

How to build a (3x3) Rubik cube robot solver, with Raspberry pi and PiCamera

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

Andrea Favero, Groningen (NL)

Rev.2 ->27/04/2022

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



It typically 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 it for mechanical gripping, as most of the fast robots use to).

Index

1. Project scope
2. Conclusions
3. Commitment
4. High level info
5. Colour's detection strategy
6. Construction
7. Made parts
8. Bought parts
9. Electrical scheme
10. Modules and connections
11. Proto board for DRV 8825 driver
12. (Raspberry pi) GPIO pinout
13. Assembly steps and attention points
14. Setting up the microSD and Raspberry pi
15. Python script, high level info
16. How to operate the robot
17. Fine tuning
18. Collection of the robot's pictures
19. Useful links
20. Memory usage
21. Revisions

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

I'm turning 50 this year, and I discovered I like to keep myself busy with coding and controls.

I thought a Rubik cube robot solver to be a good (and challenging) project for above objectives.

Conclusions:

It's usual to write conclusions at the end of a presentation; I'm writing this document after some months the robot is up and running.... And I've quite clear in mind the positives and negatives 😊

I believe I've accomplished all the objectives, especially the one related to my age.

I'm satisfied about the learning journey, and the end results.

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 when falls into the cube-holder make 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 too many global variables on this project.

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

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, and if that's the case please consider to leave a message/thumbs up on Youtube (<https://youtu.be/oYRXe4NyJqs>) or at the Instructable site.

In case you cannot find the solution by yourself (part that makes projects fun 😊), please drop a detailed question at the instructables site (<https://www.instructables.com/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....

High level info:

1. The robot uses a Raspberry pi 4, with 2 Gb of RAM, running with Raspberry Pi OS on a 32Gb microSD;
2. The robot has a vision system based on a PiCamera (v1.3), to detect the facelets positions and related colours.
3. The robot is coded in Python, and the exact same script can be used on PC (i.e. Windows laptop) to determine the cube status via a webcam and to get the solution string.
4. The vision part uses CV2 python library
5. 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
6. Cube's orientation considers the Western colour scheme: <https://ruwix.com/the-rubiks-cube/japanese-western-color-schemes/>

Western color scheme

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

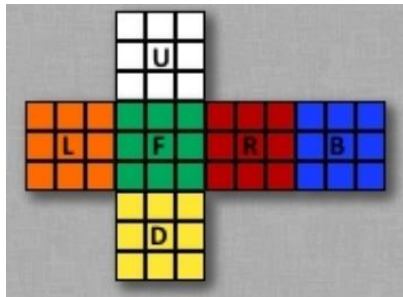
Cubers who use this color scheme usually start [solving the Rubik's Cube](#) with the white face and finish with the yellow.

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

white + yellow = yellow

red + yellow = orange

blue + yellow = green



7. Cube solver uses the Hegbert Kociemba, "two-phase algorithm in its fully developed form with symmetry reduction and parallel search for different cube orientations":
 - intro: <https://www.speedsolving.com/threads/3x3x3-solver-in-python.64887/>
 - Python script: <https://github.com/hkociemba/RubiksCube-TwophaseSolver>The solver requires as input the cube status, by following the URFDLB order (I called it "Kociemba order" along the script).
I'd like to thank Mr. Kociemba for his great solver, and for making it available: Pretty much appreciated.
8. When using the python script on a PC, the western colour scheme sequence is suggested on screen for guidance.
9. When using the robot, the cube can be dropped with any orientation; If the orientation differs from the Western colour scheme, then the cube status is "converted" right before calling the solver.
10. 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

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

Colour's detection strategy:

1. The vision system is used to detect the cube's facelets edges (contours), more or less as explained at <https://medium.com/swlh/how-i-made-a-rubiks-cube-color-extractor-in-c-551cceba80f0>
2. Average BGR is calculated for the areas defined by the contours, for the 54 facelets, and stored in a dictionary; Average HSV (of a small area at each contour's centers) is also calculated and stored on a second dictionary.

Note: On a 3x3 Rubik cube, the 6 center's facelets have useful properties:

- a. These facelets don't move (fix facelets number)
- b. These facelets have (obviously) 6 different colours
- c. Opposite faces have known colours couples, white-yellow, red-orange, green-blue (Western colour code).

This means we can make use of these 6 facelets as colour reference

3. The average HSV, detected on the 6 centers, is used to determine which colour is located on the 6 centers:
 - a. White facelet is the one having the largest V-S delta (difference between Value, or Brightness, and Saturation), while the yellow one is located at opposite face.
 - b. Remaining 4 centers are evaluated according to their Hue, and the Hue at opposite face.
 - c. Orange has very low Hue, and red should be very high (almost 180); Depending on light condition, the red's Hue could be very low (few units) and lower than the orange, in that case both Orange and red are very low, yet orange is always higher than red.
 - d. Out of the two remaining centers, blue is the one with highest Hue, and consequently the green is also known.
4. Based on previous step, the 6 cube colours (at least their centers) have a known average HVS and therefore an average BGR colour; This also informs on the cube orientation (colours) as placed on the cube-holder.
5. Facelets colour interpretation is made, by using two methods:
 - a. The first method compares the average RGB colour of each facelet in comparison with the one at the 6 centers, and the colour decision is based on the smallest colour distance.
 - b. In the second method the Hue value of each coloured (non-white) facelet are compared to the Hue of the 5 reference centers; White facelets are retrieved according to 3 parameters (Hue, Saturation, Value), in comparison to the white center HSV.

First method is in general better than the second one, yet the second one "wins" when there is lot of light; The second method is only used (called) when the first one fails.

As result both methods are used, to get reliable cube status detection under different light situations.

Note:

On the robot, the PiCamera is positioned rather close to the cube, imposing the usage of a large sensor area to have the cube fully visible; This increases the vignetting effect, resulting on darker colours at the corners.

The first colour interpretation method mitigates this effect, by using the central facelets **only as initial** colour reference:

- Facelets are first ordered by their colour distance to the reference.
- Once a facelet's colour choice is made (shortest distance from reference), that facelet's colour is averaged to the previous reference colour, and stored as the new reference for that specific colour

By averaging the reference to every new selected colour, the reference colour becomes more and more representative for that specific colour

Construction:

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

Baseline:

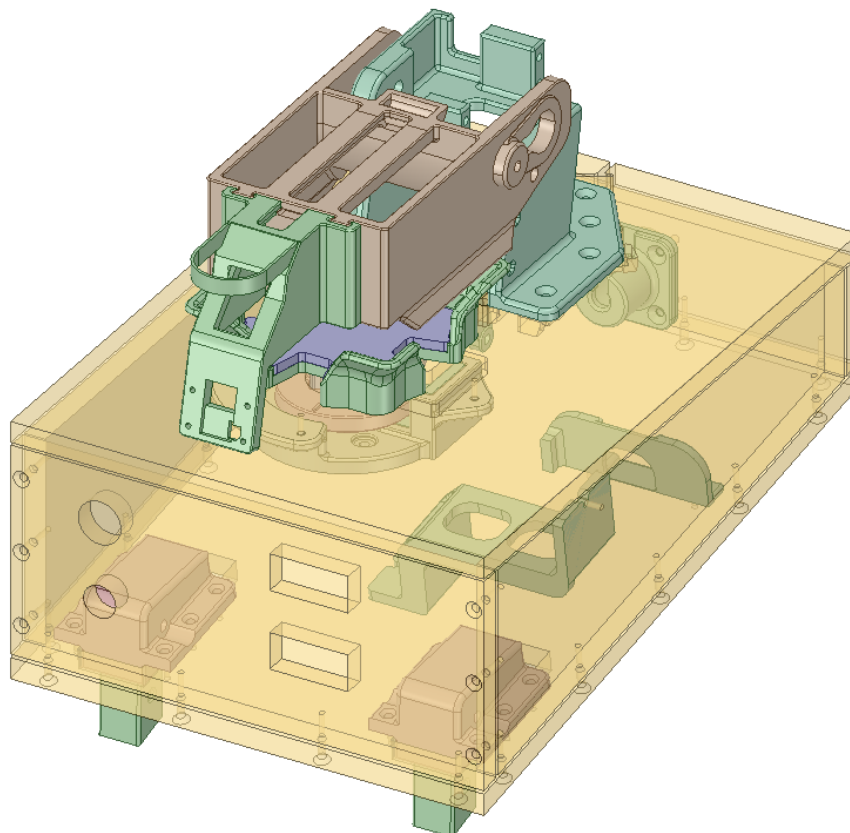
- The inclined cube-holder is inspired to Hans construction <https://www.youtube.com/watch?v=gbLnNIZ-tro>
 - The upper-cover is my own idea; It provides a constrainer for cube layer rotation, it suspends the PiCamera at sufficient distance when reading, and allows a compact robot construction when closed (robot not in use).
 - The cube flipper is also my own idea, mainly to keep a small footprint and to have independent movements.
 - 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).
 - The Box cover is hinged to the Box to quickly access/show the inner parts.
1. Upper-cover and flipping lever are actuated via two servos, therefore controlled via angle.
 2. Cube-holder is actuated via a step motor, therefore controlled by number of current steps.
 3. 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.
 4. 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.
 5. Most of the parts are made in PET-G via 3D printing.
 6. Some of the 3D printed parts are split:
 - Upper-cover and PiCamera holder, to have smaller parts to re-print in case of geometry improvement need.
 - Cube-holder and connection to the motor, mainly to have a convenient printing.
 - Flipper's servo holder and Hinge for Upper-cover and Lifter, mainly for convenient printing
- Note: PiCamera holder 3D file has an additional bracket, meant to improve support while 3D printing, and to be cut afterward
7. 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 to:
 - Integrate a couple of parts:
 - Bottom panel with foot hinges
 - Top panel with the hinge for upper-cover and lifter
 - On the stl files, for the Box, there aren't the recesses/fixing holes for the Box Top panel hinges.

