

## Inhalt

1	Allgemeines .....	3
2	Connectivität EV3 herstellen .....	4
2.1	USB .....	4
2.2	WiFi.....	4
2.3	Bluetooth.....	4
3	Python module installations.....	5
4	Konstruktiver Aufbau – Cube Solver .....	6
4.1	V1.....	7
4.2	V2.....	8
4.2.1	Gear Ratios – Käfig und Wiege .....	9
4.2.2	Konstruktion des Lift – elevator .....	10
4.2.3	Konstruktion der Wiege .....	12
4.3	Konstruktion des Käfig.....	14
4.4	Konstruktion des Ausrichtungsrahmen .....	14
5	RPI – Setup.....	15
5.1	Image herstellen.....	15
5.2	Netzwerk Verbindung & IP-Adresse .....	16
5.3	Remote Access .....	17
5.4	Dateiaustausch mit dem RPI .....	19
5.5	Kamera laufend bekommen .....	20
5.5.1	Installationsschritte .....	21
5.6	Kamera Ausrichtung und Fokus Punkt einstellen.....	22
5.6.1	Digital Still Images (raspistill).....	23
6	Lego EV3 USB Link .....	28
7	Python am RPI .....	29
7.1	EV3_DC installieren .....	30
7.2	OpenCV Installieren.....	31
7.3	PIL – Python Image Library .....	31
7.5	Kamera in Python (picamera).....	32
7.5.1	Capture to File .....	32
7.5.2	Capture to numpy .....	32
7.6	Libcamera .....	33
7.7	Libcamera in Python.....	35
7.7.1	CMD and Arguments .....	35
7.7.2	Subprocess Call.....	35
8	C / C++ am RPI .....	36
9	Color Scanning.....	37

9.1	Maschinen Bewegung .....	37
9.2	Farb-Positionsbestimmung im Bild.....	38
9.2.1	Reihenfolge.....	39
9.3	Farbdecodierung .....	40
9.3.1	Auto AWB .....	44
9.3.2	Manueller AWB (R =1, B = 1) .....	44
9.3.3	Manueller AWB (R=1, B=2).....	44
10	Rubiks Cube Facemap.....	45
10.1	Remapping scan-pics > rubikscube-facemap .....	46

# 1 Allgemeines

Es wird aus technic lego ein cube-solver konstruiert, basierend auf der Vorlage des sog. Multicuber-relay von David Gilday:

[https://www.youtube.com/watch?v=kWrJdkXp\\_n4](https://www.youtube.com/watch?v=kWrJdkXp_n4)

Es gibt verschiedenste Lego Kreationen von David, unter anderem der Cubestormer (welt-rekordhalter für maschinelles Lösen des Rubik Cube)

Es gibt leider keinen Bauplan daher wird er laut der Videovorlage rekonstruiert.

Der Lego EV3 wird verwendet als Motorcontroller, anstelle des NXT

Lego Direct-Commands zum steuern der Motoren von einem Rechner, keine DLO (device logic) am EV3 Brick.

Die Direct commands sind spezifiziert in der offiziellen API-Beschreibung von lego:

[https://www.lego.com/cdn/cs/set/assets/blt6879b00ae6951482/LEGO\\_MINDSTORMS\\_EV3\\_Communication\\_Developer\\_Kit.pdf](https://www.lego.com/cdn/cs/set/assets/blt6879b00ae6951482/LEGO_MINDSTORMS_EV3_Communication_Developer_Kit.pdf)

LEGO\_MINDSTORMS\_EV3\_Communication\_Developer\_Kit.pdf

Es braucht also keine Spezielle Firmware für den EV3-Brick oder das ev3dev package.

Es gibt die ev3\_dc library für python - von Christoph Gaukel, welche die Funktionen bereits abstrahiert anbietet. Die Lib ist gut dokumentiert.

Documentation: <https://ev3-dc.readthedocs.io/en/latest/>

Git: <https://github.com/ChristophGaukel/ev3-python3>

Das Scannen des Cube erfolgt durch eine handelsübliche Webcam. Einziges Kriterium – es muss eine UVC Webcam sein (usb video class).

## 2 Connectivität EV3 herstellen

### 2.1 USB

Als Steuerung wird USB bevorzugt (wegen der Latenz), dafür muss libusb ans laufen gebracht werden. Das ist etwas tricky, da man dafür einen eigenen Driver erstellen muss.

Die Prozedur ist beschrieben in der docu der ev3\_dc library:

#### *Windows 10*

Download libusb1.0 from the [libusb project](#) (I additionally had to install program [7 zip](#)).

Copy file *libusb-1.0.dll* from *libusb-1.0.xy.7z\MinGW64\dll\* into directory *C:\Windows\System32*.

**Anmerkung:** *das war nicht erforderlich, sollte die Fehlermeldung kommen, dass das „backend“ fehlt dann liegt es daran dass diese DLL nicht gefunden wird. Man kann sie aber in seinem Projectordner platzieren und dass backend für die libusb manuell neu setzen. Im aktuellen Versuch war es nur erforderlich die folgende Instruction umzusetzen.*

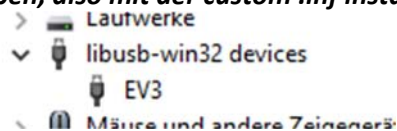
Follow this [instruction](#) and replace *Xin-Mo Programmer* by *EV3* (when I did it, I clicked the *Install Now* button in the Inf-Wizard and it was successfully installing).

<https://www.smallcab.net/download/programme/xm-07/how-to-install-libusb-driver.pdf>

„how-to-install-libusb-driver.pdf“

#### **Anmerkung:**

**In Python werden dann auch nur diese usb-device angezeigt welche den entsprechenden driver haben, also mit der custom .inf installiert wurden**



### 2.2 WiFi

WiFi hat das Problem, dass im EV3 ein USB-WiFi Adapter eingesetzt werden muss. Aufgrund des Alters der EV3-Firmware werden aber nur entsprechend alte USB-Wifi Adapter unterstützt.

Getestet funktionierend: Edimax EW-7811UN

<https://www.amazon.de/EDIMAX-EW-7811UN-Wireless-Adapter-IEEE802-11b/dp/B003MTTJOY/>

ist jedoch nicht mehr lieferbar.

Die nachfolger-Version davon: **Edimax EW-7811Un V2**

Funktioniert nicht mehr mit dem EV3. Man muss also versuchen einen entsprechend alten Stick zu erhalten

### 2.3 Bluetooth

Bluetooth am EV3 funktioniert zwar ohne einen externen Adapter, hier ist aber das Problem an der Windows-Seite und die Verwendung von USB aus Python heraus. Die gängige PyblueZ Library funktioniert nicht mehr mit windows 10, dafür hat Python aber einen nativen Bluetooth Support erhalten (ab python 3.9.6) – auf Low-Level Socket Ebene (RFCOMM).

### 3 Python module installations

Pip install ev3\_dc

Pip install pyusb

## 4 Konstruktiver Aufbau – Cube Solver

Es wurden 2 Versionen aufgebaut – bezeichnet als V1 und V2

Die Maschine besteht aus 3 bewegten Kernbauteilen, jeweils von einem mindstorms L Motor angetrieben:

- Käfig (cage)
- Wiege (cradle)
- Lift (elevator)

Zwischen Käfig und Wiege gibt es einen Ausrichtungsrahmen (alignment layer).

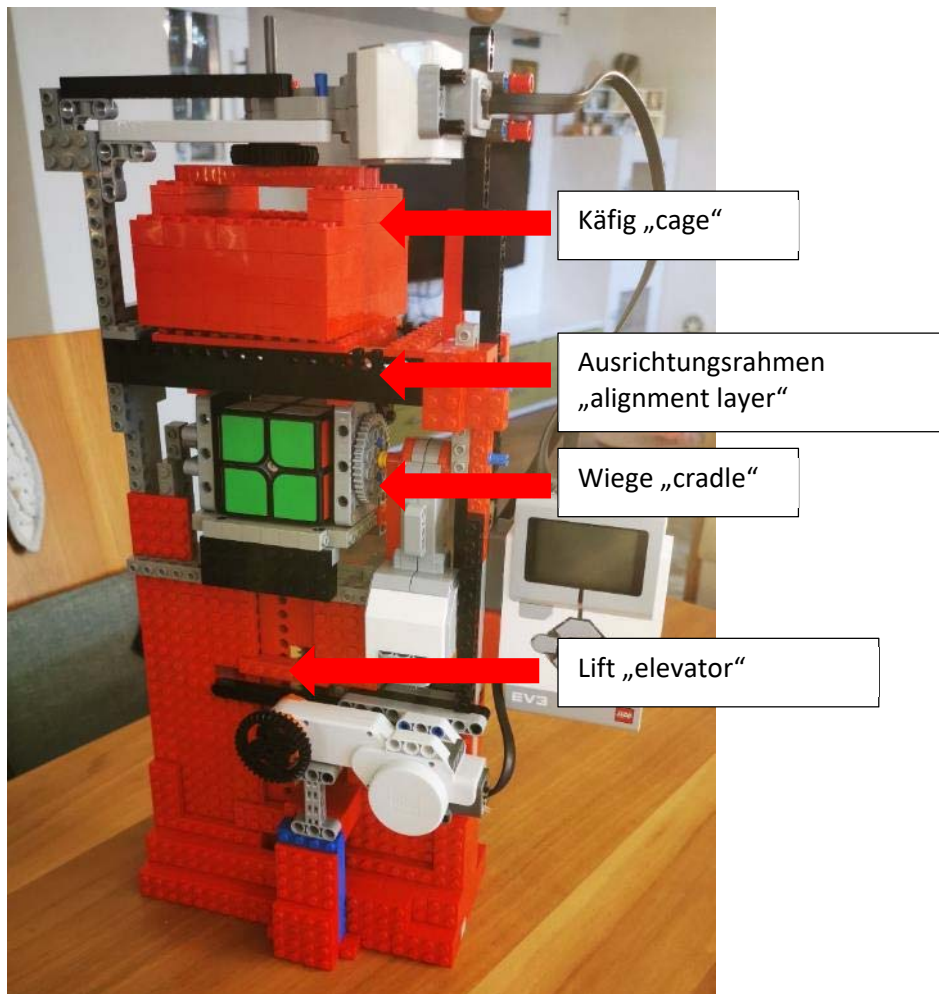
Alle Bauteile werden von einer umlaufenden Rahmenkonstruktion gehalten.

