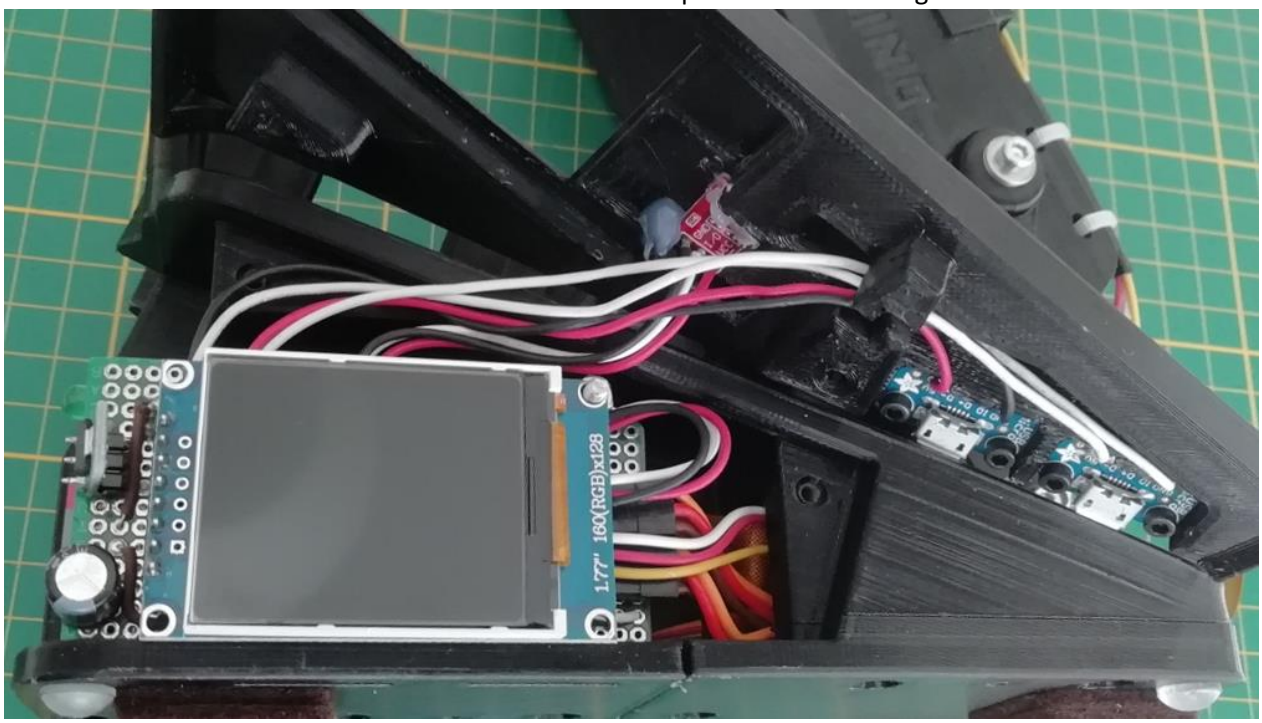
Step17 (Fix the microUSB breakout board to the PCB_cover):
2x M2.5x4mm cylindrical head
The order of the two boards is not critic, below just a proposal



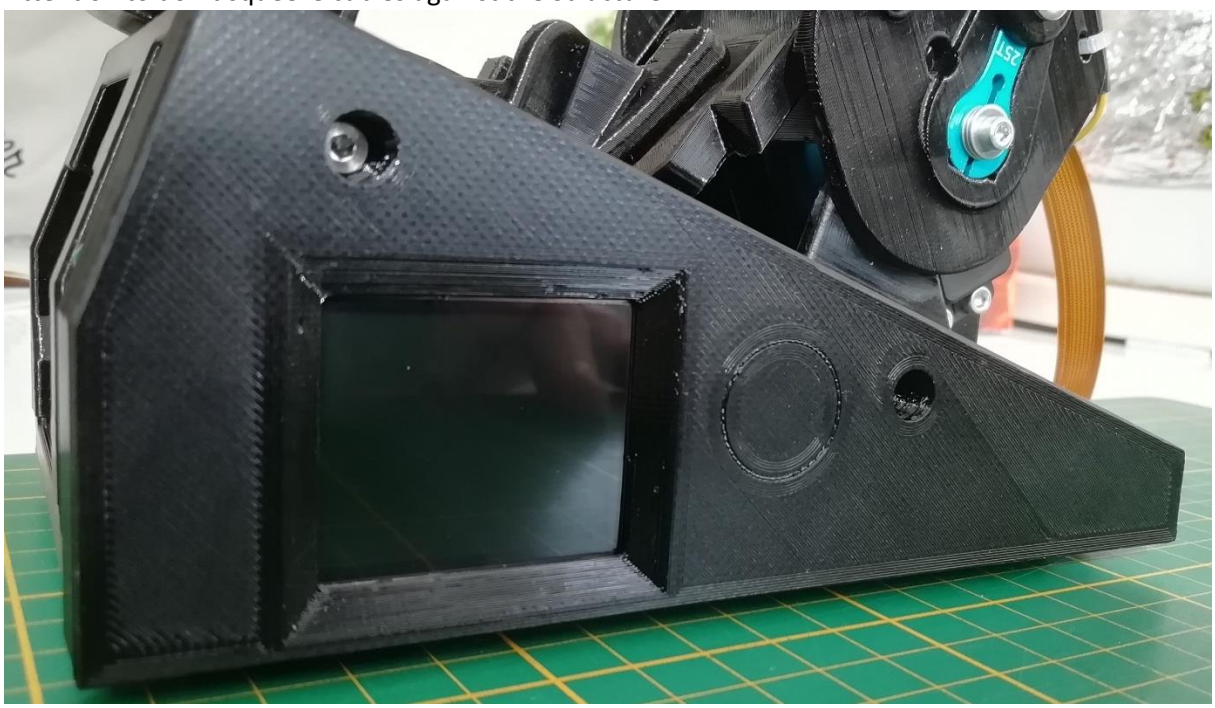
Step18 (Connect the servos, LED breakout board and Touch sensor):
In general the brown wire of servo connector is the ground, therefore to be oriented toward the bottom



