How to make…

CUBOTone: A Rubik's cube solver robot, with Raspberry Pi and PiCamera

https://www.instructables.com/Rubik-Cube-Solver-Robot-With-Raspberry-Pi-and-Pica/

Andrea Favero, Groningen (NL)

Rev.3.2 31/01/2024 (Always check if a newer version is available)

Robot (and script) demonstration at YouTube: https://youtu.be/oYRXe4NyJqs

The robot takes from 40 to 60seconds to read and solve a scrambled cube.
This is indeed not a fast robot, yet it uses a common Rubik cube (not required to modify the cube for mechanical gripping, as most of the fast robots use to).

1. Instructions: Order and organization

This document is organized in several chapters, divided into 6 main sections:

- Sections 1 to 4 to make the robot.
- Sections 5 and 6 providing useful (or interesting) info.

Sections:

1. Supplies
2. Electrical part & Raspberry Pi setup
    a. Stepper motor driver board.
    b. Wiring the parts.
    c. Setup the Raspberry Pi.
3. 3D print and assembly
    a. Print the parts.
    b. Assemble the robot.
4. Tuning and robot operation
    a. Robot tuning.
    b. Troubleshooting.
    c. How to operate the robot.
    d. Automatic start and Rpi shut-off by the robot.
5. Info (my preferred part 😊, yet not strictly needed to build the robot)
    a. Project background.
    b. High level information.
    c. Robot solver algorithm.
    d. Computer vision.
    e. Colour detection strategy.
    ….and much more
6. Appendixes
    a. Useful links I used as reference.
    b. Printout during Raspberry Pi setup.
    c. Collection of robot images.
    d. Script usage on a (Windows) PC.

Use the Summary links to quickly reach the chapters.

2. Safety

Energize the robot only via power supply having a class 2 insulation from the power supply net.
Despite the robot mechanical force is limited, it must be operated only under adult supervision.
If you build and use a robot, based on this information, you are accepting it is on your own risk.

3. Manage expectations!

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

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

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

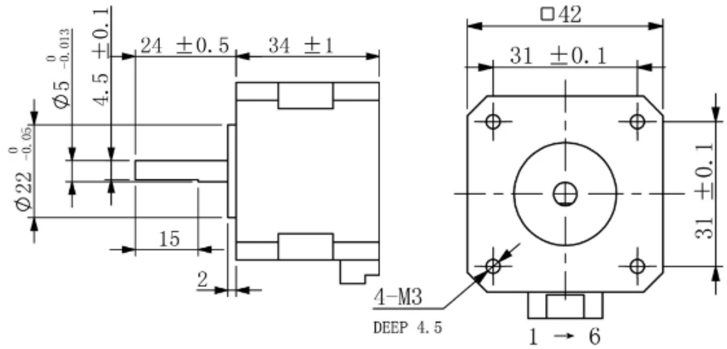
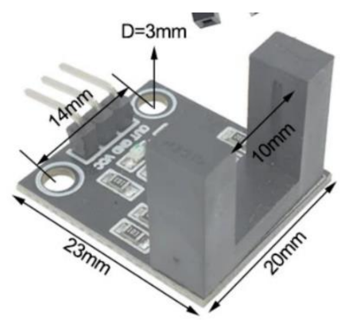
and now, let’s start!

Summary

5. Supplies

This table is a sort of minimum recommended parts; Links to the parts I bought are in the below excel file

Q.ty	Part	Notes
1	Raspberry Pi 4B 2Gb	I did not verify whether other models could do the job
1	Raspberry Pi 4 metal cover with fans (Not sure the fans are really needed)	
2	MicroSD Sandisk Extreme 32Gb (2nd one as backup, with same image)	16Gb also ok
1	PiCamera v1.3 with extension cable (50cm extension is perfect)	
1	Filament 1.75mm	PETG is very good, yet other material will do the job
2	180 deg Servo motors, with metal gear and metal lever “25T”	
1	Servo(s) driver (PCA 9685)	Better to buy 1 spare

Q.ty	Part	Notes
1	Step motor Nema 17 Stepper Motor, 34mm, 28Ncm, 1.3A, 2 phases, 1.8°, shaft Ø5mm with flat key) 200steps/rev	
1	Step motor driver, 1.5A (DRV8825)	Better to buy 1 spare
2	DC-DC transformer (1 for Raspberry pi, the 2nd for servos and remaining loads)	
1	Photo switch (Better to buy 1 spare)	 
1	Cable USB-C with screw connector	

Q.ty	Part	Notes
2	LCD displays with segments (Better to buy 2 spares)	
1	Momentary push-button with red led On / Off logo (Better to buy 1 spare)	
1	Push-button (Better to buy 1 spare)	
2	Led module 3W [link](#)	
1	Prototype boards and connectors	In alternative search for “DRV8825/A4988 42 Stepper Driver Module Motor Control Shield Drive”
1	Dc power supply, output ca 20Vdc (power ≥ 120W)	See notes below
1	Power connector	See notes below

Notes:

1. I had available a HP charger for a laptop (output 20Vdc 230W), and a HP connector adapter (link); These two parts have influenced the way I've organized the power supply system.
2. In my case I also bought some additional material (i.e., Micro HDMI to HDMI cable, bread board, etc), as this was my first experience with Raspberry pi.

In the embedded excel file the references to the parts I bought, and related info (shop, cost, delivery time, issues)

CubeSolverComponents.xlsx

Fixings:

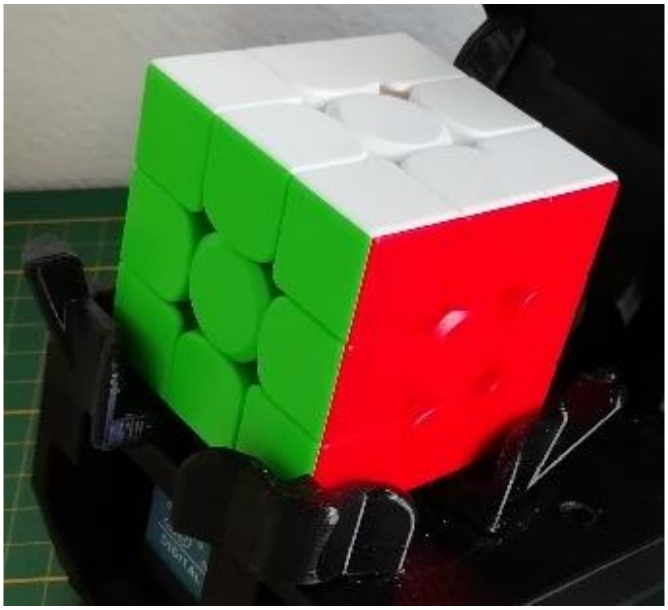
Q.ty	**Part**	**Notes**
50	2.5x13mm wood screw	For the Box construction, and thicker part toward the Box
50	2.5x10mm wood screw	For thinner parts toward the Box
2	M3x 30 + self-locking nuts	For the feet and feet hinges
1	M3x16	Lifter lever to lifter lever link
12	M3x10	Servo to servo holder (3 each) Upper_cover to servo metal lever "25T" Flipper lever link to servo lever Led modules to Top_cover
4	M3x8	Motor to motor support
2	M3x4	Metal lever "25T" to the servos. Use some spacers or reduce screw length in case too long screw available
1	M6x16	Upper_cover to upper-cover hinge (note: pre-thread the upper-cover)
1	M5x30 + self-locking nut	Lifter to lifter hinge
6	Rubber pads	4 on the Box base, 2 on the feet
4x 1cm	Filament 1.75	To fix the PiCamera to its holder (hot deforming)
2	Hinges	For easy opening of the Box top panel

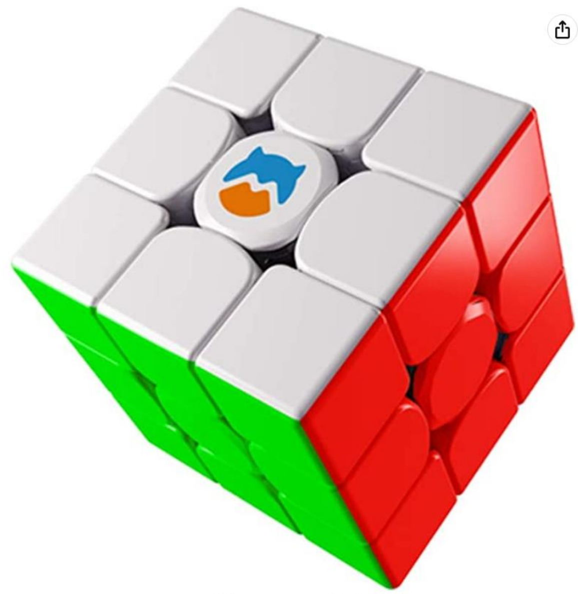
Electrical small parts:

Q.ty	**Part**	**Notes**
1x10	Headers	To connect to GPIO (odd pins)
1x6	Headers	To connect to servo driver
2x4	Headers	To connect to Displays
?	Headers	To connect to stepper motor driver interface
1	Capacitor 16V 470uF	Closer to the Raspberry pi
1	Capacitor 25V 180uF	At the stepper motor driver (or use a “DRV8825/A4988 42 Stepper Driver Module Motor Control Shield Drive”)
?	Resistors might be needed	In my case: 1 x 1 KΩ at On-off button (not clear specs on max current) 1 x 10 KΩ at stepper motor driver, to pull up the *enable* signal 1 x 4.7 KΩ at stepper motor driver if A4988 driver, to pull up the *step* signal

Off course some other common materials are needed (Wires, solder and solder device, tire wraps, etc)

6. A convenient cube





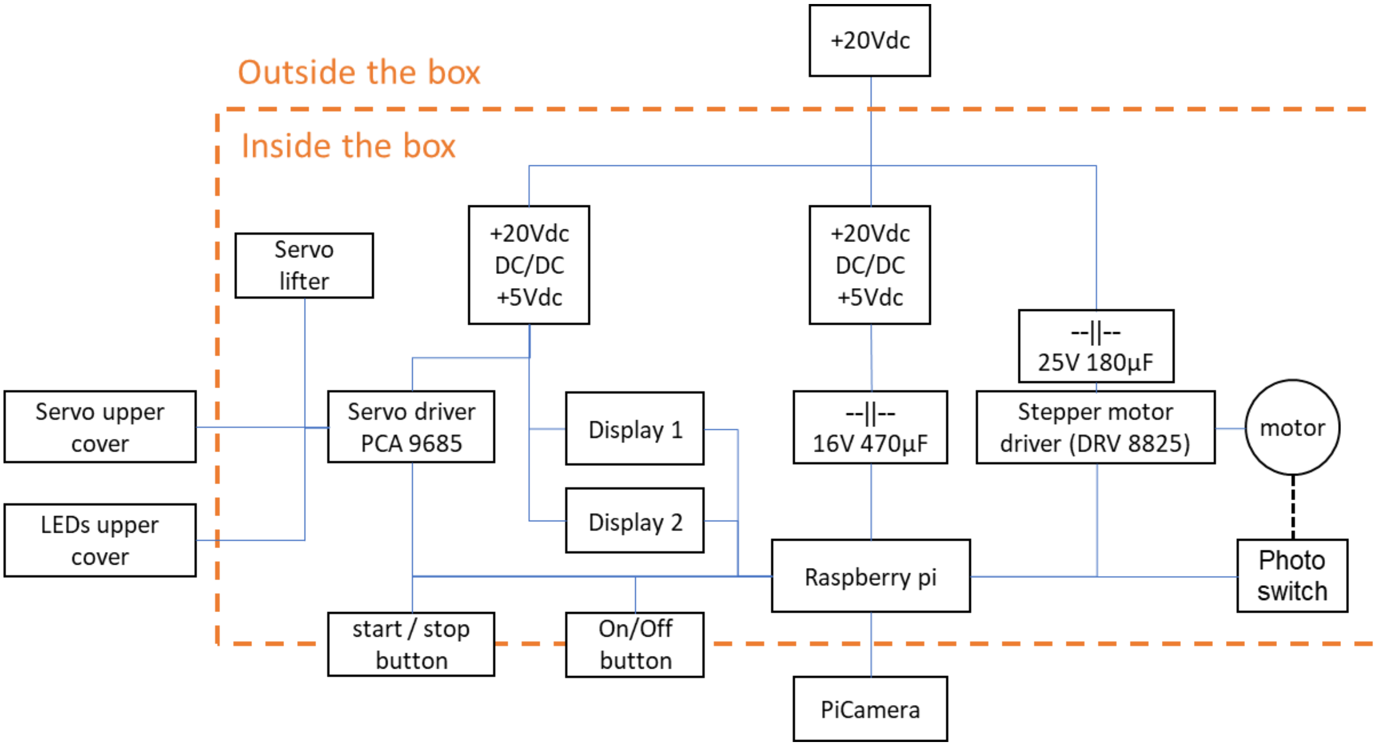
Thanks to a note from Art, I got to know the existence of “frameless” cubes.
Thanks to a note from Martin, I got to know about the existence of “magnetic” cubes.

These two informations were shared close to each other, therefore I decided to buy one cube having both those characteristics.
I ordered a “Monster Go Magnetic 3x3 Cube” in Amazon for about 16€ (https://www.amazon.com/Monster-Go-Magnetic-Learning-Beginners/dp/B087RMGJ12/ref=sr_1_1_sspa?keywords=monster%2Bgo%2Bmagnetic%2B3x3%2Bspeed%2Bcube&qid=1664722887&qu=eyJxc2MiOiIxLjUwIiwicXNhIjoiMS4zNyIsInFzcCI6IjEuMDcifQ%3D%3D&sprefix=monster%2Bgo%2B%2Caps%2C308&sr=8-1-spons&spLa=ZW5jcnlwdGVkUXVhbGlmaWVyPUEyRzhVMFNYUFQ4TDdGJmVuY3J5cHRlZElkPUEwNTk3OTcwMkw5QjM0SE9RWDlGNyZlbmNyeXB0ZWRBZElkPUEwODYyMTM0MjFUOFVWM0dONzBTViZ3aWRnZXROYW1lPXNwX2F0ZiZhY3Rpb249Y2xpY2tSZWRpcmVjdCZkb05vdExvZ0NsaWNrPXRydWU&th=1)

Well, my children and myself are very impressed by this cube, to the point I immediately ordered a second one.
This cube is very smooth, requires little force to turn, it easily cuts corners
The reason I’m especially pleased by this cube is because of the magnets, that keep the layers very well aligned: This makes a perfect cube for the robot.
In terms of dimensions, this cube is halfway the two I had home, on which I based the robot parts.
Of course, the logo must be scratched away.

Note: I’m not affiliated with Amazon, nor any other Company, I’m just sharing my positive experience with this cube 😊

7. Electrical part



Note:

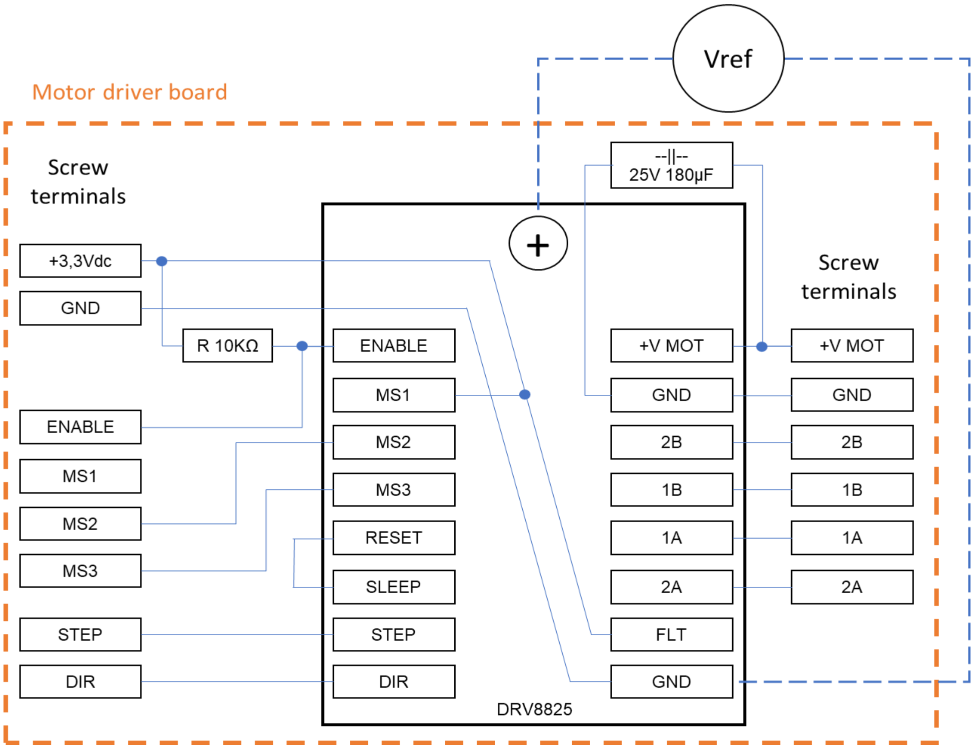
HP (and other) manufacturers of laptop's chargers, use of a sensing pin on the connector to enable a sort of smart power management; The HP power supply I have, goes in low power mode if the sensing pin is "floating".

Based on https://www.fixya.com/support/t1877467-hp_zd8000_laptop_power_supply# I've added a 47kΩ between the central pin and the +19Vdc, and it simply works fine.

Raspberry Pi 4b GPIO pinout

Proto board for DRV 8825 driver:

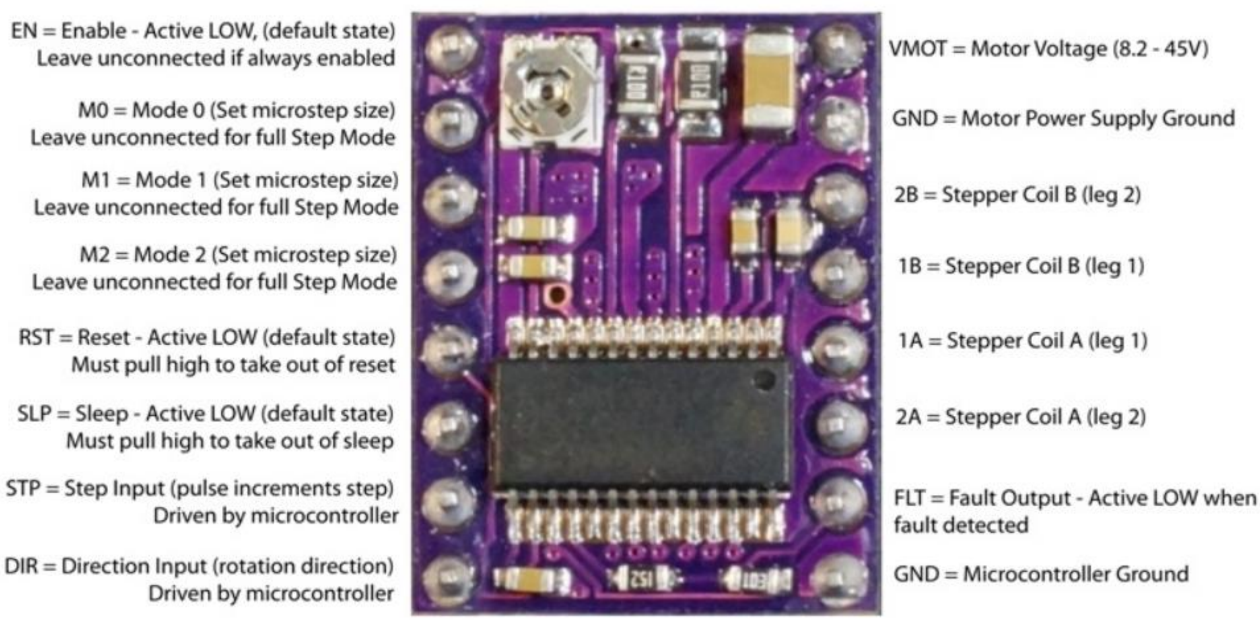
Alternatively an extension board for DRV8825 will do the job (ref: link1, or link2)



Setting the max motor current (as per https://www.pololu.com/product/2133):

Vref is set on 0.63V, the max reached via the potentiometer.

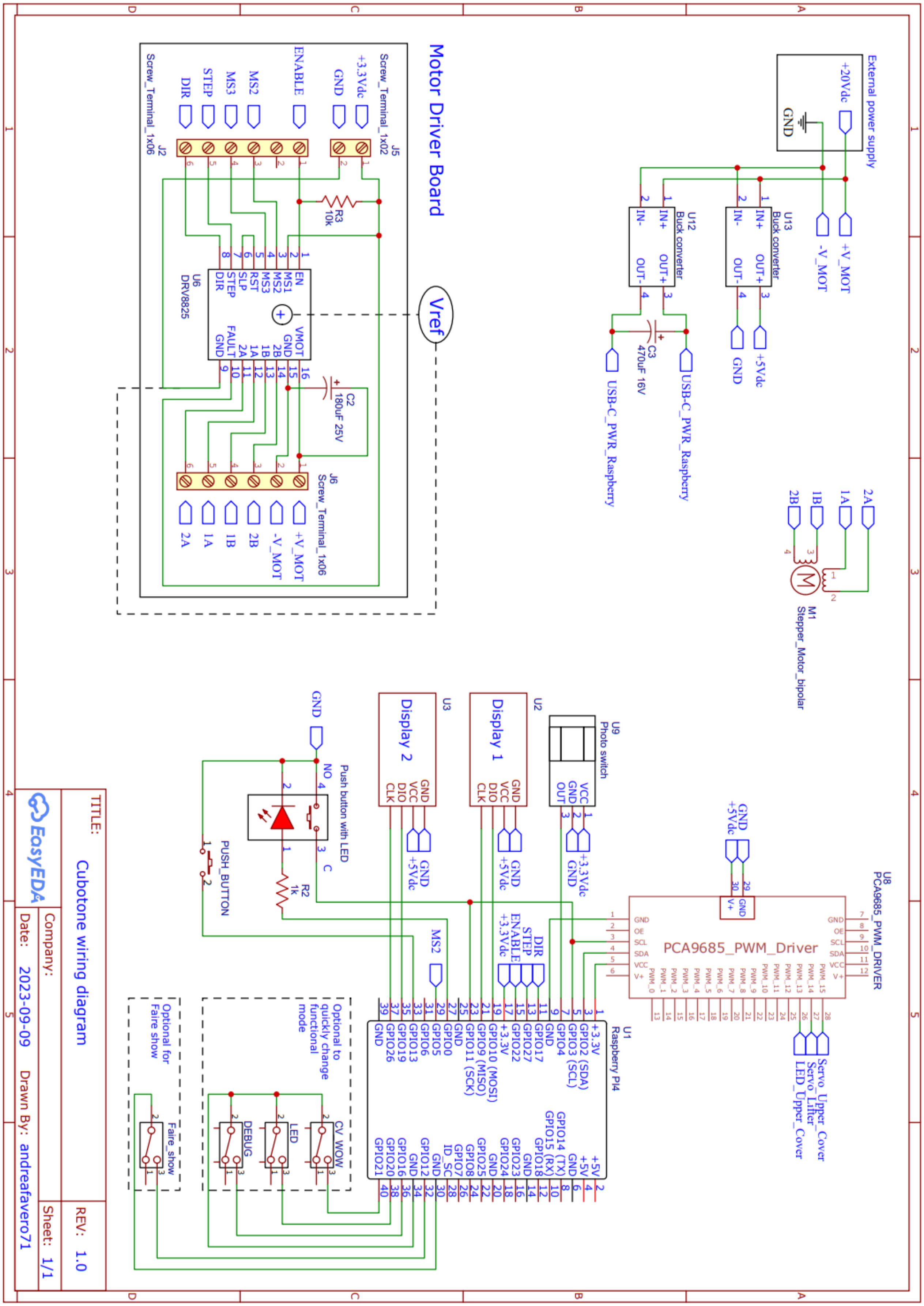
R sense=0.1Ω
Vref=Imax/2
Max current = 1.3A (in line with the stepper motor I bought)



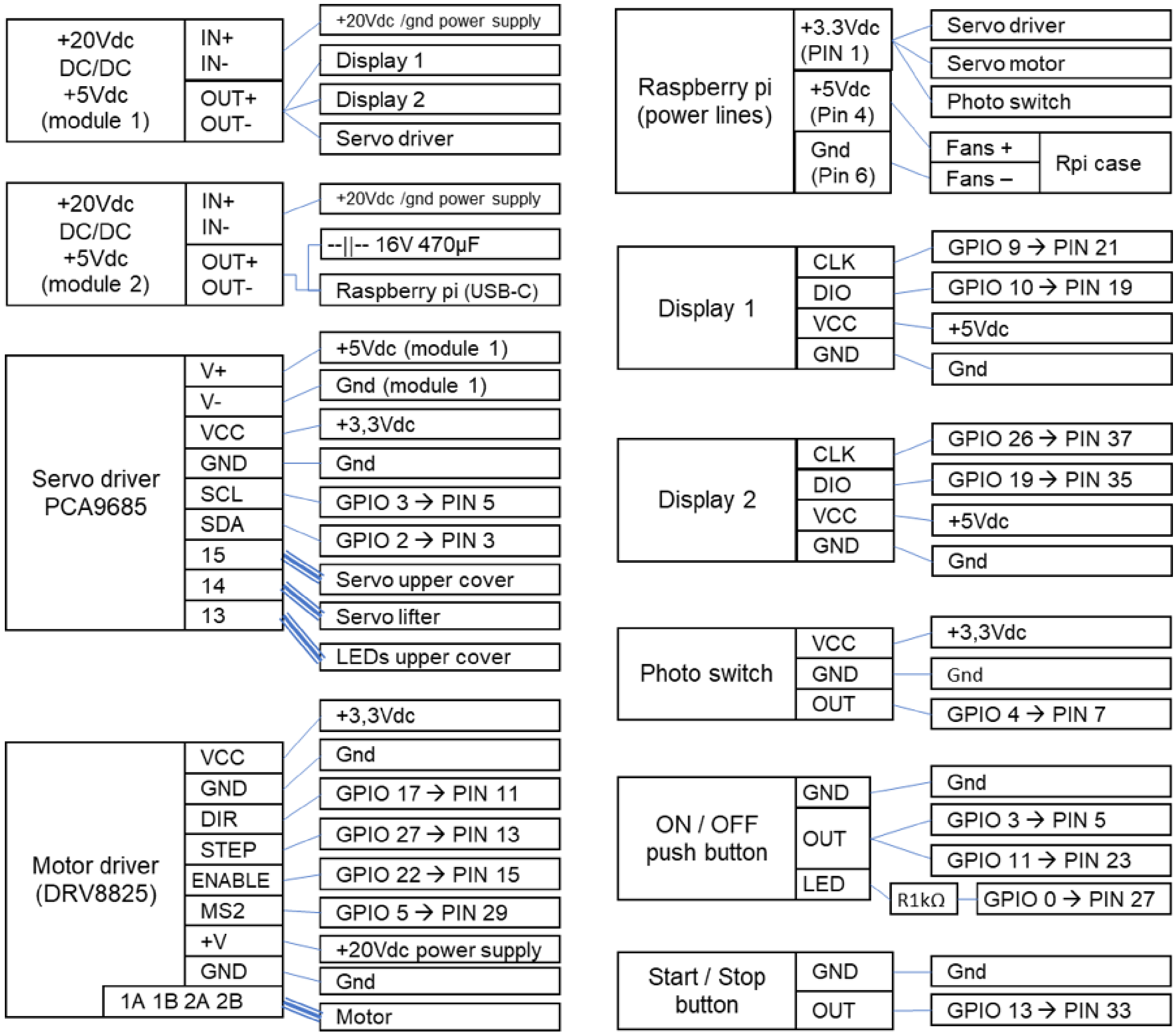
Notes:

1. MS1 (micro step1) is connected to +3.3Vdc, therefore high: It forces the driver to microstep ½ (from 200 steps to 400 steps per revolution. This setting is used for the cube spinning and rotation.
2. MS2 (+MS1): When high it forces the driver to microstep 1/8 (from 200 to 800 steps per revolution. This setting is used during motor alignment to the synchronization disk
3. MS3: Not used
4. SLEEP connected to RESET.
5. ENABLE: Added a 10KΩ pullup, to prevent the driver to activate the motor due to noise (Servos activations). This input is used to activate/de-activate the motor current, according to the robot phase.
6. STEP: Input used to steer the revolution amount and its speed
7. DIR: Used to steer the rotation direction

Connections diagram:



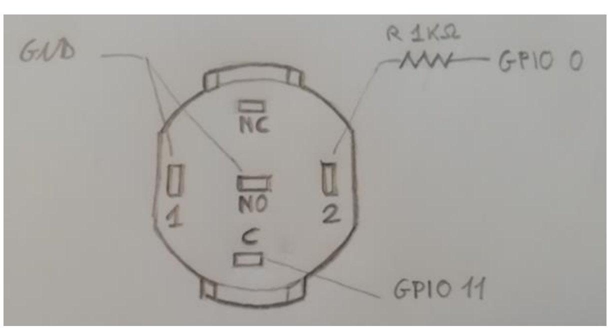
Modules and connections list:



Notes:

GPIO 3 (pin 5) is connected to both the ON / OFF push button and the SCL line of the Servo Driver

The push button I bought (from https://it.aliexpress.com/item/32956631402.html?s...) isn't matching the terminal description as per the web shop; On below image the terminals, oriented and marked as per the received part, and the connection made (of course C and NO can be swapped).



8. Setting up Raspberry Pi 4b

Cubotone has been developed with OS10, and it doesn't work with later OS versions.
The OS must be downloaded from the Raspberry Pi official site; Since the release of Bookworm, the Buster OS is no longer listed by the Raspberry Pi Imager (also not as Legacy).

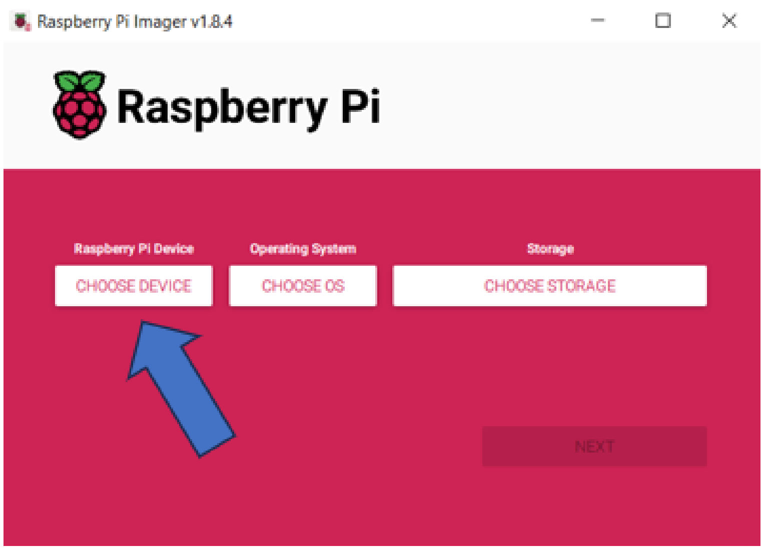
All the needed files, and Raspberry Pi settings, are stored in a GitHub repository.