Die Motoren sind so eingebaut, dass sie durch ein Zahnrad manuell gedreht werden können. Das ist wichtig da es keine Auto-Kalibration gibt. D.h. man muss die Mechanik zu Beginn in die Ruhelage bringen, bevor man das Programm startet:

- Lift am unteren Anschlag
- Wiege in der 0° Position
- Käfig exakt über Ausrichtungsrahmen

#### 4.1 V1

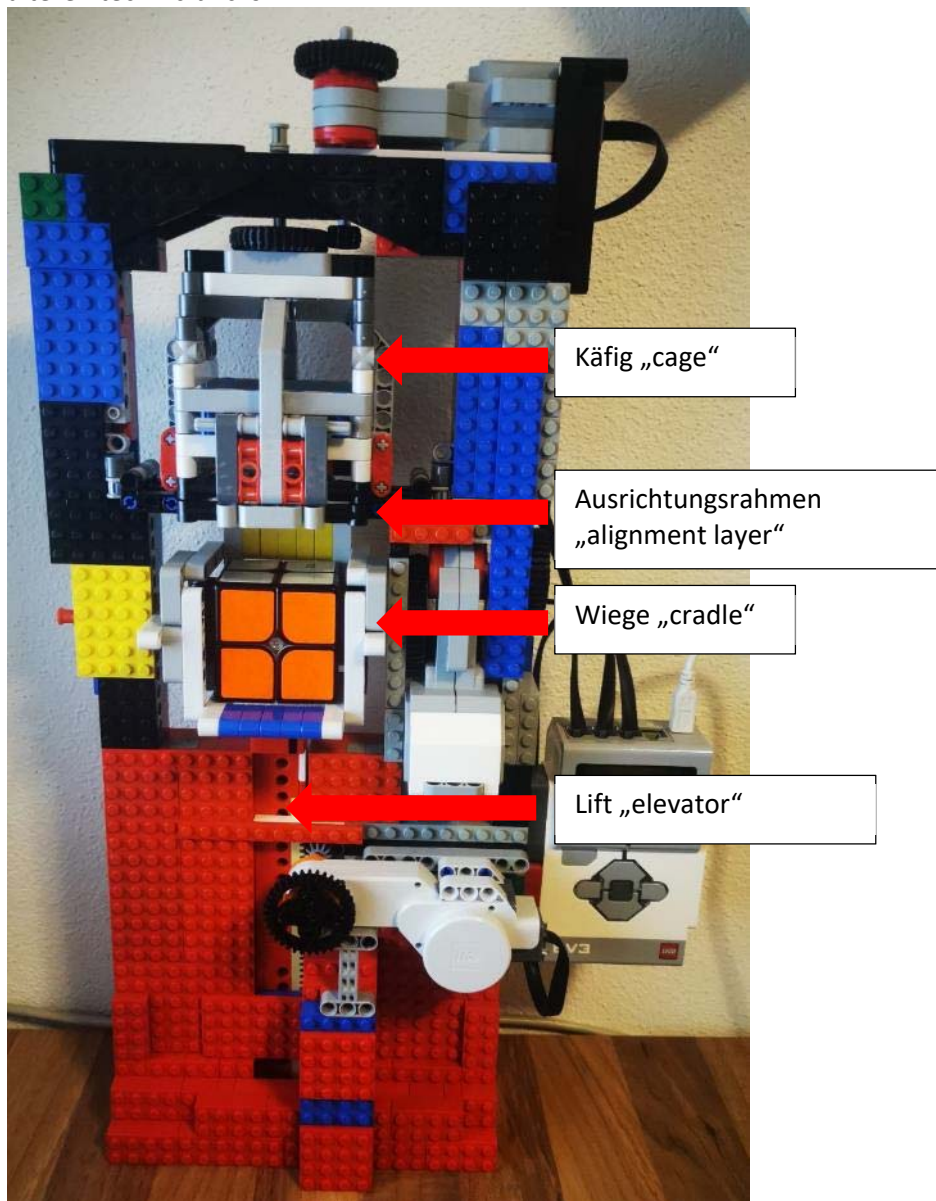
Im Bild ist die Maschine V2 zu sehen. Es sind die Kernbauteile erkennbar. Die Maschine ist hauptsächlich aus bricks konstruiert inkl. der bewegten Teile, der Rahmen fast ausschließlich mit herkömmlichen bricks bzw. älteren technic bricks.



Die V1 Konstruktion wurde verworfen, es gab einige Probleme an der Konstruktion die sehr häufig zum steckenbleiben des Cube führten. Der Lift sowie große Teile des Rahmens konnten für die V2 wiederverwendet werden. Wiege, Käfig sowie Ausrichtungsrahmen mussten neu aufgebaut werden sowie der gesamte obere Teil des Rahmens.

## 4.2 V2

Im Bild ist die Maschine V2 zu sehen. Es sind die Kernbauteile erkennbar. Die Bewegten Teile sind mit den neueren beams konstruiert, der Rahmen fast ausschließlich mit herkömmlichen bricks bzw. älteren technic bricks.





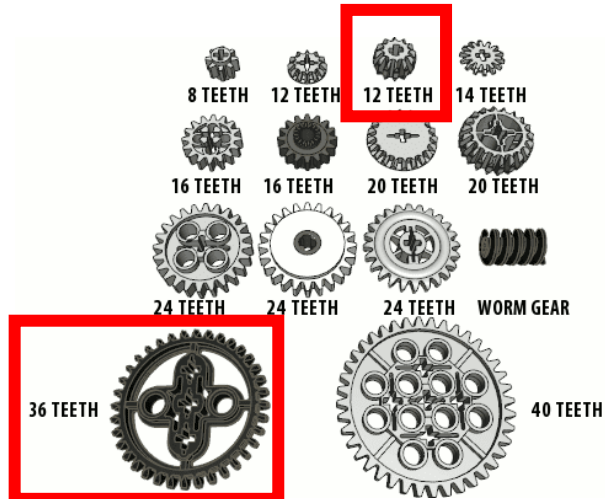
#### 4.2.1 Gear Ratios – Käfig und Wiege

Verwendet werden die dickeren Zahnräder mit 12 und 36 Zähnen für Käfig und Wiege.

Das Getriebe ist dabei als Untersetzung konfiguriert mit  $36/12 = 3:1$ .

Eine Rotation um  $90^\circ$  der Mechanik entspricht daher  $90^\circ \cdot 3/1 = 270^\circ$  auf der Motorwelle.

Für den Lift kommt auch das dicke Zahnrad mit 12 Zähnen zum Einsatz, es wird jedoch eine Zahnstange (10 Zähne) damit geschoben

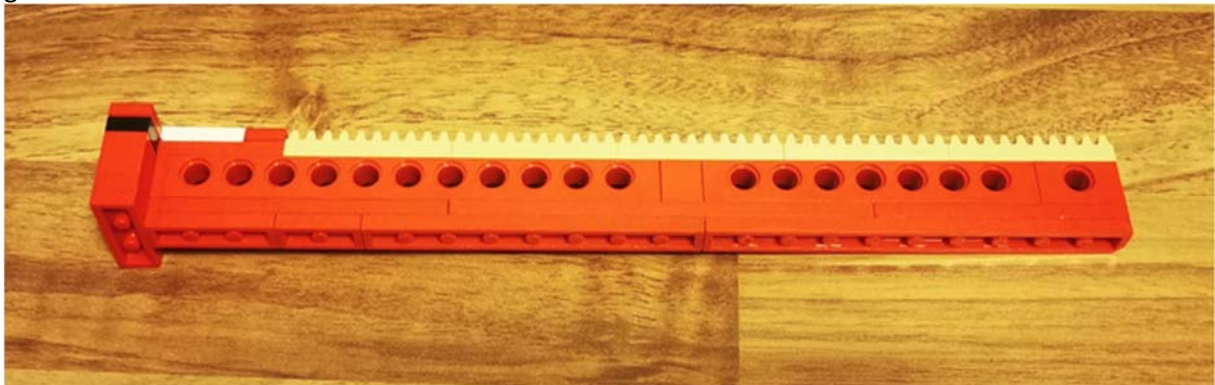


DRIVER GEAR											
FOLLOWER GEAR											
		1:1	1:1.5	1:2	1:2.5	1:3	1:3.5	1:4.5	1:5	1:7	8:1
		1.5:1	1:1	1:1.33	1:1.67	1:2	1:2.33	1:3	1:3.33	1:4.76	12:1
		2:1	1.33:1	1:1	1:1.25	1:1.5	1:1.75	1:2.25	1:2.5	1:3.5	16:1
		2.5:1	1.67:1	1.25:1	1:1	1:1.2	1:1.4	1:1.8	1:2	1:2.8	20:1
		3:1	2:1	1.5:1	1:2:1	1:1	1:1.17	1:1.5	1:1.67	1:2.33	24:1
		3.5:1	2.33:1	1.75:1	1.4:1	1:1.17	1:1	1:1.29	1:1.43	1:2	28:1
		4.5:1	3:1	2.25:1	1.8:1	1.5:1	1.29:1	1:1	1:1.11	1:1.56	36:1
		5:1	3.33:1	2.5:1	2:1	1.67:1	1.43:1	1:1.11	1:1	1:1.4	40:1
		7:1	4.67:1	3.5:1	2.8:1	2.33:1	2:1	1.56:1	1.4:1	1:1	56:1

#### 4.2.2 Konstruktion des Lift – elevator

Mit dem Lift wird der 2x2x2 cube in die Wiege oder in den Käfig gehoben.