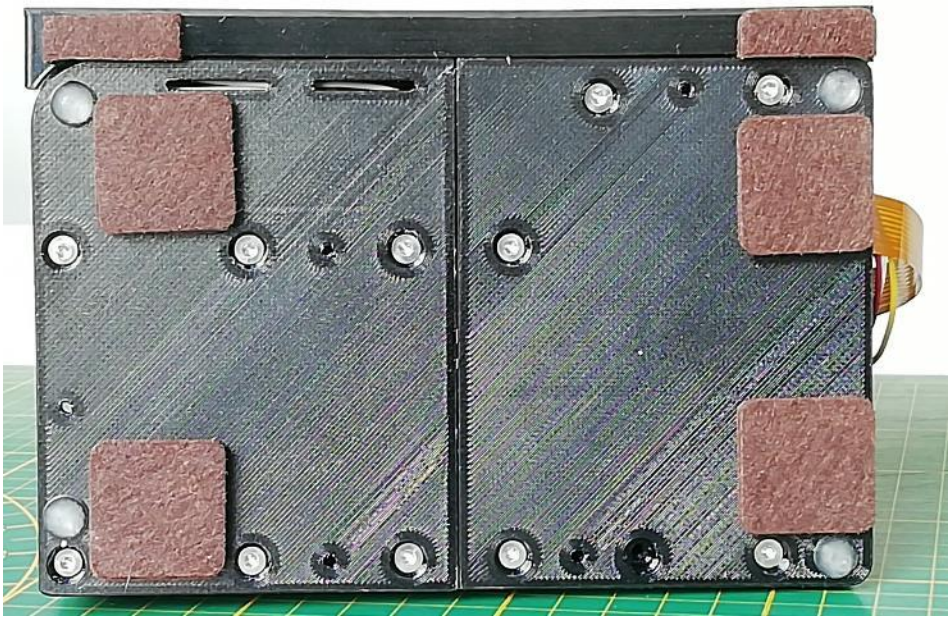
Step19 (dress the cable and connect the display):
Cables at the microUSB breakout boards have to be well positioned into the groove



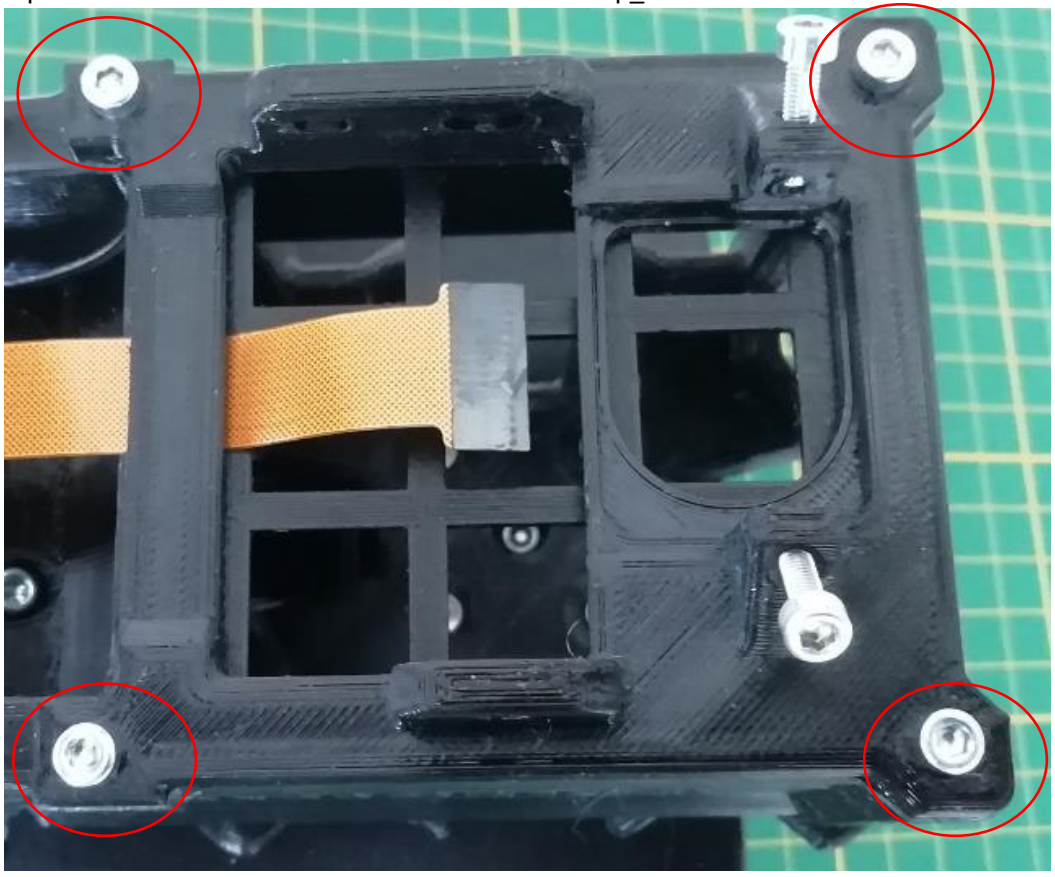
Step20 (Assemble the PCB_cover):
2x M3x12mm conical head
Attention to don't squeeze cables against the Structure



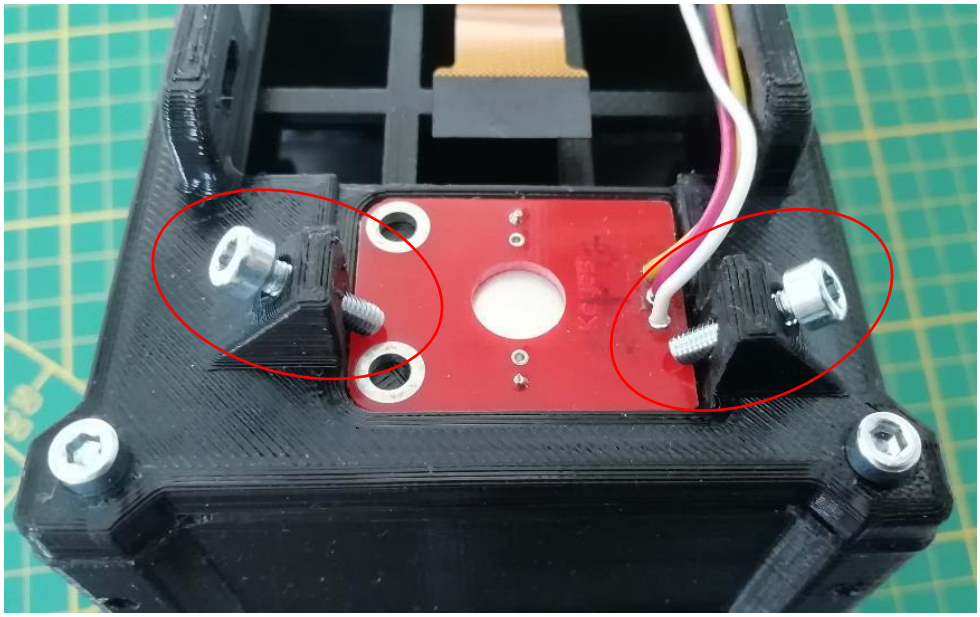
Step21 (Assemble the Baseplate_front to the Structure):
6x M3x12mm conical head (at the bottom, 4 screws at corners are sufficient)
Stick self-adhesive rubber feet (or felt pads) to the baseplate; Those at the back should be placed as close as possible to the corners