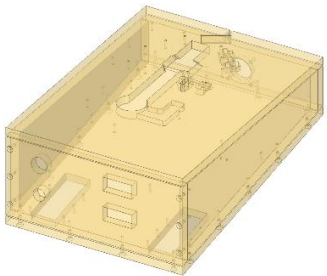
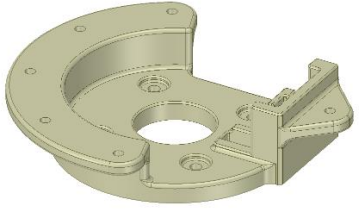
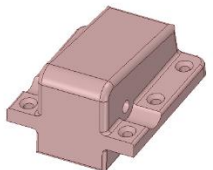
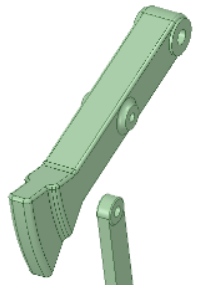


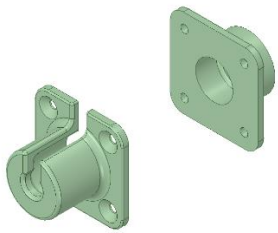
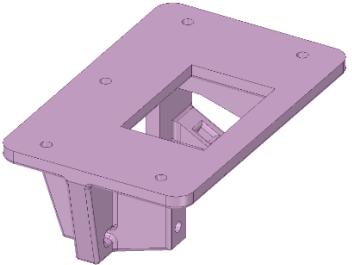
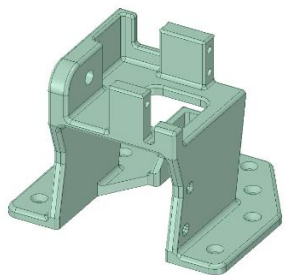
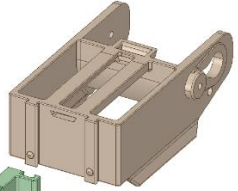
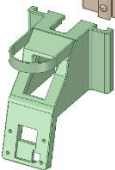
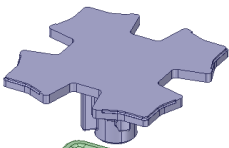
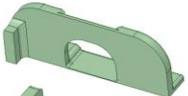
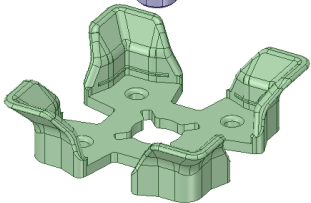
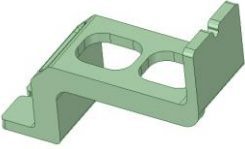
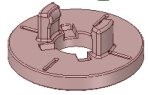
Parts to make:

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 all the above parts are provided at the instructables site




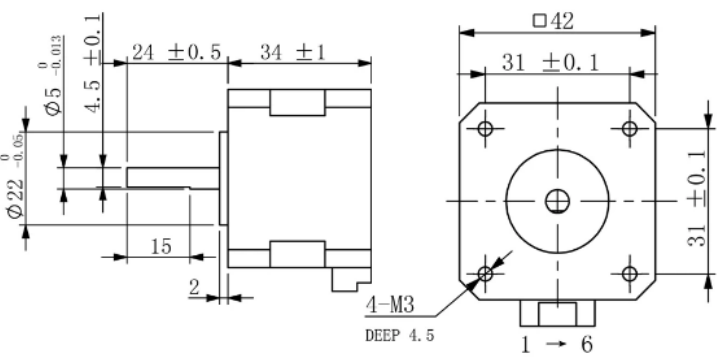
<https://www.instructables.com/Rubik-Cube-Solver-Robot-With-Raspberry-Pi-and-Pica/>


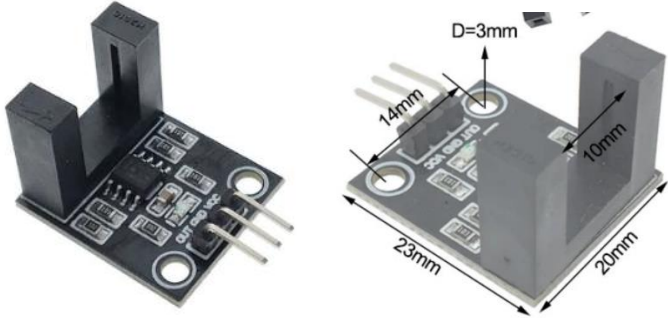





Box (6 stl files)		Motor support	
Foot hinge		Lifter	
Foot		Lifter-link	
Outer connector holder		Lifter servo holder	
Inner connector holder			
Hinge for upper-cover and Lifter		Upper-cover	
		PiCamera holder	
Cube-holder bottom part		Raspberry pi back holder	
Cube-holder upper part		Raspberry pi front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

Bought parts:

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 <u>cover</u> with fans (Not sure the fans are really needed)	
2	MicroSD Sandisk Extreme 32Gb (2 nd 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"	180 Degree Servo 2PCS + 25T Arm 2PCS (Control by Remote Control) 
1	Servo(s) driver (PCA 9685)	Better to buy 1 spare
1	Step motor Nema 17 Stepper Motor, 34mm, 28Ncm, 1.3A, 2 phases, 1.8°, shaft Ø5mm with flat key	
1	Step motor driver, 1.5A (DRV8825)	Better to buy 1 spare

2	<p>DC-DC transformer</p> <p>(1 for Raspberry pi, the 2nd for servos and remaining loads)</p>	
1	<p>Photo switch</p> <p>(Better to buy 1 spare)</p>	
1	<p>Cable USB-C with screw connector</p>	
2	<p>LCD displays with segments</p> <p>(Better to buy 2 spares)</p>	
1	<p>Momentary push-button with red led</p> <p>On / Off logo</p> <p>(Better to buy 1 spare)</p>	 <p>Size:in MM</p>

1	Push-button (Better to buy 1 spare)	
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 \geq 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:

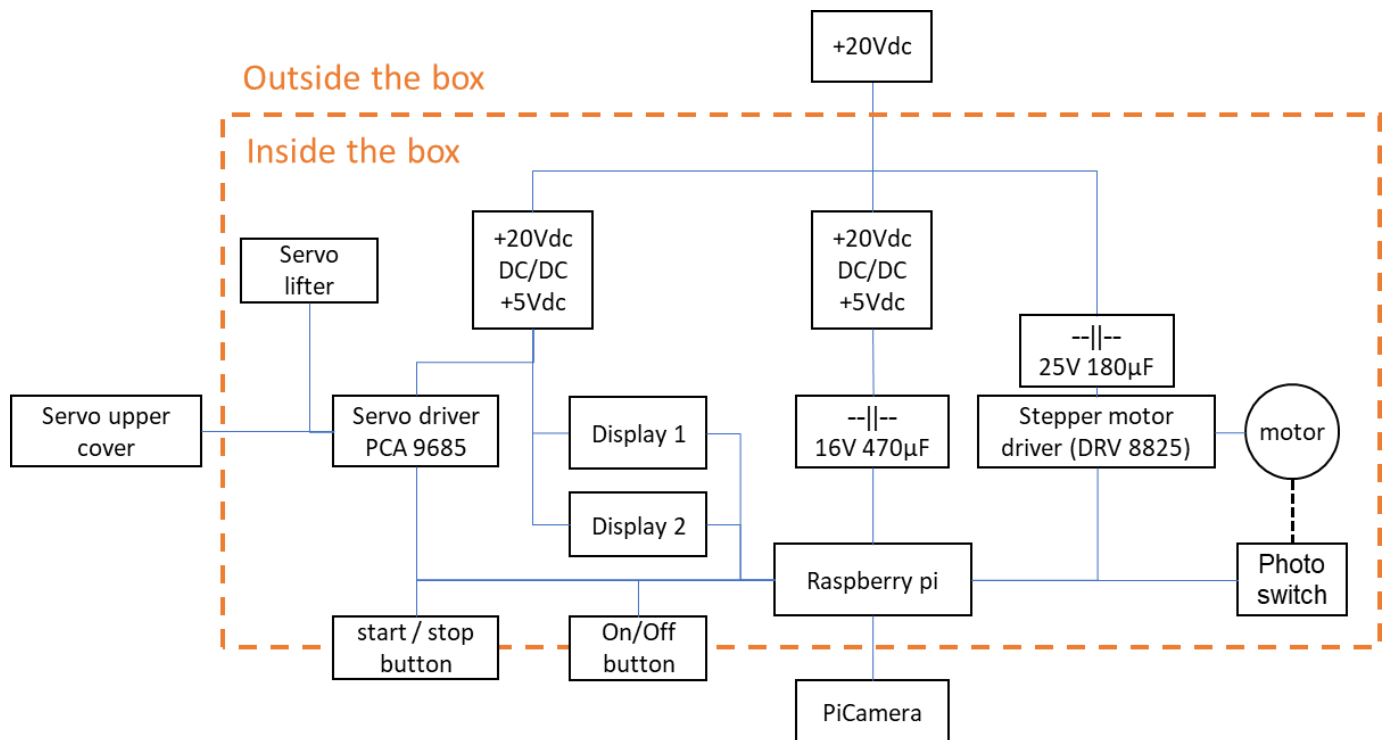
Q.ty	Part	Notes
50	2.5x13mm wood screw	For the Box construction, and thicker part toward the Box
50	2.5x10mm wood screw	For thinner parts toward the Box
2	M3x 30 + self-locking nuts	For the feet and feet hinges
1	M3x16	Lifter lever to lifter lever link
8	M3x10	Servo to servo holder (3 each) Upper-cover to servo metal lever "25T" Flipper lever link to servo lever
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 K Ω at On-off button (not clear specs on max current) 1 x 10 K Ω at stepper motor driver, to pull up the <i>enable</i> signal 1 x 4.7 K Ω at stepper motor driver if A4988 driver, to pull up the <i>step</i> signal

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

Electrical scheme:

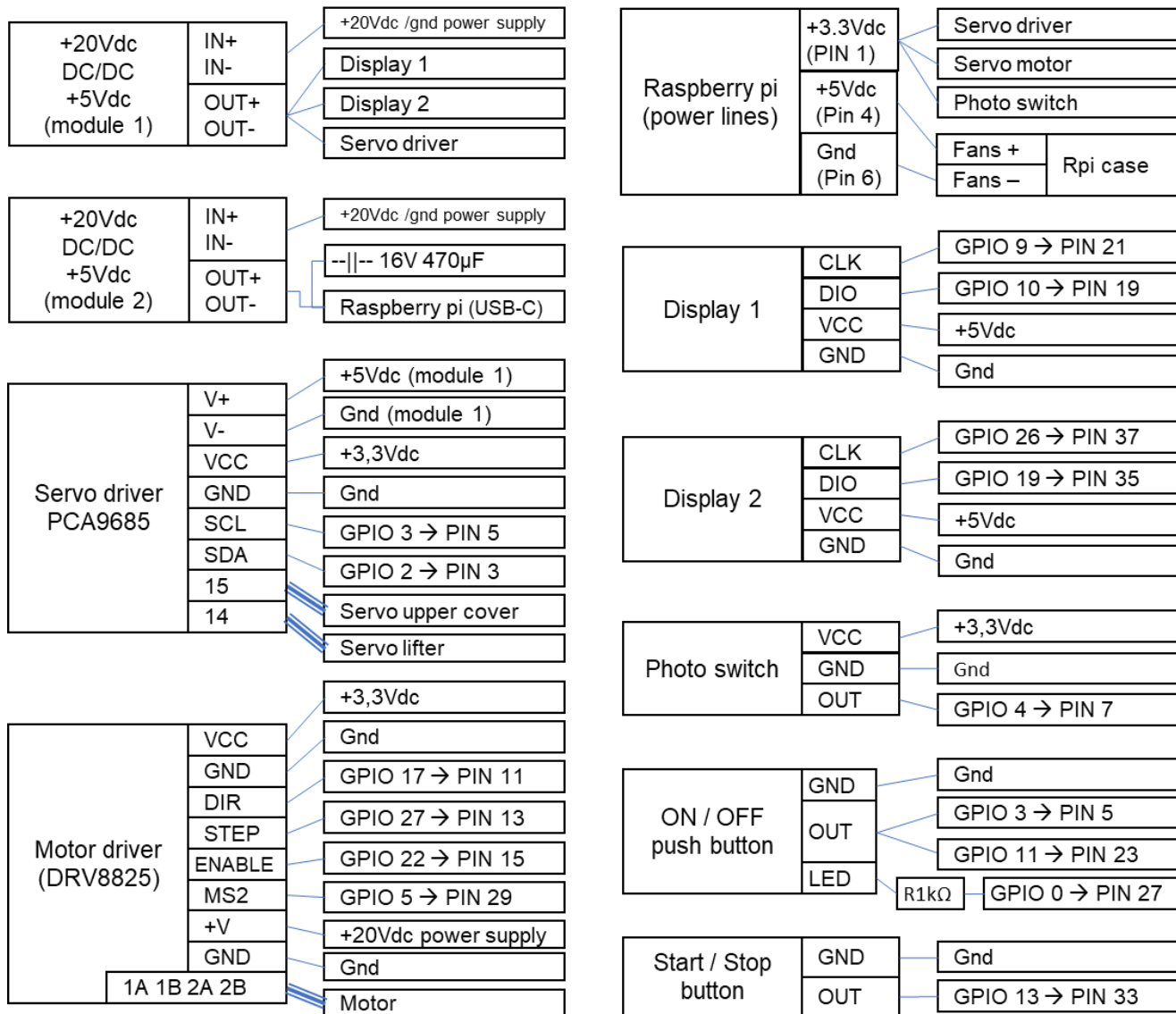


Note:

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.

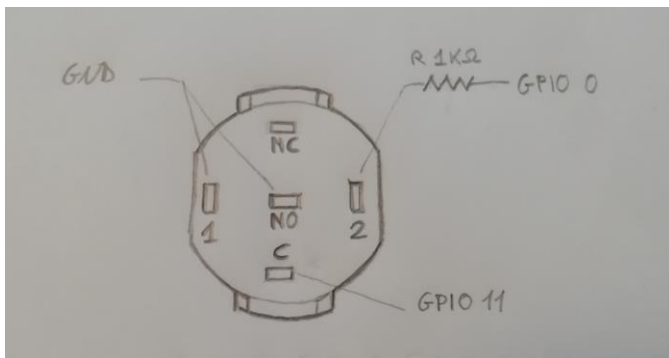
Modules and connections:



Few notes:

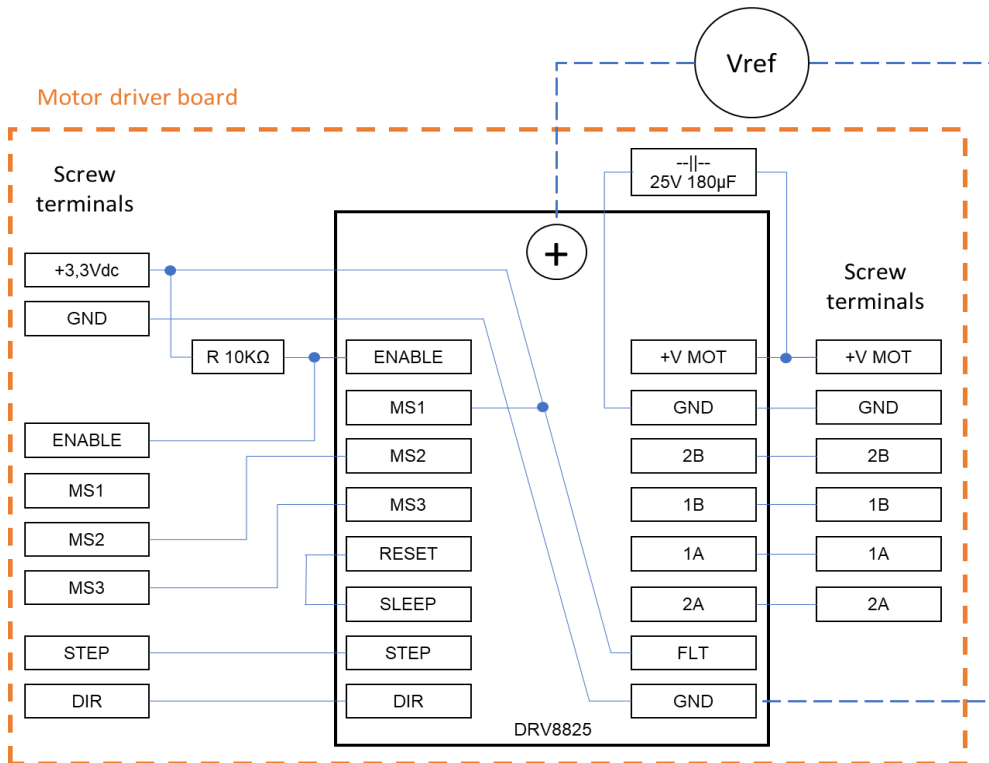
GPIO 3 (pin 5) is connected to both the ON / OFF push button and the SCL line of the Servo Driver

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).



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.

$R_{sense} = 0.1\Omega$
 $V_{ref} = I_{max} / 2$
 Max current = 1.3A (in line with the stepper motor I bought)

EN = Enable - Active LOW, (default state)
 Leave unconnected if always enabled

M0 = Mode 0 (Set microstep size)
 Leave unconnected for full Step Mode

M1 = Mode 1 (Set microstep size)
 Leave unconnected for full Step Mode

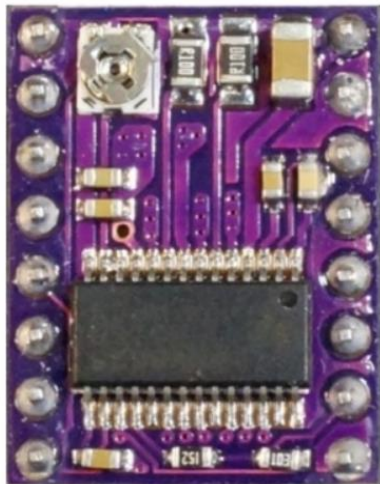
M2 = Mode 2 (Set microstep size)
 Leave unconnected for full Step Mode

RST = Reset - Active LOW (default state)
 Must pull high to take out of reset

SLP = Sleep - Active LOW (default state)
 Must pull high to take out of sleep

STP = Step Input (pulse increments step)
 Driven by microcontroller

DIR = Direction Input (rotation direction)
 Driven by microcontroller



VMOT = Motor Voltage (8.2 - 45V)

GND = Motor Power Supply Ground

2B = Stepper Coil B (leg 2)

1B = Stepper Coil B (leg 1)

1A = Stepper Coil A (leg 1)

2A = Stepper Coil A (leg 2)

FLT = Fault Output - Active LOW when fault detected

GND = Microcontroller Ground

Notes:

1. MS1 (micro step1) is connected to +3.3Vdc, therefore high: It forces the driver to microstep $\frac{1}{2}$ (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 $\frac{1}{8}$ (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

(Raspberry pi) GPIO pinout:

3v3 Power	1	2	5v Power
GPIO 2 4 SPI3 MOSI SDA1 4 SDA3	3	4	5v Power
GPIO 3 4 SPI3 SCLK SCL1 4 SCL3	5	6	Ground
GPIO 4 4 SPI4 CE0 N 4 TXD3 4 SDA3	7	8	TXD0 4 TXD1 GPIO 14 4 SPI5 MOSI
Ground	9	10	RXD0 4 RXD1 GPIO 15 4 SPI5 SCLK
GPIO 17 SPI1 CE1 N	11	12	GPIO 18 SPI1 CE0 N 4 SPI6 CE0 N
GPIO 27 4 SPI6 CE1 N	13	14	Ground
GPIO 22 4 SDA 6	15	16	GPIO 23 4 SCL6
3v3 Power	17	18	GPIO 24 4 SPI3 CE1 N
GPIO 10 SPI0 MOSI 4 SDA5	19	20	Ground
GPIO 9 SPI0 MISO 4 RXD4 4 SCL4	21	22	GPIO 25 4 SPI4 CE1 N
GPIO 11 SPI0 SCLK 4 SCL5	23	24	4 SDA4 GPIO 8 SPI0 CE0 N 4 TXD4
Ground	25	26	4 SCL4 GPIO 7 SPI0 CE1 4 SPI4 SCLK
GPIO 0 4 SPI3 CE0 N 4 TXD2 SDA0 4 SDA6	27	28	4 SPI3 MISO GPIO 1 SCL0 4 SCL6 4 RXD2
GPIO 5 4 SPI4 MISO 4 RXD3 4 SCL3	29	30	Ground
GPIO 6 4 SPI4 MOSI 4 SDA4	31	32	4 SDA5 GPIO 12 4 SPI5 CE0 N 4 TXD5
GPIO 13 4 SPI5 MISO 4 RXD5 4 SCL5	33	34	Ground
GPIO 19 SPI1 MISO 4 SPI6 MISO	35	36	GPIO 16 4 SPI1 CE2 N
GPIO 26 4 SPI5 CE1 N	37	38	GPIO 20 SPI1 MOSI 4 SPI6 MOSI
Ground	39	40	GPIO 21 SPI1 SCLK 4 SPI6 SCLK



General Purpose
Input Output



SPI (Serial Peripheral Interface)



I2C (Inter-Integrated Circuit)



UART (Universal Asynchronous
Receiver / Transmitter)



Ground (GND)



5v Power



3v Power



2 Physical Pin Number



4 Pi 4 Only

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*, check whether the cube is on focus (see Fine Tuning part).

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)

Setting up the microSD and raspberry pi:

Overall steps are:

Step #1: Setting up RASPBERRY PI (4B)

Step #2: Install dependencies for CV (virtual environment)

Step #3: Install packages for cube solver

Step #4: Get the robot starting, after raspberry pi boots (this step can be done at later stage)

Step #5: Make an image backup of the microSD

Details of these steps are on the embedded document *SD and Rpi settings_20211003.pdf*



SD and Rpi
settings_20211003.pdf

I'm not an expert, yet I had done these steps twice as the first microSD card crashed; In both cases the process was smooth but not short!