Es ist eine sehr einfache Konstruktion mit einer sog Zahnstange, welche innerhalb eines Schachtes geführt wird.



Am unteren Ende abgeflacht da es gelegentlich zum Verkanten kommt beim runterfahren

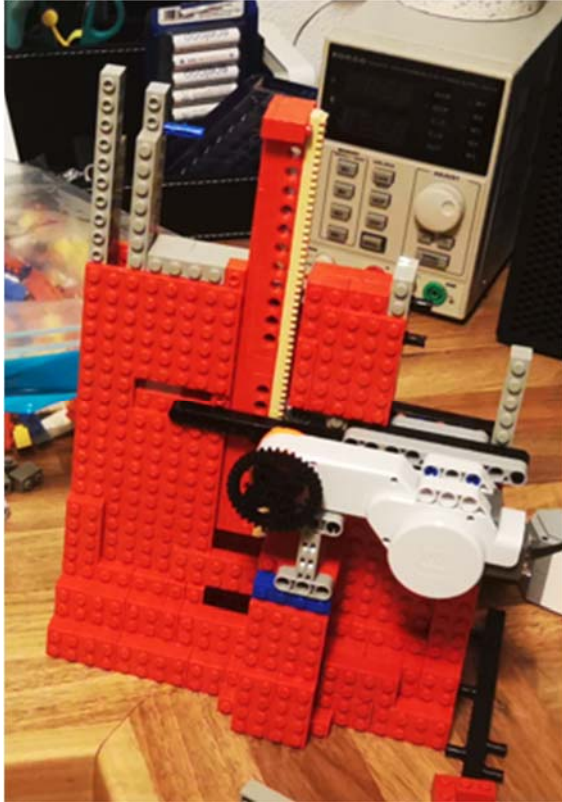


Die Führung ist nicht exakt, dadurch kann die Stange leicht kippen – hauptsächlich links/rechts.



Als Antrieb kommt ein L-Motor zum Einsatz mit dem dicken Zahnrad mit 12 Zähnen (wie bei den anderen). Da es kein bekanntes Übersetzungsverhältnis zur Zahnstange gibt wird das einfach durch probieren ausgemessen wo die wichtigen Positionen liegen.

Überschlag: Die Zahnstange hat 10 Zähne und das Zahnrad hat 12 Zähne. Es sind 5 Zahnstangen hintereinander eingebaut – in Summe also 50 Zähne.  $50/12 = 4.166$  Umdrehungen des Motors für den **vollen Hub** also ungefähr **1500° auf der Motorwelle**.



#### 4.2.3 Konstruktion der Wiege

In der Wiege kann der 2x2x2 Cube um 90° gedreht werden.

Die Drehrichtung ist dabei F > U (Front to Up) oder U > F

Die Wiege ist so konstruiert dass der Lift durchfahren kann egal ob die Wiege 0°, 90° oder dazwischen steht.

Es gibt keine mechanischen Anschlag, die Wiege kann daher frei gedreht werden solange der Lift eingefahren ist. Ein eingelegter cube fällt jedoch heraus bei >90° oder <0°

Steht die Wiege zwischen 0 und 90° so liegt der Cube darin wie in einer Wiege, woher auch der Name kommt.

##### V1-Konstruktion

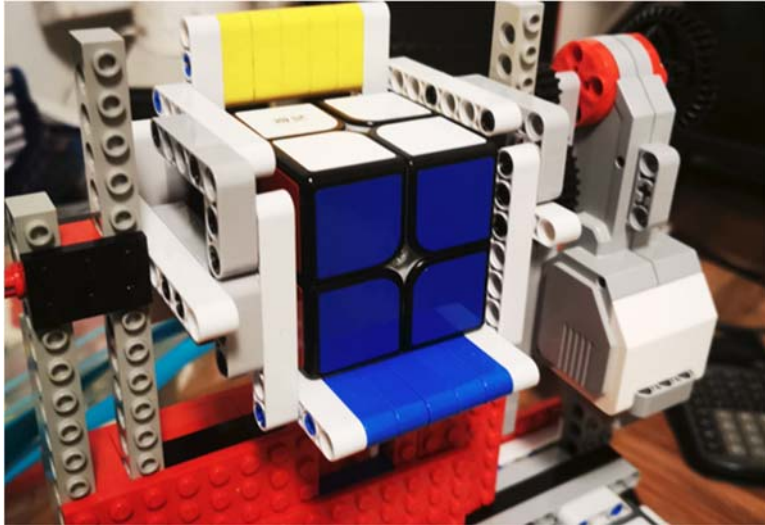
Die erste Version der Wiege wurde gemischt aus bricks und beams gebaut. Dies hatte jedoch den Nachteil dass der Drehpunkt nicht zentriert war, wodurch es häufig zur Fehlfunktion kam.





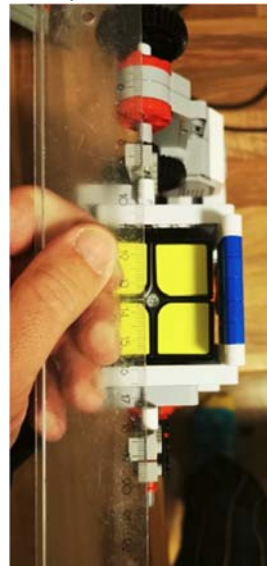
## V2 Konstruktion

Die Wiege wurde hier ausschließlich mit beams gebaut und der Drehpunkt für beide Endlagen kontrolliert. Im Bild erkennbar die 0° Lage der Wiege (blaue beams sind unten)

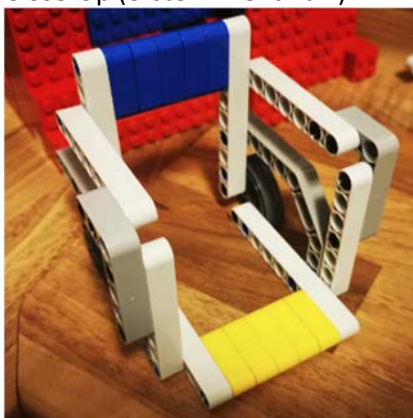


Drehpunkt bei 0°

Drehpunkt bei 90°



Close-Up (erster V2 entwurf)



#### 4.3 Konstruktion des Käfig

#### 4.4 Konstruktion des Ausrichtungsrahmen

## 5 RPI – Setup

Verwendet wird ein RPI3 Model B in Kombination V1.2 mit der NoIR Camera V2.1.

Als Power Supply kommt ein Lab-Supply mit zumindest 1.5A bei 5.1V DC zum Einsatz.

Der RPI ist mit einem Aux. DC-Supply Jack (Rundstecker) bestückt.

### 5.1 Image herstellen

Mit dem Raspberry Pi Imager Tool wird das PI OS-Lite (32 Bit) auf einer SD-Card eingerichtet (min 4GB Karte)



Auf der Boot partition der SD-Card wird ein file erstellt mit namen „ssh“ – dies aktiviert den SSH Client des RPI per default.

Die Karte wird im PI eingesetzt, damit man sie leichter rauskriegt kann man einen Streifen Klebeband befestigen

## 5.2 Netzwerk Verbindung & IP-Adresse

Ein Ethernet Kabel anschließen, der RPI holt sich seine IP Adresse via DHCP.

Die IP-Adresse kann auf verschiedene Varianten ausgelesen werden, dazu gibt es genug guides:

<https://bitreporter.de/raspberrypi/raspberry-pi-ip-adresse-herausfinden/>

Beispiel: Ping auf „raspberrypi.local“

Vergibt der DHCP Server eine IPV6 Adresse, damit kann man zwar arbeiten aber sie ist halt recht lang und schwer zu merken.

Bild: IPV6 beim default ping

```
C:\Users\Kiwi>ping raspberrypi.local

Ping wird ausgeführt für raspberrypi.local [fe80::4ce8:8aeb:c113:2846%4] mit 32 Bytes Daten:
Antwort von fe80::4ce8:8aeb:c113:2846%4: Zeit<1ms
Antwort von fe80::4ce8:8aeb:c113:2846%4: Zeit<1ms
Antwort von fe80::4ce8:8aeb:c113:2846%4: Zeit<1ms
Antwort von fe80::4ce8:8aeb:c113:2846%4: Zeit<1ms

Ping-Statistik für fe80::4ce8:8aeb:c113:2846%4:
    Pakete: Gesendet = 4, Empfangen = 4, Verloren = 0
    (0% Verlust),
    Ca. Zeitangaben in Millisek.:
    Minimum = 0ms, Maximum = 0ms, Mittelwert = 0ms
```

Ping.exe kann man mit Parameter zu IPv4 zwingen

-4	Erzwingt die Verwendung von IPv4.
-6	Erzwingt die Verwendung von IPv6.