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

Step2: Flash the OS to the microSD:

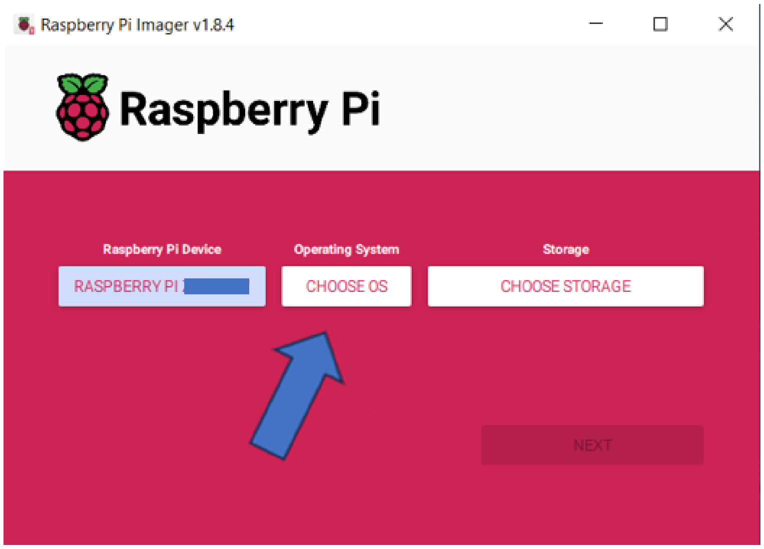
a. Download the Raspberry Pi Imager, from the official site https://www.raspberrypi.com/software/

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

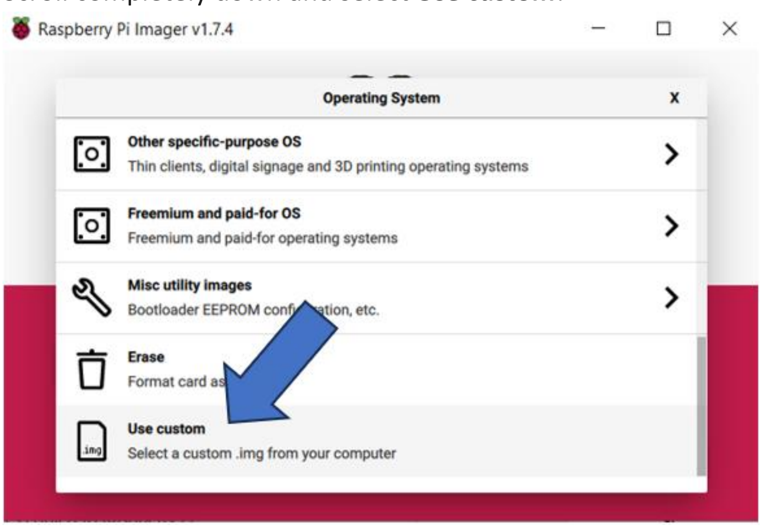


and select the board type you are using:

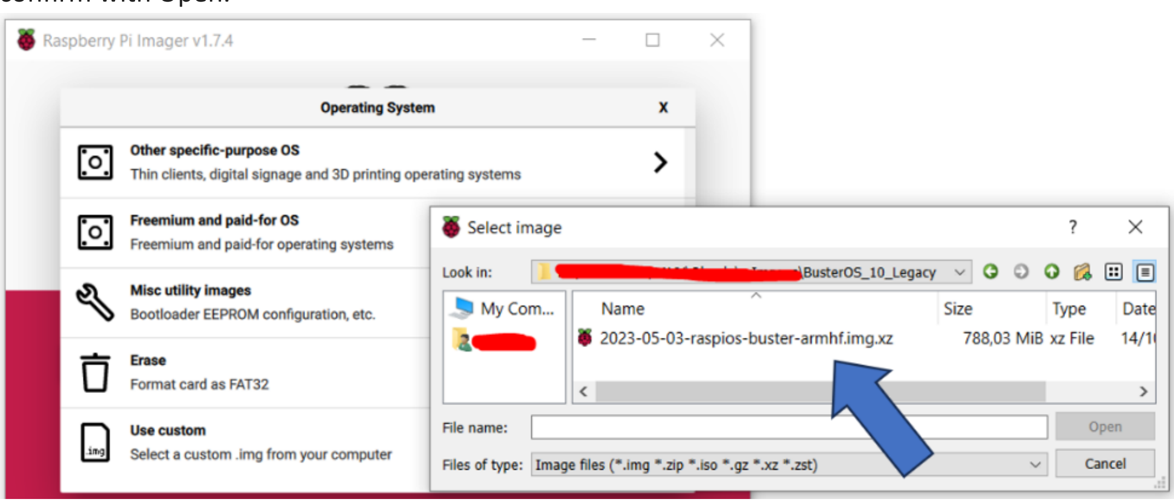
c. Select ***CHOOSE OS***:



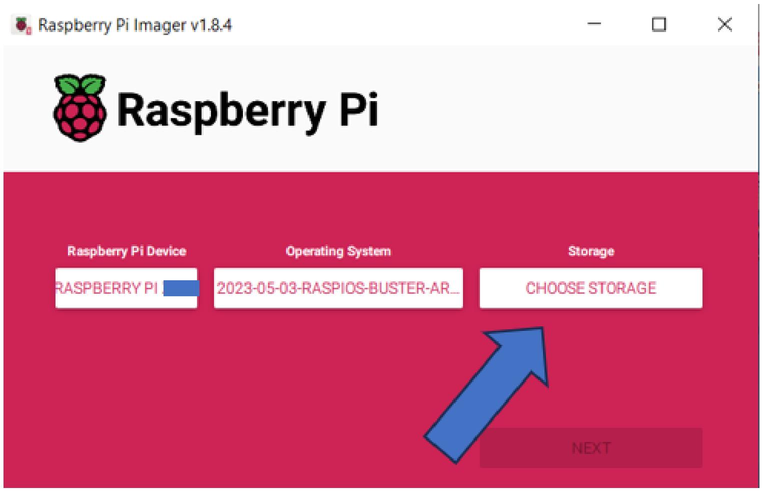
d. Scroll completely down and select ***Use custom***:



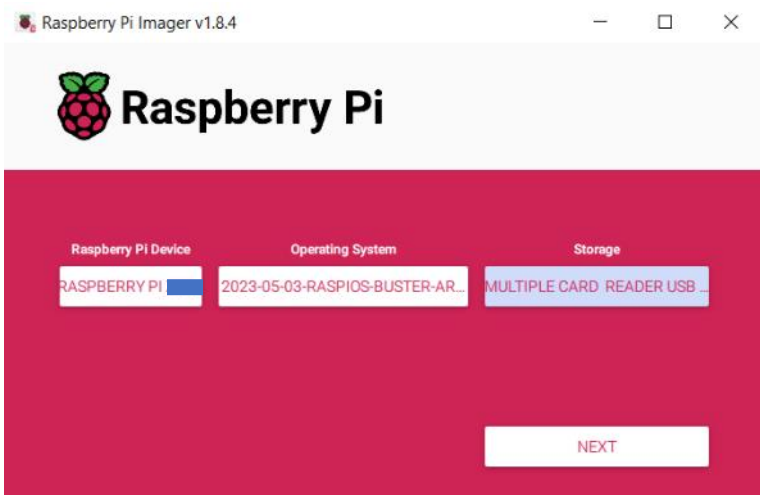
e. Navigate on your PC and select the downloaded Image (*2023-05-03-raspios-buster-armhf.img.xz*), confirm with Open.



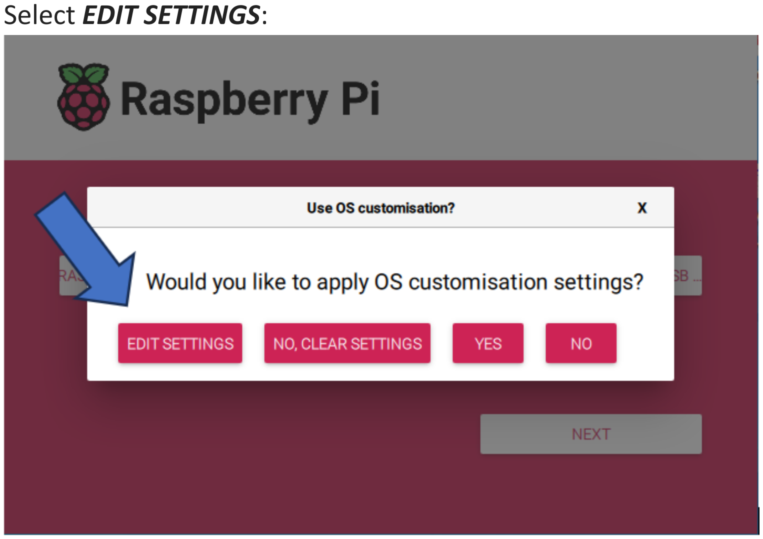
f. Select the ***Choose Storage*** disk (microSD card):



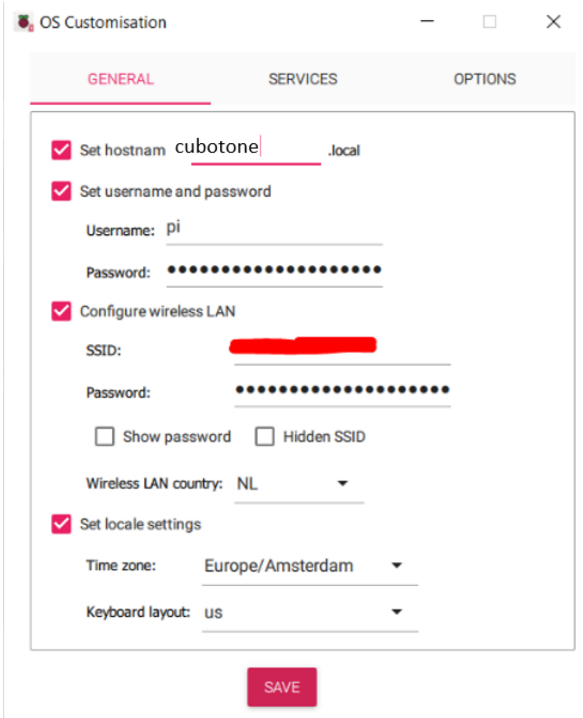
g. Select ***NEXT***:



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

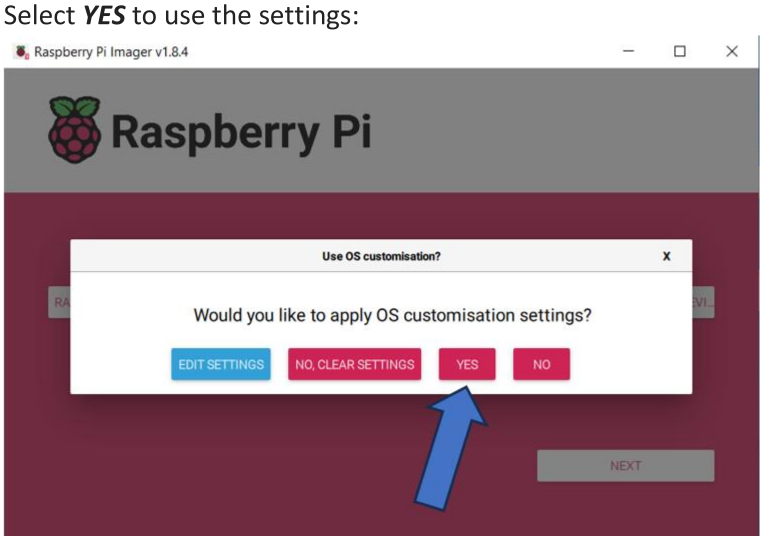


On settings, enter:

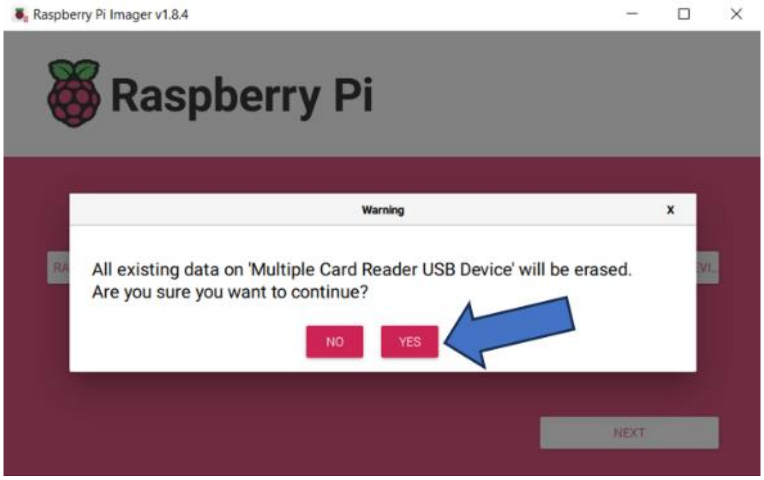


a. *Cubotone*.local
b. check SSH.
c. enter *pi* as username.
d. enter a password (my choice has been *raspberry,* as no sensitive data, but you are encouraged to use a more secure password).
e. check set Wi-Fi.
f. enter your SSID.
g. enter your network password.
h. enter your Country code.
i. set your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

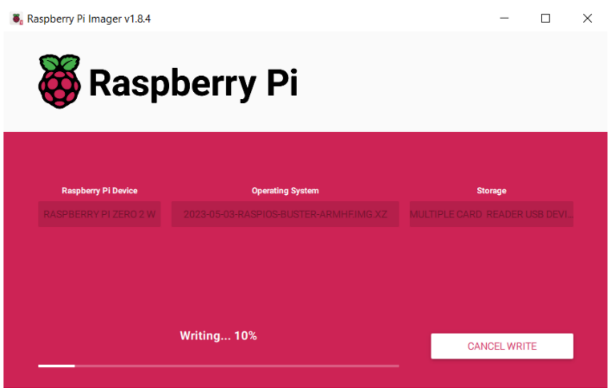
i. Select ***YES*** to use the settings:

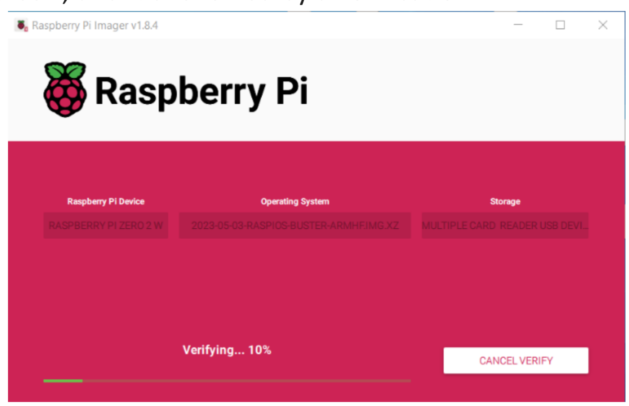


j. Select YES to start flashing the OS to the microSD:



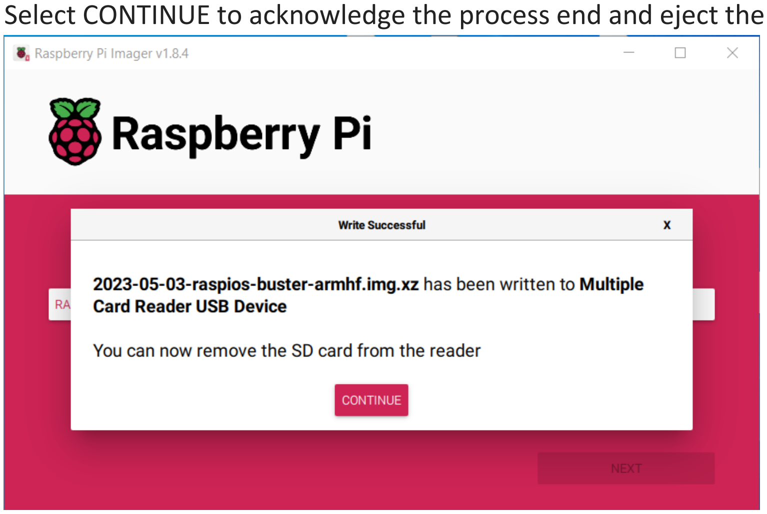
Raspberry Pi Imager starts writing the OS in the microSD, and it's followed by a verification:





Writing and verifying takes around 10 minutes.

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



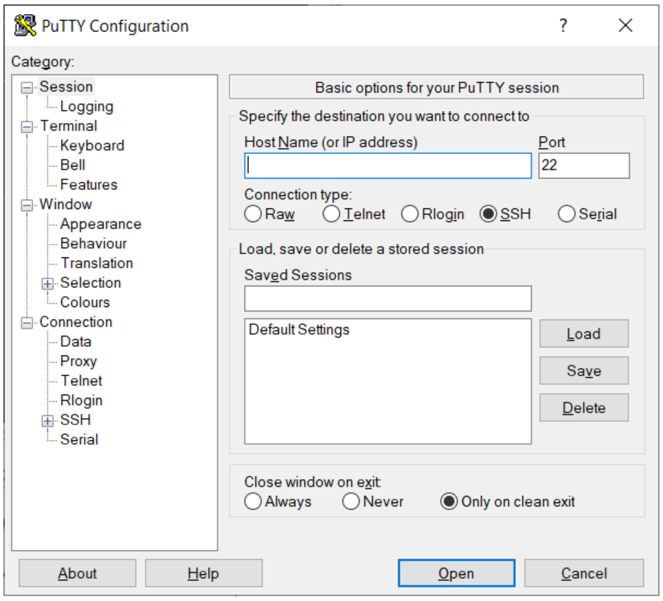
Step 3: Insert the SD card into the Raspberry Pi, and power it.

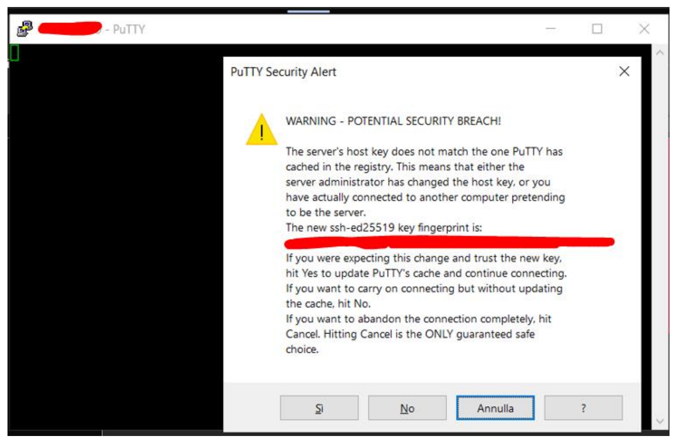
First boot takes longer, up to two minutes; Wait until the led stop blinking

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

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

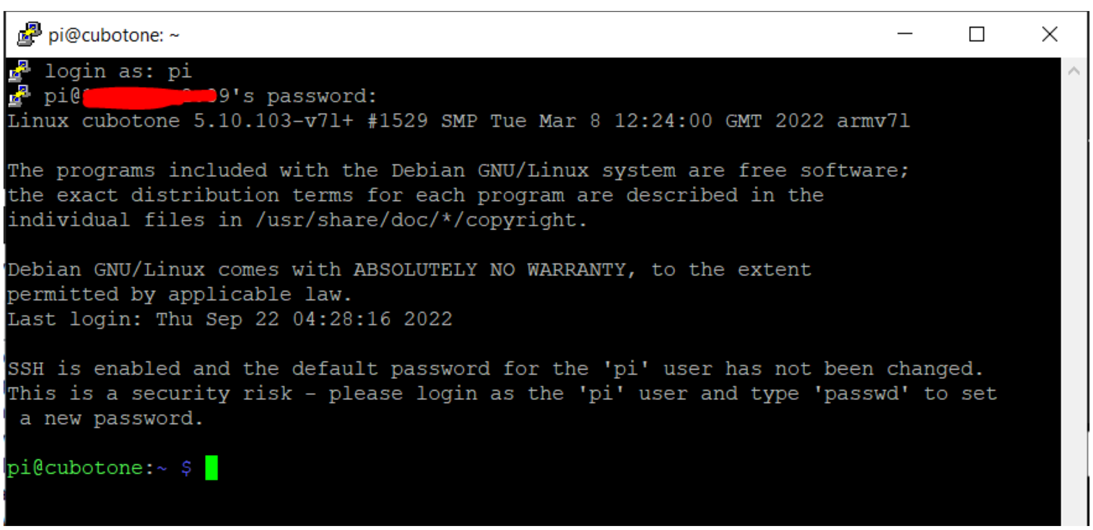
Step 5: Connect to the Raspberry Pi via SSH, i.e. by using Putty: Run Putty, with the IP address of the Raspberry Pi on the Host Name, remain settings as per Putty default. Accept the warning….





Login as: *pi*

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



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

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

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

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

```
pi@cubotone:~ $ git clone https://github.com/AndreaFavero71/cubotone.git
Cloning into 'cubotone'...
remote: Enumerating objects: 802, done.
remote: Counting objects: 100% (250/250), done.
remote: Compressing objects: 100% (179/179), done.
remote: Total 802 (delta 135), reused 111 (delta 55), pack-reused 552
Receiving objects: 100% (802/802), 109.75 MiB | 5.16 MiB/s, done.
Resolving deltas: 100% (440/440), done.
Checking out files: 100% (39/39), done.
pi@cubotone:~ $
```

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

Step 7: Start the installation:

- Enter cubotone/src folder from the root type: *cd cubotone/src*
- Start the bash file that takes care of the installation: *sudo ./install/setup.sh* (attention to the dot).
- In about 10 minutes, also depending on the internet connection speed, a Raspberry Pi 4b will complete the setup.
- Once requested confirm the reboot with a *y* and press enter.
- When the Raspberry Pi boots, the ON/OFF button LED powers ON
- See Appendix 1 for terminal printout reference

The installation is completed!

The below folder structure will result at Raspberry Pi:

/home/pi/cubotone/	Is the main robot folder.
/home/ pi/cubotone/doc	contains the How_to_buid… .pdf file.
/home/ pi/cubotone/extra	contains the link to the Instructables page of this robot.
/home/ pi/cubotone/scr/	contains all the robot specific files.
/home/ pi/cubotone/src/twophase	contains the lookup tables for Kociemba solver.
/home/ pi/cubotone/src/install	contains the setup.sh bash file: Do not use this file!
/home/ pi/cubotone/stl	contains the CAD files in stl format.

Note: This folder structure differs from the original one used until October 2022.

The installation ends, with a Raspberry Pi reboot.

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

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

Step 8: Updating the installation:

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

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

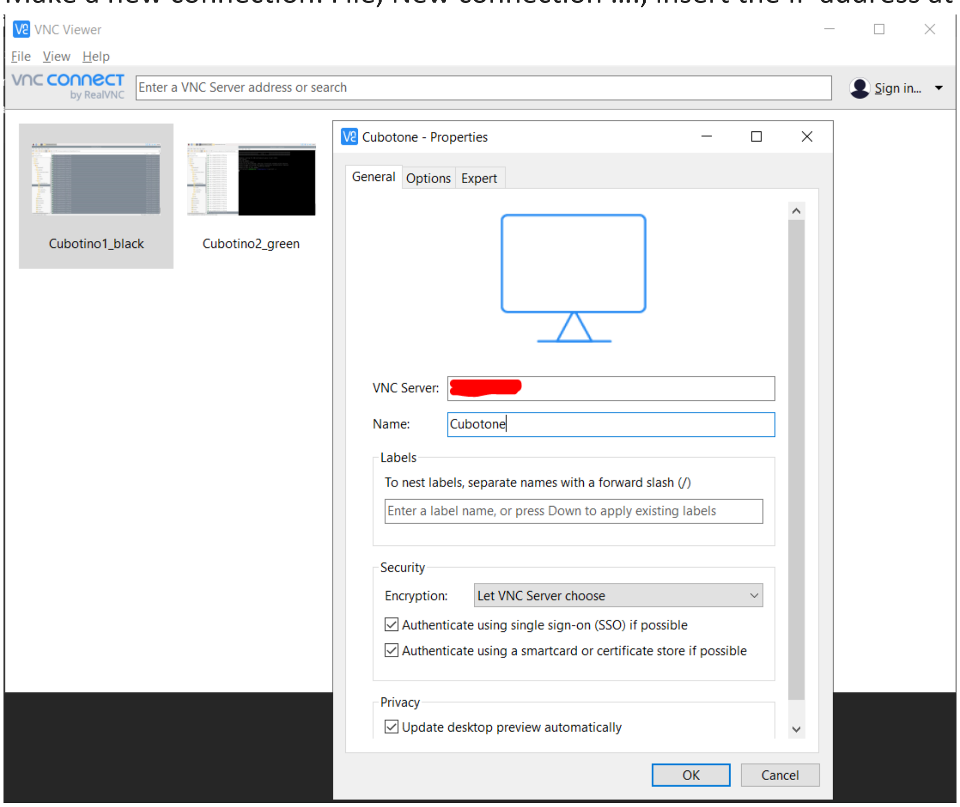
Note: Please remember to take notes (or save a copy) of your personal settings (Cubotone_settings.txt and Cubotone_servos_settings.txt) before updating the robot package;
In case you forgot…, the robot makes a backup copy for you at every boot; In this case rename the backup files:

Cubotone_settings_backup.txt → Cubotone_settings.txt
Cubotone_servos_settings_backup.txt → Cubotone_servos_settings.txt

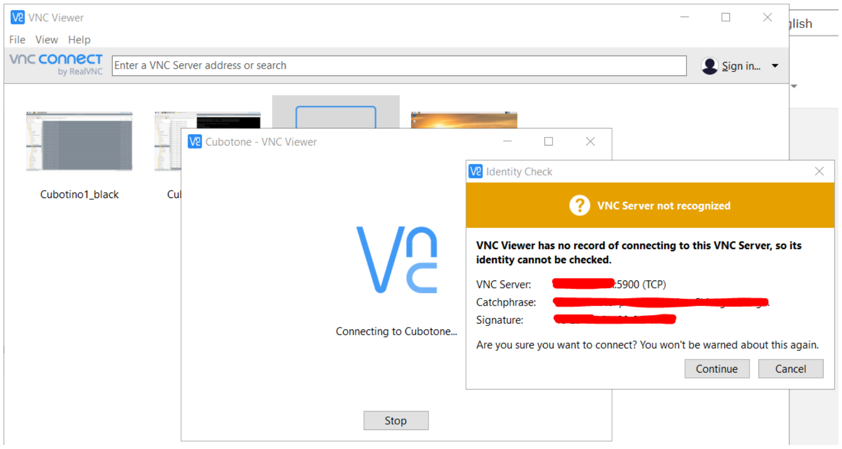
Please bear in mind the updates are based on my robot; There might be other changes you've made: Those must be manually handled by you.

Step 9: Make a new connection at VNC Viewer

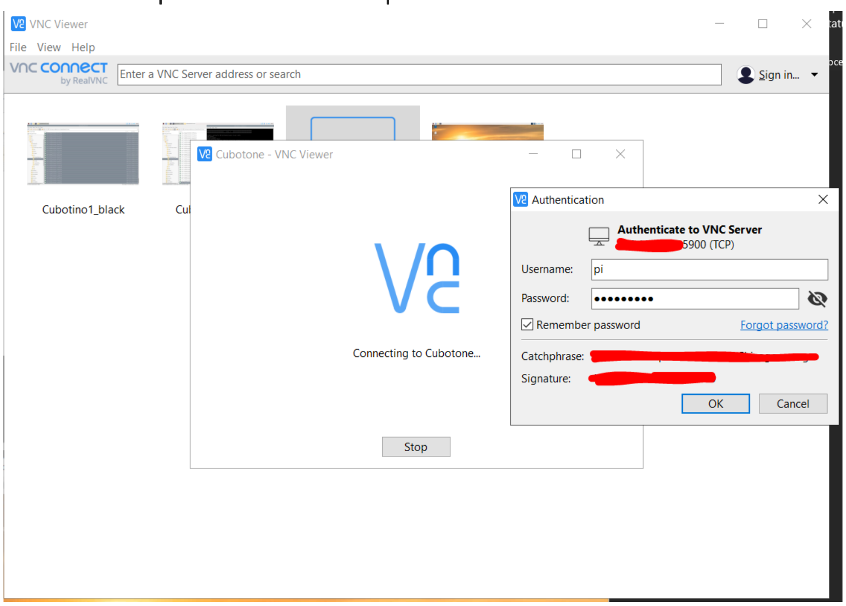
Make a new connection: File, New connection …., insert the IP address at VNC Server, and a Name



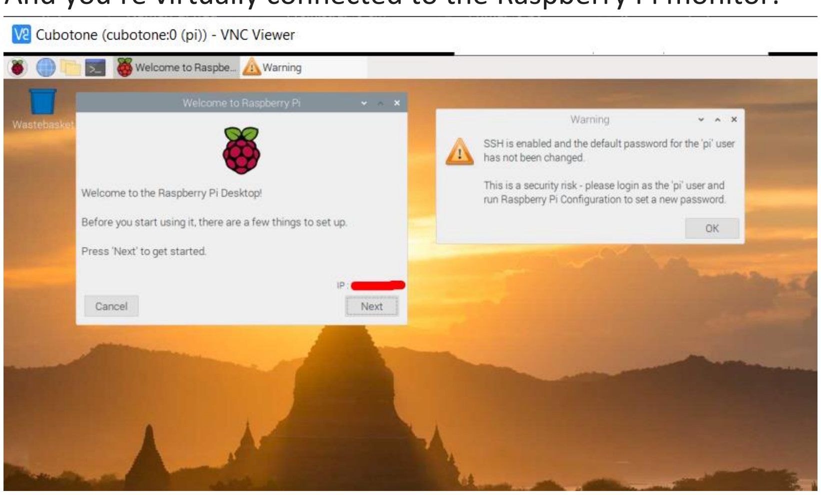
Read and acknowledge the warning:



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:



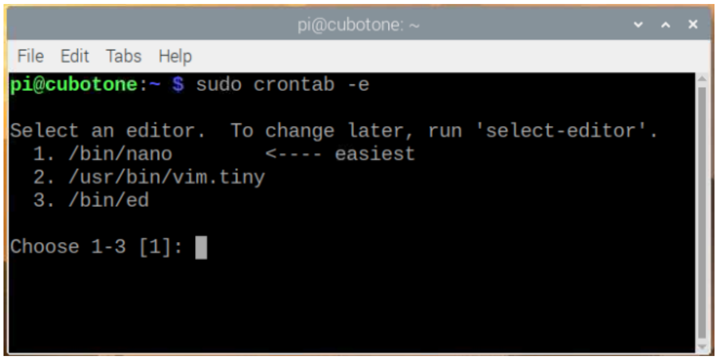
Check, and accept 😉, the warning.

You can skip the request to set up, as everything has been already set if you followed these instructions (i.e. filled all the settings at Raspberry Pi Imager)

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

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

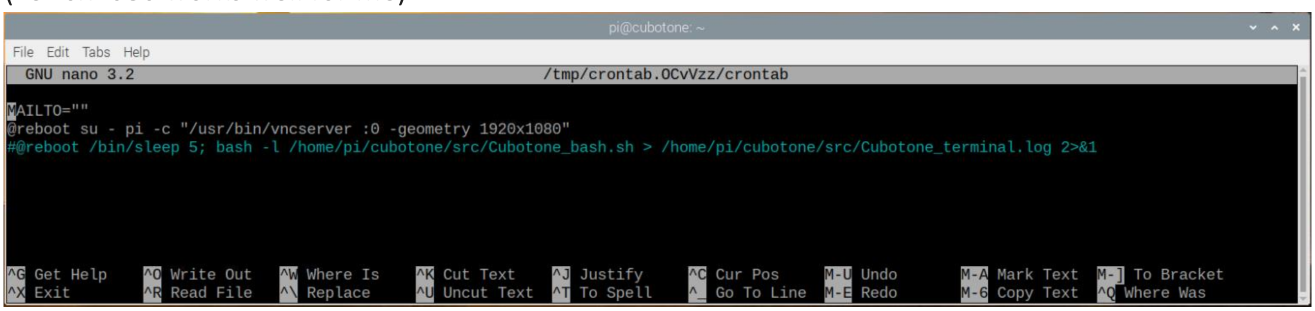
The first time you'll be ask to choose an editor, use 1 for nano



Change the geometry on the line:

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

(1920x1080 works well for me)



Step 11: Make a backup image of the microSD

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

There are many tutorials available for this task, as reference:

https://howchoo.com/g/nmexndnlmdb/how-to-back-up-a-raspberry-pi-on-windows

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

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

See 'Automatic robot start' chapter.

Step13: 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.

Step14: multiple WiFi settings:

Adding a second (or more) wifi connections, for instance to add your phone wifi hotspot, allows you to show the robot on different locations and still sharing on a screen the image processing part. For instance, by adding the phone wifi hotspot details you can use the Real VNC app on the phone to show the image processing part.

Steps:

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

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

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

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

Note: The priority command is needed when some of the wifi networks are available on the same time; The higher the value, the higher the priority.

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

Note: Once the Raspberry Pi boots, these files will be removed and the information saved in the other partition of the microSD.

Step15: card volume and memory check

Based on a 8Gb microSD, and several hundred cube status pictures:

```
pi@cubotone: ~
File  Edit  Tabs  Help
pi@cubotone:~ $ df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/root       7.1G  3.5G  3.3G  53% /
devtmpfs        750M     0  750M   0% /dev
tmpfs           911M     0  911M   0% /dev/shm
tmpfs           911M  8.5M  902M   1% /run
tmpfs           5.0M  4.0K  5.0M   1% /run/lock
tmpfs           911M     0  911M   0% /sys/fs/cgroup
/dev/mmcblk0p1  253M   49M  204M  20% /boot
tmpfs           183M     0  183M   0% /run/user/1000
pi@cubotone:~ $ free -h
              total        used        free      shared  buff/cache   available
Mem:          1.8Gi       236Mi       1.2Gi        16Mi       355Mi       1.5Gi
Swap:          99Mi          0B        99Mi
pi@cubotone:~ $
```

Conclusion:

- 8Gb microSD is sufficient for this robot, based on the legacy OS.
- Be noted 16Gb is required for more recent OS with desktop functionality.