I hope these steps won't change much in future, fortunately many helps can be found in internet.

Python script, high level info:

1. There are two main python scripts on the robot: *AF_cube_robot.py* and *AF_cube_robot_noVideo.py*
2. The difference between these two files belongs to the Boolean variable `screen`, on the "`__main__`" part: `screen=True` on *AF_cube_robot.py* while `screen=False` on *AF_cube_robot_noVideo.py*.
3. The script *AF_cube_robot_noVideo.py* starts automatically at the robot, after the Raspberry pi boots; This script doesn't use commands requiring a screen communication, thus the robot works autonomously without any screen connected to it (and without returning errors).
4. Alternatively, the script *AF_cube_robot.py* starts only if manually started, and it makes use of commands requiring screen communication; With this script there are some graphical information shared, that makes the robot working more enjoyable. In case there isn't a screen connected, the script doesn't work properly.
5. The script *AF_cube_robot.py* works in either the robot (Raspberry pi) and a python interpreter, in my case in windows laptop; In the script there is a trivial check to determine whether it is running on the robot ... if not it considers to be on a PC.
6. Depending on where the script is running, different libraries are loaded; Below table is mainly meant to show the split of needed libraries for the robot and those to play with the Vision part on a PC.

Library	HD	Main scope	notes
cv2	both	For the vision part	
numpy	both	Array related analysis	
copy (deepcopy)	both	To manipulate copy of the original array (image)	
scipy.spatial (distance)	both	Distance calculation on array	
math	both	For different math need	
statistics	both	To easily calculate the median	
time	both	To manage time	
datetime	both	To log date and time when saving data	
platform, subprocess	both	To determine the system and clear the terminal	
solver	both	Kociemba solver for the cube	see below
os	robot	To manage folder and file presence check	
RPi.GPIO	robot	To manage the GPIO at Raspberry pi	
threading (Timer)	robot	To manage repetitive timers	
picamera	robot	To manage the PiCamera	
multiprocessing	robot	To multiprocessing while the robot solves the cube	
AF_set_picamera_gain	robot	To manage settings on PiCamera	modified
AF_tm1637	robot	To manage the displays	modified
AF_robot_moves	robot	To translate the cube solution in robot's movements sequence	specific
AF_servo_and_motor	robot	To operate the robot moving parts	specific
IPython.display (clear_output)	PC	To clear the terminal in between phases	

Notes:

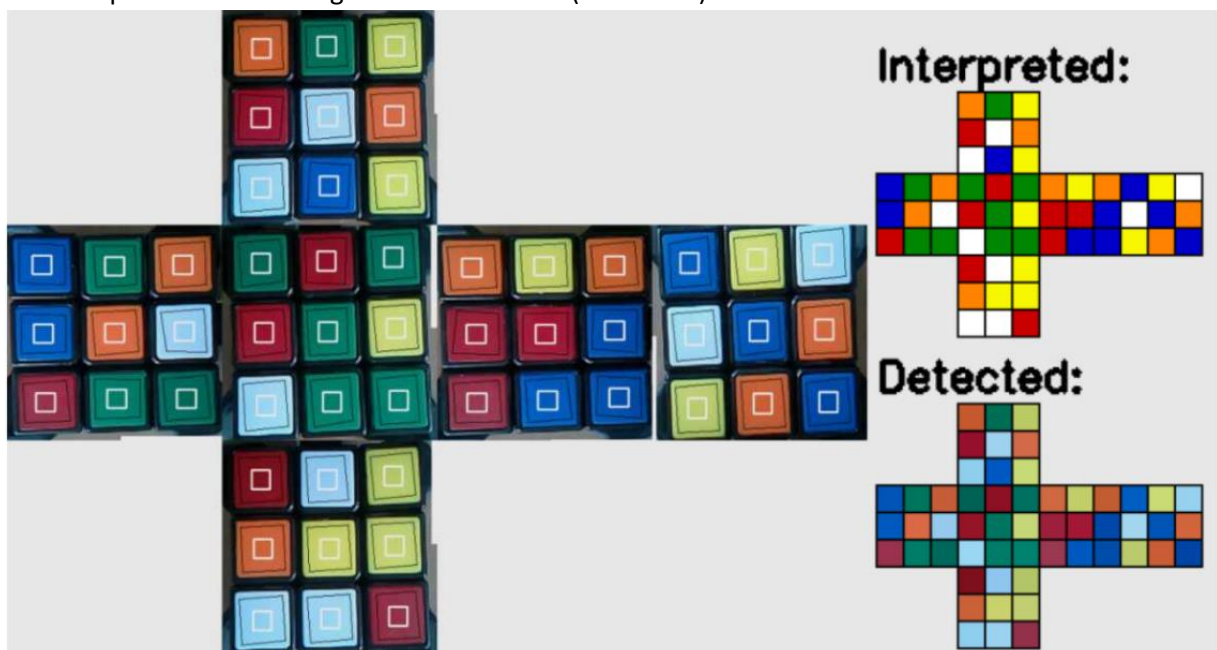
- **modified** means the file is largely based on a public version, by applying some personalization and/or corrections.
- **specific** means the file has been basically made from scratch
- **Kociemba solver:** The first time it is called, it builds many tables. This takes quite some time on a PC, and up to 6 hours on a Raspberry pi (Kociemba info); Differently, it is simply possible to generate these tables on a PC, and later copy them on the Raspberry pi (Kociemba files are about 70Mb); The needed files are those listed on the file *SD and Rpi settings_20211003.pdf* file embedded on this document.

7. The script uses a “tentative” approach, on a couple of analysis:

- a. When the image is analysed, it returns contours of facelets and many other unwanted; This happens in the function `get_facelets()`. Afterward, consecutive filters are applied to only keep contours having cube facelet's requisites. This process ends when 9 facelets, all matching the filters criteria, are retrieved from a single image (frame).
- b. When determining the cube status, according to the facelets colour; The analysis starts with a first method determining each (side and corner) facelet colour, based on the colour distance from the colours of the 6 centers. In case the cube status obtained with this first method is not coherent, then a second method is called. The second method uses the Hue value of each (non-white) facelet, by comparing it to expected (predefined) Hue ranges, adapted upon the Hue measured on the 6 centers. In case also the second method doesn't provide a coherent cube status, then an error message is returned, and relevant info logged in a text file.

8. Inner and outer contours:

- a. The outer one, in black, shows the simplified contour retrieved by the edge analysis; This analysis is used to find the 9 facelets per each cube side. The defined area is used to calculate the BGR average value, for colour interpretation according to the 1st method (BGR colour distance).
- b. The inner one, in white, depicts a smaller square area (at script \rightarrow side = 2 * edge) centred on the outer contour; This smaller area is used to calculate the HSV average colours, used for colour interpretation according to the 2nd method (Hue value).



9. To get consistent colours, while detecting the 54 cube's facelets, it is convenient to fix some PiCamera parameters, like the AWB and Exposition (and gains).

The approach follows these phases:

- a. Warm up the PiCamera: Keep the camera AWB and Exposition (and gains) in auto mode while the cube is facing the camera; Once these parameters get stable the warmup phase ends.
- b. Start retrieving the facelets on the first cube side; Once 9 facelets are retrieved the camera is set to manual by setting the latest camera parameters used to get these facelets.
- c. The camera with fixed AWB and Exposition, is used to detect the next 5 cube's faces.
- d. After the cube is solved, the camera is closed to release these manual settings.
- e. At the next cube reading, the camera will start again in auto-mode, and the above steps repeated.

10. Rotation of the cube-holder has different phases and conditions:

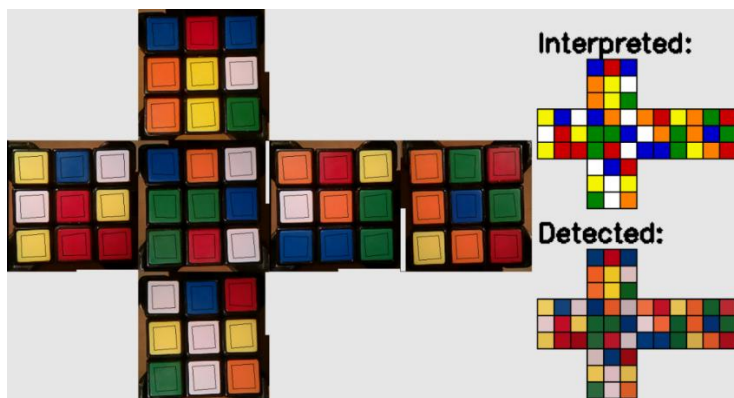
- Initial rotation speed has two levels, low and high; High speed is used when the cube is spined entirely, while Low initial speed is used when the upper-cover engages the cube, and the cube-holder is rotated (cube's bottom layer rotation).
- By linearly accelerating the first third of the requested rotation (it can be a 90-degree rotation, or a 180-degree rotation); This ensures sufficient torque to start rotating the parts
- By keeping the max speed for the second third of rotation
- By decelerating the remaining rotation part; This ensures to limit the cube inertia before the rotation stop, as differently the mid at top cube layers rotates further than the first one.
- Each rotation includes a slightly over-rotation, that ensures to get all the cube layers aligned.
- At the rotation end, the cube-holder is rotated backward by the over-rotation part; This releases majority of friction between cube-holder, cube, and upper-cover thus ensuring the upper-cover to easily open, further than enabling a new rotation starts without initial high friction.

11. When the script is running on Raspberry pi, it saves some data:

- After detecting a cube's status, it saves a png image (filename: cube_collagedate_time.png); This appears as the unfolded cube, made via a collage of images taken during the cube status detection (sketches of detected and interpreted colours are also added to the collage).

I've uploaded in YouTube a movie made with about 150 of these collected images:

<https://www.youtube.com/watch?v=BOcjhVn50s>



- After solving the cube, a string is added to a text log file with the more relevant data, for debug and statistic purposes; Data is tab separated:
 - Date and time (for reference)
 - ColorAnalysisWinner (which method has determined the "coherent" cube status)
 - RobotTime (total time: cube detection + Kociemba solver time + solving)
 - CubeStatus (BGR or HSV or BGR,HSV)
this is a 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 read by both the colour spaces are reported
 - CubeStatus (cube status according to the used notation)
 - CubeSolution (cube solution, as given by Kociemba solver)

12. Date and especially time are used by the robot:

Raspberry pi hasn't RTC, therefore when the robot isn't connected to a PC and/or internet, this info could be inaccurate.

In my experience the robot was able to complete the task without issues.