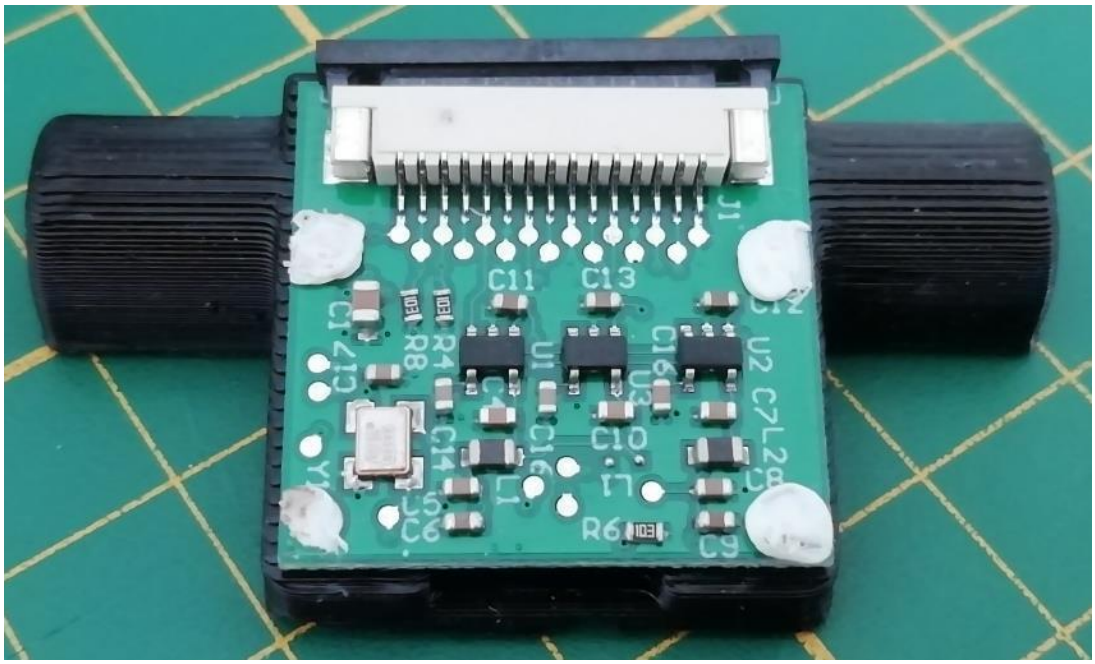
Step22 (Assemble the PiCamera holder frame to the Top_cover):
4x M3x12mm cylindrical head
Squeeze the PiCamera flex cable in between the Top_cover and the PiCamera holder frame



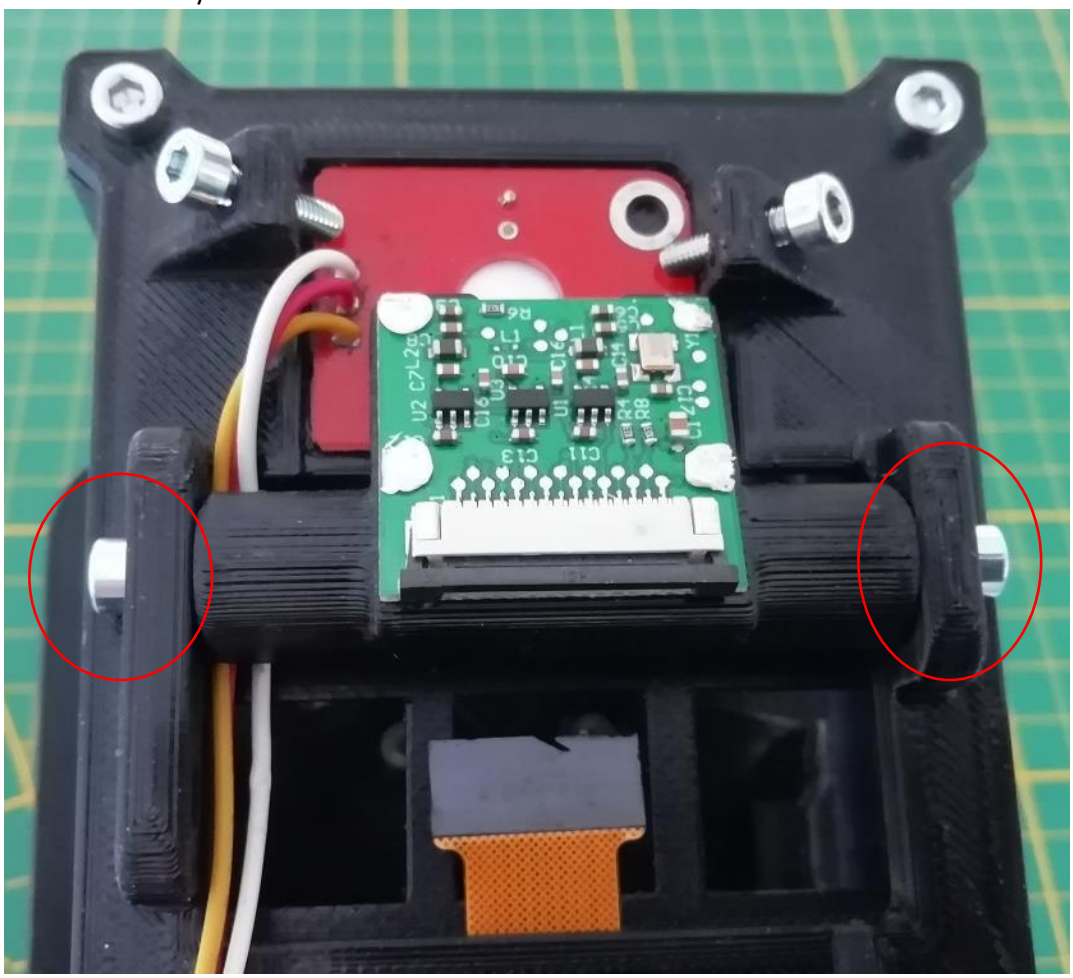
Step23 (Assemble the LED breakout board):
Add 2x M3x12mm cylindrical head
Because of limited space, it will be convenient to solver the wires instead of the connector at the board.
Stop screwing once the screw tip touches the board