Bild: IPV4 erzwungen

```
C:\Users\Kiwi>ping -4 raspberrypi.local

Ping wird ausgeführt für raspberrypi.local [192.168.0.135] mit 32 Bytes Daten:
Antwort von 192.168.0.135: Bytes=32 Zeit<1ms TTL=64
Antwort von 192.168.0.135: Bytes=32 Zeit<1ms TTL=64
Antwort von 192.168.0.135: Bytes=32 Zeit<1ms TTL=64
```

Dann bekommt man die gewohnte IPv4 Adresse

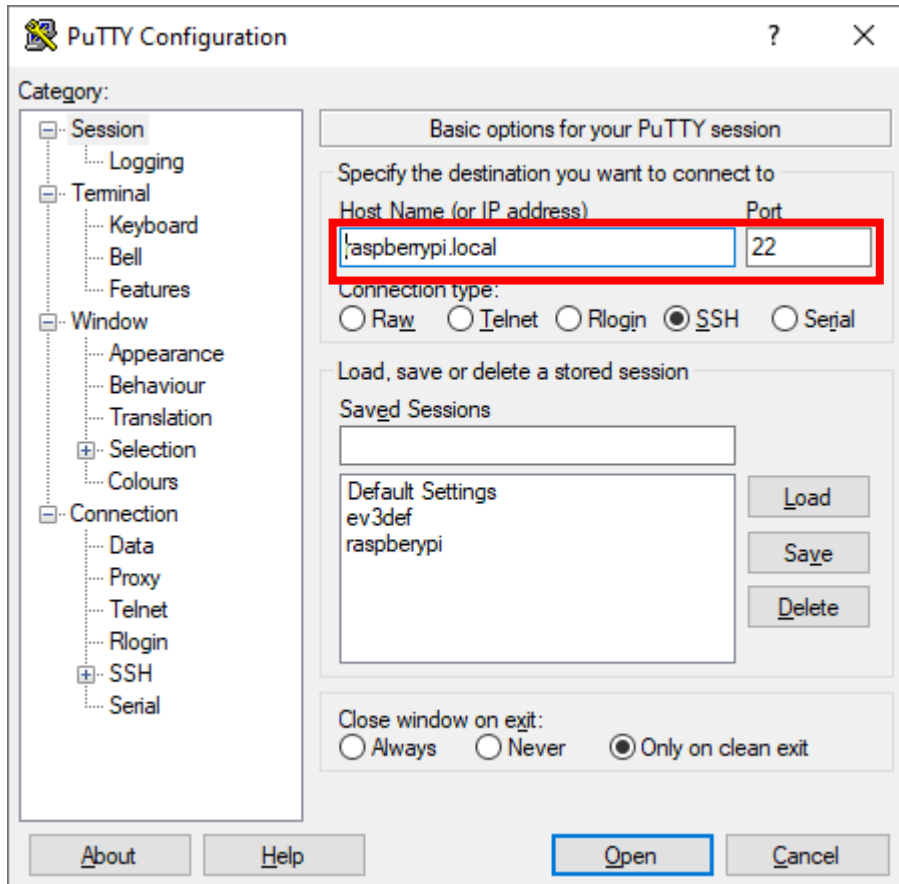


### 5.3 Remote Access

Ein Terminal Programm nach Wahl verwenden, z.b. Putty.

Host-Name oder IP Adresse eintragen und Port 22, das Profil dazu speichern

- HostName: raspberrypi.local (alternativ die IP Adresse)
- Port: 22

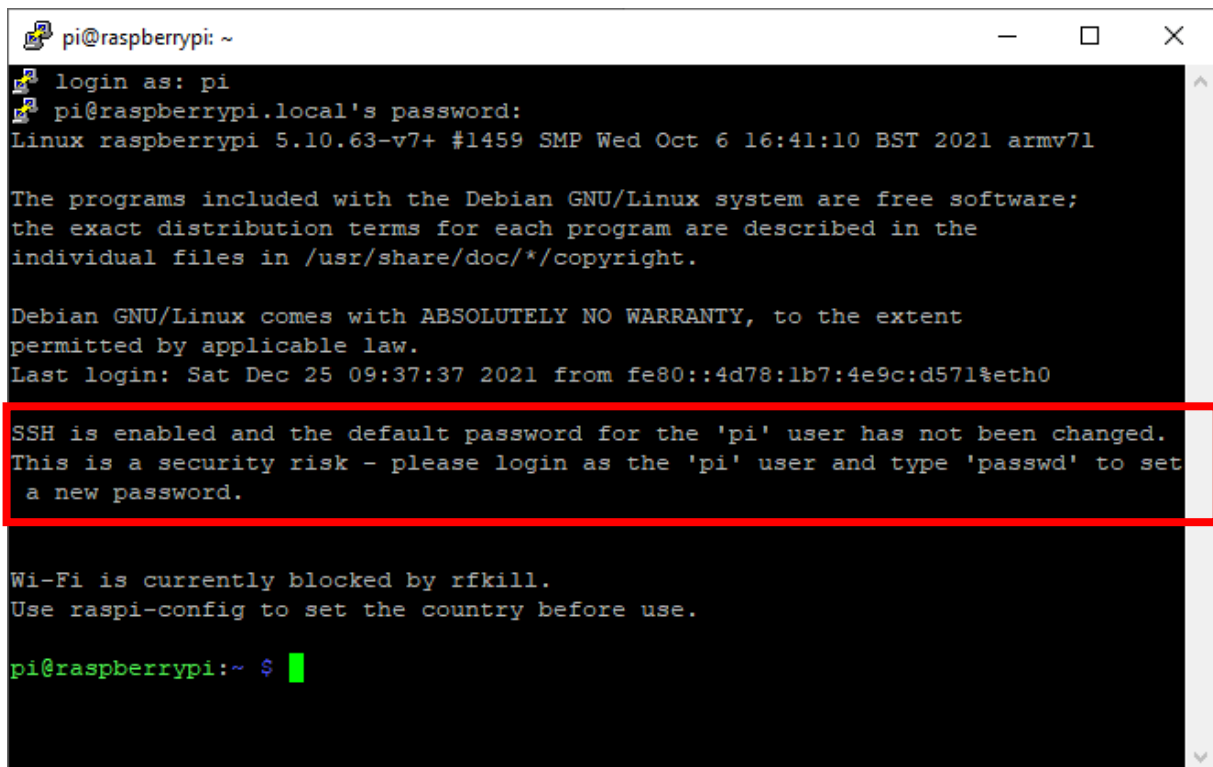


Login credentials (Defaults):

- Username: „pi“
- Password: „raspberrypi“

Nach erfolgreichem login das passwort natürlich ändern – hier geändert auf „picuber0815“  
Durch aufruf von „passwd“ wie es beim login empfohlen wird

- Username: „pi“
- Password: „picuber0815“



```
pi@raspberrypi: ~
login as: pi
pi@raspberrypi.local's password:
Linux raspberrypi 5.10.63-v7+ #1459 SMP Wed Oct 6 16:41:10 BST 2021 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Sat Dec 25 09:37:37 2021 from fe80::4d78:1b7:4e9c:d571%eth0

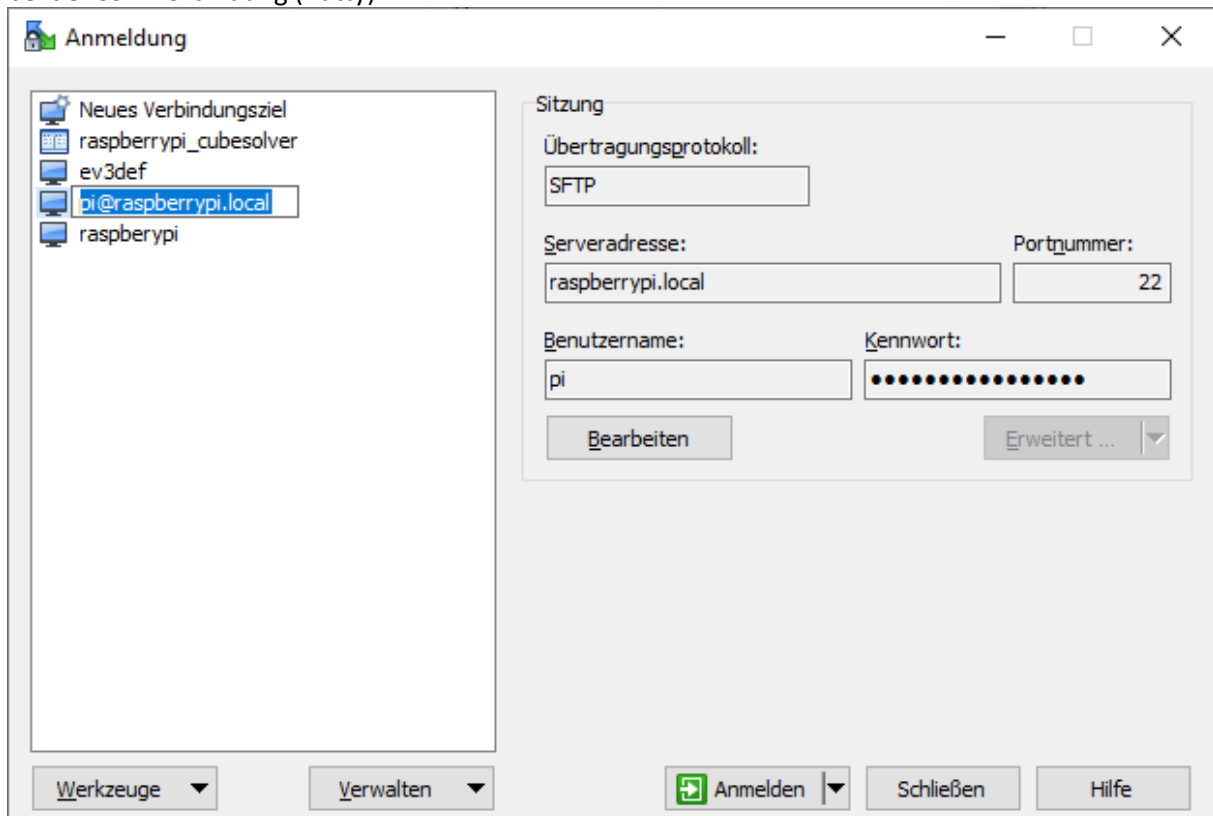
SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set
a new password.

Wi-Fi is currently blocked by rfkill.
Use raspi-config to set the country before use.

pi@raspberrypi:~ $
```

#### 5.4 Dateiaustausch mit dem RPI

Empfohlen wird das Programm „WinSCP“. Eingabe von Hostname, Port und den user Credentials wie bei der SSH Verbindung (Putty).



Es wird dann bedient wie ein Normaler File Explorer, und ermöglicht das Übertragen von Dateien in beide Richtung. Per default landet man nach dem login auf /home/pi – also dem Heimverzeichnis des angemeldeten users.

## 5.5 Kamera laufend bekommen

Hierzu gibt es einige Tutorials, hier ein Beispiel:

<http://raspberrypiguide.de/howtos/raspberry-pi-camera-how-to/>

Zuerst prüfen ob die camera vielleicht eh schon funktioniert:

- `vcgencmd get_camera`

```
pi@raspberrypi:~ $ vcgencmd get_camera
supported=1 detected=1
pi@raspberrypi:~ $
```

**Supported=1 und detected=1** > alles ok.

Ist einer der Beiden oder beide =0 gibt's ein Problem.