In case the robot is already working on a cube solving, while it establishes a connection to internet, then some timers could get heavily affected, for example the total solution time could reach nonsense values, like > 600 seconds.

If you want to prevent this, a RTC extension board (with related battery) could be add to the Raspberry pi.

13. Systems/packages:

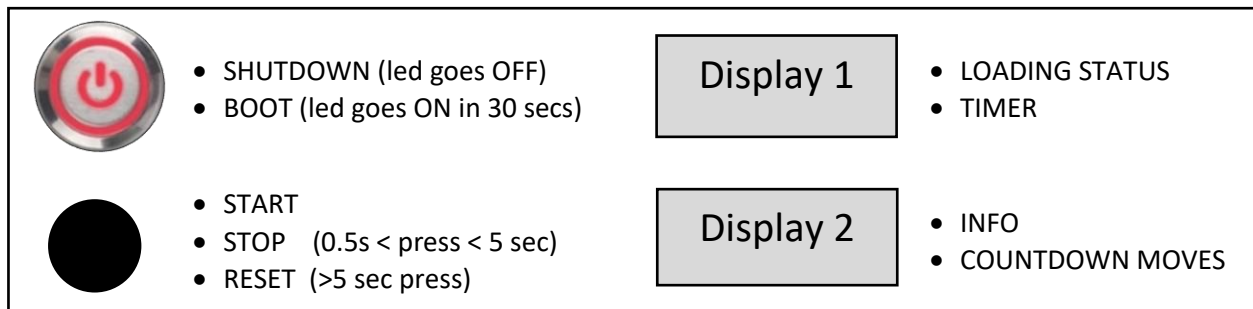
The script and robot has been developed with:

- Windows (W10) with Python version: **3.8.12** [MSC v.1916 64 bit (AMD64)] and cv2 ver: 4.5.1
- Raspberry pi 4B 2Gb (Linux-5.10.52-v7l+-armv7l-with-debian-10.10) with Python version:3.7.3 and cv2 ver: 4.1.0, and piCamera v1.3

Scripts *AF_cube_robot.py* and *AF_cube_robot_noVideo.py* has been improved on 13/11/2021, to keep working on previous systems further than:

- Windows (W10) PC with Python version: **3.9.7** | packaged by conda-forge | [MSC v.1916 64 bit (AMD64)] and cv2 ver: 4.5.1

How to operate the robot:



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).

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, by wiring or via SSH.

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

To start mode B):

- Quit mode A)
- Connect to the raspberry pi (i.e. with VNC viewer, via SSH)
- Activate the virtual environment (`workon cv`)
- From root (`/home/pi`) enter the cube folder (`cd cube/kociemba`)
- Run the python script (`python AF_cube_robot.py`)

Fine tuning:

1. PiCamera focus

This camera comes with a fixed focus, yet in case of non-satisfactory result its possible to adjust it.

I did follow this tutorial: <https://projects.raspberrypi.org/en/projects/infrared-bird-box/6>

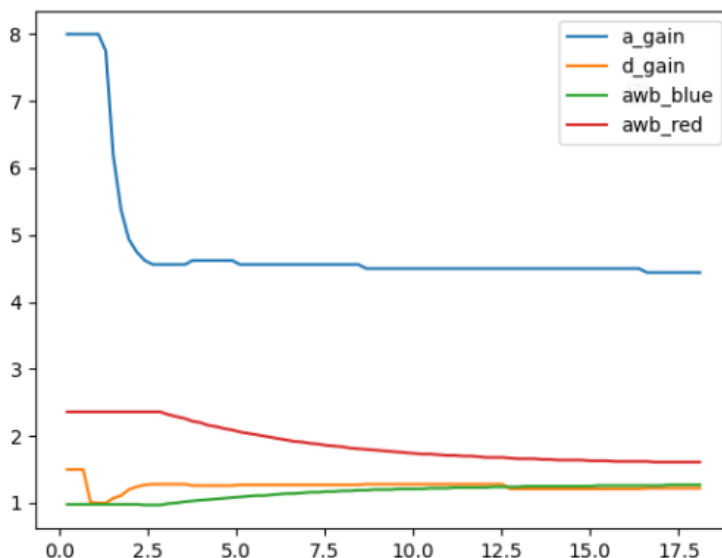
After correcting the focus, I did not glue it the lens as there was still quite some friction, preventing it from getting loose though.

2. Capturing consistent images with PiCamera:

<https://picamera.readthedocs.io/en/release-1.13/recipes1.html#capturing-consistent-images>

<https://gist.github.com/rwb27/a23808e9f4008b48de95692a38ddaa08/>

Measurements I've made on the robot: PiCamera warm-up based on gains stability



PiCamera gains (range 0 to 8) are plotted versus time (secs).

In this case the cube was placed after 2 secs from pressing robot start-button; This means the camera was initially adjusting the gains on the black cube support, and right after it had to adjust on the cube (with some white facelets): **It's clear that AWB adjustment takes quite some time to get stable**

Differently, if the cube is placed on the cube support few secs before pressing the button, then the gains are already well set.

To cover all these situations, the PiCamera warmup time has been linked to the stability of all gains: The average of last N measurements (ca 2 to 3 secs) must be within 2% variation from the average value. A timeout is anyhow set on 20 secs, as it seems more than sufficient to have the camera perfectly stable.

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 to tune my system.

6. Cube's facelet and light reflection:

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 system quite unsensitive to the light situations, in which the robot operates.

The robot also works with the cube having glossy stickers, if the light is rather diffused and not too intense.

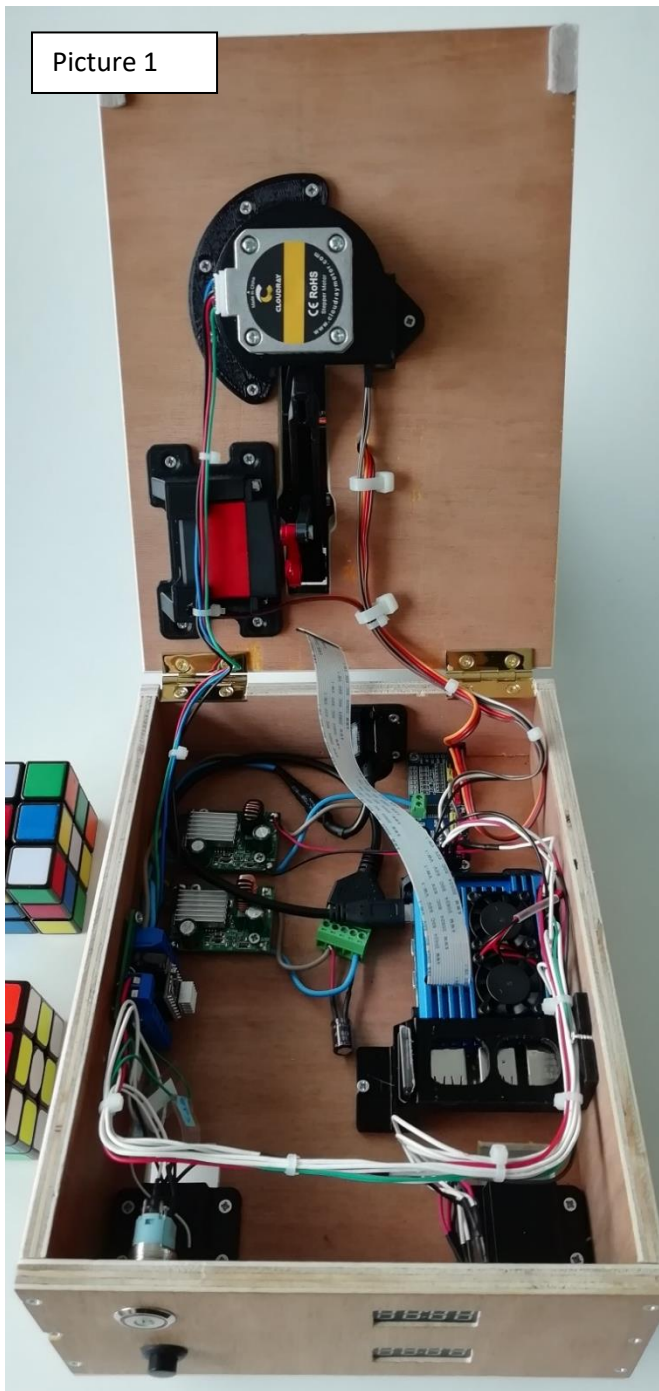
Cube with in-moulded coloured facelet, that I've made matt with sandpaper (grit 1000)