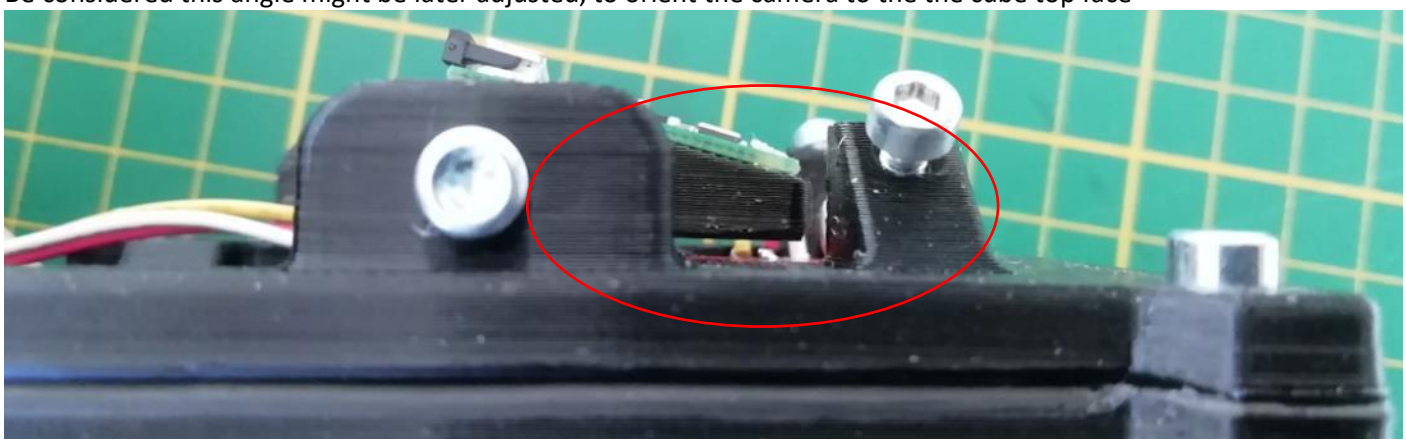
Step24 (Assemble the PiCamera to the PiCamera holder):
Necessary some little pieces of filament Ø1.75mm.
Force 4 pieces into the holder, slide the board over, with hot blade deform the filament:



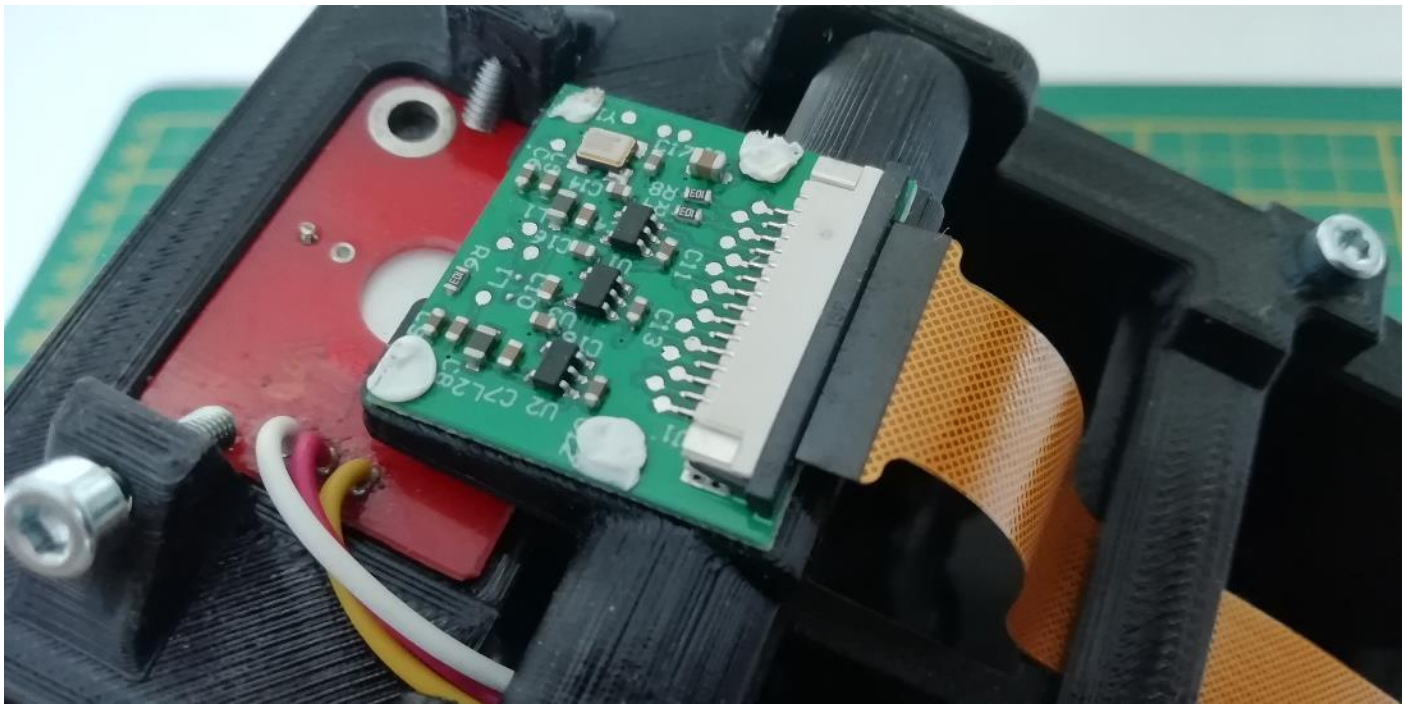
Step25 (Assemble the PiCamera holder to the PiCamera holder frame):
2x M3x12mm cylindrical head