Ein Testbild aufnehmen:

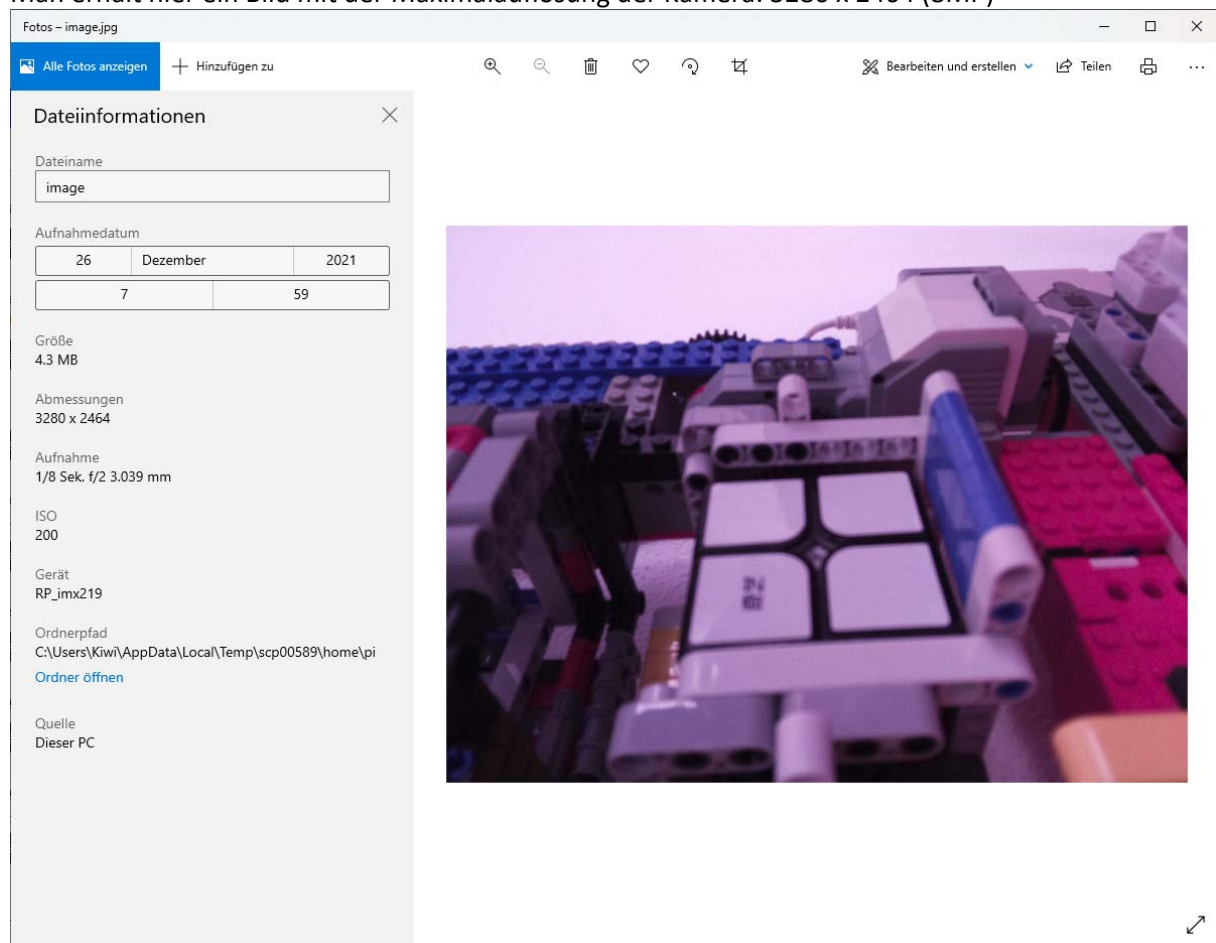
- `raspistill -o image.jpg`

Mit `ls` oder `dir` dann prüfen ob die Datei entstanden ist

```
pi@raspberrypi:~ $ ls
cube_2x2x2_front_2.jpg  cube_2x2x2_front.jpg  image.jpg
pi@raspberrypi:~ $
```

Mit WinSCP die Datei übertragen und ansehen:

Man erhält hier ein Bild mit der Maximalauflösung der Kamera: 3280 x 2464 (8MP)



### 5.5.1 Installationsschritte

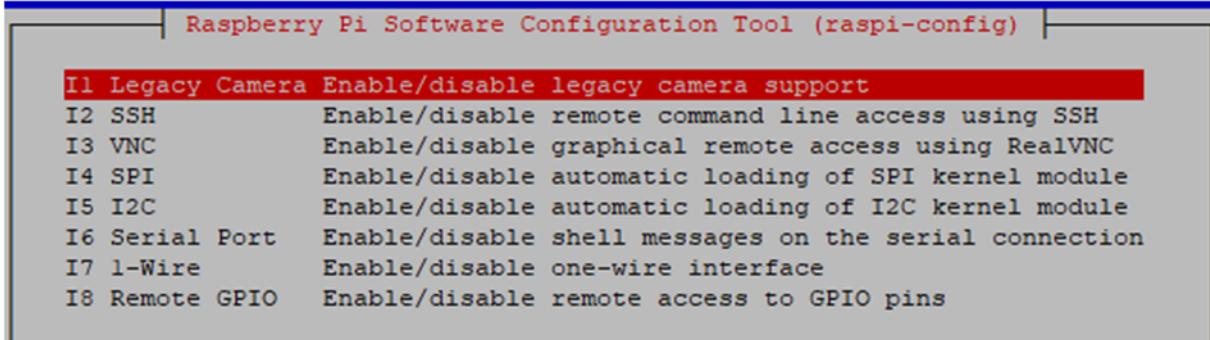
Im aktuellen Projekt hat das natürlich nicht sofort funktioniert, es wurden die Schritte laut guide ausgeführt:

Starten von Config Tool

- `sudo raspi-config`

```
pi@raspberrypi:~ $ sudo raspi-config
```

In dem aktuellen OS-Build scheint es in der Konfiguration auf al „I1 Legacy Camera“, das hat wohl mit einer Umstellung auf libcamera zu tun.



Aktivieren:

- I1 Legacy Camera Support
- I5 I2C Interface
- I6 Serial Port

Tool Beenden dann neu starten

- `sudo reboot`

Der hier verwendete Release: raspbian „bullseye“ (ver 11)

```
pi@raspberrypi:~ $ lsb_release -a
No LSB modules are available.
Distributor ID: Raspbian
Description:    Raspbian GNU/Linux 11 (bullseye)
Release:       11
Codename:      bullseye
pi@raspberrypi:~ $
```

Geht die Kamera dann schon > alles fein

Hat natürlich nicht funktioniert, es wird das Package Update ausgeführt um alle Packages zu aktualisieren. Man muss beide Kommandos in dieser Reihenfolge ausführen, es dauert sehr lange. 30min++

- `sudo apt-get update`
- `sudo apt-get upgrade`

```
pi@raspberrypi ~ $ sudo apt-get update
pi@raspberrypi ~ $ sudo apt-get upgrade
```

Nach dem erneuten Reboot hat die Kamera dann funktioniert, und wurde dann beim aufruf von `vcgencmd get_camera` richtig angezeigt

## 5.6 Kamera Ausrichtung und Fokus Punkt einstellen

Das geht am einfachsten mit einem Live-Stream, auch hier gibt es mehrere varianten wie man das macht.

Die hier getestete Variante ist via VLC laut diesem Guide:

<https://rasberry-projects.com/pi/pi-hardware/raspberry-pi-camera/streaming-video-using-vlc-player>

Dafür das VLC Package installieren am RPI:

- `sudo apt-get install vlc`

```
pi@raspberrypi:~ $ sudo apt-get install vlc
```

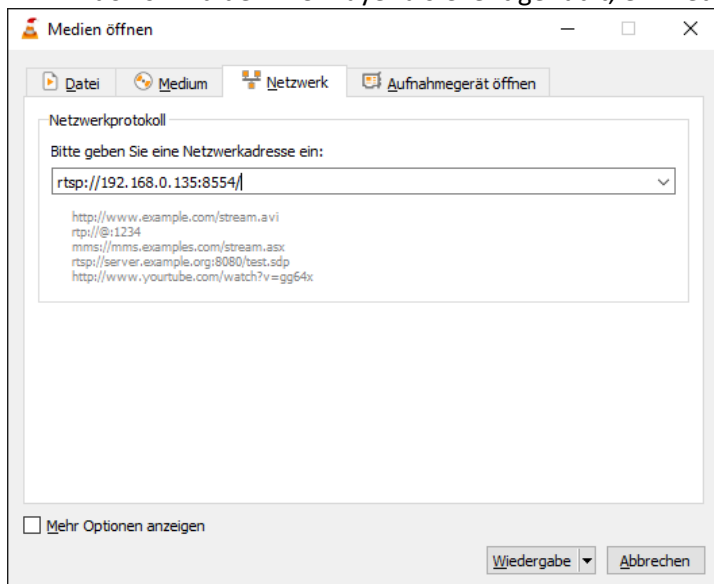
Kommand für das Streaming:

```
raspivid -o - -t 0 -n -w 480 -h 640 -rot 90 | cvlc -vvv stream:///dev/stdin --sout '#rtp{sdp=rtsp://:8554/}' :demux=h264
```

Erklärung:

Für das Live-Streaming wird die Ausgabe von `raspivid` an den Input von `cvlc` redirected durch klassische cmd-line pipes.

Im Windows wird der VLC-Player als Client genutzt, ein Netzwerkstream wird geöffnet.