Files needed at Raspberry Pi

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

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

File	Purpose	Notes
Cubotone.py	Main robot script	
Cubotone_moves.py	Translates the cube solution (Singmaster notation) in robot moves	
Cubotone_servos.py	Manages the servos	
Cubotone_set_picamera_gain.py	Manages the Camera gains settings	
Cubotone_tm1637	Manages the 7 segments displays	
Cubotone_scrambler.py	Scrambles the cube to a pre-defined or randomly chosen configuration	
Cubotone_settings.txt	Json file with settings for Cubotone.py script	Default values, to start the tuning
Cubotone_settings_AF.txt		Optimized values for my robot
Cubotone_servos_settings.txt	Json file with settings for Cubotone_servos.py script	Default values, to start the tuning
Cubotone_servos_settings_AF.txt		Optimized values for my robot
Cubotone_bash.sh	Bash file to start-up the robot script automatically after Raspberry Pi boots	

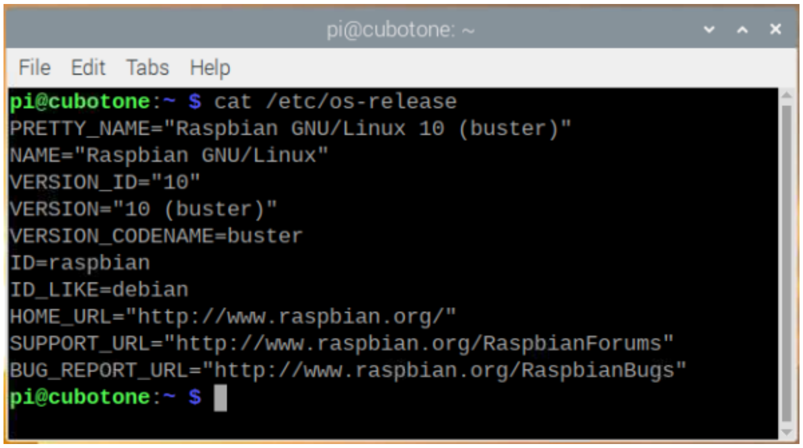
Kociemba solver library is necessary.

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

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

The project has been developed / tested with:

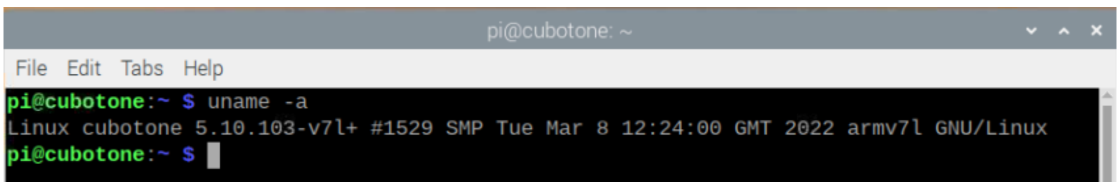
- OS Release Notes: *cat /etc/os-release*



```
pi@cubotone:~ $ cat /etc/os-release
PRETTY_NAME="Raspbian GNU/Linux 10 (buster)"
NAME="Raspbian GNU/Linux"
VERSION_ID="10"
VERSION="10 (buster)"
VERSION_CODENAME=buster
ID=raspbian
ID_LIKE=debian
HOME_URL="http://www.raspbian.org/"
SUPPORT_URL="http://www.raspbian.org/RaspbianForums"
BUG_REPORT_URL="http://www.raspbian.org/RaspbianBugs"
pi@cubotone:~ $
```

- Kernel Version: *uname -a*

  Linux cubotone 5.10.103-v7l+ #1529 SMP Tue Mar 8 12:24:00 GMT 2022 armv7l



```
pi@cubotone:~ $ uname -a
Linux cubotone 5.10.103-v7l+ #1529 SMP Tue Mar 8 12:24:00 GMT 2022 armv7l GNU/Linux
pi@cubotone:~ $
```

- 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.

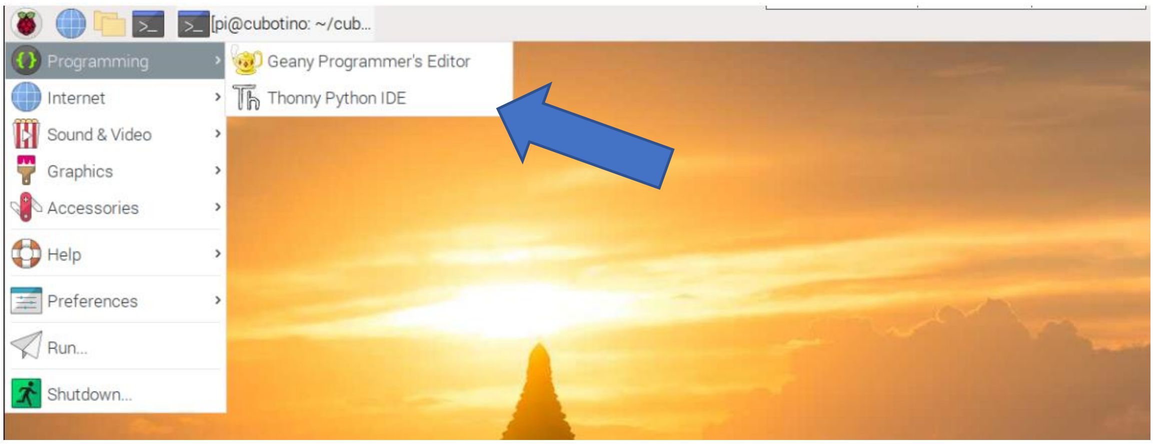
9. Set Thonny IDE interpreter

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

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

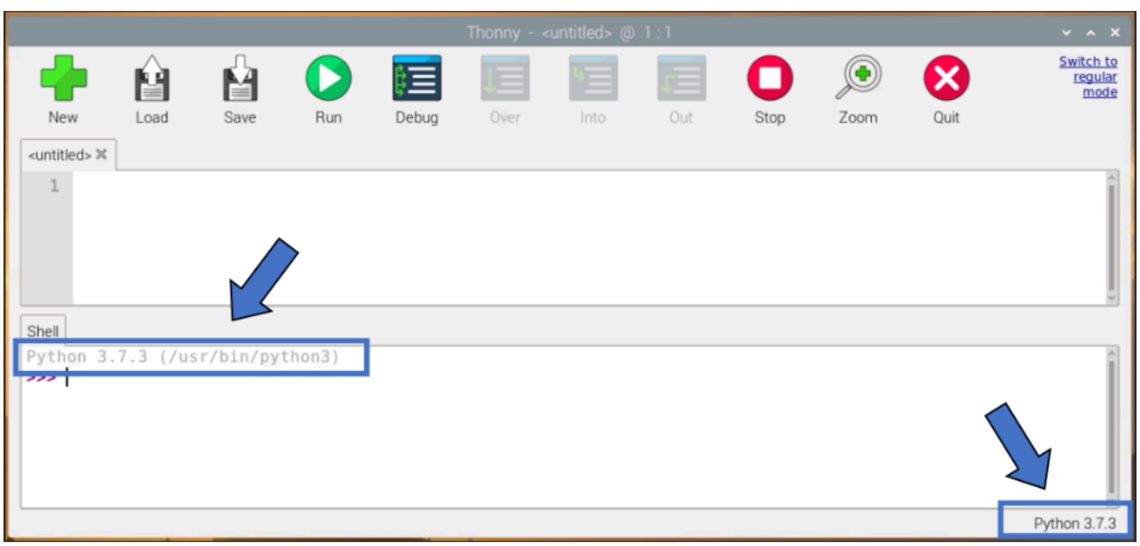
1. Access the Raspberry Pi via VNC, for instance via VNC Viewer.
2. At Raspberry Pi, open the applications menu.
3. Select Programming.
4. Choose Thonny Python IDE.

Note: On picture please consider *cubotone* instead of *cubotino* (pictures taken from Cubotino instructions)

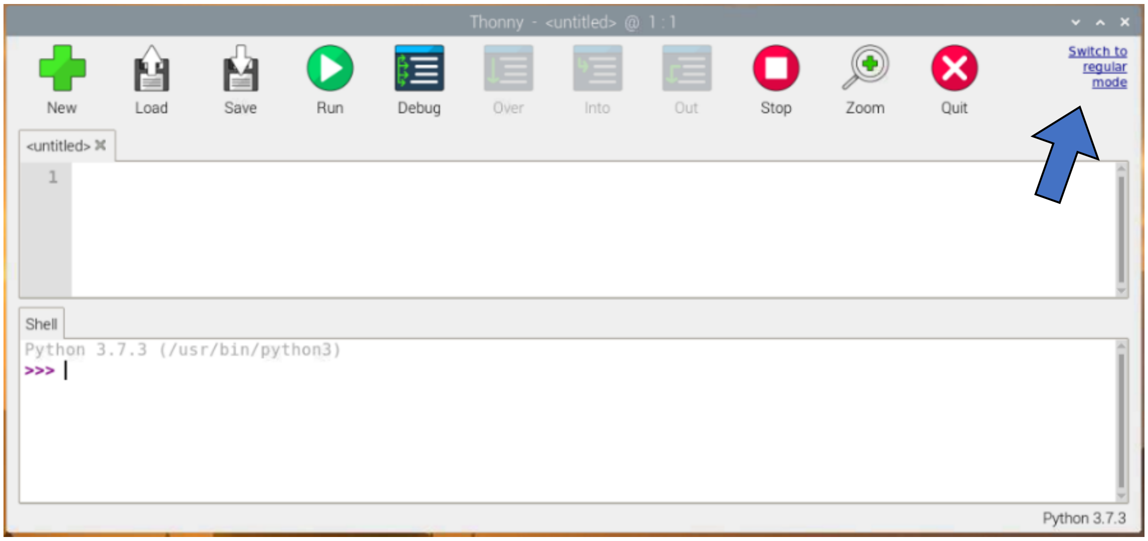


Setting up Thonny IDE interpreter, to work with the venv, it will be handy to tune the parameters hard coded in the scripts.

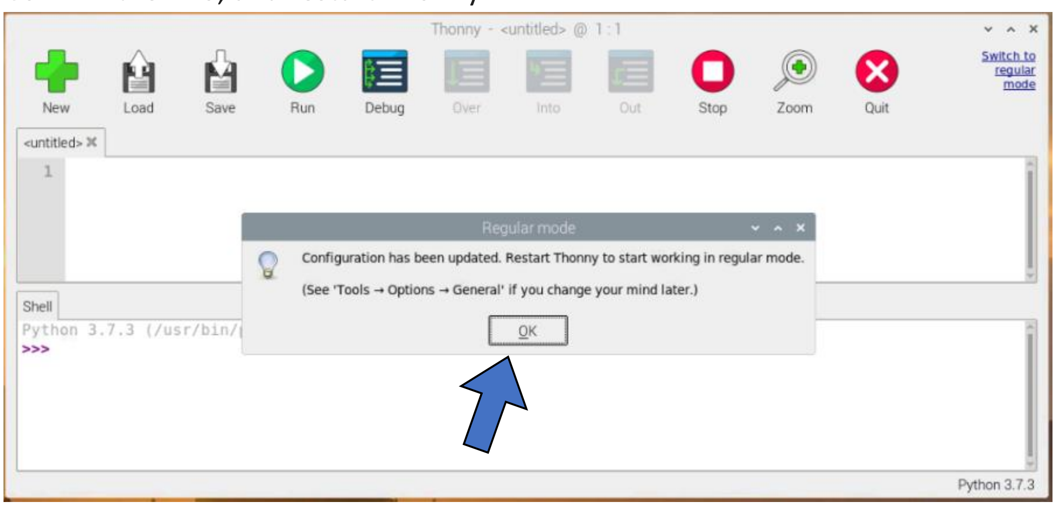
Thonny opens with the standard interpreter (/usr/bin/python3), and if you run Cubotone.py it won't find the libraries….



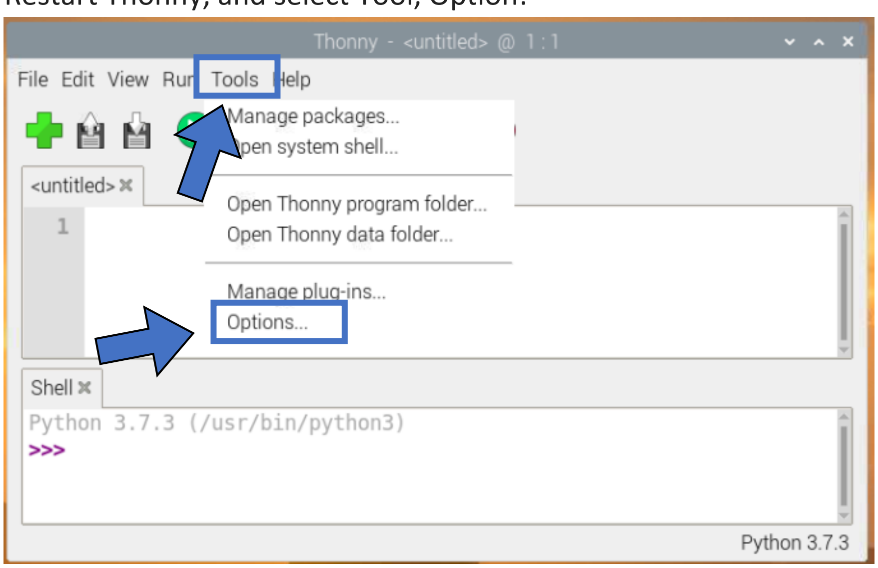
In order to have the Option menu, it is necessary to change the mode:



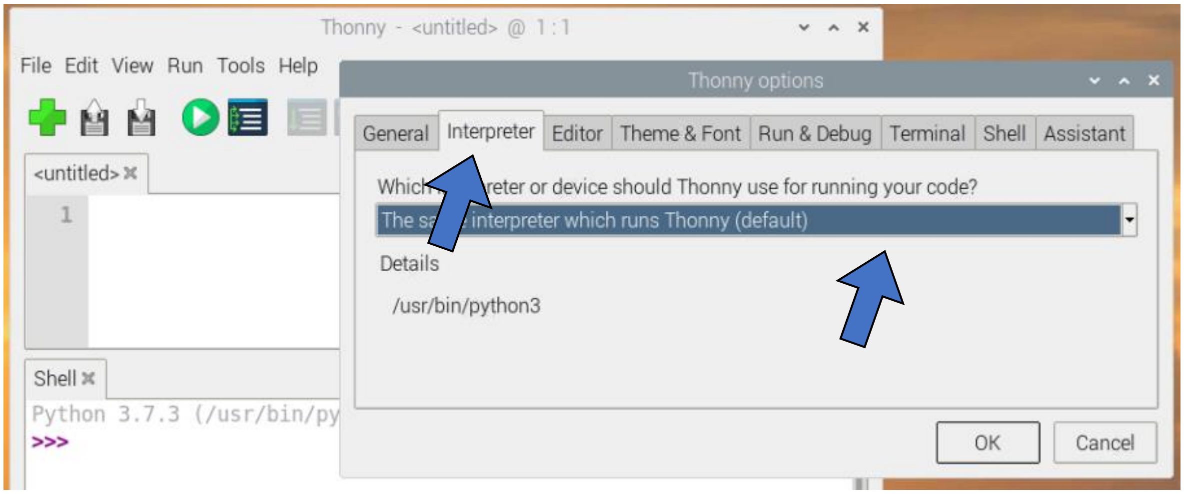
Confirm the info, and restart Thonny:



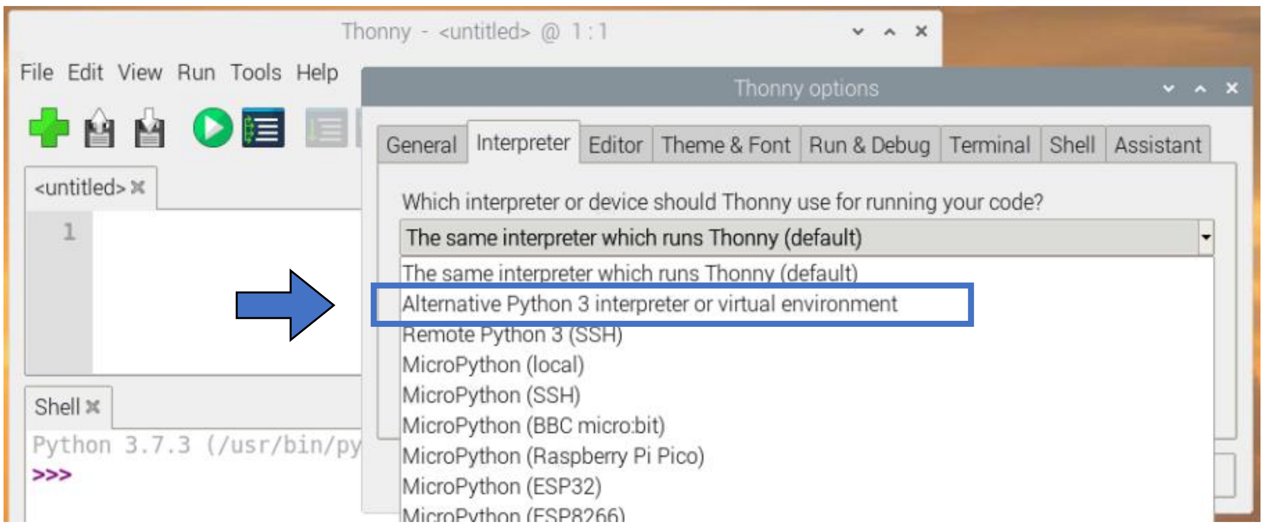
Restart Thonny, and select Tool, Option:



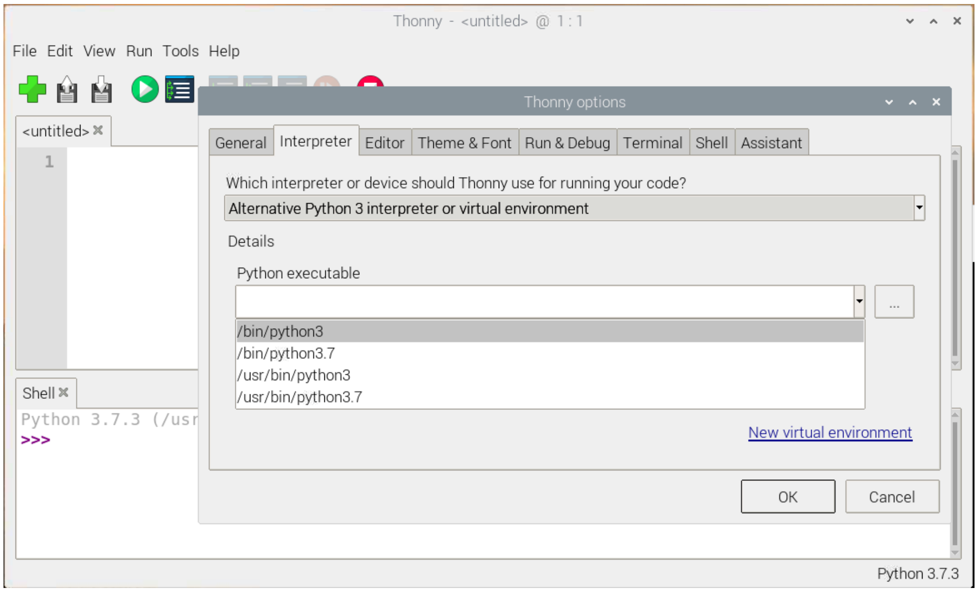
Select Interpreter where it is shown the default setting, pointing to /usr/bin/python3:



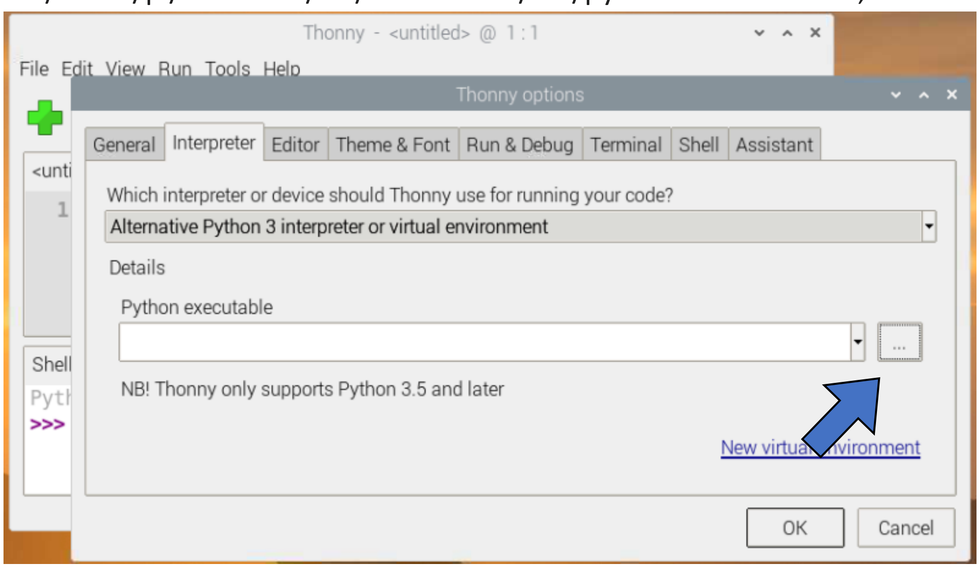
Open the drop-down menu and select 'Alternative Python 3 interpreter or virtual environment':



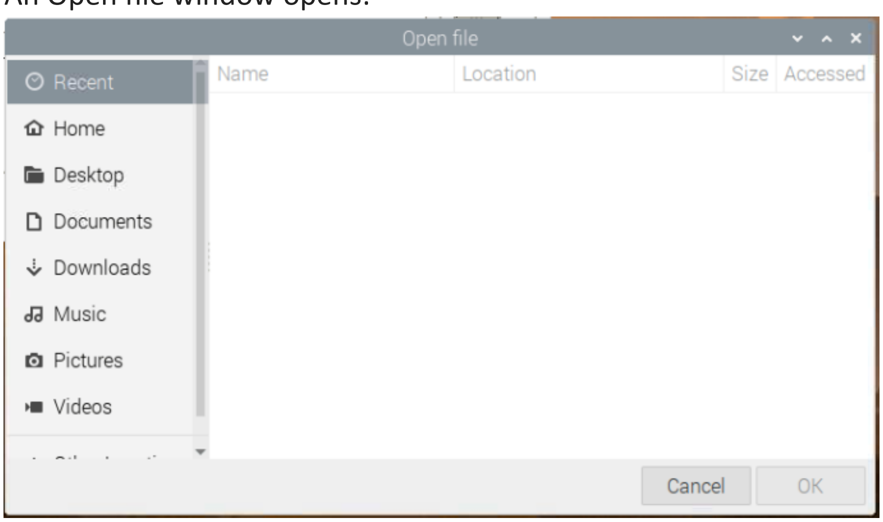
Open the drop-down menu and if '/home/pi/cubotone/src/.virtualenvs/bin/python3' is listed just select it:



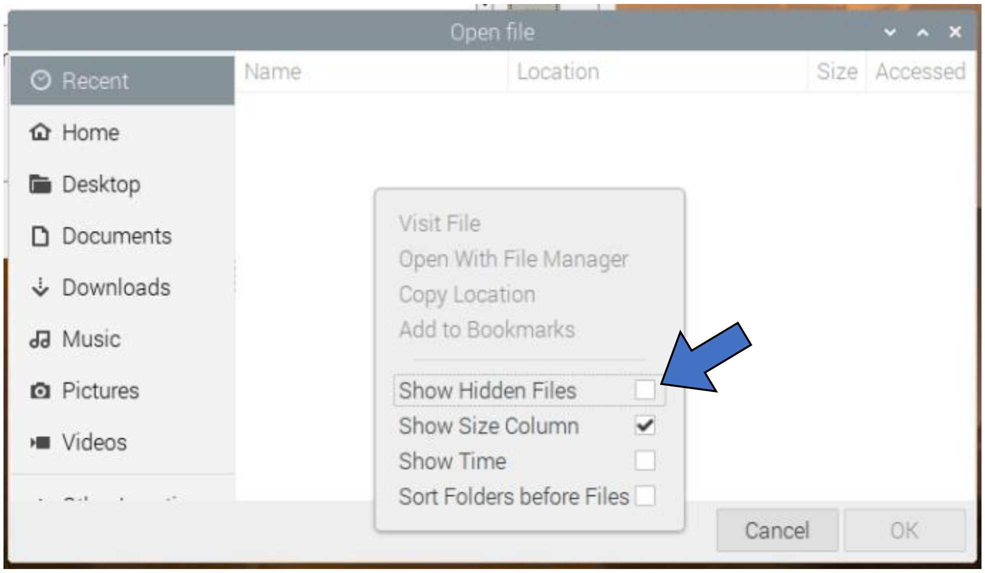
If ''/home/pi/cubotone/src/.virtualenvs/bin/python3' is not listed, select the browse button:



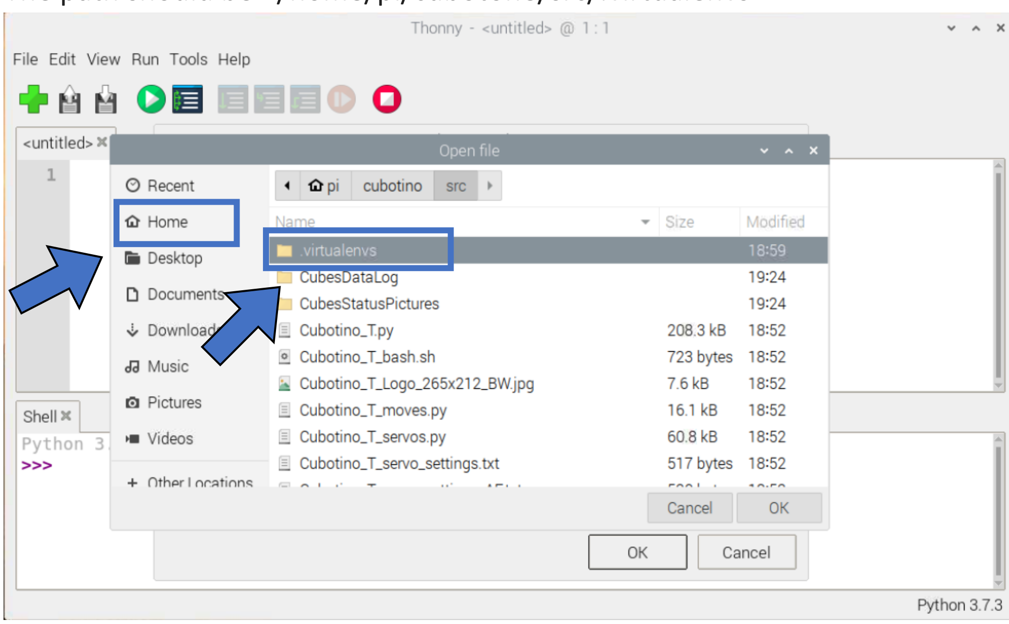
An Open file window opens:



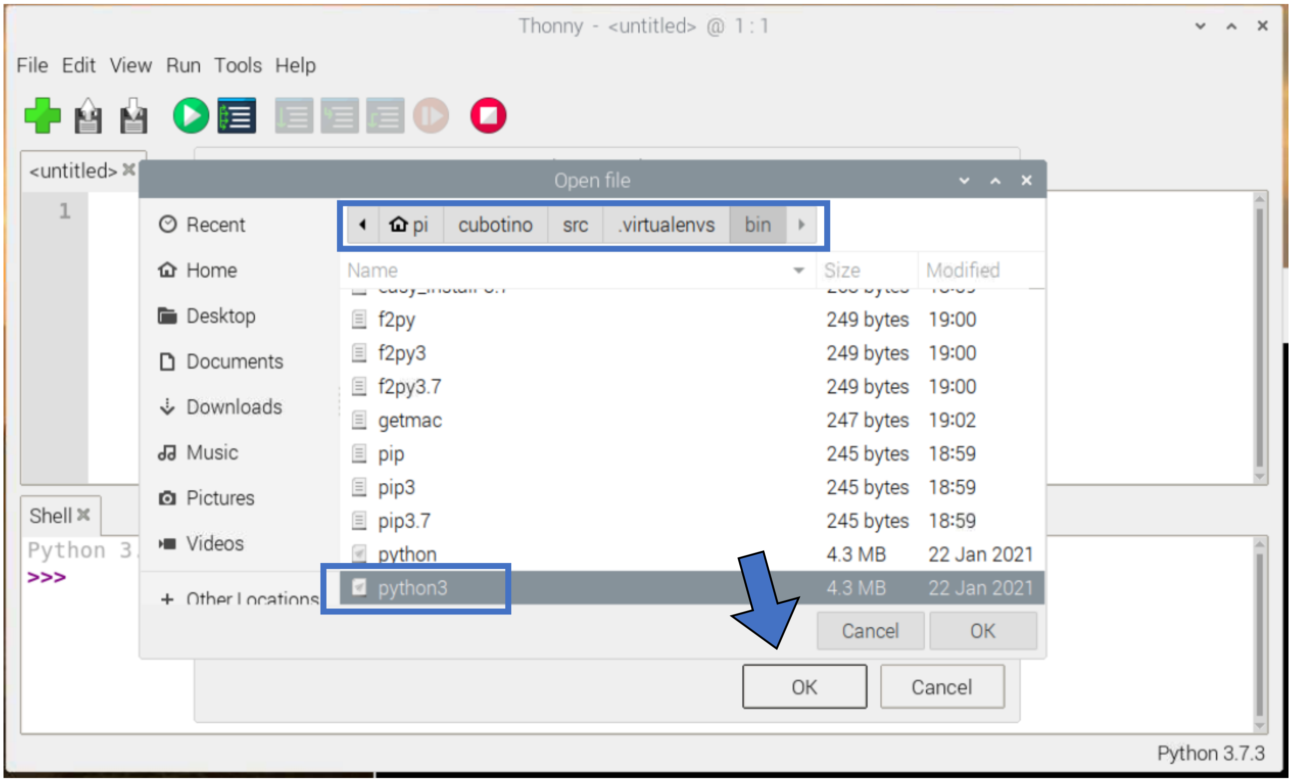
Right click on the empty window part, and check 'Shows Hidden Files':



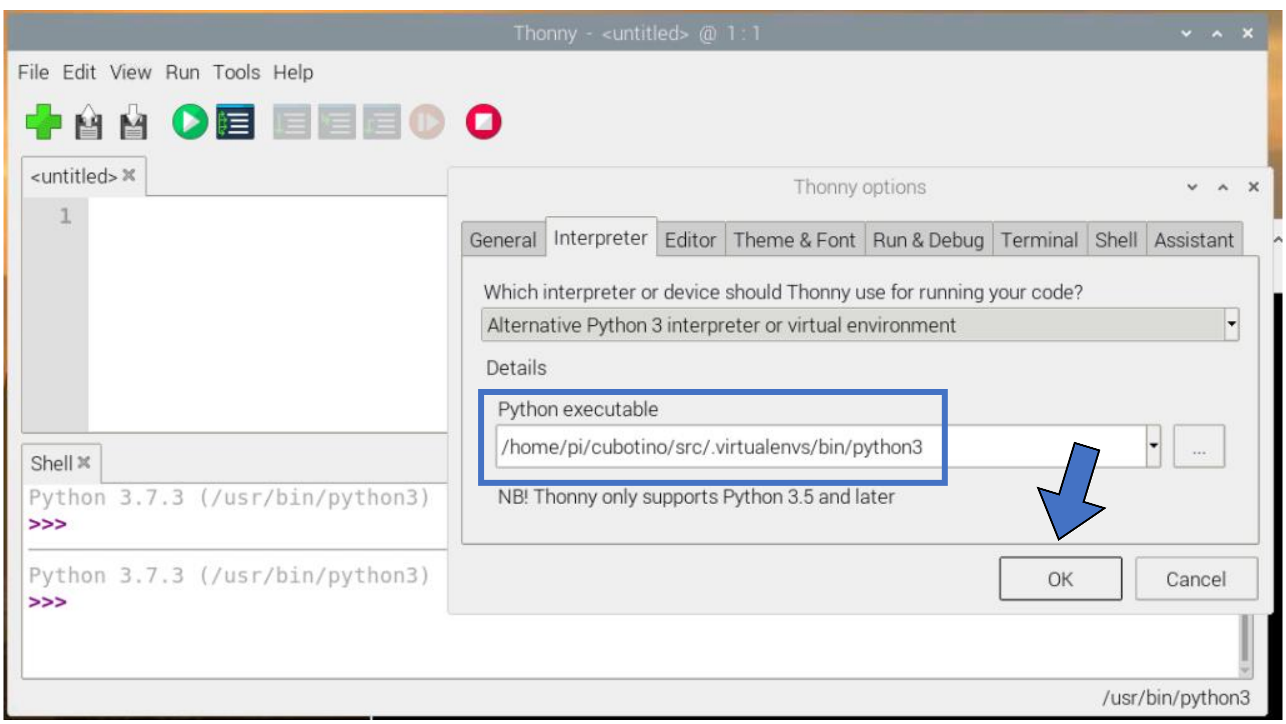
Select Home, .virtualenvs should appears (Note: all folders and files starting with a dot are hidden type)
The path should be ''/home/pi/cubotone/src/.virtualenvs'



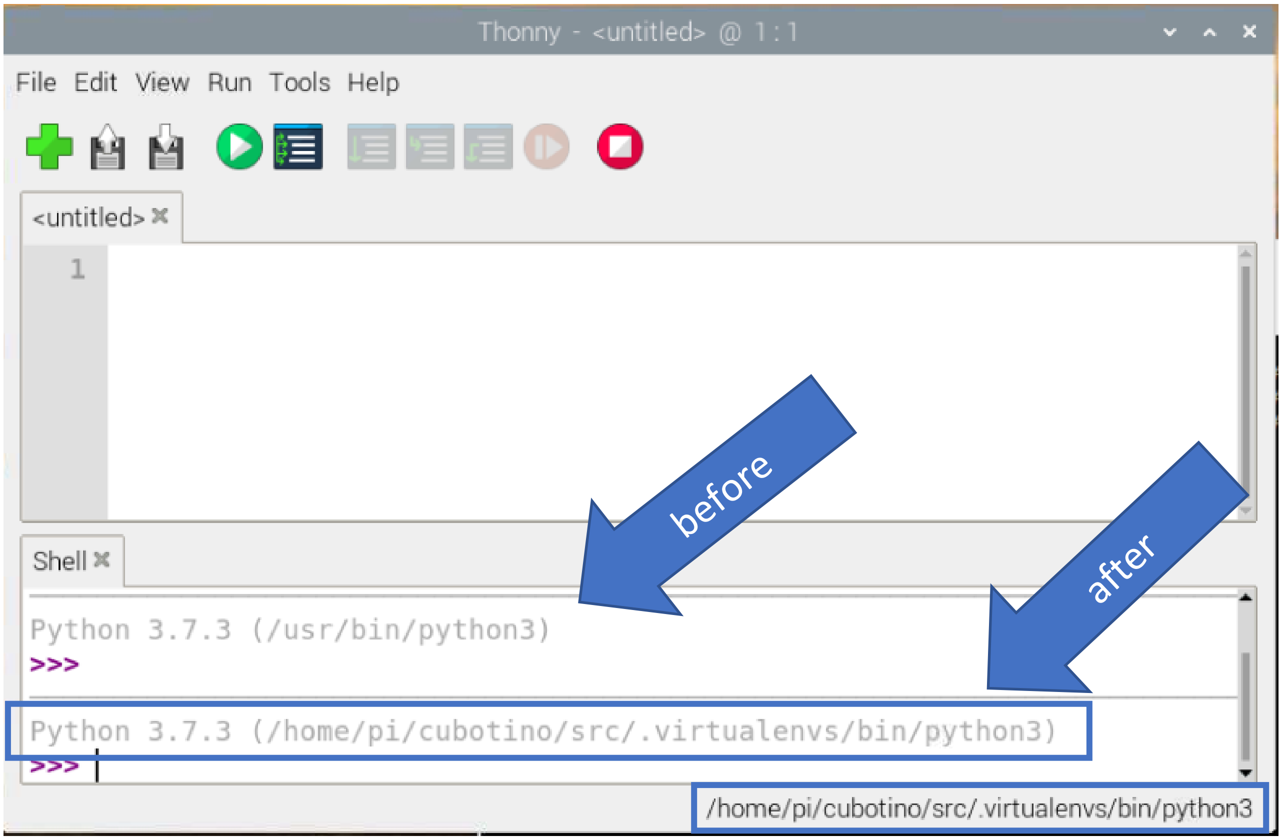
Select python3 from the path: '/home/pi/cubotone/src/.virtualenvs/bin/python3' and confirm a couple of times



Confirm one more time:



Note the interpreter is now using python3 from the venv



Notes, to get this change proposed as default:

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

10. Connect a screen to Raspberry Pi

The project has been developed without a screen connected to the Raspberry Pi, by only use the VNC viewer connected via SSH.
It is anyhow possible to connect a screen directly to the HDMI port, further than a keyboard and a mouse to the USB ports.