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



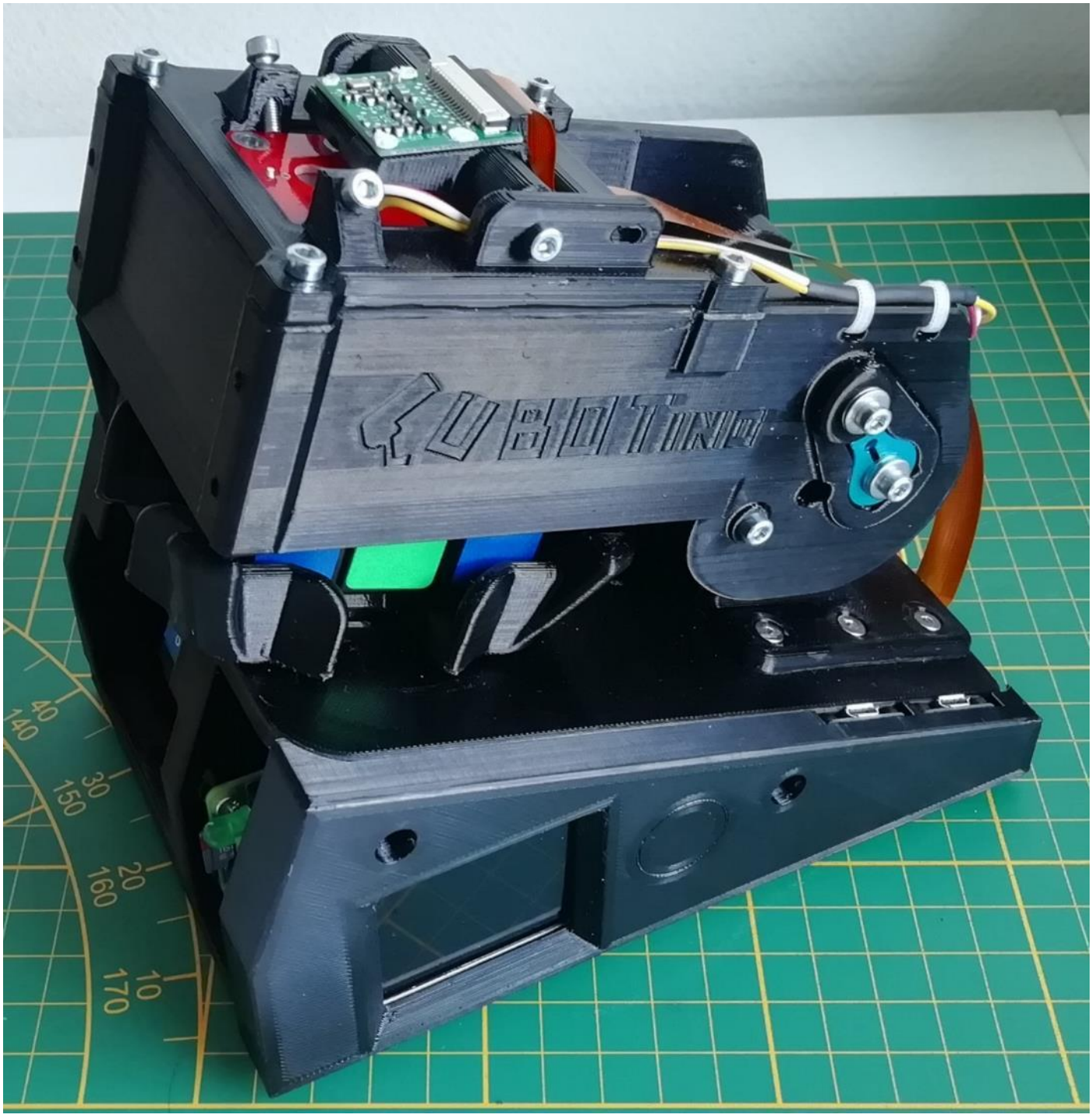
Step26 (Connect the flex cable):

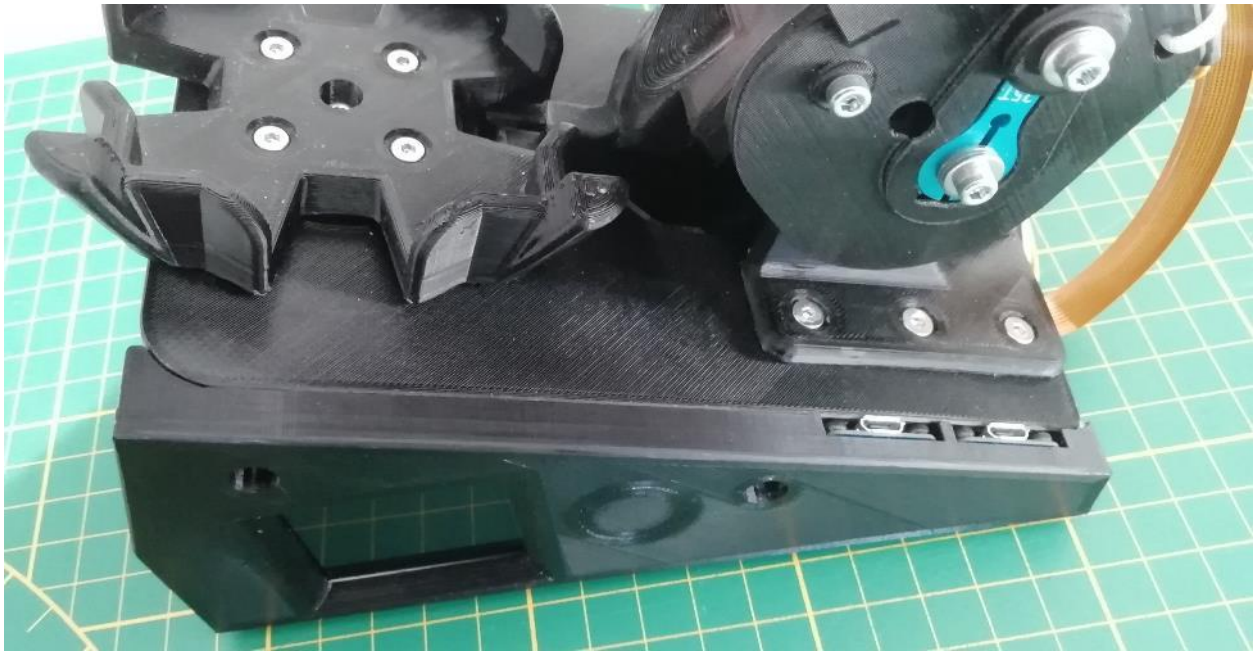


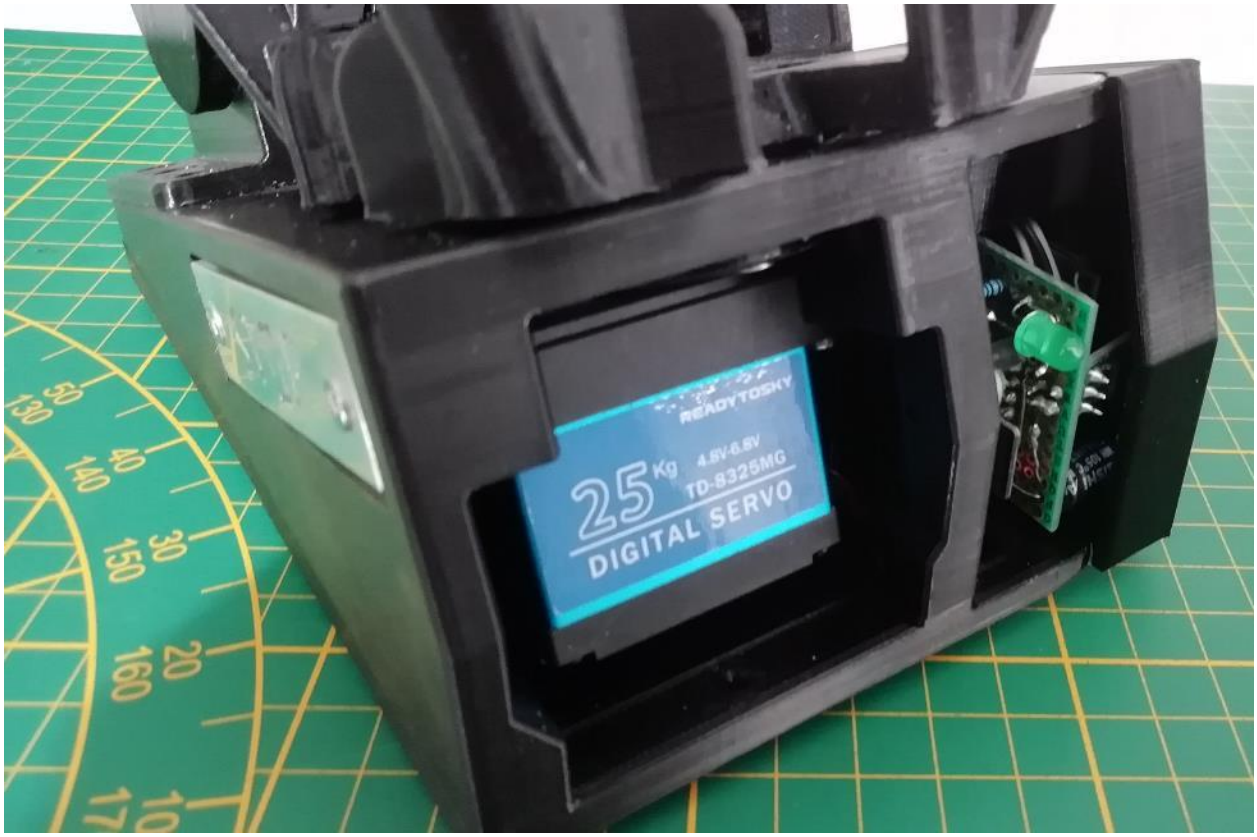
Step27 (Personalize the formal Stop plate):