Cube with glossy stickers

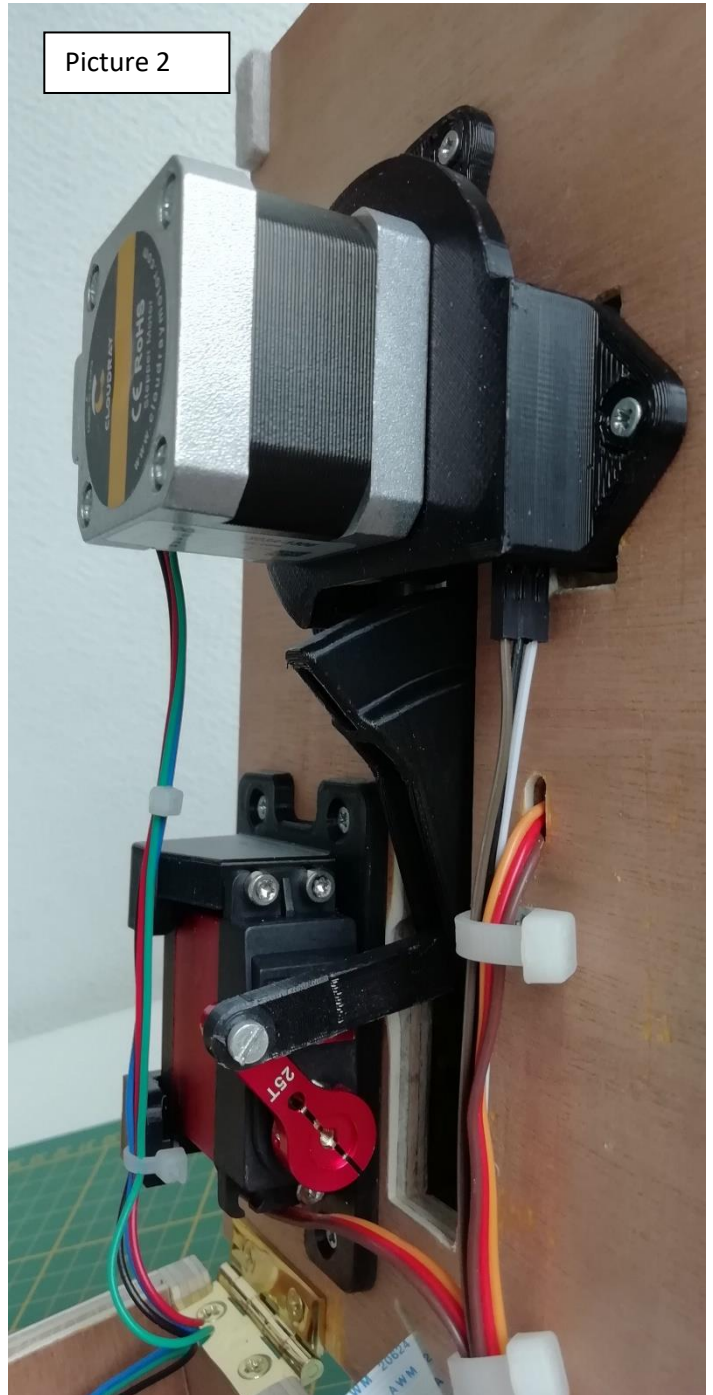


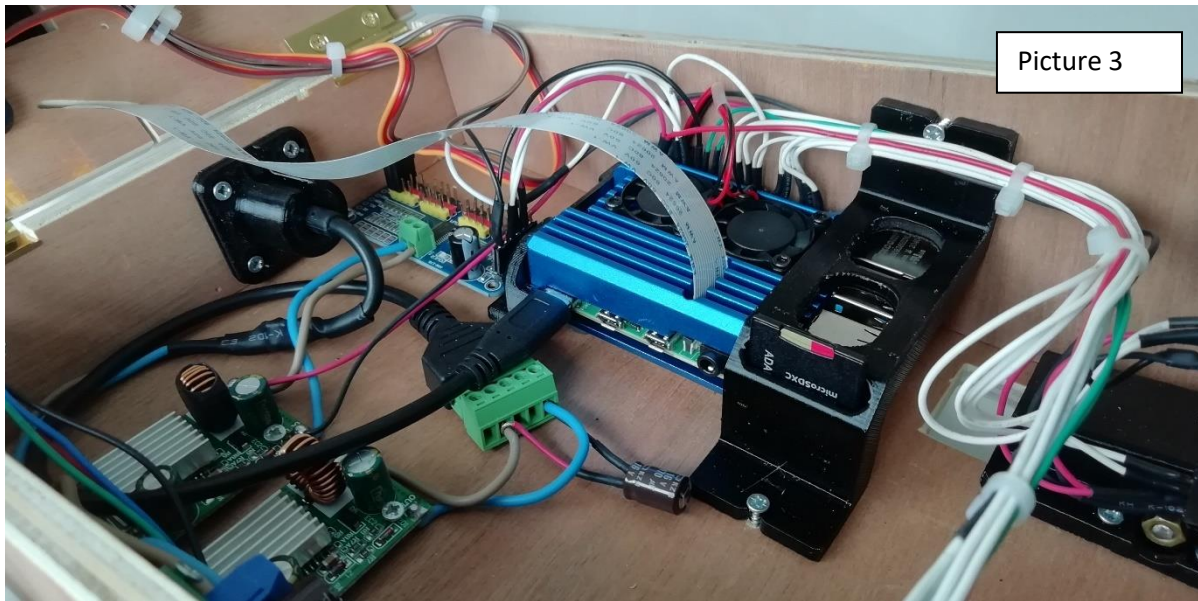
Collection of robot's pictures:

Picture 1



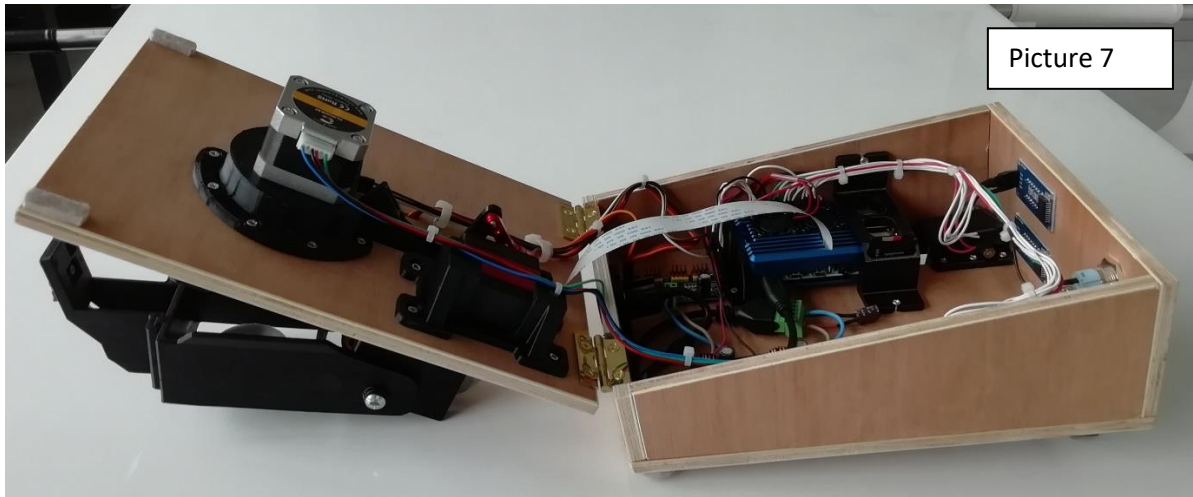
Picture 2



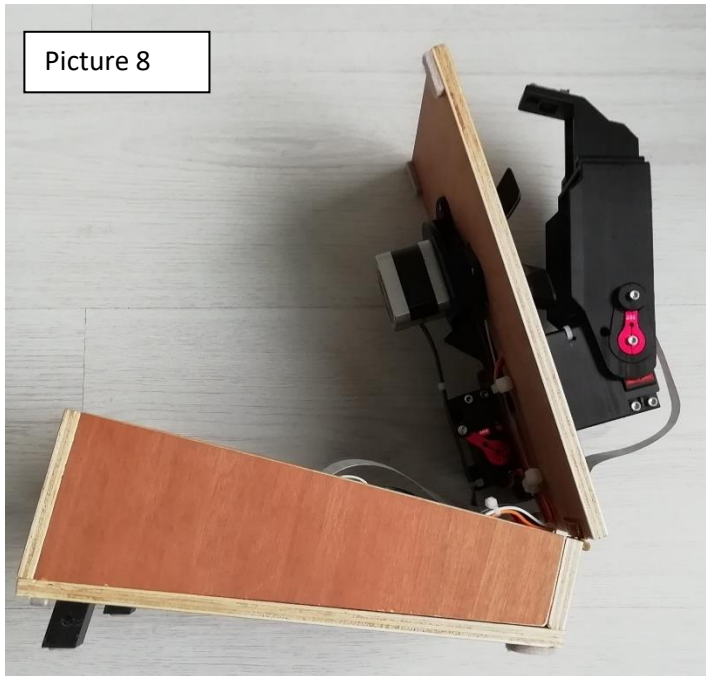




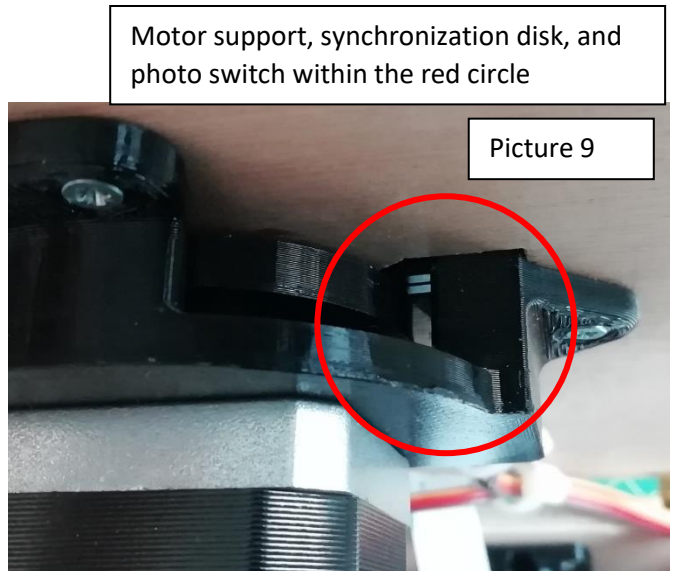
Picture 6



Picture 7

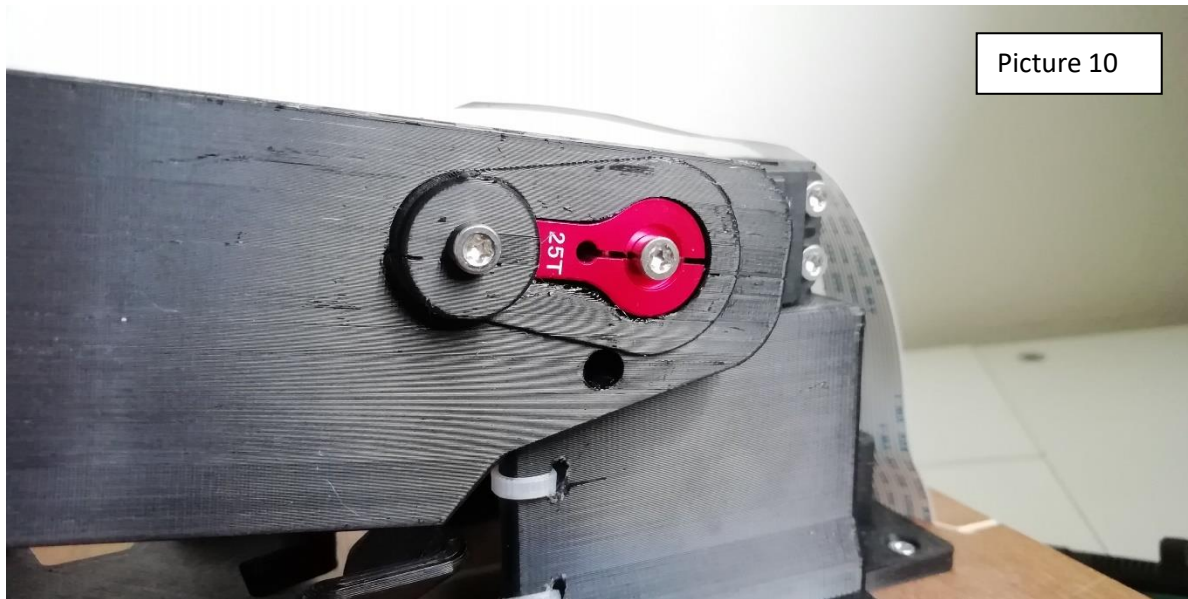


Picture 8

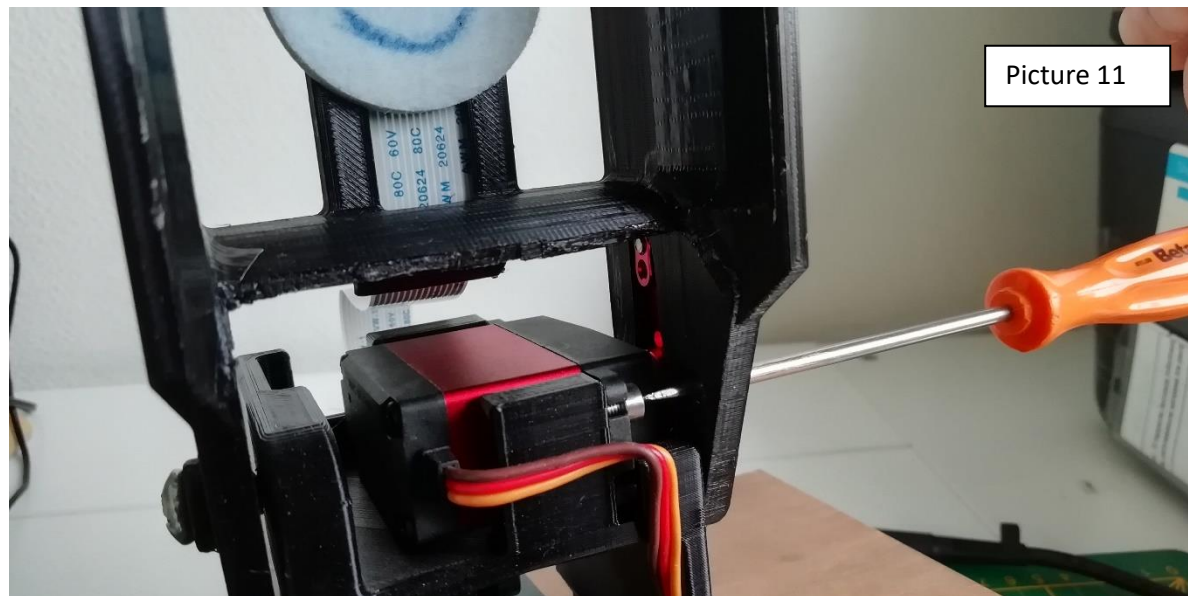


Motor support, synchronization disk, and photo switch within the red circle

Picture 9



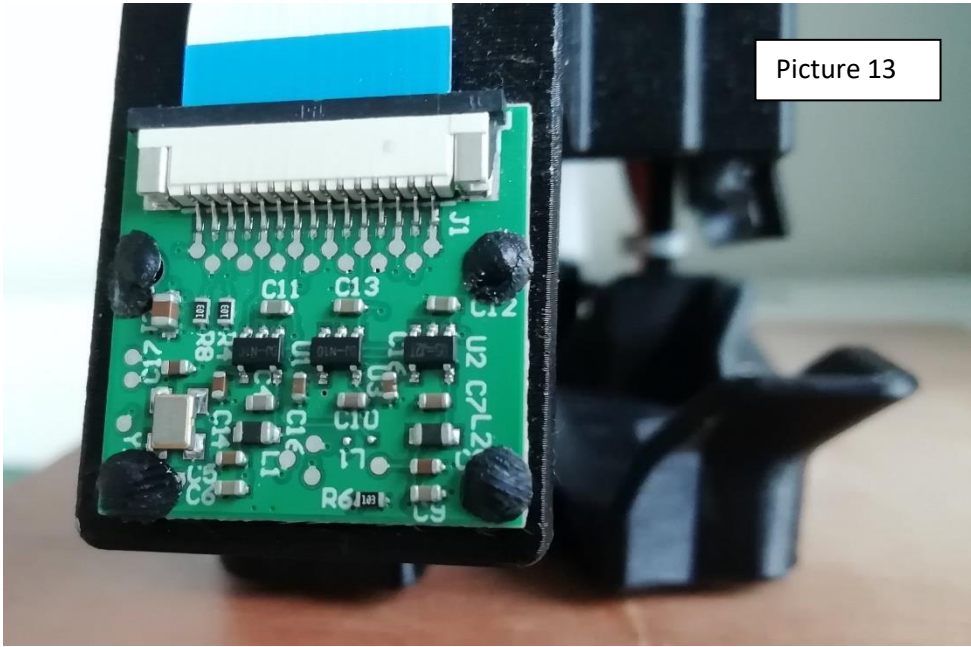
Picture 10



Picture 11



Picture 12



Picture 13

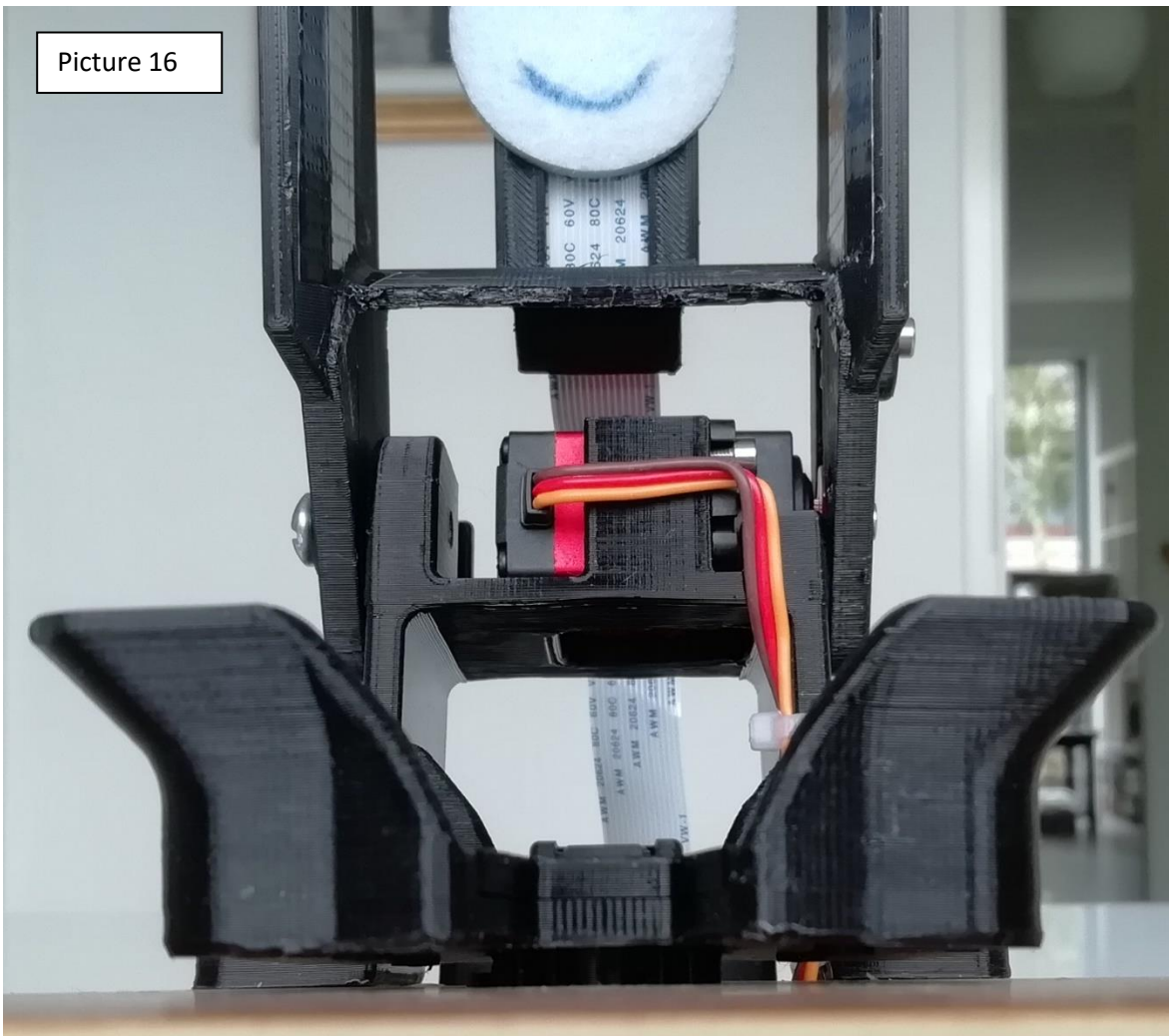


Picture 14

Picture 15



Picture 16



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/ytzjzy4m2e/build-a-simple-raspberry-pi-led-power-status-indicator>

4. Edge detection:

<https://medium.com/swlh/how-i-made-a-rubiks-cube-color-extractor-in-c-551ccea80f0>

<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 CIE Lab color space conversion: <https://gist.github.com/manojpandey/f5ece715132c572c80421febeba666ae>

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-colors-the-right-way>

10. Infinite timer:

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

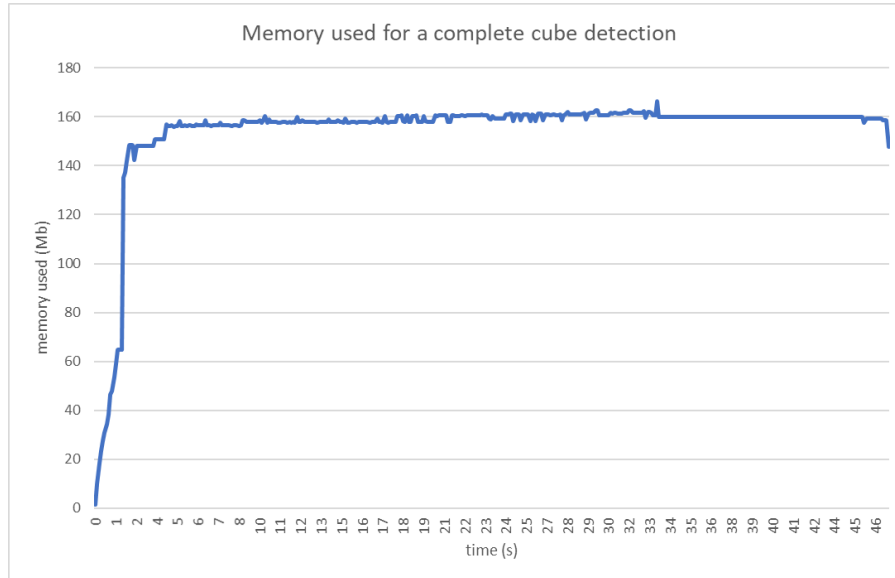
Memory usage:

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

To answer that question, I've verified the memory usage on the windows PC and on the Raspberry pi at the robot.