**Anmerkung:**

**Das Streaming ist sehr schlecht, viele Frame-Drops und lange Verzögerungszeit. Das VLC Package benötigt auch etwa 500MB am RPI**

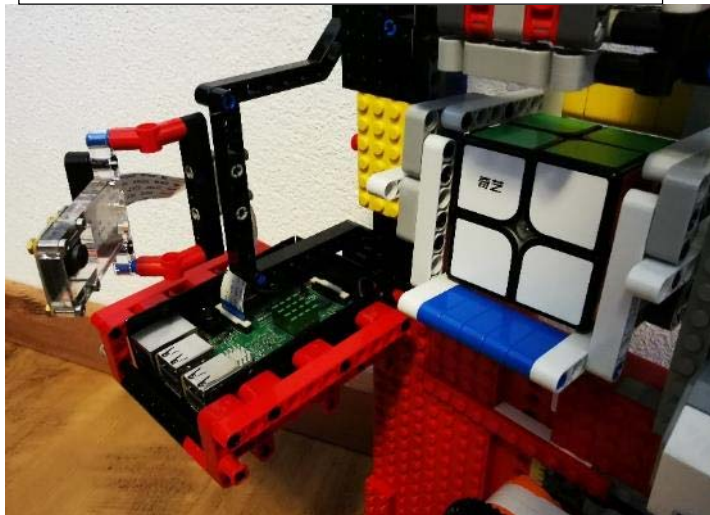
### 5.6.1 Digital Still Images (raspistill)

Bild: Montagehalterung für die Kamera & Einbausituation



Oberseite des Bildes

Einbausituation der Kamera, 90° gedreht



Das Kommando zur Bildaufnahme wurde bereits erwähnt:

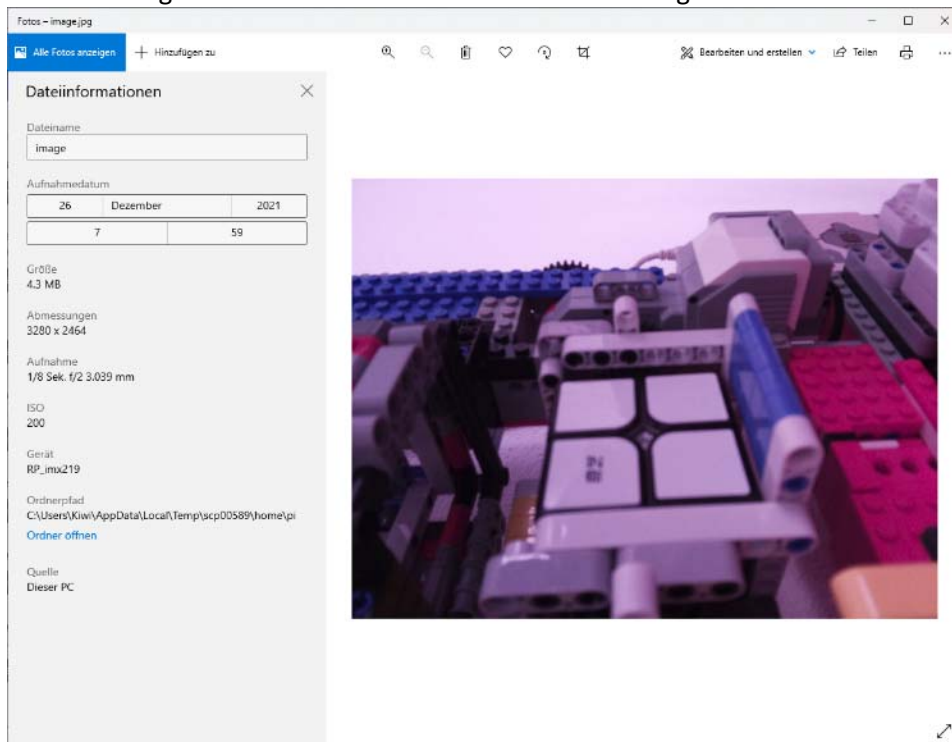
- `raspistill -o image_orig.jpg`

Mit WinSCP die Datei übertragen und ansehen:

Man erhält hier ein Bild mit der Maximalauflösung der Kamera: 3280 x 2464 (8MP) – Dateigröße ~4.3MB

Bedingt durch die Einbausituation der Kamera ist es 90° gedreht. Für die Bildanalyse wäre es egal aber es schaut halt nicht gut aus. Zur Korrektur ist eine Rechtsdrehung des Bildes um 90° erforderlich.

Die Auflösung 3280 x 2464 ist beinahe eine 4:3 Auflösung



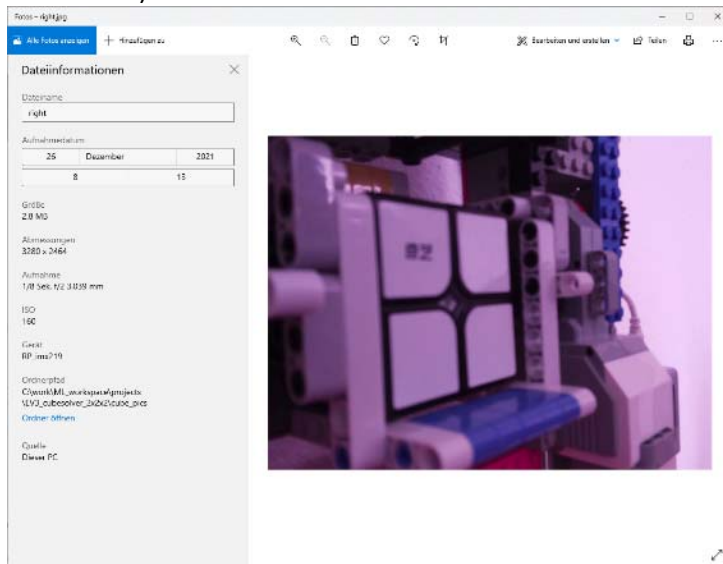


## Bild um 90° drehen

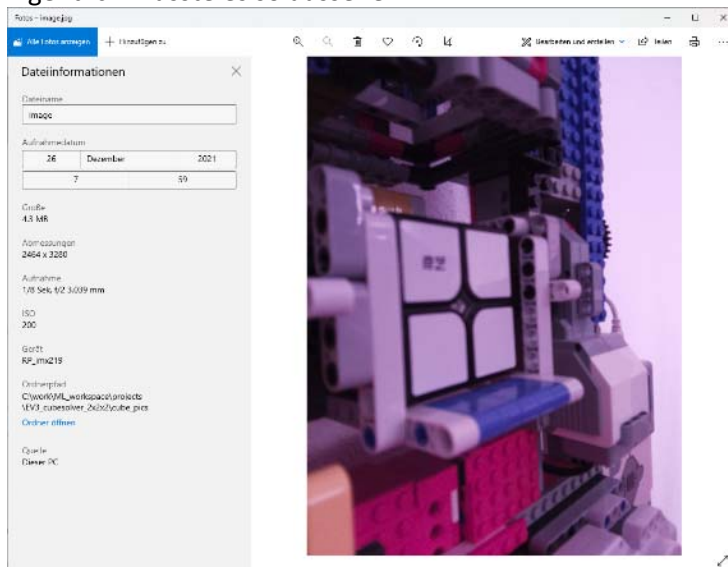
- `raspistill -rot 90 -o image_right_cropped.jpg`

Man erhält wieder ein Bild mit der Maximalauflösung der Kamera: 3280 x 2464 (8MP), Dateigröße aber nur noch ~2.8MB.

Bei Betrachtung sieht man dass es richtig gedreht ist aber ein cropping passiert ist. Also das Bild wurde von oben- und Unten beschnitten und in Längsrichtung aufgedehnt. (Digitaler Zoom auf 3280x2464)



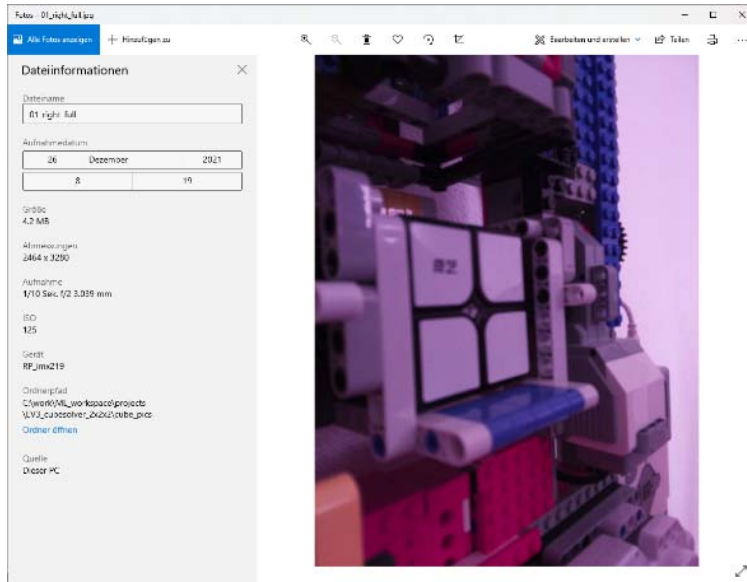
Eigentlich müsste es so aussehen:



Um das Bild in Originalgröße zu erhalten muss zusätzlich zu `-rot 90` auch angegeben werden wie die Zielauflösung aussieht. Diese muss dann dementsprechend 2464x3280 (90° gedreht werden), um ein Portrait-Bild zu erhalten.