Steps needed:

1. From the CLI edit the config file: *sudo nano /boot/config.txt*
   Uncomment 3 rows:
   - hdmi_force_hotplug=1
   - hdmi_group=1
   - hdmi_mode=16

2. Change configuration: *sudo raspi-config*
   Select
   - System Option
   - Boot /Autologin
   - select B4 (Desktop GUI, automatically logged in as 'pi' user

3. Connect the mouse, keyboard and monitor to the Raspberry Pi
4. Reboot: *sudo reboot*

- Now the screen should be controlled by the Raspberry Pi HDMI port.
- Activate the venv: *source .virtualenvs/bin/activate*
- Run the python script: *python Cubotone.py*

Enjoy the robot computer vision part, without using a PC

12. 3D printed and other parts

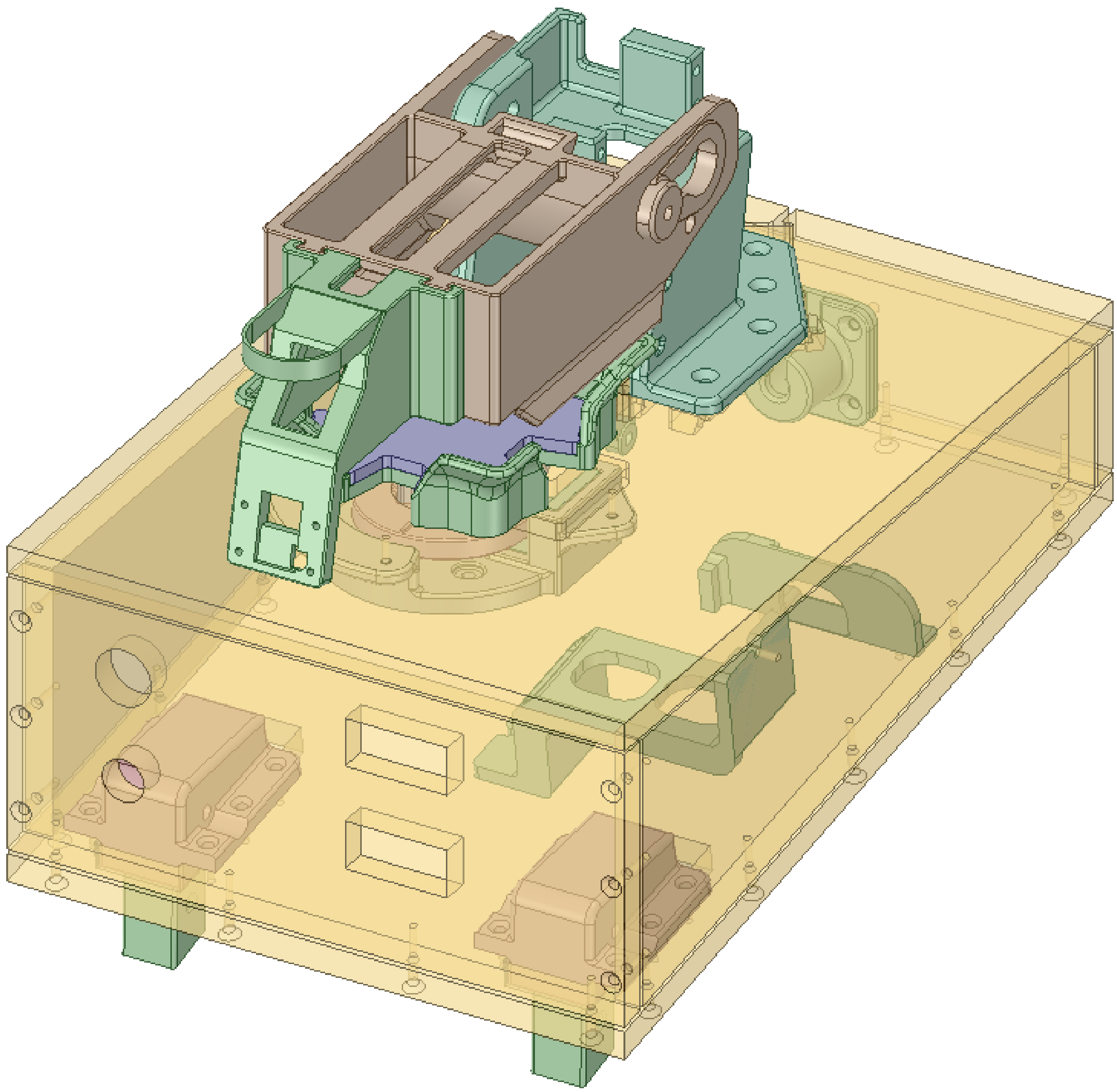
Q.ty	Part	Material	Notes
1	Box (6 stl files)	Plywood or 3D print	8mm thickness **Note**: Large difference on top panel's thickness might require changes at parts geometry
2	Foot	3D print	
2	Foot hinge	3D print	
1	Inner connector holder	3D print	
1	Outer connector holder	3D print	
1	Raspberry Pi front holder	3D print	
1	Raspberry Pi back holder	3D print	
1	Motor support	3D print	
1	Synchronization disk	3D print	
1	Cube-holder upper part	3D print	
1	Cube-holder bottom part	3D print	
1	Lifter servo holder	3D print	
1	Lifter	3D print	
1	Lifter-link	3D print	
1	Hinge for Upper_cover and Lifter	3D print	
1	Upper-cover	3D print	
1	PiCamera holder	3D print	

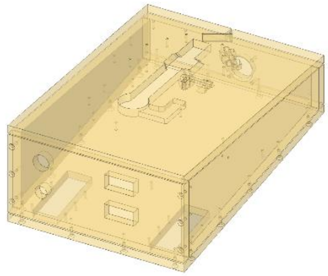
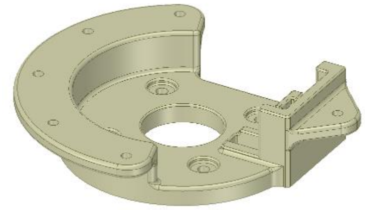
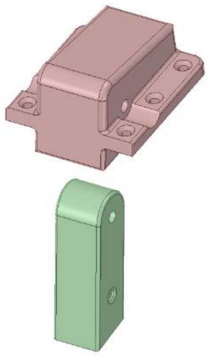
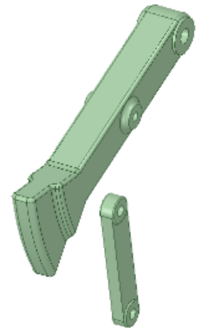
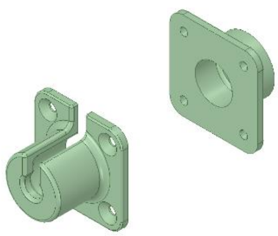
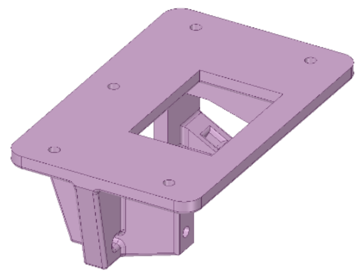
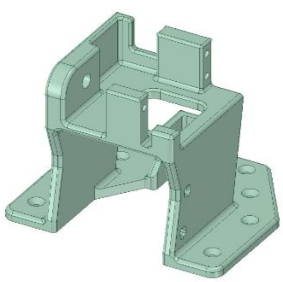
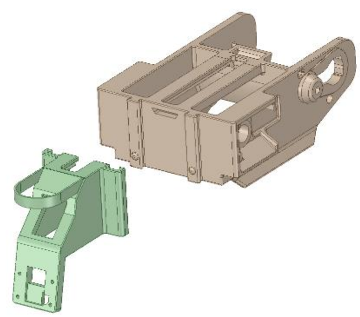
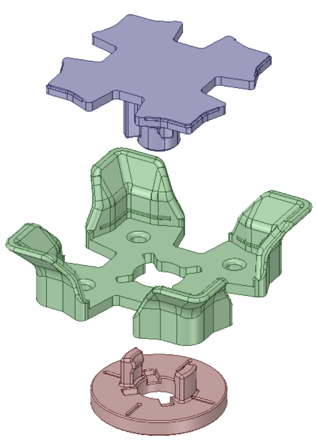
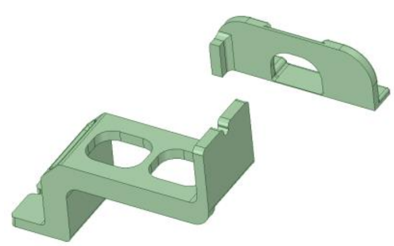
Stl files for the above parts are provided at:

1. instructables: https://www.instructables.com/Rubik-Cube-Solver-Robot-With-Raspberry-Pi-and-Pica/`
2. GitHub: https://github.com/AndreaFavero71/Cubotone/tree/main/stl

To download files from a GitHub folder:

1. Via this link a helpful online tool opens: https://downgit.github.io/
2. Paste the address of the folder to be downloaded.
3. Click Download, and a zip file of the folder will be downloaded into your download folder.



Box (6 stl files)		Motor support	
Foot hinge Foot		Lifter Lifter-link	
Outer connector holder Inner connector holder		Lifter servo holder	
Hinge for upper-cover and Lifter		Upper-cover PiCamera holder	
Cube-holder bottom part Cube-holder upper part Synchronization disk		Raspberry Pi back holder Raspberry Pi front holder	

13. Assembly steps

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

Assembly steps and attention points:

Preparation and pre-checks, to be done before final assembly:

1. *Cube-holder bottom part* must enter the *motor* axis (the flat part must match!):
    a. If there is too much friction, pass wax candle over the *motor* axis.
    b. It must be possible to insert fully these 2 parts, resulting in ca 28mm clearance between the *motor* flange and the *Cube-holder upper part* large surface.
2. *Photo Switch* must enter the *motor support.*
3. *PiCamera holder* must be fixed to the *Upper-cover*; Insertion direction is the *PiCamera* sliding down to the *Upper-cover.*
4. *Servo's metal lever* must match the seat on the *Upper-cover*.
5. Orient the *servo*'s outlet, by using the *servo's metal lever "25T"*, to have enough stroke toward the direction later required.
6. Assemble the *servo* for the *Lifter*, to the *Lifter servo holder* (3 bolts M3x10mm).
7. Assemble the *Lifter-link* to the *Lifter* (M3x16mm, insertion from right to left by standing in front of the robot); Make sure the *Lifter* can rotate without friction excess.

Final assembly:

Cube-holder, motor, Photo Switch

8. Prepare the *Box*.
9. Assemble the *motor* to the *Motor support* (4 bolts M3x8mm).
10. Press the *Cube-holder bottom part* to *the Cube-holder upper part*: There are some small ribs and undercuts all around, to hold these two parts well together. The two parts should have no visible gap in between. If these parts are too loose, apply some glue.
11. Inserts the *Cube-holder* through the *Box top panel.*
12. Insert the *Synchronization disk* to the *cube-holder*; These two parts have a couple of ribs/recesses suggesting the right orientation. If these parts are too loose, apply some glue.
13. *Motor support* and *Cube-holder* to the *Box top panel*; This step requires a bit more attention:
    a. Place the *Photo Switch* on the *Box top panel* recess.
    b. Orient the *Motor axis* to match the *Cube-holder bottom part*.
    c. Slide the *Motor* + *Motor support* completely, while keeping the *Cube-holder*.
    d. Keep the *Motor support* forced toward the *Box top panel* and verify that *Synchronization disk* rotates without touching the *Photo switch* and other parts (i.e., *Motor bolts*).
    e. Screw the *Motor support* to the *Box top panel.*

Lifter and Lifter servo

14. Fix the *Hinge for Upper_cover and Lifter* to the *Box top panel*.
15. Fix the *Lifter* to the Hinge for Upper_cover and Lifter (M5x30mm and self-locking nut).
16. Fix the *Lifter servo holder* + *servo* to the *Box top panel*.
17. Fix the *servo "25T" metal lever* to the servo for *Lifter* (M3x4mm), and tight the tangential screw.
18. Fix the *servo "25T" metal lever* to the *Lifter-link* (M3x10mm).

Upper cover and related servo

19. Fix the *servo "25T" metal lever* to the *servo* for *Upper_cover* (M3x4mm), and tight the tangential screw.
20. Fix the *servo "25T" metal lever* to the *Upper_cover* (M3x10mm).
21. Slide the *servo* + *Upper_cover* onto the *Hinge for Upper_cover and Lifter*; Fix the servo with 3 bolts M3x10 (one of the three screw can be tighten via the hole on *Upper-cover*, when placed in vertical position.
22. Place and tight the M6x16mm bolt (opposite side of the Upper_cover servo lever), that acts as fulcrum for the *Upper-cover*. The bolt must be fixed to the *Upper-cover*, and free to rotate on the *Hinge for Upper_cover and Lifter.*
23. Pass the servo cable through the Box Top panel hole; Keeps the cable out of the Lifter way, by making a nice "L" bend to the right and fix the cable with two small tire wraps.

Electrical part

24. Place and fix the remaining electrical parts into the Box.
25. On the two *Displays*, it’s convenient to de-solder the connector and re-solder it on the opposite board side; This makes easier the displays placement on the *Box front panel*, as well as the connection with the wiring.
26. Fix *the Raspberry Pi.*
27. Complete the connections.

PiCamera

Connecting the PiCamera and its cable should be done as one of the latest actions.
Before fixing the *PiCamera* to the *PiCamera holder*, fix the camera to the board via the self-adhesive tape.

28. *PiCamera* must be fixed to the *PiCamera holder*, by using 4 little pieces of filament, Ø1.75mm; Deform the protruding parts with a hot blade. Do not insert the *PiCamera flat cable* to the parts yet.
29. Slide the PiCamera flat cable (50cm long) along the slot on the *Upper_cover* and *PiCamera Holder*
30. Pass the PiCamera flat cable though the *Box Top panel* opening.
31. Connect the cable to the Raspberry pi; Electrical contact are on one side only of the cable (many tutorials helping on this).

14. 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 Upper_cover 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 Upper_cover angle of this robot.
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 Cubotone_T_servos_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)

3. Cube-holder rotation speed:

This robot isn't very fast, it typically detects the cube's status and solves it within 1 minute.
Stepper motor torque decreases while increasing the speed.
Depending on the motor torque, input voltage, max driver current, and cube rotation friction, it might result in loosing steps by the motor; In that case a lower speed might solve the issue.
The script provided with this instruction has been tuned to get the max possible speed, on my setup; It will be convenient to use a lower speed at the beginning, and progressively increase it.

4. Lifter and Upper_cover (servos) angular speed:

As per the step motor, also the servos don't provide feedback when they have completed the requested angular rotation.
The script that controls the servos, have some delays at each servo activation.
The script provided with this instruction has been tuned to get the max possible speed, on my setup; It will be convenient to use larger delays at the beginning, and progressively reduce them.

5. Reference angles for servos:

The servos I bought, have 180-degrees of rotation, that is more than sufficient for the (lifter and Upper-cover) angles of this robot.

The point is that the connection between the metal arm "25T", and the servo's outlet gear, have many possible positions; This means the reference angles set on Servo_and_Motor.py are likely not the same on other systems.
To tune these parameters on your system, the advice is to load only this script, and to adjust one parameter at the time; At *Servo_and_Motor.py* beginning, there are the angles of reference for the different parts and needs.
At the end of the script (__name__ == "__main__"), there are some examples I've used while tuning my system.

15. Parameters and settings

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

- Cubotone_settings.txt and Cubotone_servos_settings.txt

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

- Cubotone_settings_AF.txt and Cubotone_servos_settings_AF.txt

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

Cubotone_settings.txt (and Cubotone_settings_AF.txt):

Parameter (dict key)	Default value	AF value	Data type	Info
camera_width_res	800	800	Int	Picamera resolution on width.
camera_height_res	544	544	int	Picamera resolution on height.
zoom_x	0.0	0.0	float	Image cropping at the left (proportion of camera width).
zoom_y	0.0	0.0	float	Image cropping at the top (proportion of camera height).
zoom_w	1.0	0.75	float	Image cropping width (proportion of camera width).
zoom_h	1.0	0.82	float	Image cropping height (proportion of camera height).
kl	0.95	0.95	float	Coefficient for PiCamera stability acceptance. Lower values are more permissive (range 0 to 1).
scale_perc	65	65	int	Scale (%) to reduce the cube window size on screen.
square_ratio	2	2	float	Facelet contour squareness check filter. Larger values are more permissive (0 is perfect square, 2 is rather permissive).
rhombus_ratio	0	0	float	Facelet contour rhombus check filter. Smaller values are more permissive (1 is perfect Rhombus, 0 is rather permissive).
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	0.1	float	Cropping margin (%) around the faces images to generate the unfolded cube collage.
cam_led_bright	300	300	int	PWM for the 3W led at Upper_cover. Range from 0 (no PWM) to 4095 (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.
gap_w	60	60	int	Horizontal gap in pixels, between windows on screen when cv_wow.
gap_h	100	100	int	Vertical gap in pixels, between windows on screen when cv_wow.

Cubotone_servos_settings.txt (and Cubotone_servos_settings_AF.txt)

Notes: Time related parameters are in seconds

Parameter (dict key)	Default value	AF value	Data type	Info
cover_close	395	395	int	PWM (range 0 to 4095) for Upper_cover angle in close position. This position constraints the two top cube layers.
cover_open	328	328	int	PWM (range 0 to 4095) for Upper_cover angle in open position. This position the Cube_holder spin with the cube un-constrained.
cover_read	305	305	int	PWM (range 0 to 4095) for Upper_cover angle in read position. This position the PiCamera is rather parallel to the upper cube face.
flipper_low	405	405	int	PWM (range 0 to 4095) for flipper angle in its low position; This position the Lifter top part is 3 to 5mm below the Cube_holder
flipper_high	324	324	int	PWM (range 0 to 4095) for flipper angle in its high position; This position has to be sufficiently high to flip the cube.
time_cover_closing	0.25	0.25	float	Time for the Upper_cover to move to close position
time_cover_opening	0.2	0.2	float	Time for the Upper_cover to move to open position
time_cover_reading	0.18	0.18	float	Time for the Upper_cover to move to read position
time_flipper_low	0.35	0.35	float	Time for the Flipper to move to the low position
time_flipper_high	0.35	0.35	float	Time for the Flipper to move to the high position
time_consec_flip	0.18	0.18	float	Time for the Flipper to move to the low/high position when consecutive flips (cube holder remains constrained)
ramp	0.015	0.015	float	Time variation, at each motor step, to ramp. Smaller values reduces the acceleration.
spin_fast_ramp	0.024	0.024	float	Time variation, at each motor step, to ramp when fast speed. Smaller values reduces the acceleration.
spin_slow_ramp	0.018	0.018	float	Time variation, at each motor step, to ramp when slow speed. Smaller values reduces the acceleration.
time_on_spin_fast	0.001	0.001	float	Pulse ON time for stepper, when high speed. Smaller values reduces the speed.
time_on_spin_slow	0.0014	0.0014	float	Pulse ON time for stepper, when slow speed. Smaller values reduces the speed.
time_on_align_fast	0.0004	0.0004	float	Pulse ON time for stepper, during alignment at high speed. Smaller values reduces the speed.
time_on_align_precise	0.0012	0.0012	float	Pulse ON time for stepper, during alignment at low speed. Smaller values reduces the speed.
stp_rev	200	200	int	Stepper per revolution of the motor

extra_steps_s	8	8	int	Extra steps for stepper for “single” holder rotation (90deg), to recover the angular gaps and get the cube layers well aligned
extra_steps_m	4	4	int	Extra steps for stepper for “multiple” holder rotation (180deg), to recover the angular gaps and get the cube layers well aligned
extra_steps_align	3	3	int	Number of motor additional steps when aligning the motor
align_blind_steps	6	6	int	Number of initial motor steps, at precise alignment, to 're-enter' the synchronization disk slots
align_repeats	2	2	int	Max number of consecutive motor alignment attempts
timeout	3	3	int	Timeout in seconds while motor alignment attempt
motor_reversed	0	0	int	To reverse the motor rotation, instead of changing the wiring. The “fun” function should spin the cube_holder CW from motor point of view (CCW from user point of view). Accepted values are 0 and 1

Note:
On Cubotone.py and Cubotone_servos.py, the string ‘#(AF‘ is placed as comment start, where the above listed parameters are used. This choice because those variables weren’t initially collected in json file and were simply scattered along the code; Once the parameters were collected into json files, I decided to comment those rows instead of cancelling them.

Boolean parameters, at Cubotone.py __main__

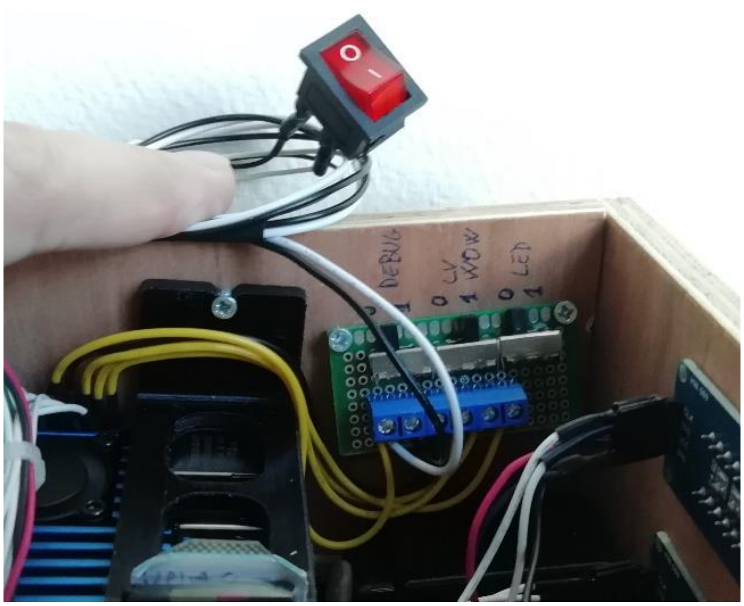
Variable	**Default value**	**Data type**	**Where**	**Info**
debug	False	Bool	PC and Rpi	enable/disable the debug related prints to the terminal; Useful during the debug phase.
cv_wow	False	Bool	PC and Rpi	enable/disable the visualization of the image analysis used to find the facelets; This is meant to make visible the image analysis steps, for educational or debug purpose
led_usage	False	Bool	Rpi	enable/disable the usage of the led lights at Upper_cover. The led usage slows down the robot, for that reason not always convenient

Additions for the Maker Faire:

The first time I brought this robot to a Fair, I wanted to be fast on changing some parameters. For that reason, I made a little board with 3 switches that override those variables:

```
if GPIO.input(16):          # case the GPIO16 is at level 1 (s1 switch is closed)
    debug = True            # debug related prints are activated
if GPIO.input(20):          # case the GPIO20 is at level 1 (s2 switch is closed)
    cv_wow = True           # visualization of the image analysis used to find the facelets
if GPIO.input(21):          # case the GPIO21 is at level 1 (s3 switch is closed)
    led_usage = True        # usage of the led lights at upper_cover
```

In addition, a 4th switch with long wires, has been connected to the GPIO.input(12)
When this switch is closed, the *cv_wow* related windows are kept on screen, and the detection timeout counter paused.
This allows to show the image analysis process, with a manual control on time, for a proper discussion with the visitors.



16. Motor alignment info

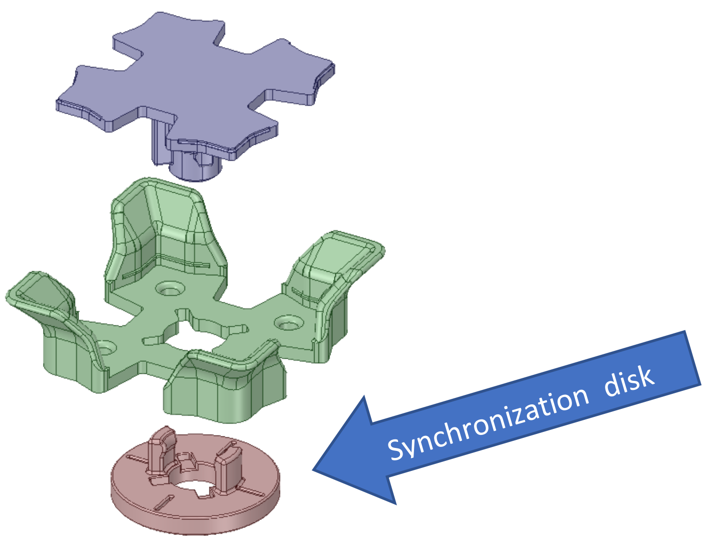
The stepper motor is supposed to reach a precise position according to the steps sent, but this is only valid if:

1) The initial position is known.
2) No steps are lost.

More in general the stepper motor doesn't feedback the SBC (Raspberry Pi) on its angular position, and it works in open loop.

For above reasons a so called "synchronization disk" is used, with below characteristics:

1) It is connected to the Cube_holder.
2) It has four slim slots, aligned to the Cube_holder recesses for the Lifter.
3) The slots are sensed by a light gate to feedback every quarter or turn.
4) The light gate is positioned in a way that when it senses a slot, the Lifter can be actuated.



At the start of a new cube solving process, each time the Lifter has to be actuated, or the Upper_cover has to be lowered, the light gate is verified; In case the Synchronization disk slot isn't under the light gate, then a motor alignment procedure starts.

Motor alignment procedure.
The motor is energized in CW direction (motor point of view, CCW from user point of view) with the expectation the light gate will sense the slot within a certain number of steps.
If the slot isn't found within $1/8^{th}$ of turn, the motor changes direction and it keeps rotating until a slot is detected (or timeout).
Once the light gate senses a slot, the motor alignment process enters a "precise alignment" mode by lowering the motor speed.
At this point the aim is to measure the slot width, by searching the slot edges and by positioning the motor at the slit centre.

Notes:

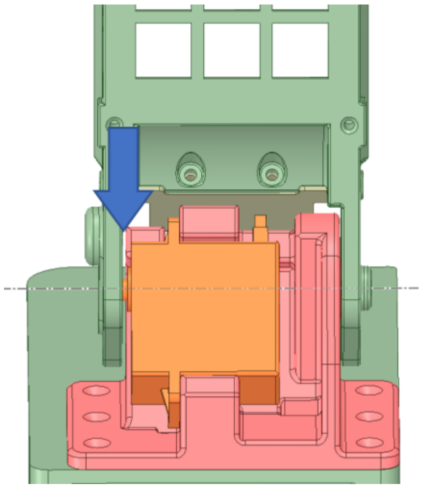
- The process makes a second attempts if the first one fails; A timeout stops the motor in case the robot/cube are stuck.
- You'll notice the alignment process being performed during the cube solving process, when the Cube_holder wasn't perfectly aligned; I do have the feeling the slots are too narrow and excessively restrictive.
- Because I could find a working method with the 1st Synchronization disk I printed, I did not try to enlarge the slots.
- To ensure reading the slot edges properly, there are some "extra steps" parameters that might need to be tuned.

17. Troubleshooting

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

1. Servos not moving smoothly.
2. Bottom cube layer doesn't align nicely.
3. Top cover usage to flat the cube.
4. Cube status detection error.
5. Robot stuck on reading the same face.
6. Cube's facelet and light reflection (cube status detection).
7. Program doesn't work as intended.
8. Error while setting up the Raspberry Pi.
9. Cube doesn't flip.

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 a 20 to 25Kg/cm servo.
    5. Minimize Upper_cover rotation friction:
        i. Ensure the hole for the M6 screw (pivot) has some gap on the Hinge hole (Ø6.1 to Ø6.3mm).
        ii. Rub some candle wax on the screw.
        iii. Ensure gap presence between Upper_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 Upper_cover

2. Bottom cube layer doesn’t align nicely:

   Adjust the “extra_steps_s” and “extra_steps_m” parameters, to force more/less extra rotation of the cube holder.

   Extra rotation of the cube holder is necessary to recover the play between cube and cube_holder and between cube and Upper_cover.

   “extra_steps_s” is the parameter used when the cube_holder makes a single rotation (90deg) .

   “extra_steps_m” is the parameter used when the cube_holder makes a multiple rotation (180deg).

   Take a movie with the phone, and play it a few times to visualize when the cube doesn’t align nicely.

3. The Upper_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 Upper_cover to level the cube. By lower the Upper_cover close position to have a little interference with the cube, will improve the cube layer alignment in particular after flipping the cube.

4. Cube detection error (“Err” on display2):

   This happens when the interpreted cube status isn’t coherent, meaning not having 9 facelets per colour or other inconsistencies.

   Possible causes:

   1. The camera also reads the back cube face: Adjust the Upper_cover angle and/or the camera cropping.
   2. Table background is the cropping is not applied: 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.
   3. 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.
   4. Too little light conditions cannot be compensated by the LED light source.
   5. In case the cube has some prints (i.e. brand) , typically on the white centre, it is suggested to carefully scratch out.
   6. The cube has been manipulated, ie. labels misplaced or a corner cubie turned.

5. Robot stuck on reading the same face, until timeout:
    a. If the robot doesn't change the cube face, it is because some of the pre-conditions aren't met (at least 5 facelets, areas of the facelets, distance between the facelets, etc)
    b. When the ambient light is rather low and the U face is rather clear: Increase the ambient light or change the cube orientation to allow the camera to set to a more "balanced" face.
    c. When the ambient light is rather hight and the U face is rather dark: Decrease the ambient light or change the cube orientation to allow the camera to set to a more "balanced" face.