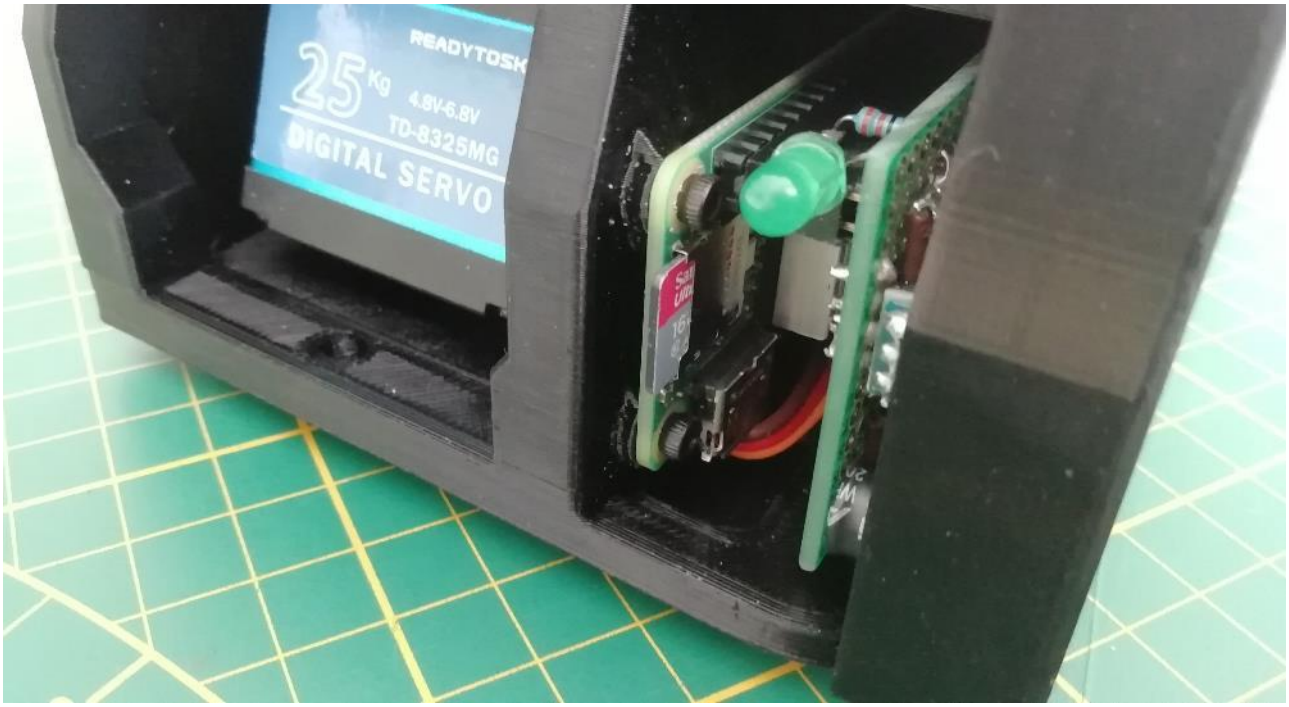
The Structure has a recess, meant to hold a metal plate working as a Stop touch sensor on the Base version.
On this version, the stop function is available on the PCB_cover, making that recess quite useless and unesthetic.
I made available two alternative stl files of a plate to 3D printed as cover such 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 😊.

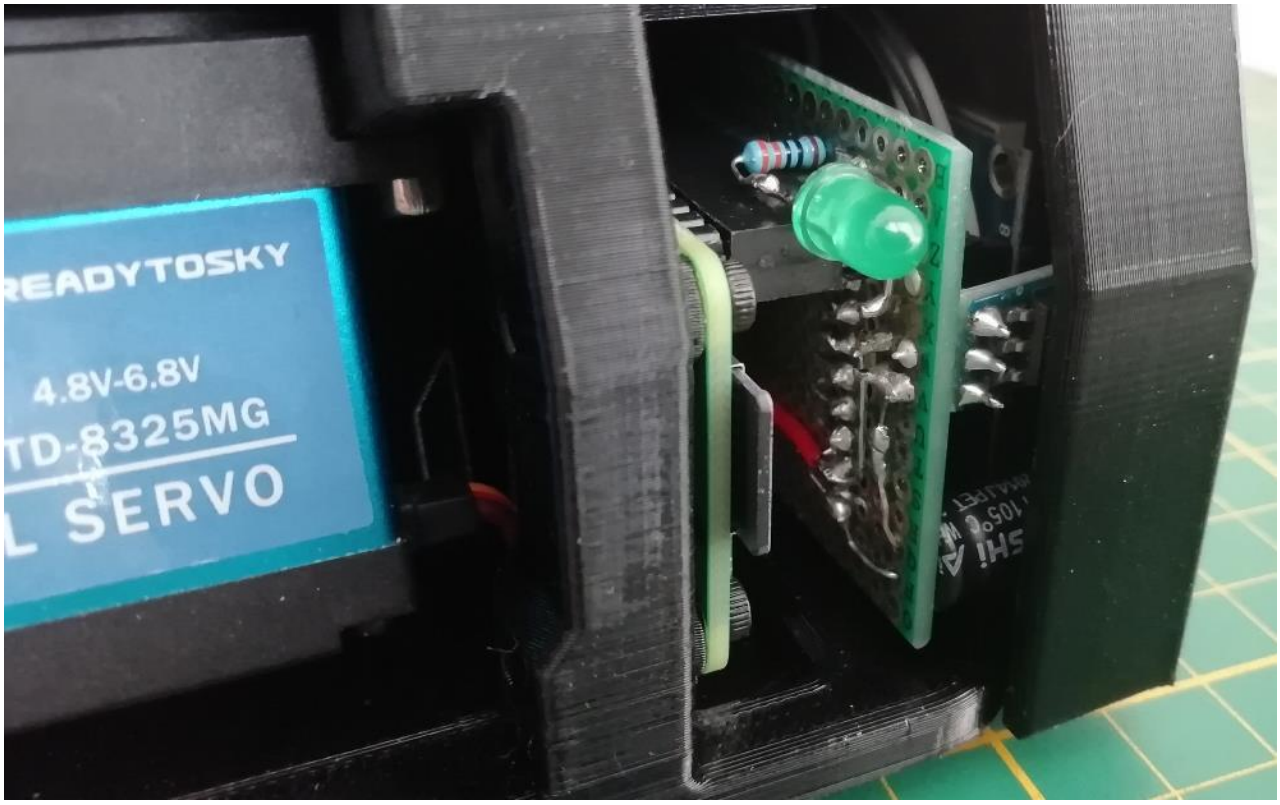
32) Collection of robot’s pictures:

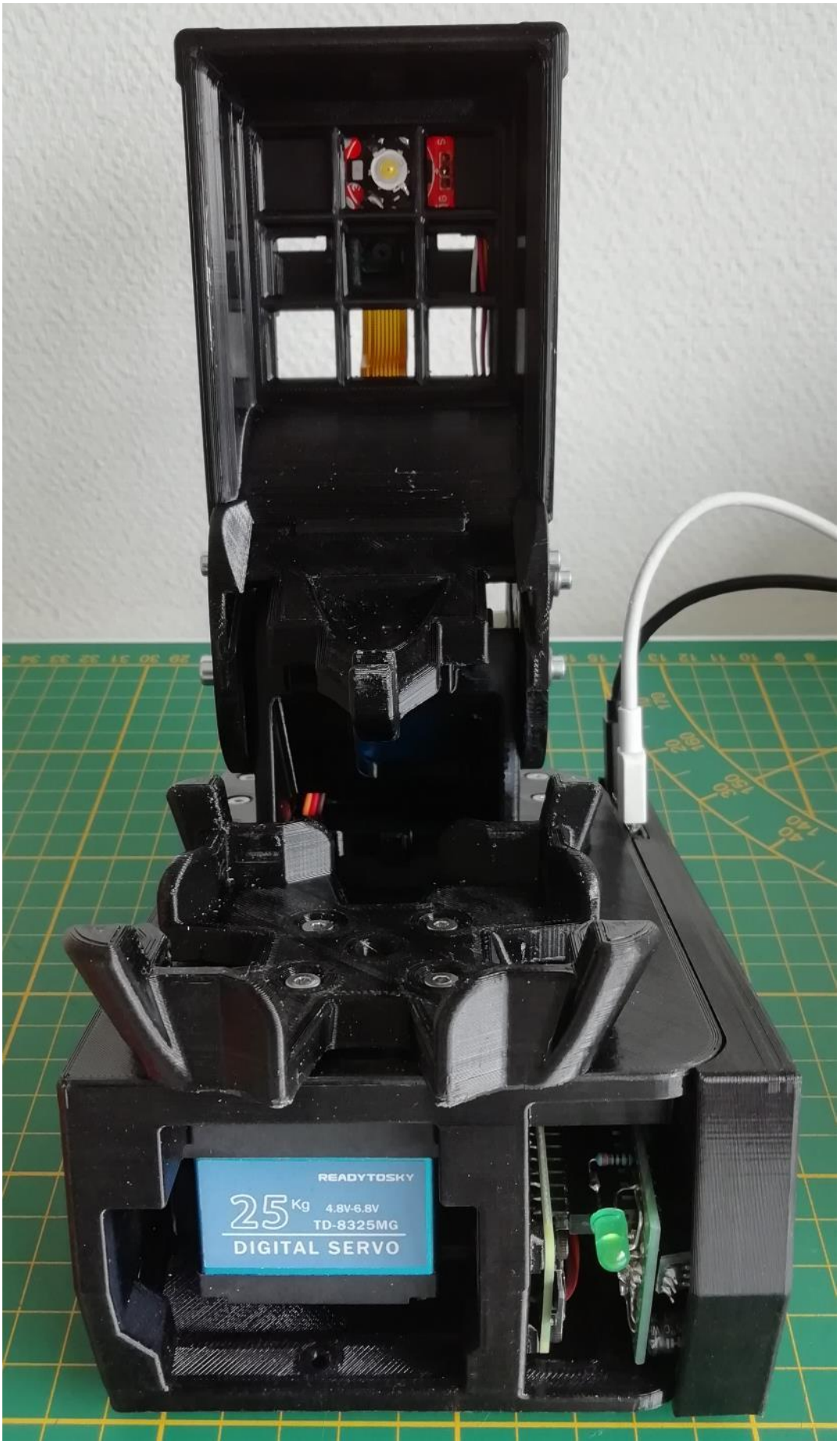





READYTOSKY
25 Kg
4.8V-6.8V
TD-8325MG
DIGITAL SERVO


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

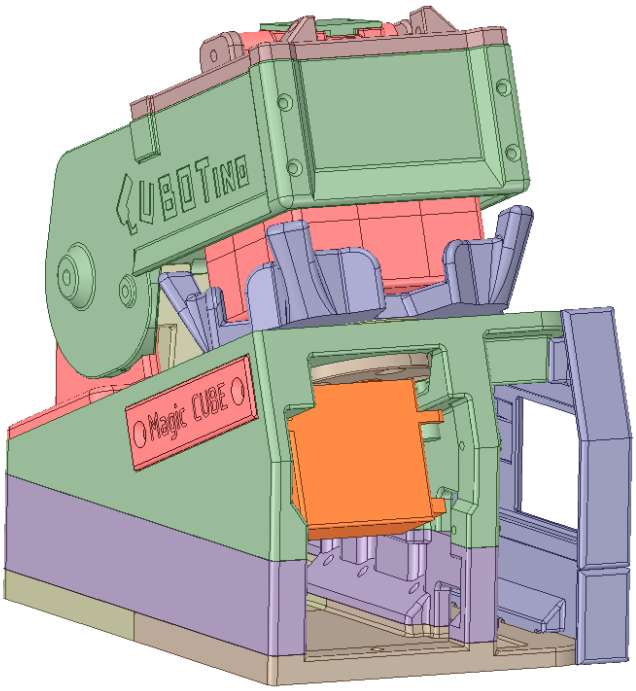
33) Using a Raspberry 3b or 4b instead:

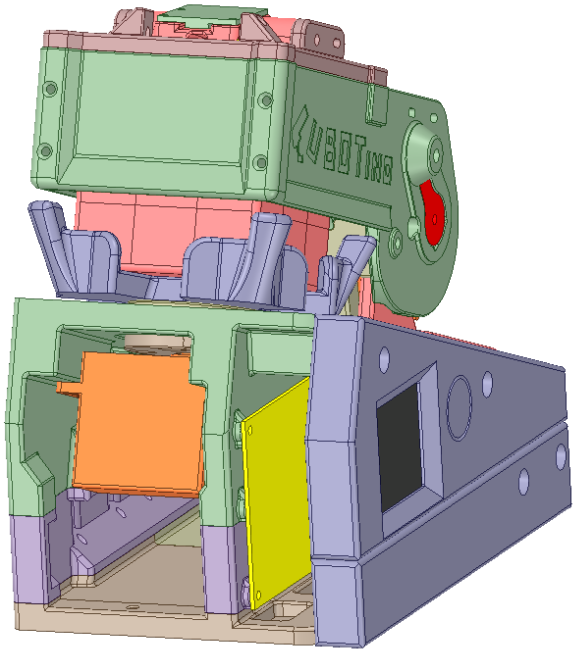
Because of the current Raspberry Pi Zero 2 shortage, it is possible to use a Raspberry Pi 3b or 4b as a temporary solution.

By adding three spacers the robot height is increased by 26mm
This makes possible to embed the larger Raspberry Pi SBCs, while keeping all the original parts unchanged:

- Connections_board and Display placement as per original project
- Possible to come back to the Raspberry Pi Zero 2, once it will be available, by “only” wasting the three additional spacers (Extensions) and related printing time…..
- One little manual modification needed, yet it doesn’t prevent to come back to the original design.

Please note this proposal is fully based on CAD analysis: I haven’t printed/tested the parts.
I’m confident it will work reasonably well.
Please feedback for either positive or negative results.



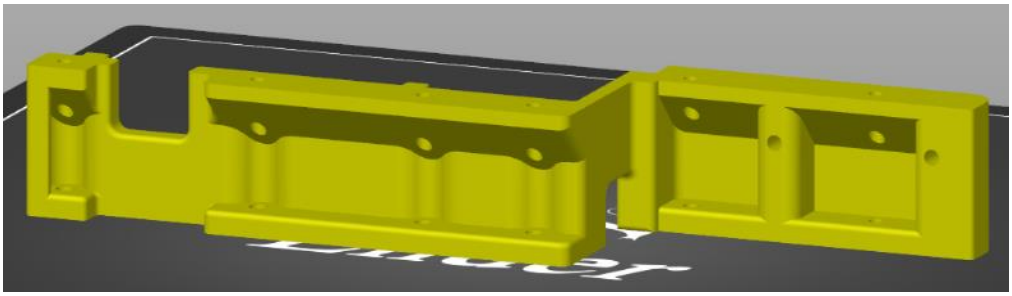
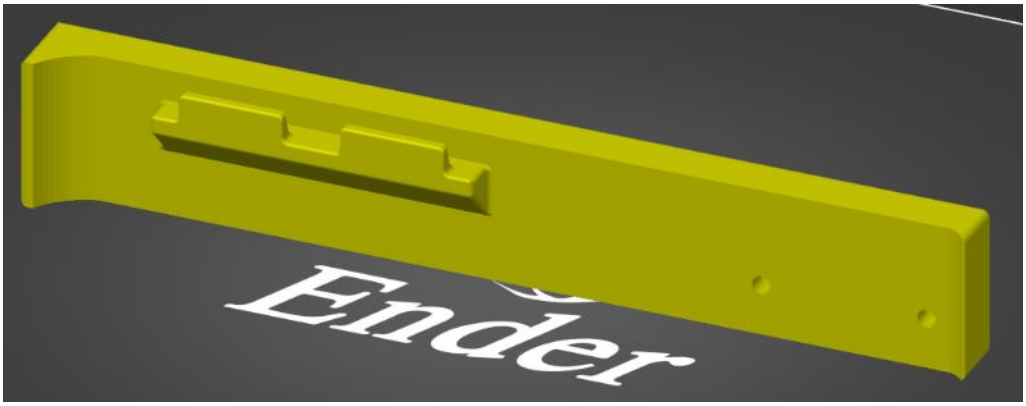


Filament quantity, and printing time …

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:

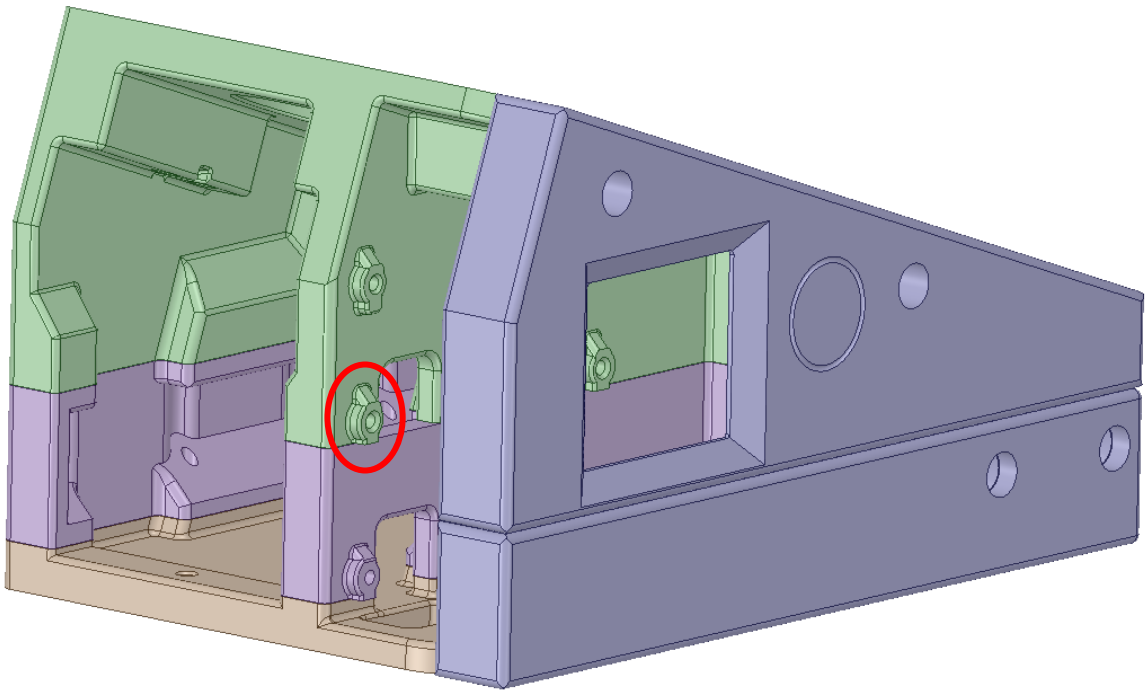
1. Still valid the notes on chapter 3D printed parts (i.e. no support needed).
2. Above Filament consumption and printing time are based on a quality printing setting; If these parts are really meant to be temporary in your case, a much lower quality could be considered.
3. The suggested part orientation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