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

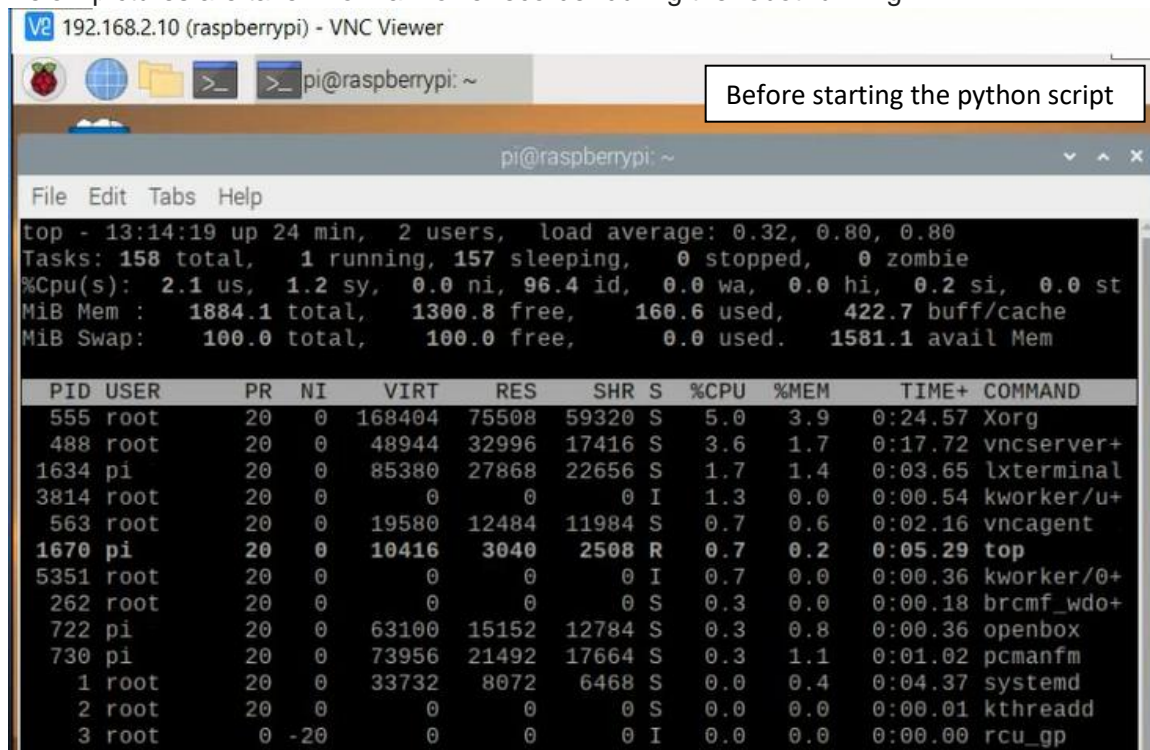
By plotting the values from the generated text file, the used memory is up to 170Mb



At Raspberry pi, checked with `top`.

I've a complete installation of Raspberry Pi OS, that 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:



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

192.168.2.10 (raspberrypi) - VNC Viewer

pi@raspberrypi: ~

File Edit Tabs Help

```
top - 13:15:31 up 25 min, 2 users, load average: 0.27, 0.69, 0.76
Tasks: 162 total, 1 running, 161 sleeping, 0 stopped, 0 zombie
%Cpu(s): 2.1 us, 1.3 sy, 0.0 ni, 96.6 id, 0.0 wa, 0.0 hi, 0.0 si, 0.0 st
MiB Mem : 1884.1 total, 1141.0 free, 291.6 used, 451.4 buff/cache
MiB Swap: 100.0 total, 100.0 free, 0.0 used, 1433.3 avail Mem
```

PID	USER	PR	NI	VIRT	RES	SHR	S	%CPU	%MEM	TIME+	COMMAND
5690	pi	20	0	262680	82984	58832	S	4.3	4.3	0:03.28	python
488	root	20	0	48944	32996	17416	S	4.0	1.7	0:21.62	vncserver-x11-c
555	root	20	0	174288	84036	67620	S	3.3	4.4	0:30.20	Xorg
5583	pi	20	0	436756	161224	58080	S	2.3	8.4	0:18.73	python
1634	pi	20	0	85764	28016	22656	S	0.7	1.5	0:04.17	lxterminal
1670	pi	20	0	10416	3040	2508	R	0.7	0.2	0:05.76	top
210	root	-2	0	0	0	0	S	0.3	0.0	0:00.48	v3d_render
4974	root	20	0	0	0	0	I	0.3	0.0	0:00.74	kworker/u8:3-brcmf_wq/mmc1:0001+
1	root	20	0	33732	8072	6468	S	0.0	0.4	0:04.37	systemd
2	root	20	0	0	0	0	S	0.0	0.0	0:00.01	kthreadd
3	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_gp
4	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	rcu_par_gp
6	root	0	-20	0	0	0	I	0.0	0.0	0:00.28	kworker/0:0H-mmc_com
8	root	0	-20	0	0	0	I	0.0	0.0	0:00.00	mm_percpu_wq
9	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcu_tasks_rude_
10	root	20	0	0	0	0	S	0.0	0.0	0:00.00	rcu_tasks_trace
11	root	20	0	0	0	0	S	0.0	0.0	0:00.27	ksoftirqd/0
12	root	20	0	0	0	0	I	0.0	0.0	0:00.40	rcu_sched
13	root	rt	0	0	0	0	S	0.0	0.0	0:00.00	migration/0
14	root	20	0	0	0	0	S	0.0	0.0	0:00.00	cpuhp/0
15	root	20	0	0	0	0	S	0.0	0.0	0:00.00	cpuhp/1
16	root	rt	0	0	0	0	S	0.0	0.0	0:00.00	migration/1
17	root	20	0	0	0	0	S	0.0	0.0	0:00.12	ksoftirqd/1
20	root	20	0	0	0	0	S	0.0	0.0	0:00.00	cpuhp/2
21	root	rt	0	0	0	0	S	0.0	0.0	0:00.00	migration/2

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

Conclusions:

It seems possible to run the robot script on a Raspberry pi with 512Mb of RAM.

As I don't have the right knowledge, I'm not sure about all the possible issues in case of limited RAM quantity, for instance already during the Package installation; For this reason, I've decided to embed the above pictures, as reference for people who wants to check whether a lighter board could be sufficient.

It might be worth to read <https://qengineering.eu/install-opencv-lite-on-raspberry-pi.html>, in which the OpenCV Lite version is suggested in combination with Raspberry pi having limited RAM.

Document revisions:

Rev.1 13/11/2021

- 1) **Python scripts:** On 11/11/2021 luisl33 has spotted an error on `AF_cube_robot.py`; Verified this error raises with higher python version due to wrong data type usage (an easy to fix problem).

Few more improvements have been introduced on `AF_cube_robot.py` and `AF_cube_robot_noVideo.py` together with the fix for the spotted error

1. When `.get('contour_example')` is used with array pointers (i.e. `.get('cont_ordered')[0][0]`), it returns a single value; On the original script this value was left in its `np.intc` data type, now it's converted to `int` data type; farther than preventing errors on more recent python versions, this is the correct approach.
2. With newer python version, the program quitting wasn't anymore clean; Improved this by preventing camera reading while quitting the script. An additional Boolean variable is set during quitting.
3. At the start up the python and CV2 versions are printed to the interpreted terminal.
4. Imported/used *platform* and *subprocess* packages, for a better system detection and to clear the terminal in between sequential cubes reading.
5. Added and explicit print at the script quitting when the laptop is used.
6. Added a Boolean variable *fixWindPos*, largely used along the scripts to force the CV2 windows at screen top left position; This way it's much easier to see the graphical part when a smartphone is used instead of a larger screen.

- 2) **Wiring and mechanical part:** Added a hdmi socket to the rear panel of the robot, connected to the hdmi0 port at raspberry pi 4B

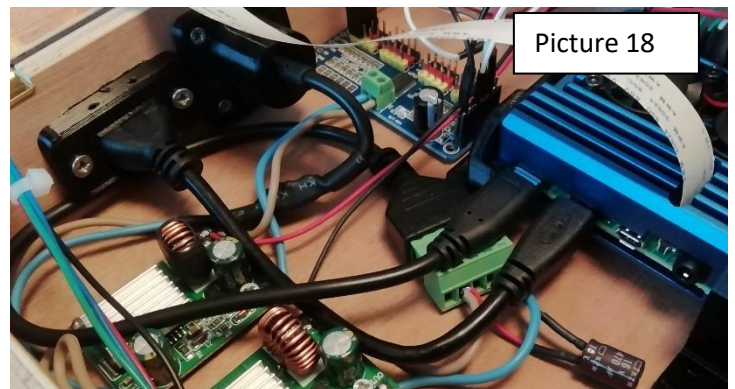
In this way it is easy to connect an external screen, in case of robot demonstration out of home: The smartphone can be used as Access Point, to interact with the robot, and an external screen to show what the camera sees. For this it will be convenient to add a second network (phone AP) on `wpa_supplicant.conf` file, by also setting the priority.

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

- 3) **Information:**

1. Added the memory usage related chapter
2. Reported the python / OpenCV versions used and tested; Info at the end of "Python script info" chapter.

1) Python scripts:

Files *AF_cube_robot.py* and *AF_cube_robot_noVideo.py*

Indentation error on two rows (1289,1296), and wrong word on one of the command; These two errors were preventing the HSV color detection approach to be effective, if the boolean "debug" (on main function) was set False.

Changed from:

```
1286         if debug:
1287             print(f'\nCube_status_detected_colors: {cube_status_detected}')
1288             print(f'\nCube_status_conventional_colors: {cube_status_kociemba}\n')
1289             breake
1290
1291     elif cube_color_sequence == std_cube_color:
1292         cube_status_kociemba=cube_status_detected
1293         if debug:
1294             print('\nCube colors and orientation according to kociemba order')
1295             print(f'\nCube_status_detected_colors: {cube_status_detected}\n')
1296             break
1297
```

Changed to:

```
1286         if debug:
1287             print(f'\nCube_status_detected_colors: {cube_status_detected}')
1288             print(f'\nCube_status_conventional_colors: {cube_status_kociemba}\n')
1289             break # wrong identionation until 27 April 2022, additionally to wrong word (breake)
1290
1291     elif cube_color_sequence == std_cube_color:
1292         cube_status_kociemba=cube_status_detected
1293         if debug:
1294             print('\nCube colors and orientation according to kociemba order')
1295             print(f'\nCube_status_detected_colors: {cube_status_detected}\n')
1296             break # wrong identionation until 27 April 2022
1297
```

2) Wiring: Added few notes on “Modules and connections” to make clearer.