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

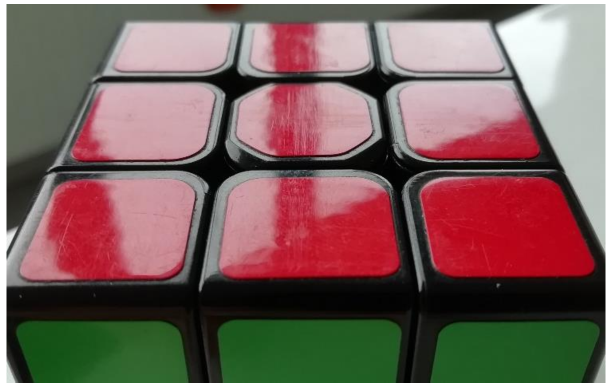
A contour will be drawn, over the camera image, on every location interpreted as facelet (excess of contours are filtered out, lack of contours is critic...)
This should help to have an understanding on the reason, or reasons, the robot stuck on the same cube face.

Possible reasons for the facelets detection failure:
A) the camera doesn't see the complete top face of the cube: In this case change the camera orientation angle, via Upper_cover angle, or reduce/adjust the cropping.
B) facelets on the back cube side are also detected (detected means that on the image contours are drawn on the back cube facelets): In this case change the camera orientation angle, via Upper_cover angle, or reduce/adjust the cropping.
C) the critic face has a logo on the central facelet: Carefully scratch that out or cover it.
D) too low light conditions: Increase ambient light.
E) light reflection: Avoid localized light source from the ceiling, better from the side or even better if diffused. Consider the option to make matt the facelets.

6. 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.

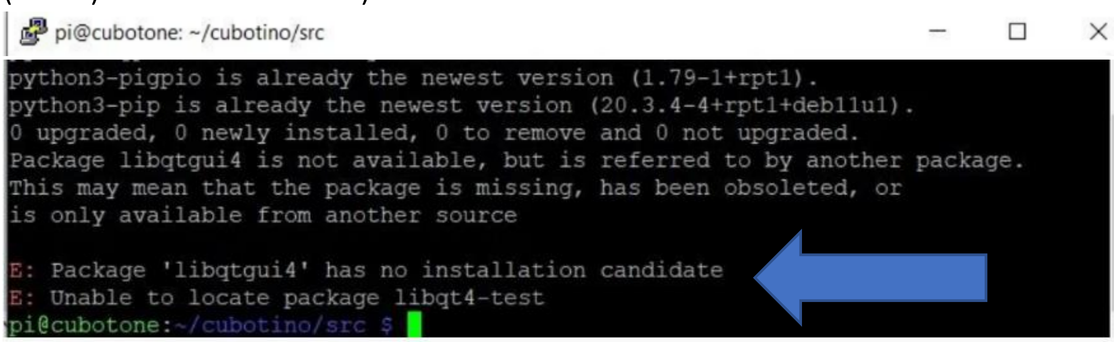
Cube with in-moulded coloured facelet, that I've made matt with sandpaper (grit

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



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 Cubotone.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 Cubotone.py
   d. Check the prints on the terminal
   e. If the print out doesn't suggest much to you, share it at the Instructables chat

8. Error while setting up the Raspberry Pi:
   If you flash the microSD card with the default OS proposed by Raspberry Pi Imager ("Raspberry Pi OS (32-bit)" in November 2022) then an error is returned:



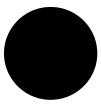
```
pi@cubotone: ~/cubotino/src
python3-pigpio is already the newest version (1.79-1+rpt1).
python3-pip is already the newest version (20.3.4-4+rpt1+deb11u1).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
Package libqtgui4 is not available, but is referred to by another package.
This may mean that the package is missing, has been obsoleted, or
is only available from another source

E: Package 'libqtgui4' has no installation candidate
E: Unable to locate package libqt4-test
pi@cubotone:~/cubotino/src $
```

To solve the problem, flash the SD card by selecting the sub-menu "Raspberry Pi OS (other)" and then scrolling down and selecting “RASPBERRY PI OS (LEGACY)”: This is the operative system to be used (as per detailed info on “Setting up Raspberry Pi 4b”.

9. Cube doesn’t flip:
   If the cube doesn’t flip, despite the lifter raising completely, it might depend on excessive friction between the cube and the cube_holder; Possible actions:
   - Polish the cube_holder base surface.
   - Increase the robot inclination, i.e. by adding a spacer underneath the robot’s front feet.
   - Improve the cube layers alignment.

18. How to operate the robot

- SHUTDOWN (led goes OFF)
- BOOT (led goes ON in 30 secs)



- START
- STOP (0.5s < press < 5 sec)
- RESET (>5 sec press)

Display 1

- LOADING STATUS
- TIMER

Display 2

- INFO
- COUNTDOWN MOVES

SHUTDOWN: Closes the Raspberry Pi OS, wait 15 seconds before to unplug the robot.
BOOT: After a shutdown, a short press starts the boot (takes 30secs to complete); Boot starts automatically after the robot is plugged.
START: Only when Display2 shows Press.
STOP: Stops the robot, at any moment, when it is "moving"; If the button is released within 5 seconds the python script is reloaded and the robot gets ready again for a new cycle.
RESET: Ends the python script. Raspberry Pi OS remains up (use SHUTDOWN before unplugging).

After the Raspberry Pi shutdown, detach the power supply, and gently lower the Upper_cover.

The robot has 2 main working modes:

A) Without any screen connected; This is the default mode, that starts once the robot is energized.
B) With a screen connected, via hdmi cable or via SSH.

To quit mode A) or B), press STOP for at least 5 seconds. Displays segments go full ON and OFF shortly after yet let wait for at least 5 seconds.

To start mode B):

- Quit mode A).
- Connect to the Raspberry Pi (i.e. with VNC viewer, via SSH).
- Enter cubotone/src folder from the root type: *cd cubotone/src*
- Activate the virtual environment: *source .virtualenvs/bin/activate*
- Run the python script: *python Cubotone.py*

Note that Cubotone.py script can be launched with arguments, see next page.

Below arguments can be added (without parameters):

- - -help (lists the possible arguments with a short explanation)
- - -debug (printout info and variables for debug purpose)
- - -led (activates the LED modules at Top_cover)
- - -cv_wow (shows the image processing steps, it requires a FHD screen/setting)
- - -delay (when Cubotone scripts are used on a laptop)
- - -scramble (to start a scrambling function,)
- - -picamera_test (tests the camera function)
- - -no_motors (excludes motor and servos for test purpose)

Below arguments can be added (with a parameter):

- - -size N (cube size deviation in tents of mm, from reference cube used for settings. Compensates the Cube_holder rotations for Rubik's cube size differences. **N is an integer and can be positive or negative**; Mandatory sign only for negatives).

Notes:

1. Arguments don't have an order.
2. Most of them can be combined.

   Examples:

   python Cubotone_T.py - -size -5 (compensation for a cube 0.5mm smaller)

   python Cubotone_T.py - -size 5 (compensation for a cube 0.5mm bigger)

   python Cubotone_T.py - -debug - - cv_wow

   python Cubotone_T.py - -cv_wow - -led

   python Cubotone _T.py - -picamera_test - -no_motors

3. These arguments can be added to the Cubotone_bash.sh file in order to get the arguments automatically uploaded when the script starts at Raspberry Pi boot.

19. Cube scrambling function

The Cubotone_scrambling.py is a scripts that allows to scramble cubes:

- Enter cubotone/src folder from the root type: *cd cubotone/src*
- Activate the virtual environment : *source .virtualenvs/bin/activate*
- Run the python script: *python Cubotone_scrambler.py -- moves n* where n is an integer starting from 0.

When 0 is used as argument the robot scrambles the cube with a pre-defined sequence of 23 moves (always the same moves); This approach is useful when trying to speed up the overall robot solving time, by playing with different parameters, by always having the same cube starting condition.

When n is >0, the robot applies random movements, for an equivalent of n cube movements.

Once the scrambling sequence is terminated, the script asks for another n quantity of moves, as this function can also be applied to check how well the robot manipulate the cube.
This script can be quitted by entering any letter.

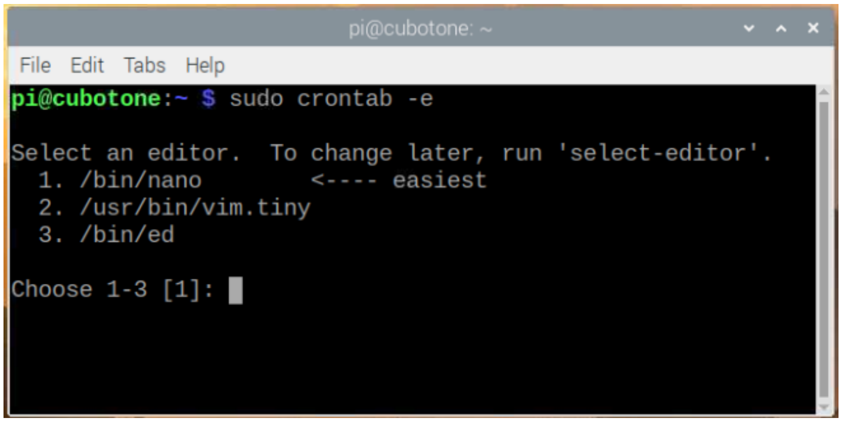
This function can be also used to check the servo/motor settings for cube layers alignment, without the need to first detect for the cube status.

20. Automatic robot start

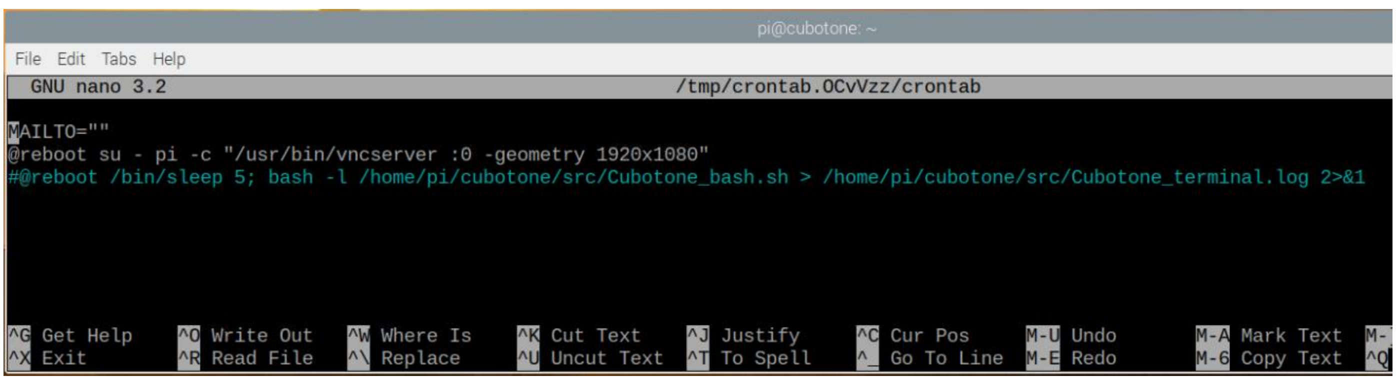
It is possible to have the robot starting up automatically once the Raspberry Pi boots.

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

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



Un-comment the last row



MAILTO=""

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

#@reboot /bin/sleep 5; bash -l /home/pi/src/cuboone/Cubotone_bash.sh > /home/pi/cubotone/src/Cubotone_terminal.log 2>&1'

Note: Eventual arguments (see "How to operate the robot") you'd like to get at the automatic robot start, can be added to the *Cubotone_bash.sh* file.

Example: change "*python Cubotone.py*" to "*python Cubotone.py - -size 5*"

The following chapters where initially at the instruction start, introducing and explaining the project bases.
Since September 2023, I changed the order to prioritize the robot construction, in line with maker's interest: If you find weird the instructions order, at least you know the reason.
I still hope the below information will be useful and appreciated 😊

21. Project scope

Despite I'm still a programmer beginner, I wanted to learn Computer Vision and keep on learning Python.
I've used Arduino boards before, and wanted to learn about Raspberry Pi
By turning 50-year-old (2021), while discovering I like to keep myself busy with coding and controls, I thought a Rubik cube robot solver to be a good, and challenging, project.

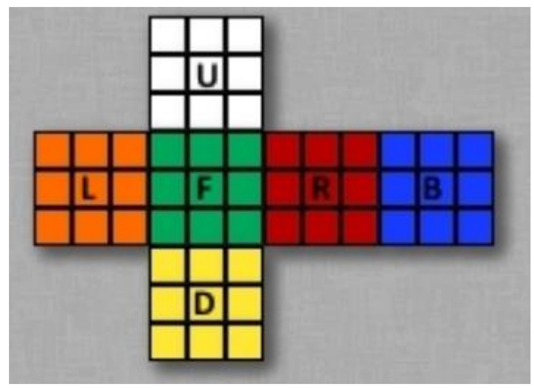
22. Robot name

This project got a name only after a long time, to be more precise only once I decided to update this project with the learnings from a successor project.
The chosen name for this robot is CUBOTone.
The name is the combination of **CU**be + ro**BOT** + **one**, wherein 'one' is the Italian suffix for augmentative.
This robot isn't huge but its successor CUBOTino is so much smaller, and CUBOTino micro is just a small fraction of it.

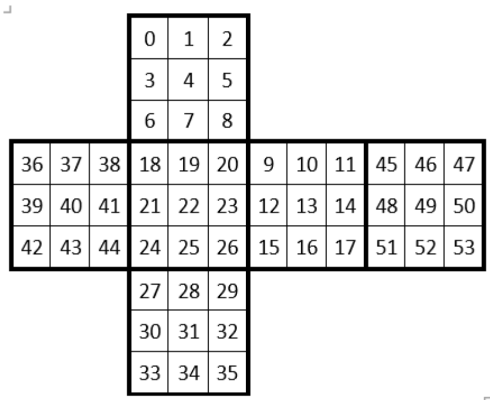
23. High level robot info

1. The robot is based on a Raspberry Pi 4B 2Gb RAM with a 16Gb microSD.
2. A PiCamera (ver 1.3) is used to detect the cube status.
3. Python CV2 library is used for the computer vision part.
4. A led module is used to reduce the influence from the ambient light conditions.
5. All coded in Python.
6. The same main script also works on a PC with webcam, to detect the status and to return the cube solution.
7. 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
8. Cube notations are from David Singmaster, limited to the uppercase (one "external layer rotation" at the time): https://en.wikipedia.org/wiki/Rubik%27s_Cube#Move_notation
9. Cube's orientation considers the Western colour scheme (https://ruwix.com/the-rubiks-cube/japanese-western-colour-schemes/):

   The Western colour 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
10. Cube solver uses the Hegbert Kociemba, "two-phase algorithm in its fully developed form with symmetry reduction and parallel search for different cube orientations", with an almost optimal target:
    - intro: https://www.speedsolving.com/threads/3x3x3-solver-in-python.64887/
    - Python script: https://github.com/hkociemba/RubiksCube-TwophaseSolver
11. When rotations are 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.
12. The robot detects the cube status on cubes with and without the black frame around the facelets.
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.



24. Construction

The robot mechanical principles are simplicity, compact design and to solve a Rubik cube without modifying the cube (no need for special gripping)

Main info:

1. The inclined cube-holder is inspired to Hans' construction [Tilted Twister 2.0 – LEGO Mindstorms Rubik's Cube solver – YouTube](#);
   This is a clever concept, as it allows to flip the cube around one of the horizontal axes by forcing a relatively small angle change (about 30 degrees, over the 20 degrees of the starting cube holder angle); Once the cube centre of gravity is moved beyond the foothold, the cube falls on the following face thanks to the gravity force.
   Overall, it allows to flip the cube via a relatively small and inexpensive movement.
2. The Upper_cover moving solution is my own idea; It provides a constrainer for cube layer rotation, it suspends the PiCamera at sufficient distance for reading, it allows a compact robot construction when closed (robot not in use) and it only requires one pivot.
3. The cube flipper (lifter) is also my own idea, mainly to keep a small footprint and to have independent movements.
4. Underneath the Box there are two flipping feet, to reach a sufficient inclination to flip the cube, and to keep an overall smaller construction when the robot is closed (not in use).
5. The Box cover is hinged to the Box to quickly access/show the inner parts.
6. Upper_cover and flipping lever (lifter) are actuated via two servos, therefore controlled via angle.
7. Cube-holder is actuated via a step motor, therefore controlled by number of steps.
8. Cube-holder has 4 stationary positions, synchronized via a photo sensor: A printed disk with 4 open slots is fixed to the cube-holder, right underneath the Box cover. This ensures a proper cube/cube-holder position when the flipping lifter or the upper-cover are operated.
9. Cube-holder is hold in position by the motor (motor is energized to get holding torque) when the flipper or the upper-cover are activated.
10. Parts with a complex shape 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.
11. The Box is made by plywood, as it is an easy material to work with; The Box could also be 3D printed (stl files are also provided), and in that case I'd suggest merging some of the parts:
    - Bottom panel with foot hinges
    - Top panel with the hinge for upper-cover and lifter

    Note: On the stl files, for the Box, there aren't the recesses/fixing holes for the Box Top panel hinges.

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

- **Raspberry Pi** 4b 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 '**Cubotone.py**' is responsible for the interaction with these elements.

Below listed aspects, are presented on the next pages:

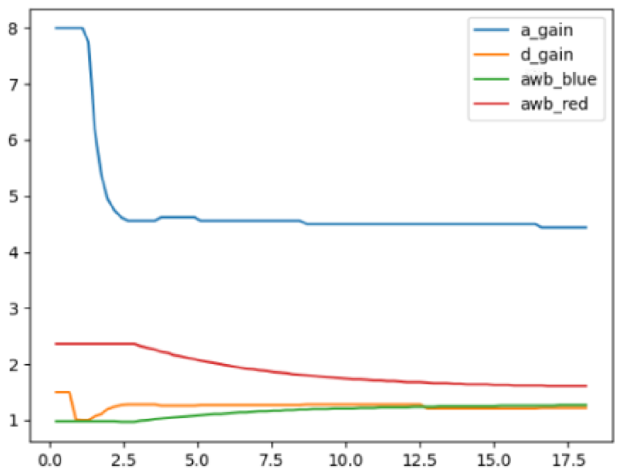
1. Camera positioning.
2. Taking consistent images.
3. Image analysis.
4. Contour analysis.
5. Colour retrieved.
6. Is all this really needed?

Colours detection strategy is described on a dedicated chapter as, in my case, it has proved to be the most challenging part of the complete project.

A. To get images, everything starts with positioning the camera on the right location:
On this first robot the camera is positioned parallel to the cube upper face.
This choice makes easy to analyse the facelets shape and area, due to the perspective absence.

B. Taking consistent images:
This is a crucial aspect for proper colour analysis.
The light source addition is a way to mitigate the environment light conditions, typically out of our control; Anyhow I found preferable to activate/disactivate the additional light source.

When the robot is requested to detect the cube status the Camera is 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 (blue, red) 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 Cubotone.py script: Once all these parameters are within 2% from the average value, then the camera is switched 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 for each of these cube faces.

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.

C. Image analysis:

The approach uses a similar technic as explained at https://medium.com/swlh/how-i-made-a-rubiks-cube-colour-extractor-in-c-551cceba80f0

1. The camera image is converted to grayscale: *gray = cv2.COLOUR_BGR2GRAY(frame….*
2. The grayscale image is filtered with a low pass filter to reduce noise: *blurred = cv2.GaussianBlur(gray, …*
3. The de-noised grayscale image is analysed with a Canny filter; This function transform the image to binary, assigning 1 (white) the pixels detected as edges: *canny = cv2.Canny(blurred….*
4. The binary image is analysed with Dilate, a morphological operation, aimed to join eventual interruptions of the thin edges returned by the Canny filter: *dilated = cv2.dilate(canny,….*

   The edges are now thicker (or much thicker) according to the kernel definition. Having Thicker edges is a way to reduce the quantity of edges, and gain speed.
5. The "Dilated" binary image, is analysed with Erode, a morphological operation that works opposite of Dilate: *eroded = cv2.erode(dilated….*

   Anyhow I preferred to use a different kernel than Dilate, and still keep rather thick edges
6. The Eroded binary image is now used to find contours: *cv2.findContours(image, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)*

D. Contour analysis:

Despite the image preparation, it is very common to get many more, and unwanted, contours; This requires filtering out the contours not having the potential to be a facelet.

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

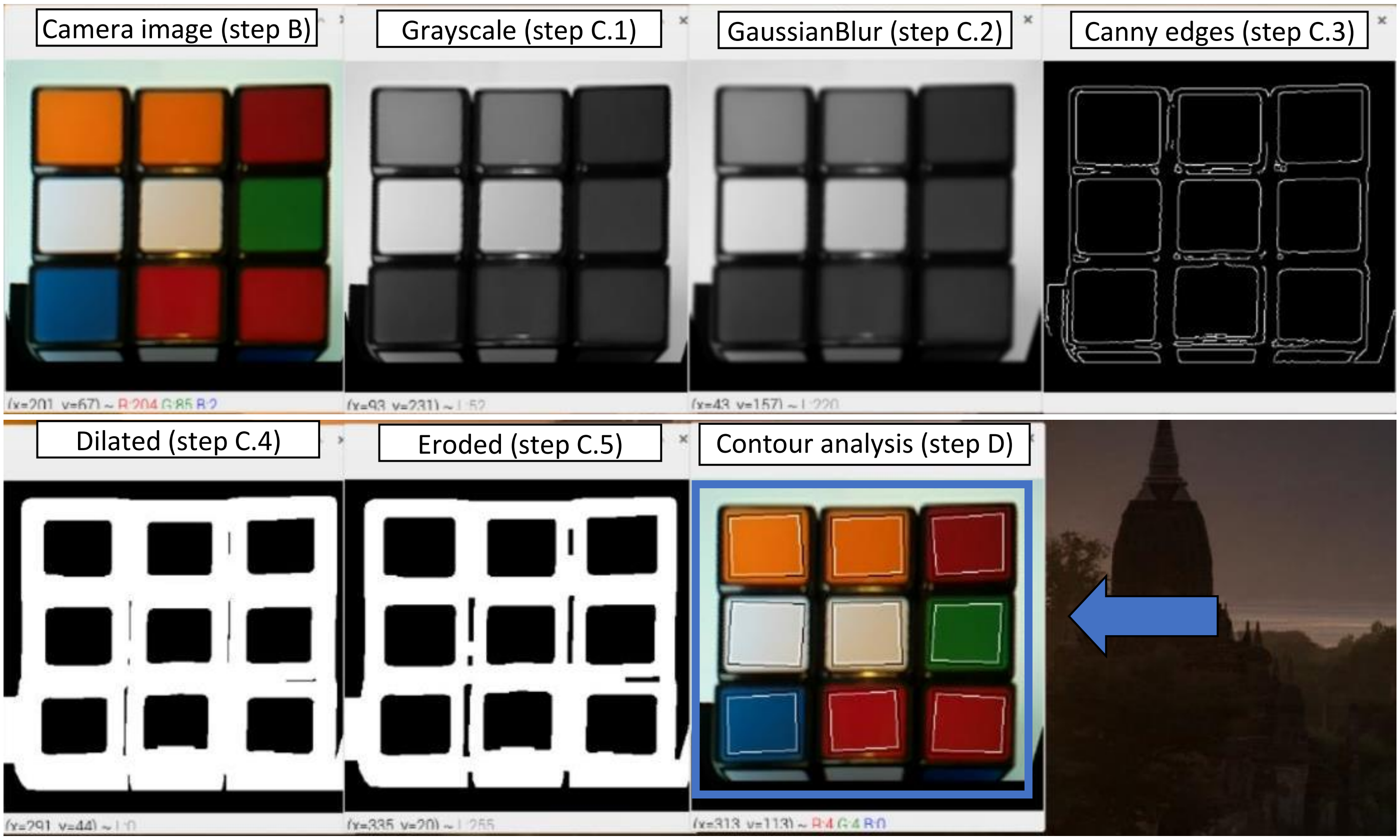
E. Colours retrieved:

On each facelet, are the retrieved 2 main info:

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

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

(If you'd like to see these images on screen, change the Boolean "cv_wow" at __main__ to True)



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

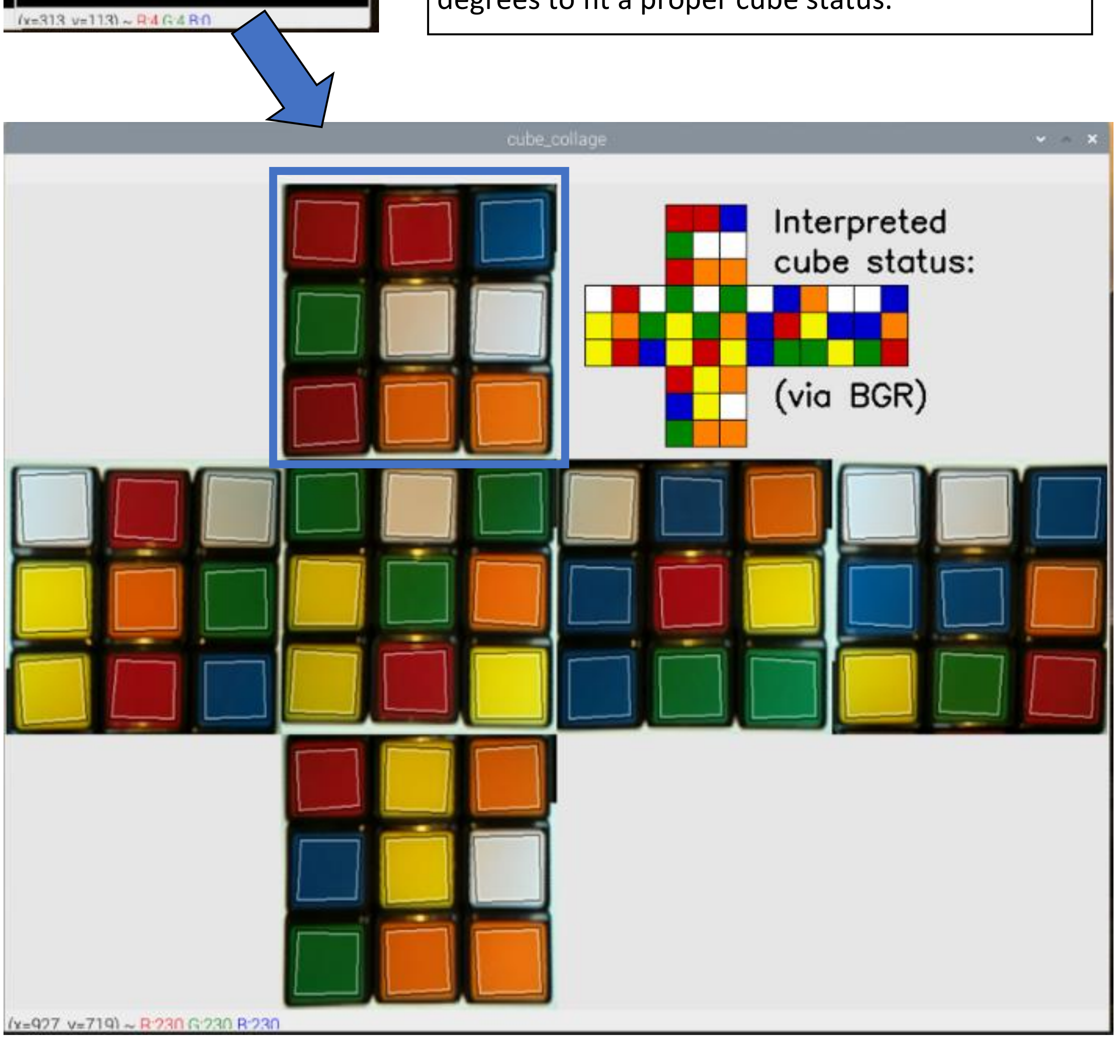
Be noted not all images are oriented as per user point of view: Cube's sides UBDF are 180deg rotated wrt camera reading.

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 image at side is the upper cube side (U side) as seen by camera during the facelets detection process.

After all the faces are detected, the images of the faces are cropped/rotated to form a collage with the cube status.

Highlighted below the same U side, rotated 180 degrees to fit a proper cube status.



F. Is all this really needed?

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

I haven't tried the 'simpler' approach; Below the below reasons I to stick to the chosen approach:

1. The robot construction is rather basic, and the cube/camera positions cannot be expected to be very precise/repetitive.
2. Not all the time the top cube layer is perfectly aligned.
3. Below picture shows one extreme case, in which the edge detection excluded a large area affected by light reflection; This cube face was correctly interpreted!

Notes:
Light reflection drastically affects the colour interpretation.
On the example at the side, the accepted contour is on the very low contour acceptable area, yet sufficiently large to don't be considered as noise.

4. There is one more reason for all this: Fun…. Having the facelets detected by a piece of AI is quite cool!

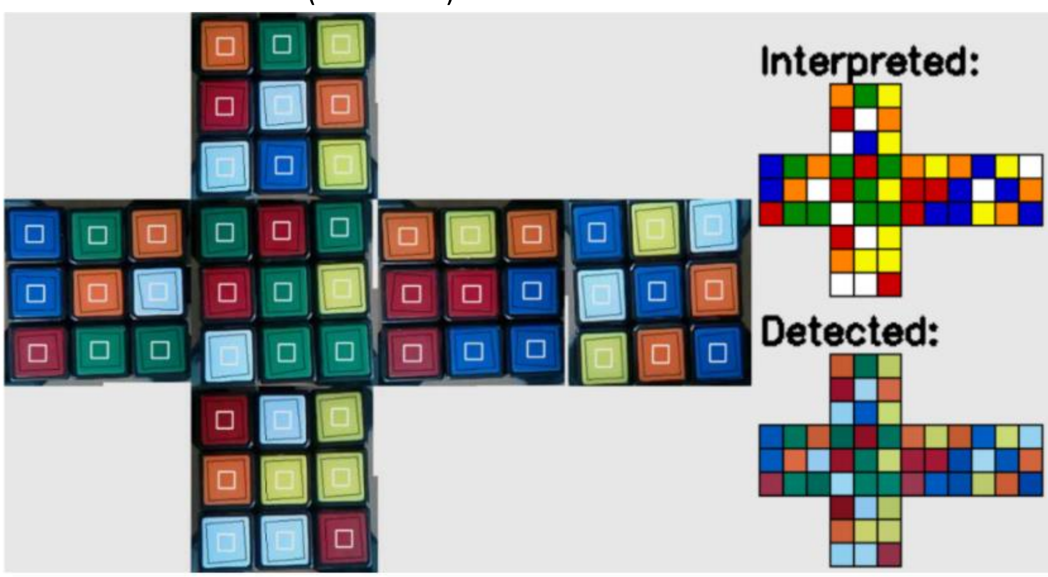
26. Colour's detection strategy

Some of the strategies used by this robot aren't described on the papers I could freely find in internet, in particular:

- To collect all the 54 facelets colours info prior to assign them to a reference face colour.
- To switch to an alternative colour analysis when the primary approach doesn't deliver a coherent cube status.

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 centre 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. Calculate the BGR average value, used for the colour interpretation according to the 1st method (BGR colour distance)
      ii. calculate the HSV average colours, used for colour interpretation according to the 2nd method (Hue value).



2. Properties of the faces centre facelet:
   On a 3x3x3 Rubik's cube, the 6 centre'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.

4. The average HSV, detected on the 6 centres, is used to determine which colour is located on the 6 centres:
   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 centres 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 centres, blue is the one with highest Hue, and consequently the green is also known.

5. Based on previous step, the 6 cube colours (at least their centres) have a known average HVS and therefore an average BGR colour; This also informs on the cube orientation (colours) as placed on the cube-holder.