**Bild um 90° drehen und Zielauflösung definieren (max):**

- `raspistill -rot 90 -w 2464 -h 3280 -o image_right_fullsize.jpg`



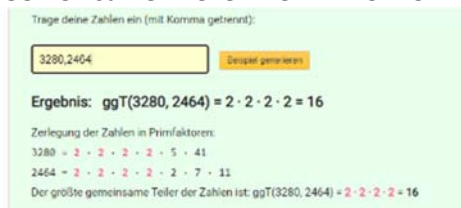
**Bild um 90° drehen und Zielauflösung definieren (verkleinern):**

- `raspistill -rot 90 -w 2464 -h 3280 -o image.jpg`

Um keine Pixel zu verlieren wird der Aspect Ratio gleich behalten. Mögliche Auflösung sind daher z.b:

- |          |       |       |
|----------|-------|-------|
| • /2     | 1232  | 1640  |
| • /4     | 616   | 820   |
| • /8     | 308   | 410   |
| • /10.25 | 240.4 | 320   |
| • /16    | 154   | 205   |
| • /32    | 77    | 102.5 |

205 x 154 ist die kleinste Auflösung welche sich ohne Rest teilen lässt, darunter gibt es keinen Gemeinsamen Teiler mehr. Wie Die Primfaktorenzerlegung bestätigt:

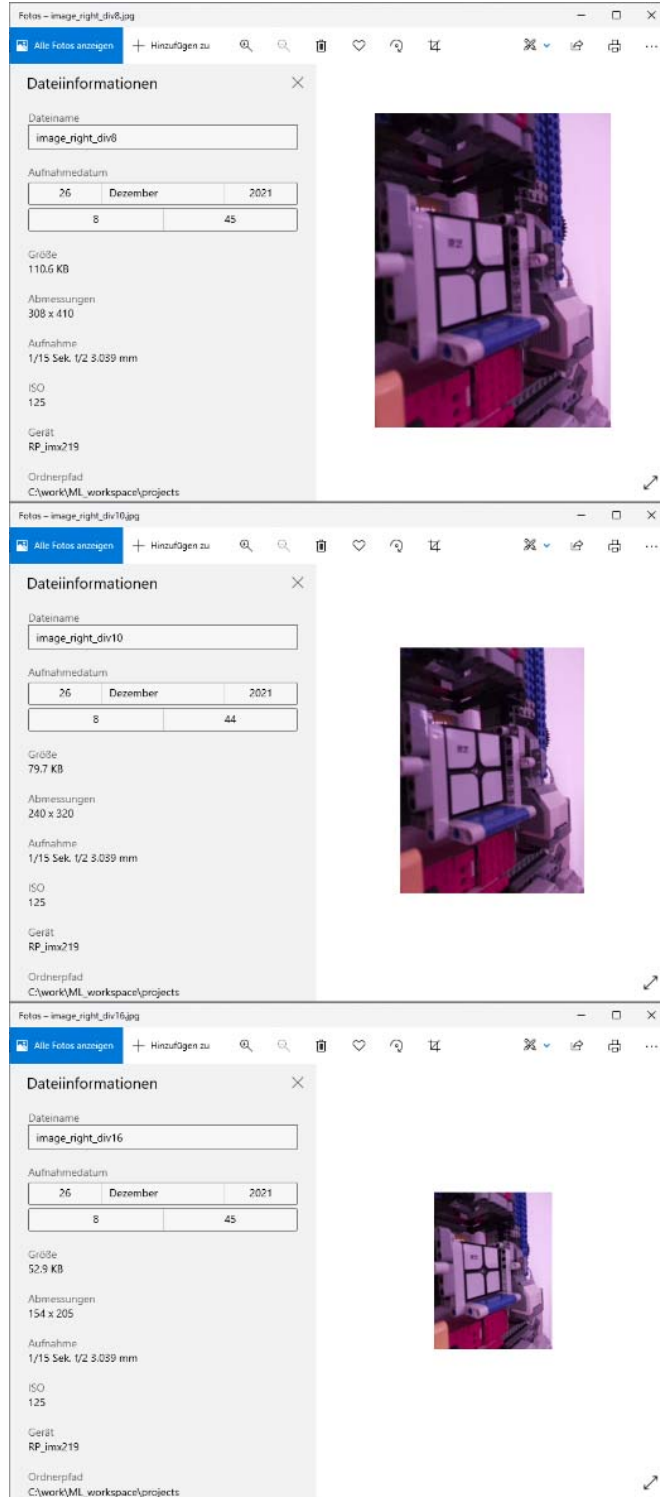


320 x 240 ist eine äußerst gängige Auflösung für Bildanalyse und hat nur einen Minimalen Divisionsfehler

### Aufnahme von 3 verkleinerten Bildern:

- `raspistill -rot 90 -w 308 -h 410 -o image_right_div8.jpg`
- `raspistill -rot 90 -w 240 -h 320 -o image_right_div10.jpg`
- `raspistill -rot 90 -w 154 -h 205 -o image_right_div16.jpg`

Die Bilder zeigen dass keine Ausschnitte fehlen



## 6 Lego EV3 USB Link

Den EV3 am RPI anschließen und einschalten

lsusb

```
pi@raspberrypi:/etc/udev/rules.d $ lsusb
Bus 001 Device 004: ID 0694:0005 Lego Group Mindstorms EV3
Bus 001 Device 003: ID 0424:ec00 Microchip Technology, Inc. (formerly SMSC) SMSC9512/9514 Fast Ethernet Adapter
Bus 001 Device 002: ID 0424:9514 Microchip Technology, Inc. (formerly SMSC) SMC9514 Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
pi@raspberrypi:/etc/udev/rules.d $
```

Der EV3 wird hier angezeigt, dann passt das schon

## 7 Python am RPI

Python 3.x ist vorinstalliert, Abfrage der Versionen:

- `python3 -V`
- `pip3 -V`

```
pi@raspberrypi:~ $ python3 -V
Python 3.9.2
pi@raspberrypi:~ $ pip3 -V
pip 20.3.4 from /usr/lib/python3/dist-packages/pip (python 3.9)
pi@raspberrypi:~ $
```

Wenn pip3 fehlt: (python3 verwendet daher pip3)

- `sudo apt-get install python3-pip`

```
pi@raspberrypi:~ $ sudo apt-get install python3-pip
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
python3-pip is already the newest version (20.3.4-4+rpt1).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
```

Wenn libusb fehlt:

- `sudo apt-get install libusb-1.0-0`

```
pi@raspberrypi:~ $ sudo apt-get install libusb-1.0-0
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
libusb-1.0-0 is already the newest version (2:1.0.24-3).
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
```

## 7.1 EV3\_DC installieren

`sudo pip3 install ev3_dc`

Es braucht also das pip3, lässt sich nicht über apt-get installieren

```
pi@raspberrypi:~ $ sudo pip3 install ev3_dc
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Requirement already satisfied: ev3_dc in /usr/local/lib/python3.9/dist-packages (0.9.9.1)
Requirement already satisfied: gTTS in /usr/local/lib/python3.9/dist-packages (from ev3_dc) (2.2.3)
Requirement already satisfied: thread-task>=0.9.7 in /usr/local/lib/python3.9/dist-packages (from ev3_dc) (0.9.7)
Requirement already satisfied: pyusb in /usr/local/lib/python3.9/dist-packages (from ev3_dc) (1.2.1)
Requirement already satisfied: click in /usr/local/lib/python3.9/dist-packages (from gTTS->ev3_dc) (8.0.3)
Requirement already satisfied: six in /usr/lib/python3/dist-packages (from gTTS->ev3_dc) (1.16.0)
Requirement already satisfied: requests in /usr/lib/python3/dist-packages (from gTTS->ev3_dc) (2.25.1)
```

Test:

`sudo python`

**Achtung:** Python muss im superuser mode laufen da sonst der Zugriff auf das USB-device nicht klappt  
Dann einfach die folgenden Lines-of Code in python eingeben:

```
import ev3_dc as ev3

my_robot = ev3.EV3(protocol=ev3.USB)
print(str(my_robot))
```

Wenn alles klappt wird die MAC-Adresse des EV3 Brick angezeigt

```
pi@raspberrypi:~ $ sudo python
Python 3.9.2 (default, Mar 12 2021, 04:06:34)
[GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import ev3_dc as ev3
>>>
>>> my_robot = ev3.EV3(protocol=ev3.USB)
>>>
>>> print(str(my_robot))
USB connected EV3 00:16:53:46:E9:BB (EV3)
>>>
>>>
```

Documentation zu ev3\_dc

<https://ev3-dc.readthedocs.io/en/latest/>

## 7.2 OpenCV Installieren

sudo pip3 install opencv-python

```
pi@raspberrypi:~$ sudo pip3 install opencv-python
Looking in indexes: https://pypi.org/simple, https://www.piwheels.org/simple
Collecting opencv-python
  Downloading https://www.piwheels.org/simple/opencv-python/opencv_python-4.5.4.60-cp39-cp39-linux_armv7l.whl (10.2 MB)
    | 10.2 MB 2.8 kB/s
Requirement already satisfied: numpy>=1.19.3 in /usr/lib/python3/dist-packages (from opencv-python) (1.19.5)
Installing collected packages: opencv-python
Successfully installed opencv-python-4.5.4.60
```

- Das funktioniert so aber nicht! Scheint recht kompliziert zu sein, build from source.

## 7.3 PIL – Python Image Library

<https://www.geeksforgeeks.org/working-images-python/>

Reading Pixel Data:

<https://stackoverflow.com/questions/138250/how-to-read-the-rgb-value-of-a-given-pixel-in-python>

PIL Data > numpy:

# Third party modules

import numpy

from PIL import Image

```
def get_image(image_path):
    """Get a numpy array of an image so that one can access values[x][y]."""
    image = Image.open(image_path, "r")
    width, height = image.size
    pixel_values = list(image.getdata())
    if image.mode == "RGB":
        channels = 3
    elif image.mode == "L":
        channels = 1
    else:
        print("Unknown mode: %s" % image.mode)
        return None
    pixel_values = numpy.array(pixel_values).reshape((width, height, channels))
    return pixel_values
```

## 7.5 Kamera in Python (picamera)

`sudo apt-get install python3-picamera`

```
pi@raspberrypi:~ $ sudo apt-get install python3-picamera
Reading package lists... Done
```

Basic usage:

<https://picamera.readthedocs.io/en/latest/recipes1.html>

### 7.5.1 Capture to File

```
import time
import numpy as np
from picamera import PiCamera
camera = PiCamera()
camera.resolution = (1024, 768)
camera.start_preview()
time.sleep(2)
camera.capture('foo.jpg', resize=(320, 240))
```

### 7.5.2 Capture to numpy

```
import time
import numpy as np
from picamera import PiCamera
camera = PiCamera()
camera.resolution = (320, 240)
camera.start_preview()
time.sleep(2)
output = np.empty((240, 320, 3), dtype=np.uint8)
camera.capture(output, 'rgb')
print(output[0][0])
print(output[239][319])
```



## 7.6 Libcamera

Die Verwendung von raspicam als dem legacy camera interface wird nicht mehr empfohlen. Neue Anwendungen sollten mit libcamera gemacht werden.

raspicam und libcamera sind leider nicht gleichzeitig verfügbar

> in raspi-config das camera interface umstellen (legacy mode deaktivieren) und neu starten.