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

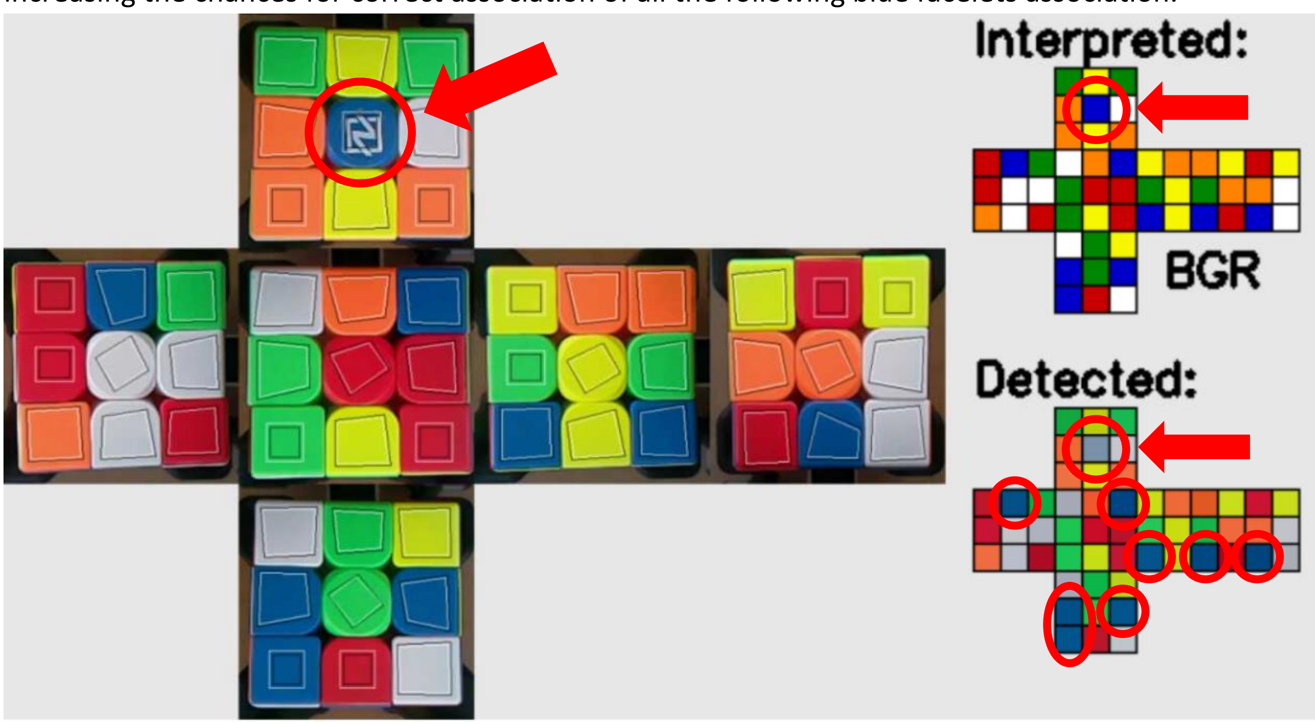
   First method is in general better than the second one, yet the second one "wins" when there is lot of light; The second method is only used (called) when the first one fails.
   As result both methods are used, to get reliable cube status detection under different light situations.

7. Dynamic colour reference:
   Facelets association to the cube face (to the cube faces centre colours) is made after ordering the facelets by colour distance, and by starting with those having the least distance (the less uncertain choice).
   Once a facelet is associated to the reference, the reference is updated by averaging it with the just associated facelet; In this way the reference keeps updating with the other facelets from the (assumed) same colour; In a way, the reference becomes more and more representative of that cube face colour.
   I haven't found any documentation in internet about this approach, that might be therefore novel.

   This approach is useful to mitigate:
   - camera vignetting effect (darker colour at the camera's corners).
   - centre face having a logo.
   - little facelets defect.

   On below case it can be appreciated the thick white logo on the blue centre (U face)
   Because of the logo presence, that facelet position has been estimated.
   The facelet average colour resulted in a very light blue, as plot on the "Detected" sketch.

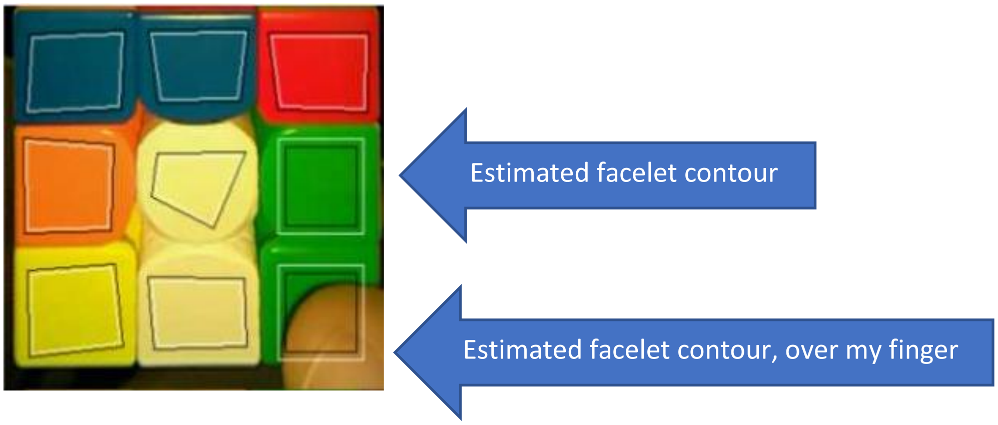
   Despite its very light blue the colour distance of one of the blue facelets (probably L face) was still closer than facelets colours from other cube sides, and the first association went well; Because of the dynamic colour reference, once the first association is made, the logo progressively loses its influence, increasing the chances for correct association of all the following blue facelets association.



9. Facelets position estimation:

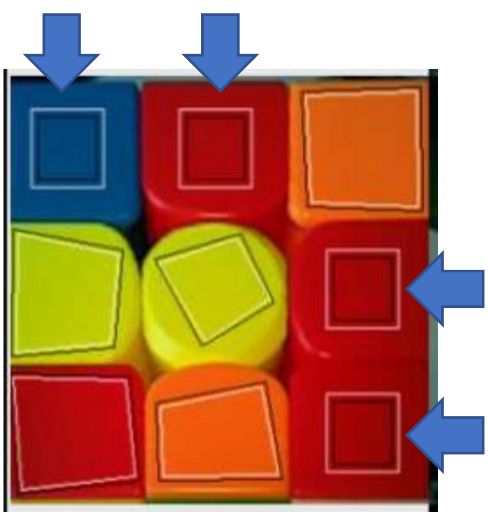
This robot version has proved working on different types of Rubik’s cube: with and without the black frame around the coloured facelets.
When the script works on the robot as soon as there are at least 5 detected facelets, without an empty row or column, the remaining facelets will be estimated on their position.

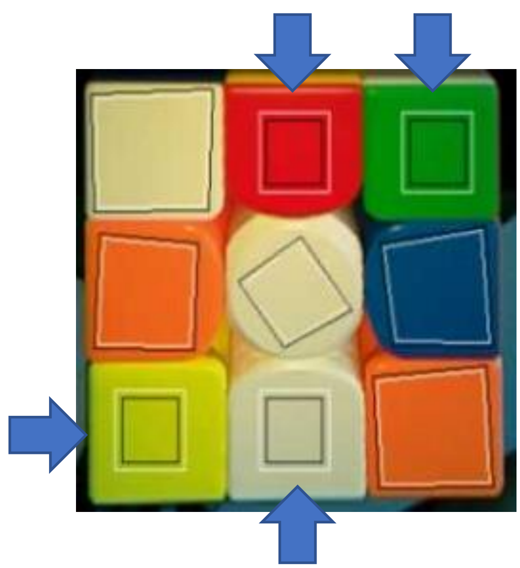
This approach helps to gain speed on establishing where the cube’s facelets are, in particular when adjacent facelets have the same colour.
This approach is not used the the script runs on laptop, as it doesn’t prevent your finger (or cube logo) to get captured: See below image, with my finger not been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



Examples of estimated facelets:

On below examples, 4 facelets have been estimated; To be noted all the rows and column have at least one detected facelet for a total of at least 5 detected facelets.

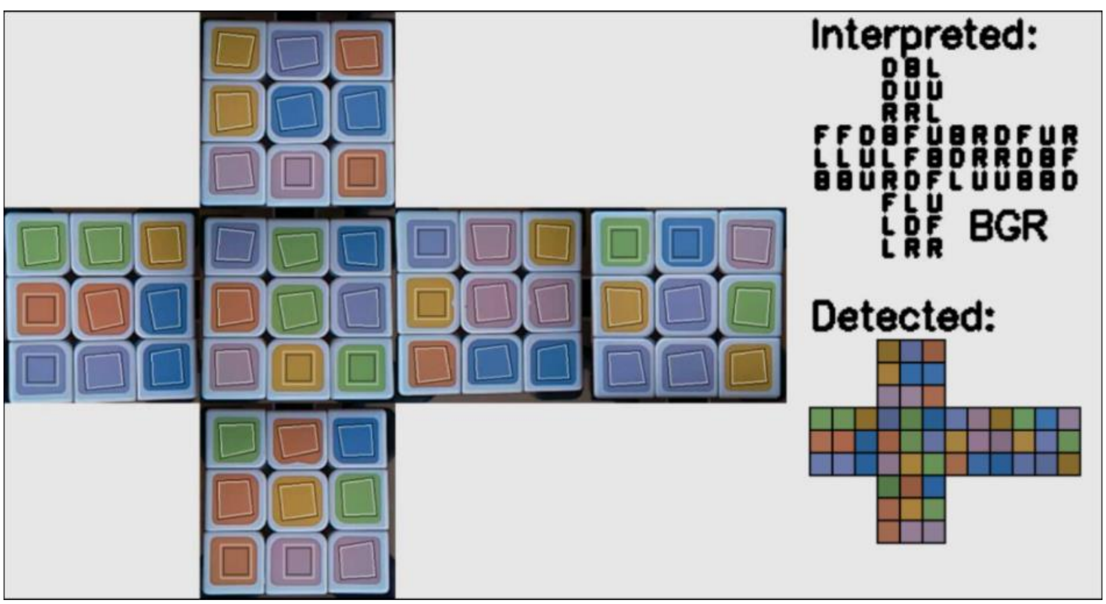




Be noted one of the estimated facelets goes to the central white one with the logo.
When the facelets have a printed logo, a defect, or some pollution, it will make more difficult to be detected as facelet: These typically are facelets that will be estimated on their position.

Despite the position of the central facelet will be correctly estimated, it will be rather possible to get a detection error because of the logo influences the average colour retrieved from that facelet.
Apart from the printed logo, below example shows again 4 adjacent estimated facelets.

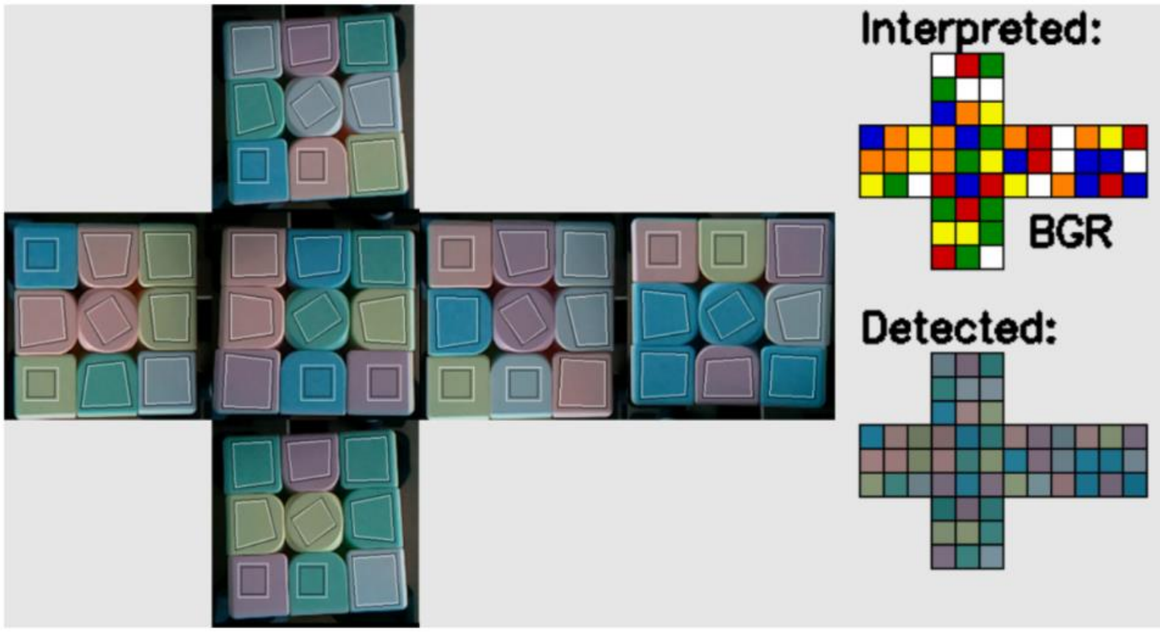
10. Non-western colour scheme cubes:
    This robot has been developed by considering western colour scheme cubes, but that is only relevant to plot a nice "interpreted" cube.
    In case the cube centres colours detection isn't coherent with a western colour scheme cube, the plotted "Interpreted" cube uses the face location letters instead of colours.
    Because of the colour detection strategy used, the robot detects and solves also non-western colour scheme:



Be noted the interpreted cube sketch uses (URFDLB) letters

11. Western colour scheme with very light colours:
    On below images how the robot interprets this Rubik's cube, characterized by using the western colour scheme yet having extremely light colours.

    Case the centres are detected as per the western colour scheme:



The Interpreted cube sketch propose vivid colours of those detected.

Case the centres aren't detected as per the western colour scheme:

Be noted the interpreted cube sketch uses (URFDLB) letters.

Notes:

- The cube detection status doesn't always succeed with this Rubik's cube.
- well diffused light conditions are especially needed with this cube 😊.

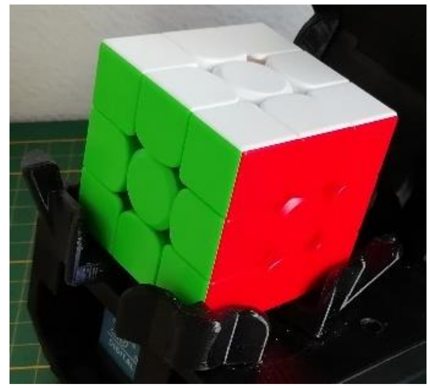
12. Frameless cubes:

The cube facelets detection algorithm has been initially developed for the classic cubes, those having the black frame around the facelets.
Starting from October 2022, the code has been adapted to also work with frameless type of cubes.
There is no need to change settings, when changing type of cube.

classic (with stickers)

classic (in-molded)

frameless



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 Cubotone_moves.py file.

It is clear this robot has very limited degrees of freedom, as it can only rotate the bottom layer, farther than flipping the cube around the L-R horizontal axis; This obviously requires an algorithm that limit 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. The cube orientation is defined by two faces: The face lying on the Cube_holder and the face facing Front; a "Down-Front" cube orientation is used.
2. The sequence of these two faces is part of 6 possibles sequences ('DFUBDFUB', 'DLURDLUR', 'DRULDRUL', 'DBUFDBUF', 'FLBRFLBR', 'FRBLFRBL'), by considering the flipping of the cube along one axis.
3. The algorithm checks if the next cube orientation can be better reached by flipping the cube, or if a rotation is needed, or via a 180degrees rotation instead of 2 flipping.
4. The initial cube orientation is known after the cube scanning phase and, after each cube movement, the "Down-Front" orientation is updated.
5. When the cube solution string is parsed, the above analysis is performed to return:
    - complete set of movements
    - total quantity of movements (used to de-count while solving).
6. Cubotone.py sends the complete set of movements to Cubotone_servos.py, that activates the servos and motor accordingly.

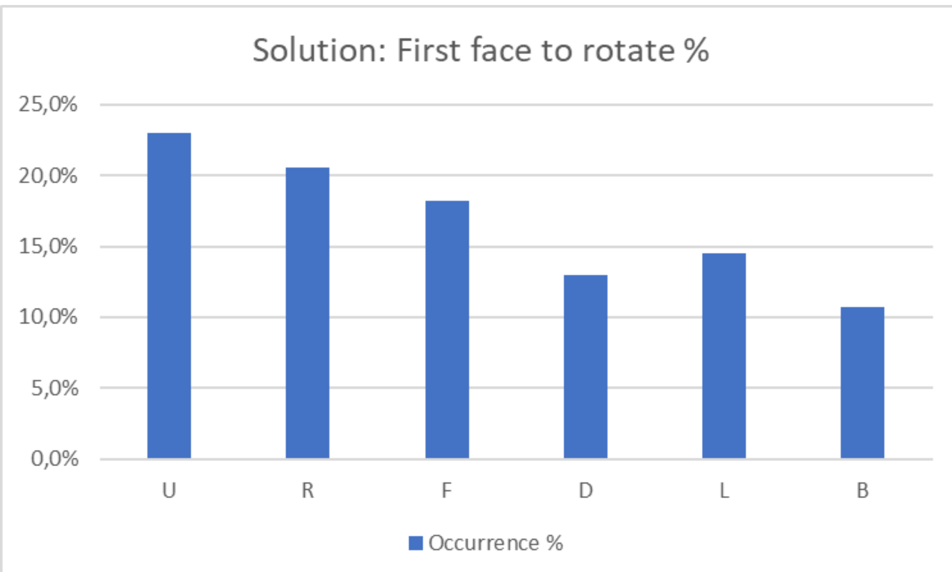
Small optimization made on September 2023.

1. Based on data saved in the log files of my robots (1xCubotone, 2xCubotino)
2. Total of about 1300 manually scrambled cubes, after removing those with less than 10 rotations.
3. Below table shows the occurrences of the first rotation face, returned by the Kociemba solver:

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

~ 62% (U, R, F)

~ 38% (D, L, B)



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

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

Starting from September 2023, after the scanning phase, the cube is not anymore moved to the “Initial position”. This means the cube is laying on its R face (20% chances to be the first face to rotate) and with a single flip the U face will be on the Cube_holder.
With this new approach it should be possible to save some robot movements ….

Of course, this robot is still highly inefficient, yet a little bit less 😊

28. Python main scripts, high level info

1. *Cubotone.py* is the main python script, that imports other custom files.
2. *Cubotone.py* and *Cubotone_servos.py* scripts use parameters/settings from two json files; The choice to group the parameters has been made for easier management, setting and communication.
3. When the script *Cubotone.py* is started (eventually automatically at the Raspberry Pi boots), the script checks if there is a screen connected. The screen can also be via VNC, i.e. with VNC Viewer. The presence/absence of a screen 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 the 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 match the filters criteria 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 centres.
    When the cube status from 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 centres.
    In case also the second method doesn't provide a coherent cube status, then an error message is returned, and relevant info logged in a text file.

6. Kociemba solver:
   Kociemba solver is uploaded at the start.
   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 given time-out; The solver doesn't provide the absolute best solution, as it is too computational (and time) expensive, yet it typically returns a solution with 20 movements or less. Very rarely, the solution has 21 movements, mostly because of the chosen time-out of 'only' two seconds.
   The solver returns an error if the cube status is not coherent; This info is then used to attempt the second colour assignment method, or to stop by providing error feedback to the display.
7. From cube solution to robot movements:
   (see Robot solver algorithm chapter for more info) Cube solutions, in Singmaster notation, sent to Cubotone_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 Cubotone notation, is sent to Cubotone_servos.py that operates the servos to actuate all the intended movements.

9. Data logged:

Each time the robot solves a cube, or when it gets stopped, the below data is logged in a text file:

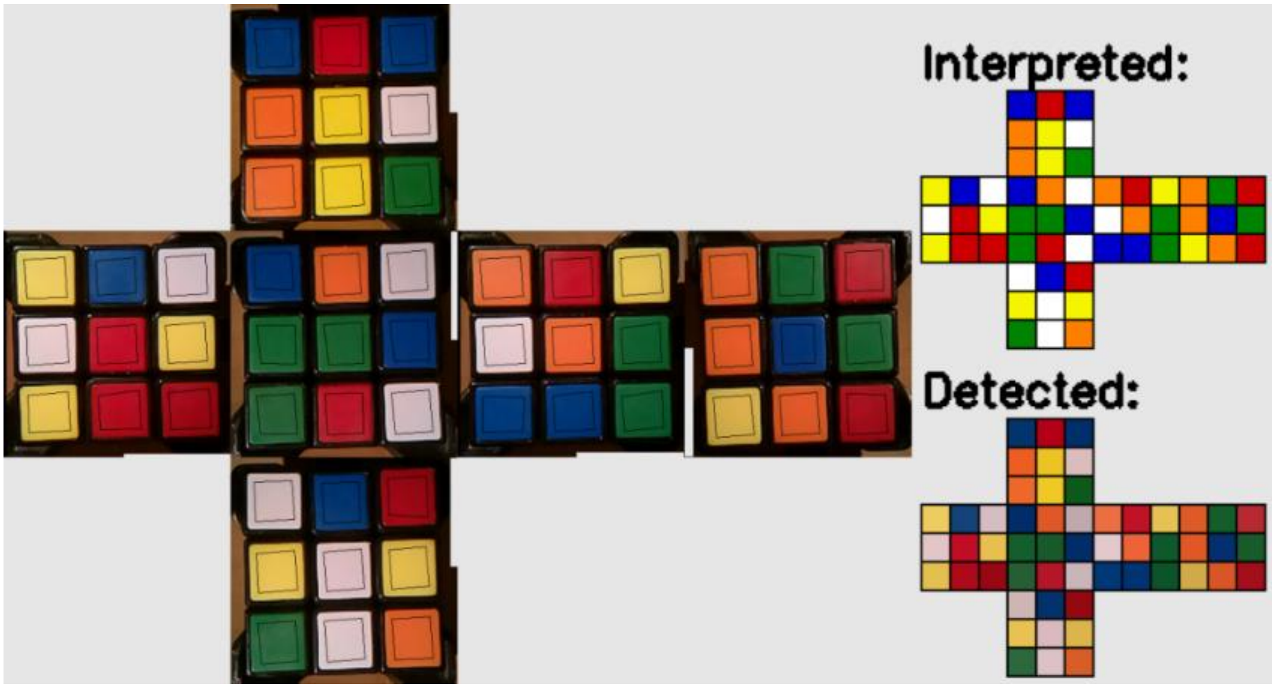
Column name	**Info**
Date	Date and time (yyyymmdd_hhmmss), i.e. 20220428_213439
Screen	Indicates if a screen is connected
CV_wow	Shows the cube face image analysis
Debug	Forces printouts for additional info
Led_usage	Indicates whether the leds usage at upper_cover is activated
ColourAnalysisWinner	The 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)
TotalTime (s)	Time, in seconds, from pressing the start button, until the cube is solved or until the robot is stopped
CamWarmUpTime (s)	Time, in seconds, from pressing the button, until the robot ends all the camera settings
FaceletsDetectionTime (s)	Time, in seconds, for the facelets detection
CubeSolutionTime(s)	Time, in seconds, used by the Kociemba solver to return the solution
CollageImageTime (s)	Time, in seconds, to make the unfolded cube status image collage
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 *Cubotone.py* is running.
2. The logged data is saved in the *Cubotone_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.

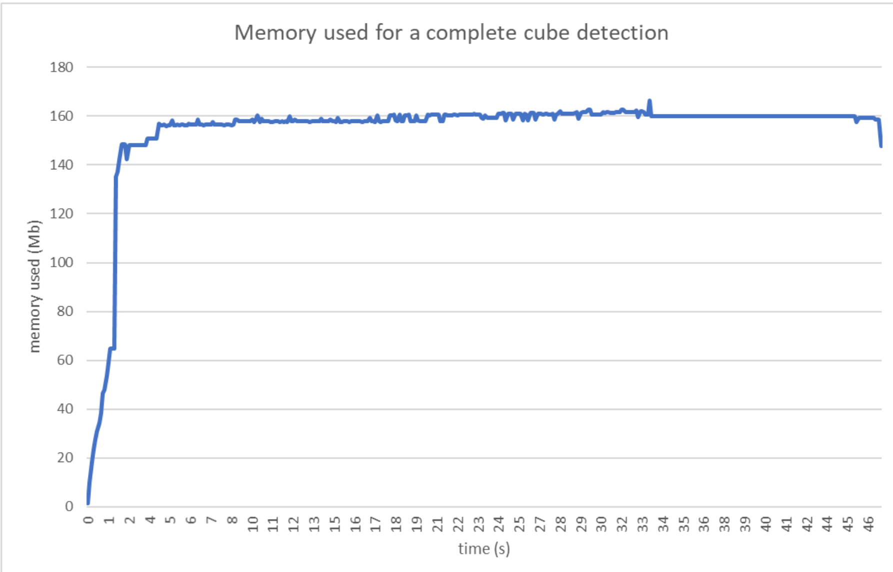
12. Memory profiling

One question at Instructables was about the possibility to use a Raspberry Pi 3 (512Mb RAM) instead of Raspberry Pi 4B with 2Gb of RAM, to lower the total build cost.

The memory usage on the windows PC and on the Raspberry Pi (the robot) is reported below.

At Window PC, checked with *python -m mprof plot AF_cube_robot.py.*

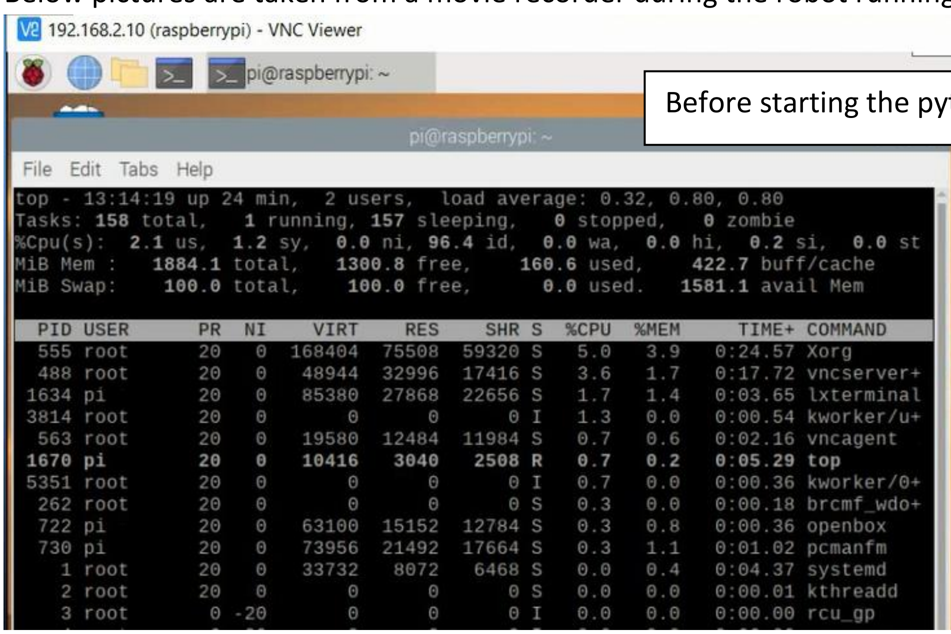
By plotting the values from the generated text file, the used memory is up to 170Mb



At Raspberry pi, checked with *top*.

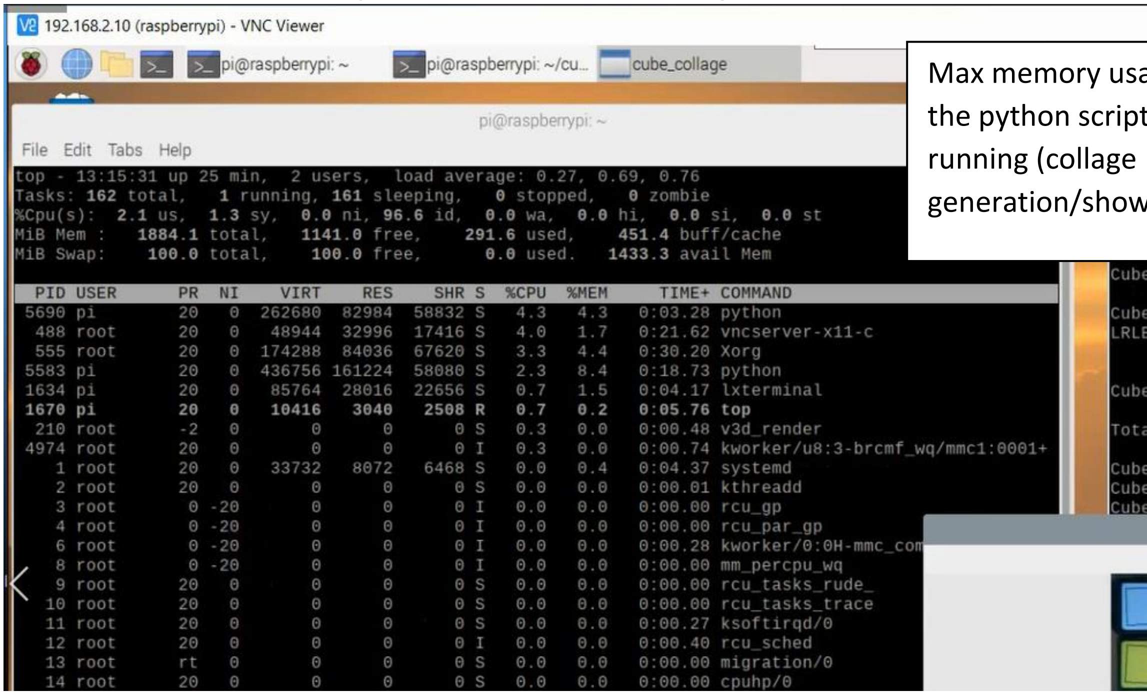
The Raspberry Pi OS installation already uses ca 160Mb before the python script is started (data sharing via SSH);

Below pictures are taken from a movie recorder during the robot running:



Before starting the python script

When the robot is running, the total RAM used is up to ca 300MiB



Max memory usage when the python script is running (collage generation/show)

Notes:

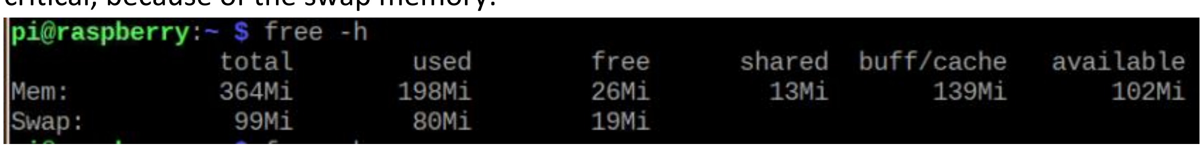
1. Raspberry Pi was set to share 64Mb with GPU.
2. Very large quantity of virtual memory could be addressed by the python script, yet not used.
3. The used memory has reached a total of ca 21% of the total, on a 2Gb system.
4. I haven't spent energy to optimize on memory usage.

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.

Conclusions:

It seems well possible to run the robot script on a Raspberry Pi with 512Mb of RAM.
On Cubotino, a successor of this robot that uses very similar scripts, a Raspberry Pi Zero or Zero 2 (both having 512Kb of RAM) are sufficient.

29. Overall conclusions

I believe I've accomplished the goals set upfront the project .

I'm satisfied about the learning journey, and the end results, to the point I've decided to write this document and to share the project at Instructables.

Colours interpretation, to determine the right cube status, has been by far the biggest challenge on this project.

Based on the logged data, the robot correctly reads the cube status on 99.5% of the cases, while the mechanical part solves the cube without issues.

Off course there are also negative results:

- This robot is quite noisy, apart from the typical noise of servos and stepper motor, the cube falling into the cube-holder makes an unpleasant noise (and the underneath box doesn't help on this).
- The chosen LCD with segments (very cheap), or the chosen Python library, don't consistently work.
- My coding skills are still very low, and I do recognize I've used way too many global variables on this project.

Tips and feedback, on all areas, are for sure very welcome.

30. Commitment

If you read these instructions, there are chances you are interested on making a similar 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, feedback or thumbs up on Youtube (https://youtu.be/oYRXe4NyJqs) or at the Instructables site.

In case you cannot find the solution by yourself (part that makes projects fun 😊), please drop a detailed question at the instructables site (https://www.instructables.com/Rubik-Cube-Solver-Robot-With-Raspberry-Pi-and-Pica/)

I can't promise I'll be able to answer your questions, as well as I cannot commit to be fast in replying.

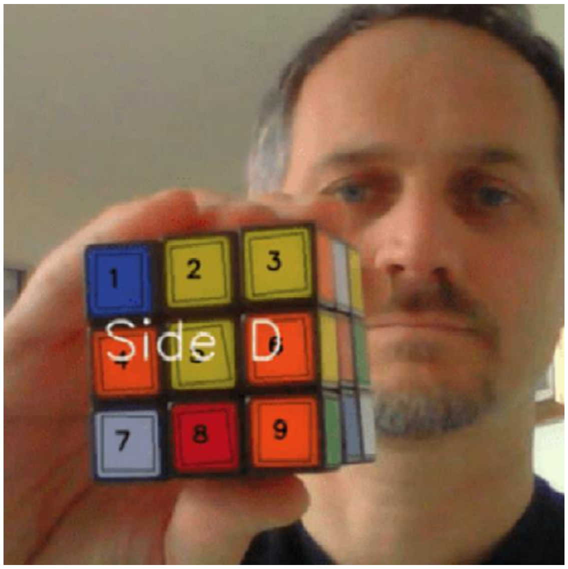
Please feel free to provide your tips and feedback, on all areas: This will help me.

31. Credits

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

Credits to all the people who have provided feedback about this robot, triggering me on starting the Cubotino project (the successor of Cubotone 😊)

32. Myself



I'm Andrea Favero.

I was born in Italy in 1971.

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

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

Since 2015 we've been living in Groningen, The Netherland.

On 2019 I had the opportunity to attend a Python class course, and I felt in love with coding.

On 2021 I decided to learn computer vision and Raspberry Pi, by giving myself the target to build a Rubik's cube solver robot.

My first Rubik's cube solver robot (CUBOTone) has been a test for myself, yet I learned so many things from others that I wanted to share it back via the Instructables site; The positive reactions and criticisms, led me thinking on how to make an easier version, with an eye to costs:
CUBOTino project started on January 2022 and CUBOTino micro was published in March 2023.

Contacts:

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

33. Revisions

Rev	Date	Notes
0	03/10/2021	First release
1	13/11/2021	Solved bug on Python scripts: *AF_cube_robot.py* and *AF_cube_robot_noVideo* Added a hdmi socket to the rear panel of the robot, connected to the hdmi0 port at Raspberry Pi 4b. Instructions: • Added the memory usage related chapter.
2	27/4/2022	Solved a bug (indentation error) on Python scripts: Files *AF_cube_robot.py* and *AF_cube_robot_noVideo.py*
3	01/10/2022	Large update, to incorporate the learnings from Cubotino: New Upper_cover geometry, supporting leds breakout boards. Scripts: 1. Semplified installation via GitHub repository. 2. Moved the most relevant parameters settings to text files. 3. Automated the screen presence check, therefore not anymore needed two file versions. 4. Camera exposition extended to UBDF faces. 5. Extended folders/files permission at their creation. 6. Removed the need for Scipy library, by extending Numpy usage. 7. Added the Get-mac library, to differentiate the optimized parameters for my (AF. 8. Added the cv_wow part (it shows the image analysis to detect the facelets). 9. Improved the motor alignment procedure. 10. Added frameless cube functionality, with related estimation on facelets positions. 11. Extended the debug printout. 12. Added physical switches to quickly change some settings (for faire demonstration). Improved instructions

Rev	Date	Notes
3.1	10/09/2023	This revision covers the last topics from the Cubotone “to do” list, and waiting for attention for almost one year. Instructions: • Changed the content order. • Added quite some new parts: 1. Connection diagram 2. Robot solver algorithm 3. Screen connection to Rpi HDMI port 4. How to use Cubotone.py on a Windows PC • Troubleshooting, added: 1. Error when SD card is flashed with OS proposed as default since November 2021. 2. Missing flips due to friction. Scripts, changed: Cubotone.py, Cubotone_moves.py, Cubotone_servo.py and Cubotone_bash.sh • The cube handling for solving starts from the last position used at scanning phase (the cube is not moved to its original position anymore). • The solver is not reloaded at each cycle (still after each reset). • When a screen is not connected, the unfolded cube image collage is made right after solving the cube instead of in parallel (multi-processing). • More information is logged at every solved cycle. • Added arguments to allows early tests (before wiring everything), to correct for cubes size difference, and others.
3.2	31/01/2024	Instructions: How to download and flash OS10 (Buster) at setting up Raspberry Pi.

34. APPENDIX 1: Raspberry Pi setup printout

Info collected on 2022/11/06

Terminal printout during Raspberry Pi setup (GitHub repository https://github.com/AndreaFavero71/cubotone.git).

pi@cubotone:~ $ *cd cubotone/src*
pi@cubotone:~/cubotone/src $ *sudo ./install/setup.sh*

Deactivating graphical login

configuring config file

Updating packages

```
Get:1 http://archive.raspberrypi.org/debian buster InRelease [32.6 kB]
Get:2 http://raspbian.raspberrypi.org/raspbian buster InRelease [15.0 kB]
Get:3 http://archive.raspberrypi.org/debian buster/main armhf Packages [392 kB]
Get:4 http://raspbian.raspberrypi.org/raspbian buster/main armhf Packages [13.0 MB]
Fetched 13.5 MB in 8s (1,638 kB/s)
Reading package lists... Done
Building dependency tree
Reading state information... Done
6 packages can be upgraded. Run 'apt list --upgradable' to see them.
The following packages will be upgraded:
  libexpat1 libexpat1-dev libpoppler-qt5-1 libpoppler82 poppler-utils unzip
6 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
Need to get 2,010 kB of archives.
After this operation, 4,096 B of additional disk space will be used.
apt-listchanges: Reading changelogs...
(Reading database ... 98957 files and directories currently installed.)
Preparing to unpack .../0-libexpat1-dev_2.2.6-2+deb10u5_armhf.deb ...
Unpacking libexpat1-dev:armhf (2.2.6-2+deb10u5) over (2.2.6-2+deb10u4) ...
Preparing to unpack .../1-libexpat1_2.2.6-2+deb10u5_armhf.deb ...
Unpacking libexpat1:armhf (2.2.6-2+deb10u5) over (2.2.6-2+deb10u4) ...
Preparing to unpack .../2-poppler-utils_0.71.0-5+deb10u1_armhf.deb ...
Unpacking poppler-utils (0.71.0-5+deb10u1) over (0.71.0-5) ...
Preparing to unpack .../3-libpoppler-qt5-1_0.71.0-5+deb10u1_armhf.deb ...
Unpacking libpoppler-qt5-1:armhf (0.71.0-5+deb10u1) over (0.71.0-5) ...
Preparing to unpack .../4-libpoppler82_0.71.0-5+deb10u1_armhf.deb ...
Unpacking libpoppler82:armhf (0.71.0-5+deb10u1) over (0.71.0-5) ...
Preparing to unpack .../5-unzip_6.0-23+deb10u3_armhf.deb ...
Unpacking unzip (6.0-23+deb10u3) over (6.0-23+deb10u2) ...
Setting up libexpat1:armhf (2.2.6-2+deb10u5) ...
Setting up unzip (6.0-23+deb10u3) ...
Setting up libpoppler82:armhf (0.71.0-5+deb10u1) ...
Setting up libexpat1-dev:armhf (2.2.6-2+deb10u5) ...
Setting up poppler-utils (0.71.0-5+deb10u1) ...
Setting up libpoppler-qt5-1:armhf (0.71.0-5+deb10u1) ...
Processing triggers for libc-bin (2.28-10+rpt2+rpi1+deb10u1) ...
Processing triggers for man-db (2.8.5-2) ...
Processing triggers for mime-support (3.62) ...
```

Removing old packages

```
The following packages were automatically installed and are no longer required:
  librtimulib-dev librtimulib-utils librtimulib7 python-olefile python-pil
  python-rtimulib python-sense-hat python3-rtimulib
Use 'sudo apt autoremove' to remove them.
The following packages will be REMOVED:
  python3-microdotphat python3-numpy python3-pgzero python3-picamera python3-pygame
  python3-scrollphathd python3-sense-hat python3-unicornhathd sense-hat
0 upgraded, 0 newly installed, 9 to remove and 0 not upgraded.
After this operation, 13.9 MB disk space will be freed.
(Reading database ... 98956 files and directories currently installed.)
Removing python3-microdotphat (0.2.1) ...
Removing python3-unicornhathd (0.0.4) ...
Removing python3-pgzero (1.2.post4+dfsg-2+~rpt1) ...
Removing python3-pygame (1.9.4.post1+dfsg-3) ...
Removing sense-hat (1.2) ...
Removing python3-sense-hat (2.4.0-1~bpo10+1) ...
Removing python3-picamera (1.13) ...
Removing python3-scrollphathd (1.2.1) ...
Removing python3-numpy (1:1.16.2-1) ...
Processing triggers for man-db (2.8.5-2) ...
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages will be REMOVED:
  librtimulib-dev librtimulib-utils librtimulib7 python-olefile python-pil
  python-rtimulib python-sense-hat python3-rtimulib
0 upgraded, 0 newly installed, 8 to remove and 0 not upgraded.
After this operation, 2,344 kB disk space will be freed.
(Reading database ... 98277 files and directories currently installed.)
Removing librtimulib-dev (7.2.1-5) ...
Removing librtimulib-utils (7.2.1-5) ...
Removing python3-rtimulib (7.2.1-5) ...
Removing python-sense-hat (2.2.0-1) ...
Removing python-rtimulib (7.2.1-5) ...
Removing librtimulib7 (7.2.1-5) ...
Removing python-olefile (0.46-1) ...
Removing python-pil:armhf (5.4.1-2+deb10u3) ...
Processing triggers for libc-bin (2.28-10+rpt2+rpi1+deb10u1) ...

Installing required packages

python3-pil is already the newest version (5.4.1-2+deb10u3).
python3-pil set to manually installed.
python3-venv is already the newest version (3.7.3-1).
python3-venv set to manually installed.
python3-gpiozero is already the newest version (1.6.2-1).
python3-pigpio is already the newest version (1.79-1+rpt1).
python3-pip is already the newest version (18.1-5+rpt1).
python3-rpi.gpio is already the newest version (0.7.0-0.1~bpo10+4).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
The following additional packages will be installed:
  freeglut3 libaec0 libatlas3-base libaudio2 libglu1-mesa libhdf5-103 libjasper1 libjpeg8 libmng1 libqt4-dbus libqt4-xml libqtcore4
libqtdbus4 libsz2 python3-numpy qdbus qt-at-spi
  qtchooser qtcore4-l10n
Suggested packages:
  libatlas-doc liblapack-doc nas libicu57 qt4-qtconfig python-h5py-doc gfortran python-numpy-doc python3-pytest python3-numpy-dbg
The following NEW packages will be installed:
  freeglut3 libaec0 libatlas-base-dev libatlas3-base libaudio2 libglu1-mesa libhdf5-103 libjasper-runtime libjasper1 libjpeg8 libmng1
libqt4-dbus libqt4-test libqt4-xml libqtcore4
  libqtdbus4 libqtgui4 libsz2 python3-h5py python3-numpy qdbus qt-at-sPi qtchooser qtcore4-l10n
```

```
0 upgraded, 24 newly installed, 0 to remove and 0 not upgraded.
Need to get 16.1 MB of archives.
After this operation, 76.7 MB of additional disk space will be used.
Selecting previously unselected package libjasper1:armhf.
(Reading database ... 98075 files and directories currently installed.)
Preparing to unpack .../00-libjasper1_1.900.1-debian1-2.4+deb8u1_armhf.deb ...
Unpacking libjasper1:armhf (1.900.1-debian1-2.4+deb8u1) ...
Selecting previously unselected package libjpeg8:armhf.
Preparing to unpack .../01-libjpeg8_8d1-2_armhf.deb ...
Unpacking libjpeg8:armhf (8d1-2) ...
Selecting previously unselected package libmng1:armhf.
Preparing to unpack .../02-libmng1_1.0.10+dfsg-3.1_armhf.deb ...
Unpacking libmng1:armhf (1.0.10+dfsg-3.1) ...
Selecting previously unselected package freeglut3:armhf.
Preparing to unpack .../03-freeglut3_2.8.1-3_armhf.deb ...
Unpacking freeglut3:armhf (2.8.1-3) ...
Selecting previously unselected package libaec0:armhf.
Preparing to unpack .../04-libaec0_1.0.2-1_armhf.deb ...
Unpacking libaec0:armhf (1.0.2-1) ...
Selecting previously unselected package libatlas3-base:armhf.
Preparing to unpack .../05-libatlas3-base_3.10.3-8+rpi1_armhf.deb ...
Unpacking libatlas3-base:armhf (3.10.3-8+rpi1) ...
Selecting previously unselected package libatlas-base-dev:armhf.
Preparing to unpack .../06-libatlas-base-dev_3.10.3-8+rpi1_armhf.deb ...
Unpacking libatlas-base-dev:armhf (3.10.3-8+rpi1) ...
Selecting previously unselected package libaudio2:armhf.
Preparing to unpack .../07-libaudio2_1.9.4-6_armhf.deb ...
Unpacking libaudio2:armhf (1.9.4-6) ...
Selecting previously unselected package libglu1-mesa:armhf.
Preparing to unpack .../08-libglu1-mesa_9.0.0-2.1_armhf.deb ...
Unpacking libglu1-mesa:armhf (9.0.0-2.1) ...
Selecting previously unselected package libsz2:armhf.
Preparing to unpack .../09-libsz2_1.0.2-1_armhf.deb ...
Unpacking libsz2:armhf (1.0.2-1) ...
Selecting previously unselected package libhdf5-103:armhf.
Preparing to unpack .../10-libhdf5-103_1.10.4+repack-10_armhf.deb ...
Unpacking libhdf5-103:armhf (1.10.4+repack-10) ...
Selecting previously unselected package libjasper-runtime.
Preparing to unpack .../11-libjasper-runtime_1.900.1-debian1-2.4+deb8u1_armhf.deb ...
Unpacking libjasper-runtime (1.900.1-debian1-2.4+deb8u1) ...
Selecting previously unselected package qtcore4-l10n.
Preparing to unpack .../12-qtcore4-l10n_4%3a4.8.7+dfsg-18+rpi1+deb10u1_all.deb ...
Unpacking qtcore4-l10n (4:4.8.7+dfsg-18+rpi1+deb10u1) ...
Selecting previously unselected package libqtcore4:armhf.
Preparing to unpack .../13-libqtcore4_4%3a4.8.7+dfsg-18+rpi1+deb10u1_armhf.deb ...
Unpacking libqtcore4:armhf (4:4.8.7+dfsg-18+rpi1+deb10u1) ...
Selecting previously unselected package libqt4-xml:armhf.
Preparing to unpack .../14-libqt4-xml_4%3a4.8.7+dfsg-18+rpi1+deb10u1_armhf.deb ...
Unpacking libqt4-xml:armhf (4:4.8.7+dfsg-18+rpi1+deb10u1) ...
Selecting previously unselected package libqtdbus4:armhf.
Preparing to unpack .../15-libqtdbus4_4%3a4.8.7+dfsg-18+rpi1+deb10u1_armhf.deb ...
Unpacking libqtdbus4:armhf (4:4.8.7+dfsg-18+rpi1+deb10u1) ...
Selecting previously unselected package qtchooser.
Preparing to unpack .../16-qtchooser_66-2_armhf.deb ...
Unpacking qtchooser (66-2) ...
Selecting previously unselected package qdbus.
Preparing to unpack .../17-qdbus_4%3a4.8.7+dfsg-18+rpi1+deb10u1_armhf.deb ...
Unpacking qdbus (4:4.8.7+dfsg-18+rpi1+deb10u1) ...
Selecting previously unselected package libqt4-dbus:armhf.
Preparing to unpack .../18-libqt4-dbus_4%3a4.8.7+dfsg-18+rpi1+deb10u1_armhf.deb ...
```

```
Unpacking libqt4-dbus:armhf (4:4.8.7+dfsg-18+rpi1+deb10u1) ...
Selecting previously unselected package libqt4-test:armhf.
Preparing to unpack .../19-libqt4-test_4%3a4.8.7+dfsg-18+rpi1+deb10u1_armhf.deb ...
Unpacking libqt4-test:armhf (4:4.8.7+dfsg-18+rpi1+deb10u1) ...
Selecting previously unselected package libqtgui4:armhf.
Preparing to unpack .../20-libqtgui4_4%3a4.8.7+dfsg-18+rpi1+deb10u1_armhf.deb ...
Unpacking libqtgui4:armhf (4:4.8.7+dfsg-18+rpi1+deb10u1) ...
Selecting previously unselected package python3-numpy.
Preparing to unpack .../21-python3-numpy_1%3a1.16.2-1_armhf.deb ...
Unpacking python3-numpy (1:1.16.2-1) ...
Selecting previously unselected package python3-h5py.
Preparing to unpack .../22-python3-h5py_2.8.0-3_armhf.deb ...
Unpacking python3-h5py (2.8.0-3) ...
Selecting previously unselected package qt-at-spi:armhf.
Preparing to unpack .../23-qt-at-spi_0.4.0-9_armhf.deb ...
Unpacking qt-at-spi:armhf (0.4.0-9) ...
Setting up libjpeg8:armhf (8d1-2) ...
Setting up libmng1:armhf (1.0.10+dfsg-3.1) ...
Setting up libatlas3-base:armhf (3.10.3-8+rpi1) ...
update-alternatives: using /usr/lib/arm-linux-gnueabihf/atlas/libblas.so.3 to provide /usr/lib/arm-linux-gnueabihf/libblas.so.3
(libblas.so.3-arm-linux-gnueabihf) in auto mode
update-alternatives: using /usr/lib/arm-linux-gnueabihf/atlas/liblapack.so.3 to provide /usr/lib/arm-linux-gnueabihf/liblapack.so.3
(liblapack.so.3-arm-linux-gnueabihf) in auto mode
Setting up freeglut3:armhf (2.8.1-3) ...
Setting up libatlas-base-dev:armhf (3.10.3-8+rpi1) ...
update-alternatives: using /usr/lib/arm-linux-gnueabihf/atlas/libblas.so to provide /usr/lib/arm-linux-gnueabihf/libblas.so (libblas.so-
arm-linux-gnueabihf) in auto mode
update-alternatives: using /usr/lib/arm-linux-gnueabihf/atlas/liblapack.so to provide /usr/lib/arm-linux-gnueabihf/liblapack.so
(liblapack.so-arm-linux-gnueabihf) in auto mode
Setting up libaec0:armhf (1.0.2-1) ...
Setting up libaudio2:armhf (1.9.4-6) ...
Setting up python3-numpy (1:1.16.2-1) ...
Setting up libjasper1:armhf (1.900.1-debian1-2.4+deb8u1) ...
Setting up libglu1-mesa:armhf (9.0.0-2.1) ...
Setting up qtchooser (66-2) ...
Setting up libsz2:armhf (1.0.2-1) ...
Setting up qtcore4-l10n (4:4.8.7+dfsg-18+rpi1+deb10u1) ...
Setting up libjasper-runtime (1.900.1-debian1-2.4+deb8u1) ...
Setting up libhdf5-103:armhf (1.10.4+repack-10) ...
Setting up python3-h5py (2.8.0-3) ...
Setting up libqtcore4:armhf (4:4.8.7+dfsg-18+rpi1+deb10u1) ...
Setting up libqtgui4:armhf (4:4.8.7+dfsg-18+rpi1+deb10u1) ...
Setting up libqt4-xml:armhf (4:4.8.7+dfsg-18+rpi1+deb10u1) ...
Setting up libqt4-test:armhf (4:4.8.7+dfsg-18+rpi1+deb10u1) ...
Setting up libqtdbus4:armhf (4:4.8.7+dfsg-18+rpi1+deb10u1) ...
Setting up qdbus (4:4.8.7+dfsg-18+rpi1+deb10u1) ...
Setting up libqt4-dbus:armhf (4:4.8.7+dfsg-18+rpi1+deb10u1) ...
Setting up qt-at-spi:armhf (0.4.0-9) ...
Processing triggers for man-db (2.8.5-2) ...
Processing triggers for libc-bin (2.28-10+rpt2+rpi1+deb10u1) ...

Creating python virtual env

Installing required python packages

Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting numpy==1.21.4
  Downloading https://www.piwheels.org/simple/numpy/numpy-1.21.4-cp37-cp37m-linux_armv7l.whl (12.3MB)
    100% |████████████████████████████████| 12.3MB 36kB/s
Installing collected packages: numpy
```

```
  Found existing installation: numpy 1.16.2
    Not uninstalling numpy at /usr/lib/python3/dist-packages, outside environment /home/pi/cubotone/src/.virtualenvs
    Can't uninstall 'numpy'. No files were found to uninstall.
Successfully installed numpy-1.21.4
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting picamera[array]
  Downloading https://www.piwheels.org/simple/picamera/picamera-1.13-py3-none-any.whl (154kB)
    100% |████████████████████████████████| 163kB 1.2MB/s
Requirement already satisfied: numpy; extra == "array" in ./.virtualenvs/lib/python3.7/site-packages (from picamera[array]) (1.21.4)
Installing collected packages: picamera
Successfully installed picamera-1.13
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting RubikTwoPhase==1.0.9
  Downloading
https://files.pythonhosted.org/packages/67/2c/a364a9f603f1165ad34f0f914a7fdfe1939d873d39017540c19346ccde5b/RubikTwoPhas
e-1.0.9-py3-none-any.whl (53kB)
    100% |████████████████████████████████| 61kB 1.8MB/s
Installing collected packages: RubikTwoPhase
Successfully installed RubikTwoPhase-1.0.9
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting getmac==0.8.3
  Downloading
https://files.pythonhosted.org/packages/ea/ad/7821cbe819079c037a1e1a89a41d9e43b767052307fa474ed0b425394fcb/getmac-
0.8.3-py2.py3-none-any.whl
Installing collected packages: getmac
Successfully installed getmac-0.8.3
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting adafruit-pca9685
  Downloading https://www.piwheels.org/simple/adafruit-pca9685/Adafruit_PCA9685-1.0.1-py3-none-any.whl
Collecting Adafruit-GPIO>=0.6.5 (from adafruit-pca9685)
  Downloading https://www.piwheels.org/simple/adafruit-gpio/Adafruit_GPIO-1.0.3-py3-none-any.whl
Collecting adafruit-pureio (from Adafruit-GPIO>=0.6.5->adafruit-pca9685)
  Downloading https://www.piwheels.org/simple/adafruit-pureio/Adafruit_PureIO-1.1.9-py3-none-any.whl
Requirement already satisfied: spidev in /usr/lib/python3/dist-packages (from Adafruit-GPIO>=0.6.5->adafruit-pca9685) (3.5)
Installing collected packages: adafruit-pureio, Adafruit-GPIO, adafruit-pca9685
Successfully installed Adafruit-GPIO-1.0.3 adafruit-pca9685-1.0.1 adafruit-pureio-1.1.9
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting opencv-contrib-python==4.1.0.25
  Downloading https://www.piwheels.org/simple/opencv-contrib-python/opencv_contrib_python-4.1.0.25-cp37-cp37m-
linux_armv7l.whl (15.7MB)
    100% |████████████████████████████████| 15.7MB 28kB/s
Requirement already satisfied: numpy>=1.16.2 in ./.virtualenvs/lib/python3.7/site-packages (from opencv-contrib-python==4.1.0.25)
(1.21.4)
Installing collected packages: opencv-contrib-python
Successfully installed opencv-contrib-python-4.1.0.25

configuring gpu mem in config file

Configuring vnc server to run at startup and prepare setup for cubotone

Reboot now? (y/n)
```

35. APPENDIX 2: Useful links:

1. Cube solver:

Hegbert Kociemba solver: https://github.com/hkociemba/RubiksCube-TwophaseSolver

2. Combined Power on / power off (single) button:

https://github.com/lihak/rpi-power-button
https://howchoo.com/g/mwnlytk3zmm/how-to-add-a-power-button-to-your-raspberry-pi

3. Led indicator for power on / power off:

https://howchoo.com/g/ytzjyzy4m2e/build-a-simple-raspberry-pi-led-power-status-indicator

4. Edge detection:

https://medium.com/swlh/how-i-made-a-rubiks-cube-colour-extractor-in-c-551cceba80f0
http://programmablebrick.blogspot.com/2017/02/rubiks-cube-tracker-using-opencv.html
https://programmer.help/blogs/rubik-cube-recognition-using-opencv-edge-and-position-recognition.html

5. Approximated contours

https://docs.opencv.org/4.5.3/dd/d49/tutorial_py_contour_features.html

6. Order coordinates clockwise

https://www.pyimagesearch.com/2016/03/21/ordering-coordinates-clockwise-with-python-and-opencv/

7. Colour space conversion:

From RGB to CIELab colour space conversion: https://gist.github.com/ manojpandey/ f5ece715132c572c80421febebaf66ae

8. Distance between two (L*a*b*) colours:

How to calculate the (CIEDE2000) colour distance between two CIE L*a*b* colours:
https://github.com/lovro-i/CIEDE2000

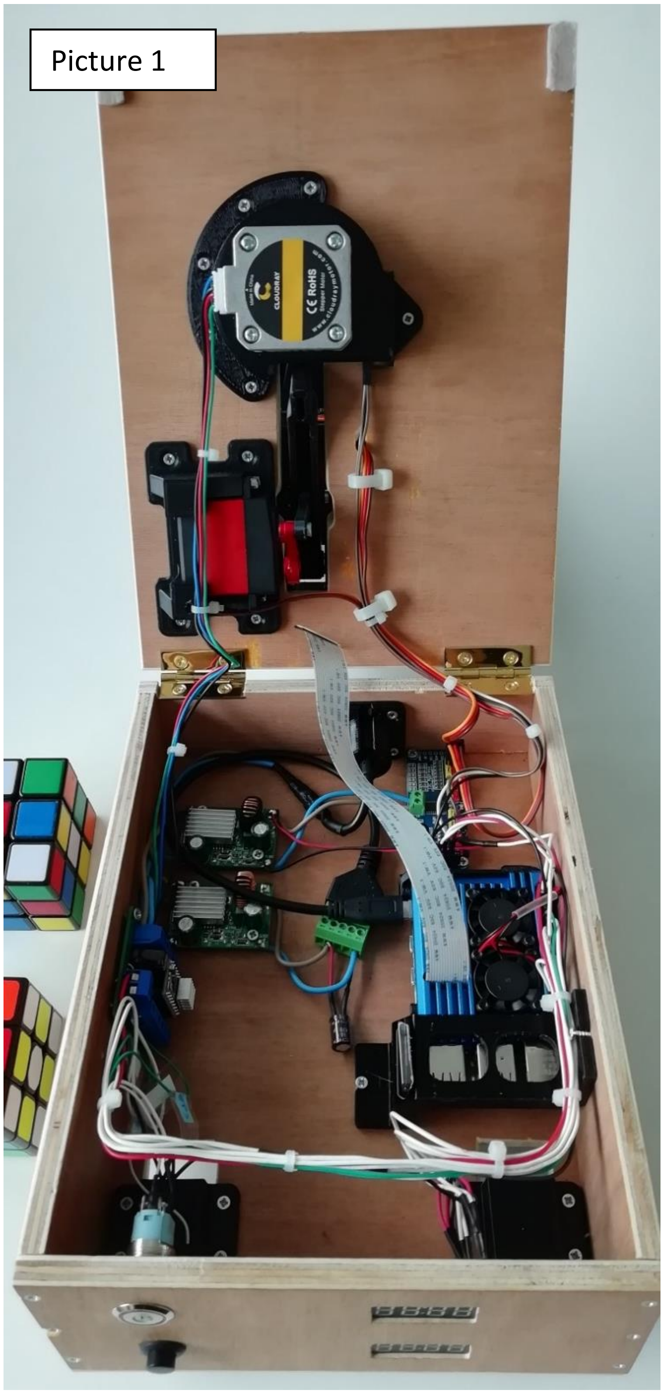
9. How to average two colours on the right way:

https://sighack.com/post/averaging-rgb-colours-the-right-way

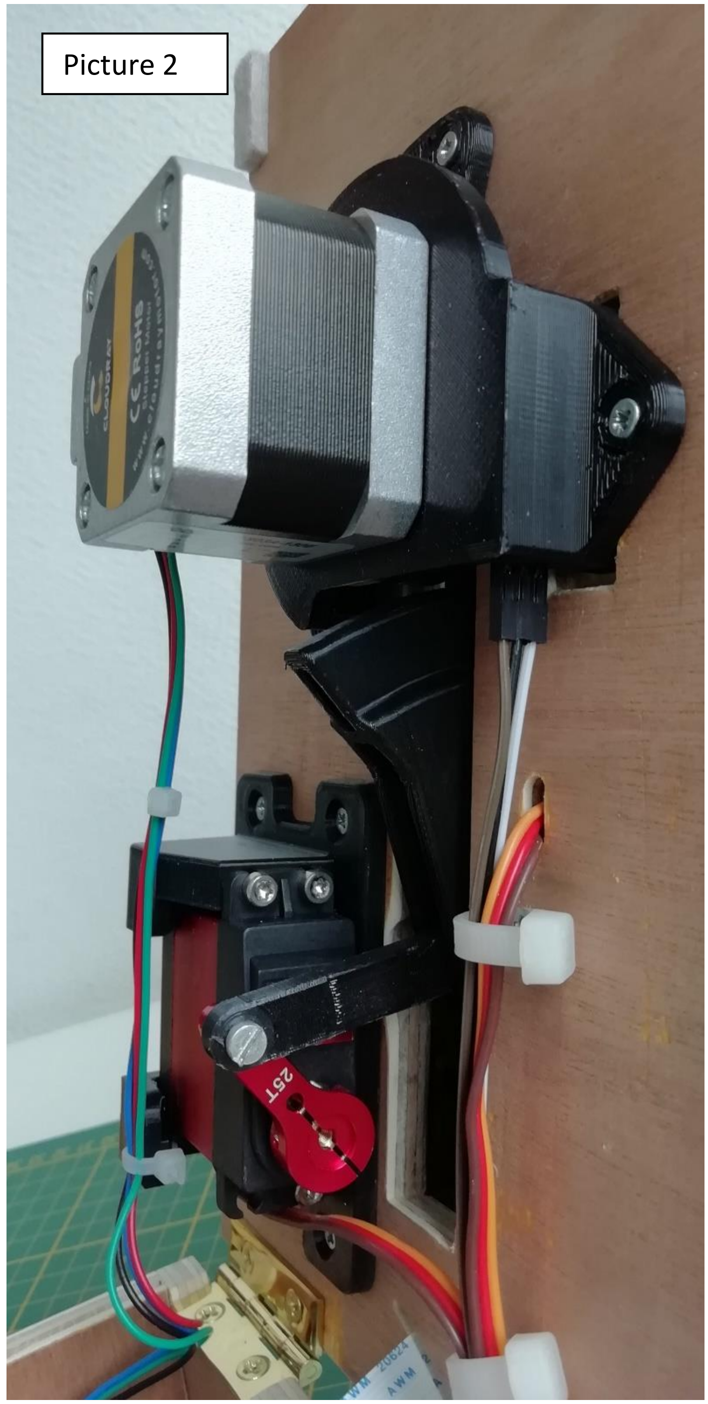
10. Infinite timer:

https://stackoverflow.com/questions/12435211/python-threading-timer-repeat-function-every-n-seconds

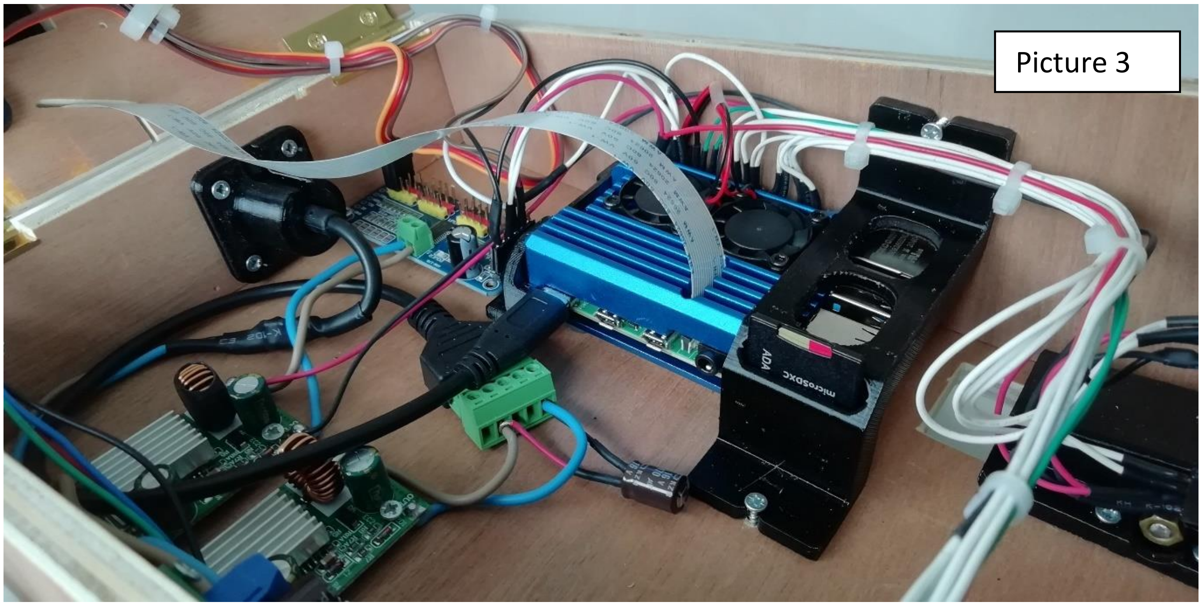
36. APPENDIX 3: Collection of robot's pictures



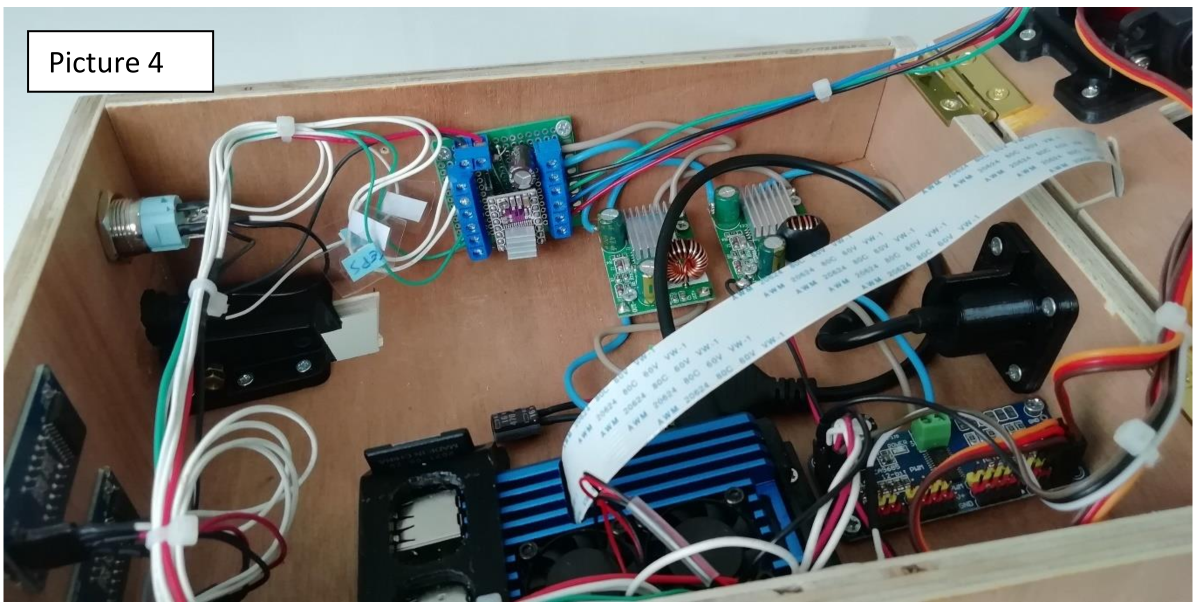
Picture 1



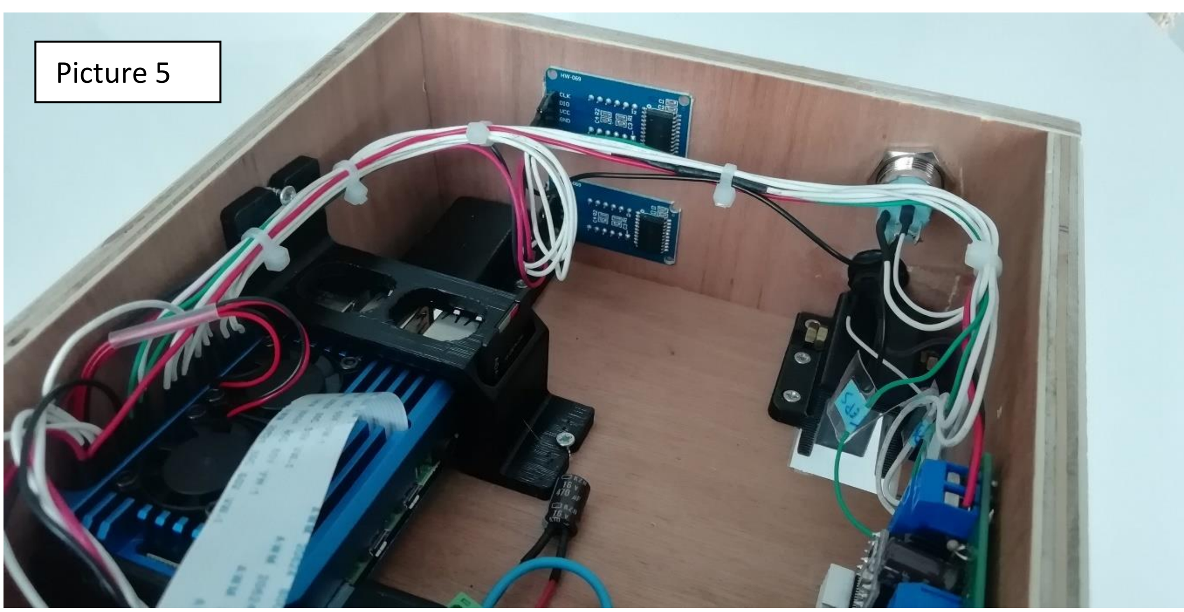
Picture 2



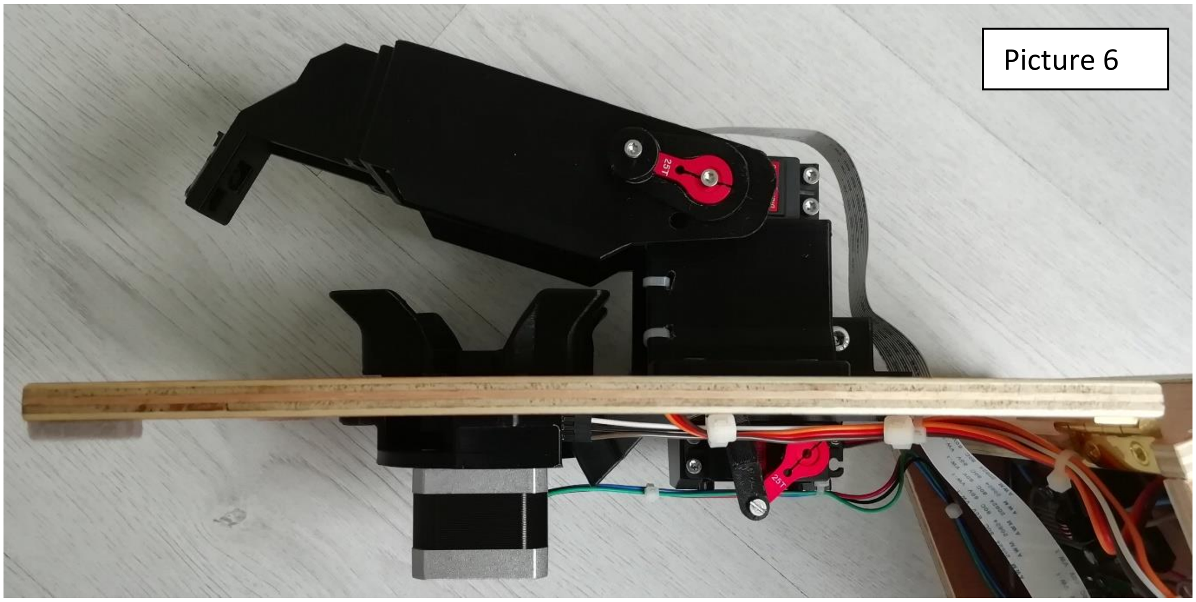
Picture 3



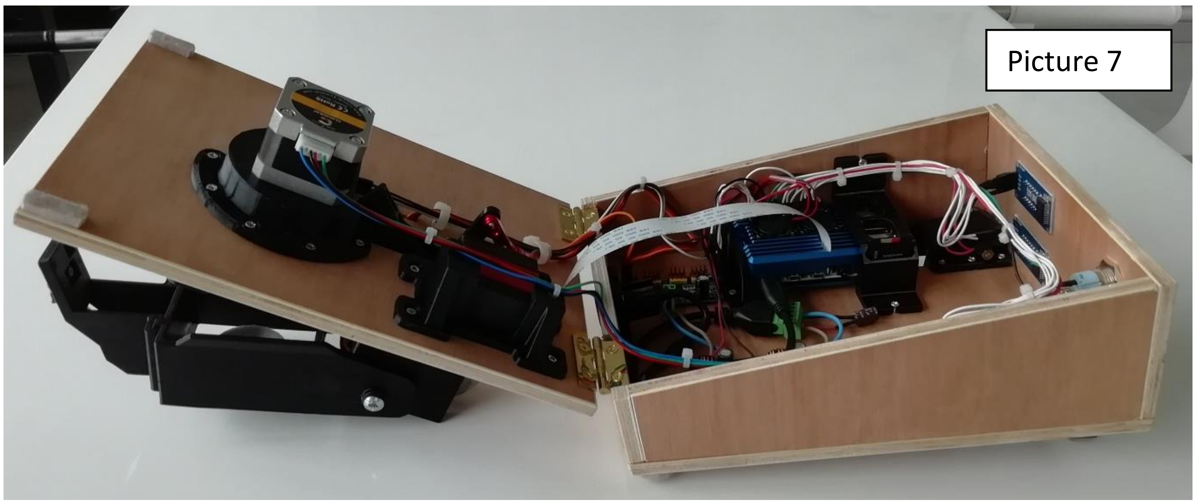
Picture 4



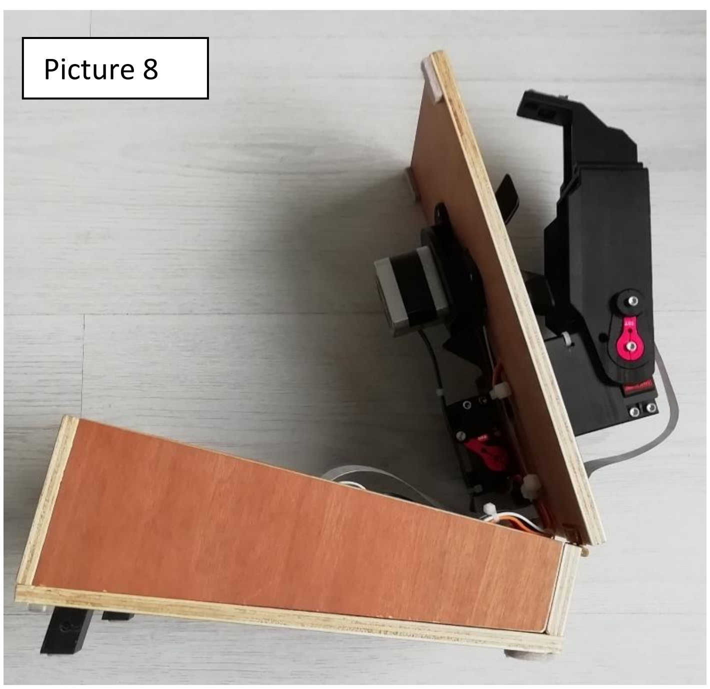
Picture 5



Picture 6

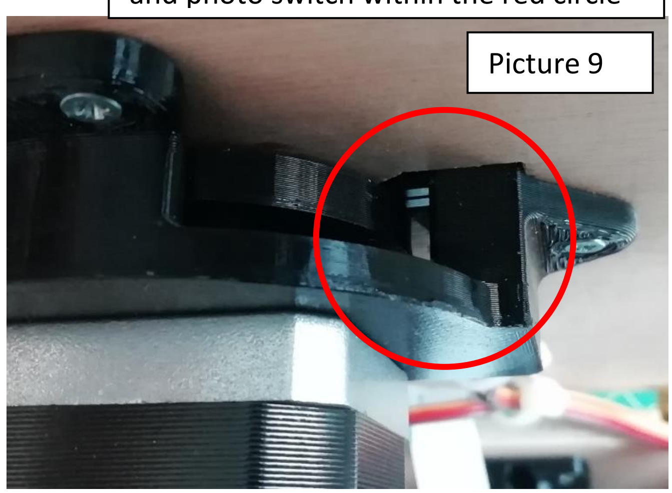


Picture 7

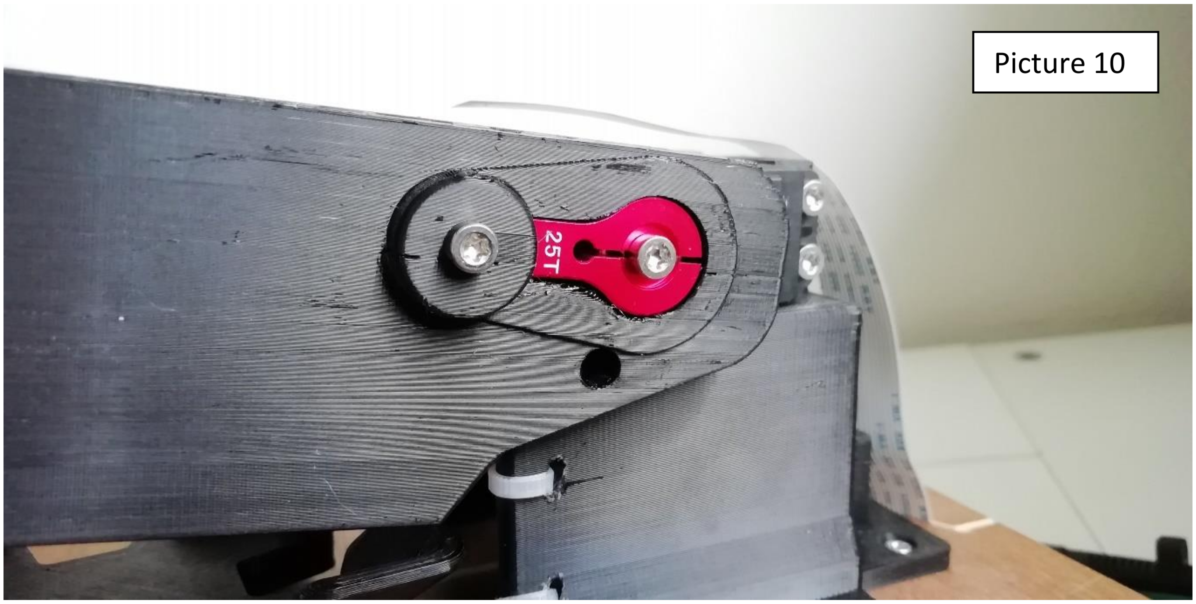


Picture 8

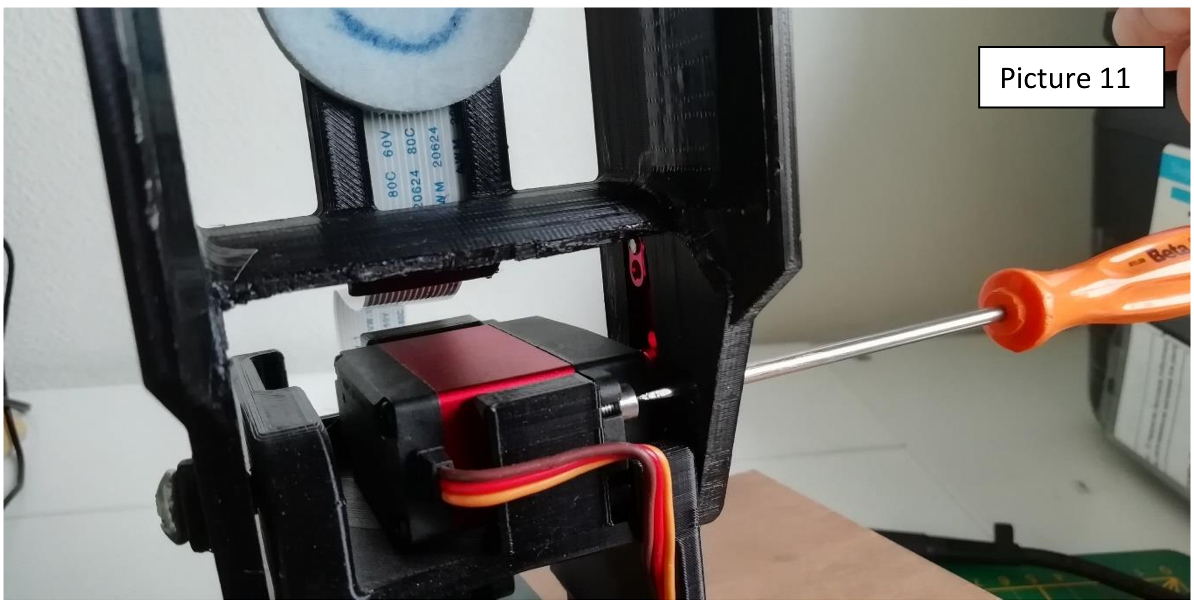
Motor support, synchronization disk, and photo switch within the red circle



Picture 9

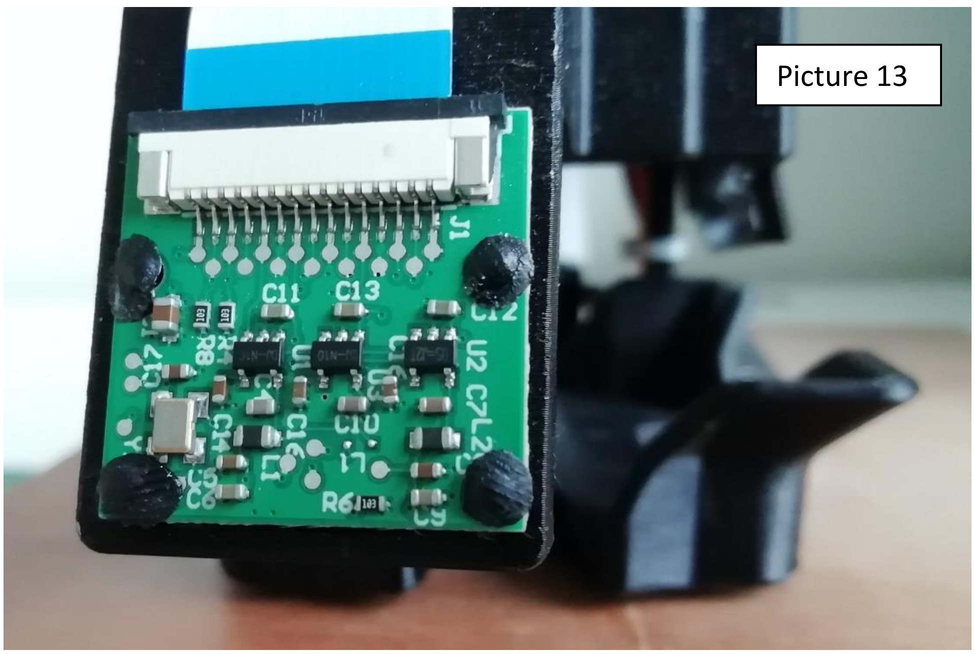


Picture 10



Picture 11

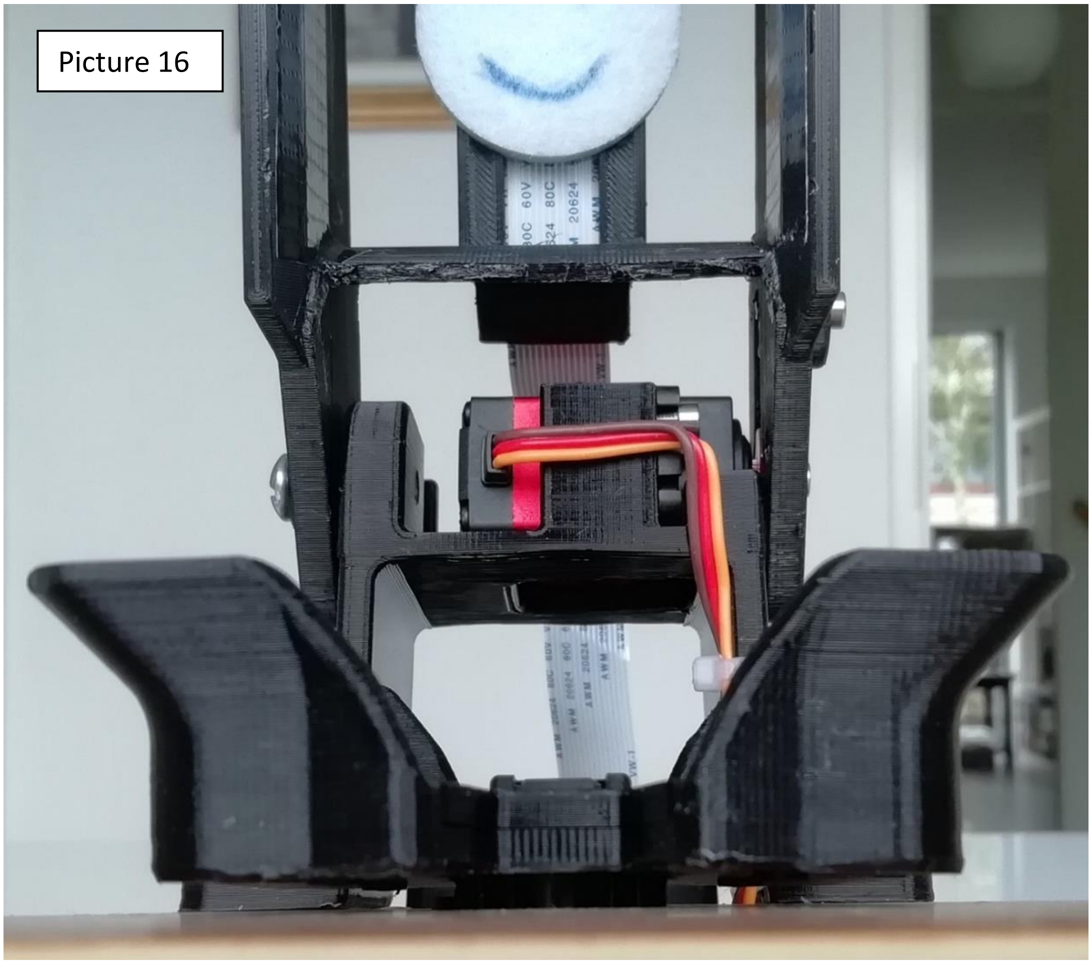
Picture 12



Picture 13

Picture 14

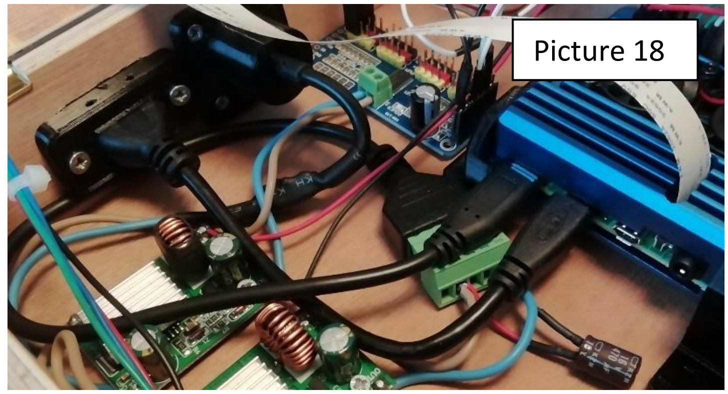
Picture 15



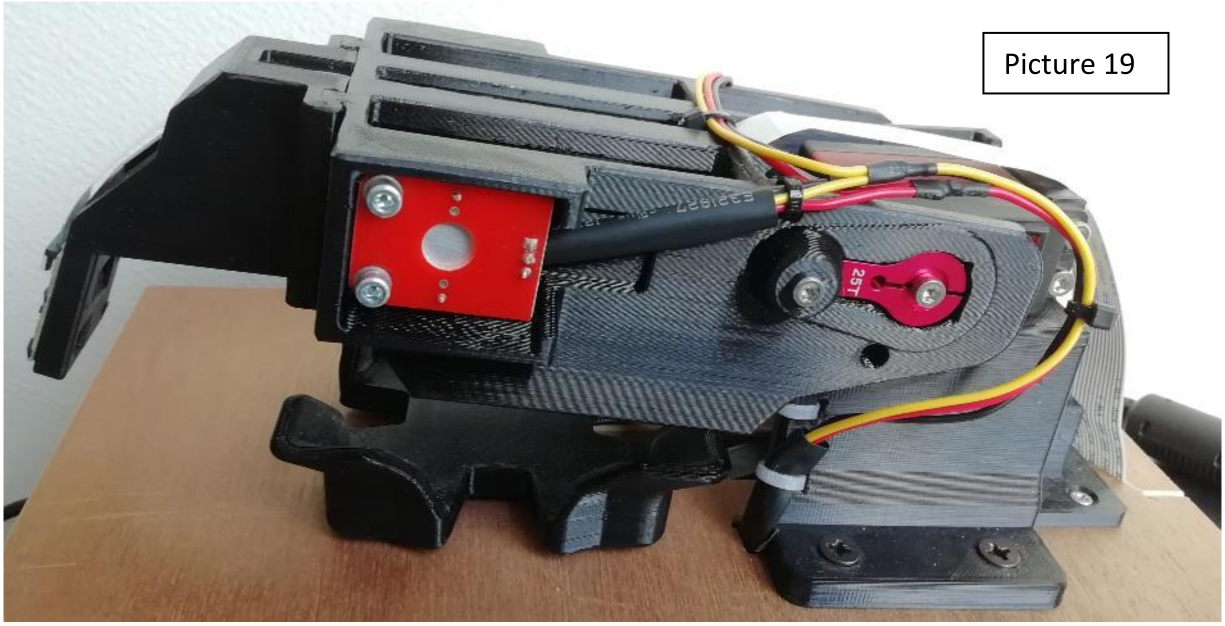
Picture 16

HDMI socket to the rear panel of the robot, connected to the hdmi0 port at raspberry Pi 4B
In this way it is easy to connect an external screen, in case of robot demonstration out of home: The smartphone can be used as Access Point, to interact with the robot, and an external screen to show what the camera sees.
Used the mini hdmi to hdmi cable listed on the bought parts list; I've 3D printed a support fort the hdmi socket, having sufficient interference to keep the connector in position, and fixit to the back panel with 2 screws.
The printed frame keeps the connector inclined (ca 15 degrees), for a better wire routing into the box (to keep distance from the lifter-link) and to prevent long hdmi male connector to eventually touch the table.

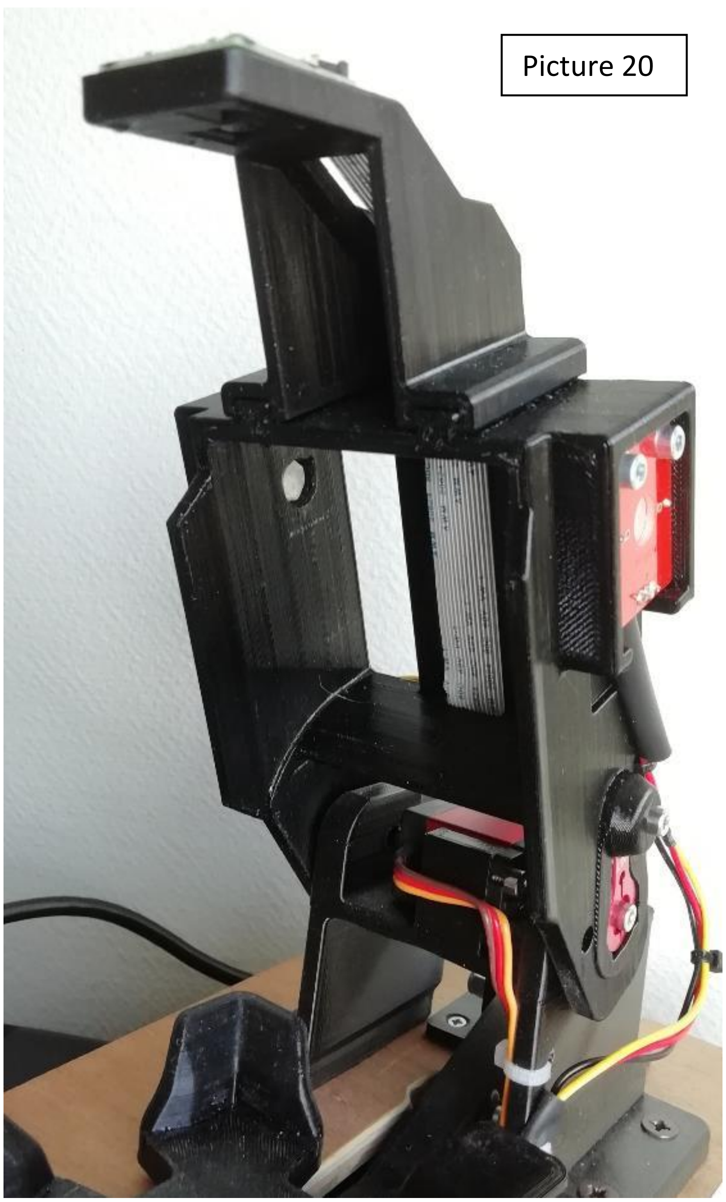
Picture 17



Picture 18



Picture 19



Picture 20

37. APPENDIX 4: Script usage on a (Windows) PC

As I'm quite a basic programmer, my first step has been to write a Python script on a laptop (with webcam), aiming to check whether I was able to detect the cube's facelets, related colours and to interact with the solver.
Only when I accomplished this part, I did start designing the robot and order the components, among them my first Raspberry Pi (also meaning my first time facing a linux similar OS)

While porting the script to the Raspberry pi, I realized it was convenient to have the exact same script (Cubotone.py), that adapts its functions according to where it's working (PC or Raspberry pi).

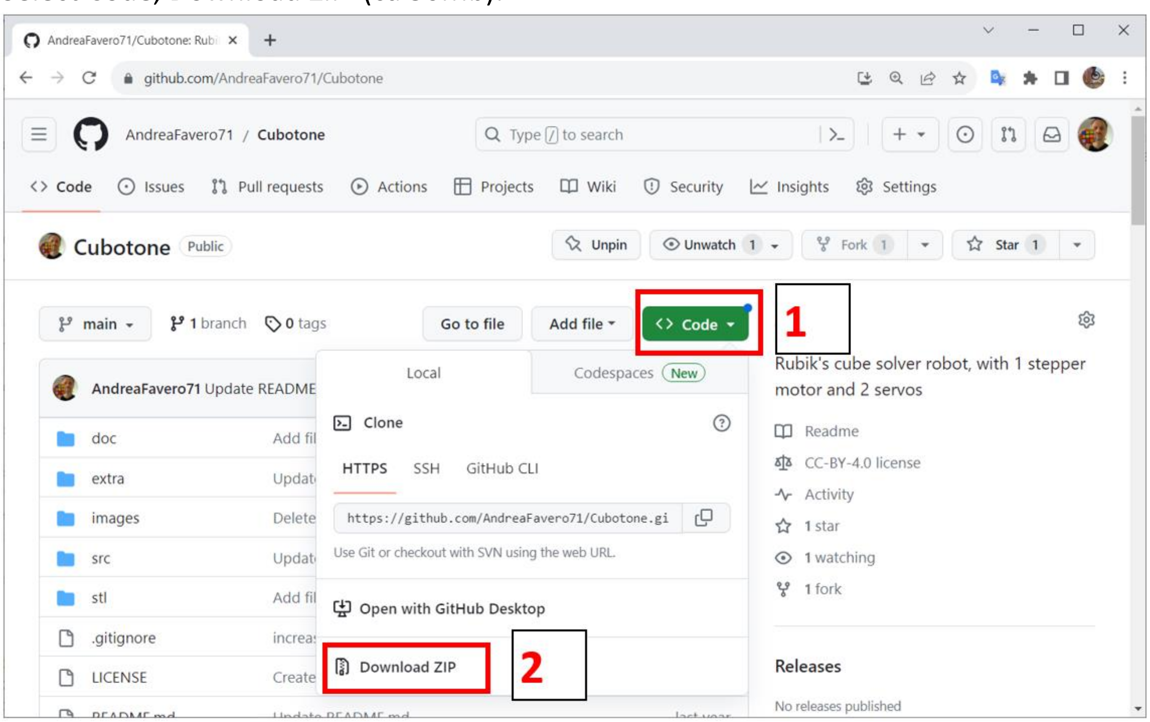
The 1st line of code at Cubotone main is: *device = hd_check()*
The called functions determines whether the script is running at Rpi (robot), if not then it is assumed the script running on a laptop: The function returns either 'Rpi' or 'laptop' string 😊

Cubotone.py script, used on a PC, has the objective to detect the cube status by showing the cube to the webcam, and it returns the solution (movements) from the solver.

The cube must be presented according to the URFDLB sequence, as explained on this tutorial: https://youtu.be/udr6tryxA_Y

In case you'd like to use, test, modify the Cubotone code in a Windows laptop (or PC with webcam):

1. Install python (3.7 or higher) in your PC.
2. Add python to the Windows PATH.
3. Download the Cubotone repository:
   a) Open the online repo https://github.com/AndreaFavero71/Cubotone
   b) Select Code, Download ZIP (ca 50Mb):

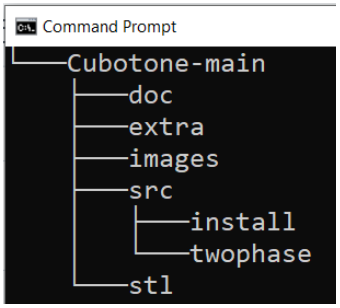


The ZIP file will lend in your download folder.

c) Un-ZIP the complete content to a known location (in my case I chose the root of my d: drive)

d) The main folder name is Cubotone-main-main. Be noted "-main" is added at the end when using the ZIP method; this is not as issue, as long as you are aware of that. You might also decide to rename the folder by removing the "-main" part.

e) The folders' structure will look something like:



4. Files needed are a subset of the complete repository; In case you prefer to keep as little files as possible, the needed files and folders are:
   - */src/Cubotone.py*
   - */src/twophase* folder (complete folder)

5. Make a virtual environment for this project (called *venv* in my case): From the CMD type *python -m venv /path_to_new_virtual_environment*

6. Activate the venv: Type *path_to_new_virtual_environment\Scripts\activate*

7. Within the activated virtual environment, install below libraries:
   a) Numpy
   b) Opencv-python
   c) RubikTwoPhase (it will use the look up tables from the *twophase* folder downloaded from the repository, instead of building them at the first time the solver is called)

8. To run the code:
   a) enter the src folder: *cd path_to_src*
   b) run the code: *python Cubotone.py*
   c) the *- -delay* argument can be added, to delay the facelets detection when changing the cube orientation in front of the webcam.
   d) the *- -debug* function can be used to printout some additional info to the CLI.

9. After closing Cubotone, the virtual environment can be deactivated: *deactivate*

10. Of course, the repository can be downloaded via the Git command, making it prone for future updates; People comfortable with Git do not need any explanation from me 😊.