Hier findet sich information dazu:

<https://www.raspberrypi.com/documentation/accessories/camera.html>

Funktionsweise der Libcamera ist hier definiert:

<https://www.raspberrypi.com/documentation/accessories/camera.html#libcamera-and-libcamera-apps>

Auch bei Libcamera zeigen sich diverse Eigenheiten (digital Zoom & cropping vs picture Dimension & cropping)

Raspicam hat ein Problem mit der Orangen Seite – so sieht die orange Seite aus mit dem smartphone aufgenommen.



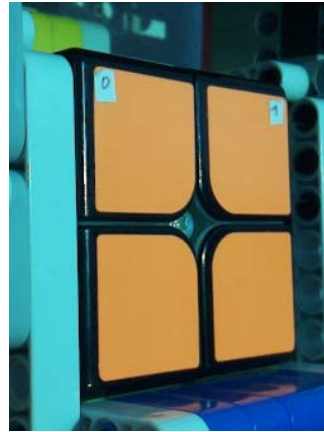
Darstellung in  
raspicam



Libcamera



Libcamera + tuning IMX219



Klar erkennbar liefert die Aufnahme von libcamera mit dem Tuning-File für den IMX219 Bild-Sensor die beste Qualität in Bezug auf Farbtreue, womit die Unterscheidung zwischen ROT und ORANGE deutlich einfacher wird.

## 7.7 Libcamera in Python

Es gibt zur Zeit kein Python Binding, libcamera wird daher einfach direkt ausgeführt als shell-cmd in Python. Dafür wird das subprocess module von python verwendet

### 7.7.1 CMD and Arguments

```
tuning_file= "--tuning-file  
/usr/share/libcamera/ipa/raspberrypi/imx219_noir.json "
```

```
exposure = "--immediate "
```

```
#img_filename = "cube_scan_results//2022-01-09_11-30-39_back.png"  
output_file = "-o " + img_filename
```

```
#width heigh, keep aspect ratio=4:3  
#actual changing the width height acts as digital zoom, so it crops the image,  
should be resolved in future version  
img_dimension = "--width=640 --height=480 "
```

```
#ROI (digital zoom)  
#roi is cropping the image around the given coordinates and afterwards resizes  
the image to the given resolution.  
#roi = "--roi 0.25,0.25,0.5,0.5 "  
roi = ""
```

```
shell_cmd = "libcamera-still " + tuning_file + exposure + img_dimension + roi  
+ pic_format + output_file
```

### 7.7.2 Subprocess Call

Für subprocess.run muss der cmd-string zunächst geteilt werden in substrings (sep = „space“)  
Der Call ist blockierend, die Outputs werden in pipes umgeleitet falls man sie noch benötigt

```
import subprocess  
import shlex  
cmd_and_args = shlex.split(shell_cmd)  
#print("Split Command Str: " + str(cmd_and_args))  
  
#CLI - dump stdout and stderr to variables  
process = subprocess.run(cmd_and_args,  
                          stdout=subprocess.PIPE,  
                          stderr=subprocess.PIPE,  
                          universal_newlines=True)  
assert(process.returncode == 0), 'libcamera-still exception'
```

## 8 C / C++ am RPI

Die Compiler sind bereits vorinstalliert

- Für C: gcc
  - gcc --version
  - gcc (Raspbian 10.2.1-6+rpi1) 10.2.1 20210110
- Für C++: g++
  - g++ --version
  - g++ (Raspbian 10.2.1-6+rpi1) 10.2.1 20210110

Es wurden Hello-World Programme in C / C++ vorbereitet (console-app) und in einer Windows IDE (Visual Studio getestet):

C

```
#include <stdio.h>

int main()
{
    printf("Hello World\n");
    return(0);
}
```

C++

```
#include <iostream>

int main()
{
    std::cout<<"Hello World\n";
    return(0);
}
```

Die Compilierungsanweisung am RPI ist wie folgt:

```
gcc -o hello_world_c hello_world.c
g++ -o hello_world_cpp hello_world.cpp
```

dadurch entsteht ein ausführbares File (binary)  
dieses wird dann gestartet durch „./<bin\_name>“

```
pi@raspberrypi:~/C_solver $ ./hello_world
Hello World
```

## 9 Color Scanning

### 9.1 Maschinen Bewegung

Die Maschine dreht den Würfel entsprechend vorgegebener Sequenz um ein Bild von allen 6 Seiten (faces) zu erhalten – welches als PNG abgespeichert wird.

Die Kamera-Verdrehung wird dabei im vorfeld korrigiert, dadurch sind alle Bilder in Hochformat 480x640.

Je Scan-Sequenz wird ein Unique Timestamp erzeugt als filename\_prefix.

Die Dateinamen enthalten ebenso einen unique identifier welche Seite des RubiksCube sie zeigen

Beispiel:

- 2022-01-16\_10-15-21\_back.png
- 2022-01-16\_10-15-21\_down.png
- 2022-01-16\_10-15-21\_front.png
- 2022-01-16\_10-15-21\_left.png
- 2022-01-16\_10-15-21\_right.png
- 2022-01-16\_10-15-21\_up.png

## 9.2 Farb-Positionsbestimmung im Bild

Es wird je „cubie“ ein Rechteck durch 2 Eckpunkte definiert, das kann man mit dieser Website zum Beispiel machen

[https://www.mobilefish.com/services/record\\_mouse\\_coordinates/record\\_mouse\\_coordinates.php](https://www.mobilefish.com/services/record_mouse_coordinates/record_mouse_coordinates.php)

die Pixeldaten der scan-rectangles werden alle eingelesen und der häufigste Wert daraus gebildet (Histogramm verfahren)

Als Encoding wird „HSV“ (Farbkreis 0-360°) verwendet anstatt RGB



Koordinaten Erstbestimmung

Links oben:

160,165

220,254

Rechts oben:

322,162

372,248

Links unten:

169,394

227,457

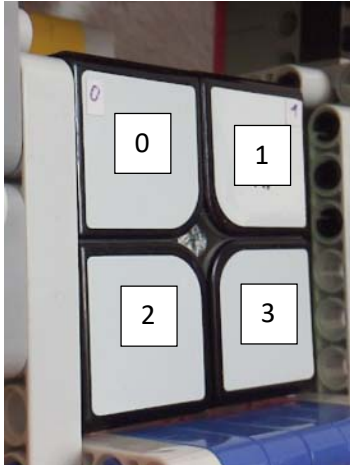
Rechts unten:

335,377

374,435

### 9.2.1 Reihenfolge

Es wird eine Reihenfolge definiert in welcher die Daten der Cubies kommen



```
face_hsv = cube_image_hsv(filename)
```

Hier nur als Beispiel, die Daten kommen als flat-array, 1 dimension – 3 Werte (tuple) in HSV codierung. (hue, saturation, value)

Filename: 2022-01-16\_10-15-21\_up.png Picture Dimension: 480 x 640

[0] = 178, 001, 081

[1] = 093, 000, 084

[2] = 198, 002, 081

[3] = 180, 002, 080

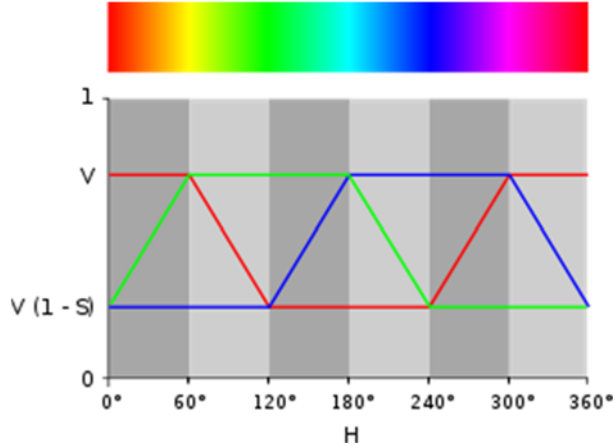
### 9.3 Farbdecodierung

Die Farbdecodierung mit HSV (hue, saturation, value) folgt nach einem einfachen Schema:

- Bestimmung der Weißen Farbe, diese ist Erkennbar an niedriger saturation
- Bestimmung der verbleibenden Farben anhand des hue (Farbkreis)

Das Bild zeigt das Mapping der RGB channels zu dem hue (H), was hier zu bemerken ist:

ROT ist im hue bereich an einer numerischen Unstetigkeitsstelle 0°/360°



Wertebereiche:

- Hue            0 – 360
- Saturation    0 -100
- Value         0-100

Auszug aus dem Quelltext

```
if(sat <= 25): #low saturation means white
    face_color = "white"
else:
    if(hue < 9) | (hue > 330):
        face_color = "red"
    elif(hue >= 9) & (hue < 45):
        face_color = "orange"
    elif(hue >= 45) & (hue < 90):
        face_color = "yellow"
    elif(hue >= 90) & (hue < 180):
        face_color = "green"
    elif(hue >= 180) & (hue < 270):
        face_color = "blue"
    else:
        face_color = "magenta"
```



Bei der Erkennung der eigentlichen Farbe gibt es einige Schwierigkeiten, hier aufgelistet die aufgetretenen Probleme:

- Beleuchtung > je nach Art des Umgebungslichtes gibt es hier verschiedene Probleme
  - Reflektion am Cube > dann wird als Farbe eher weiß erkannt

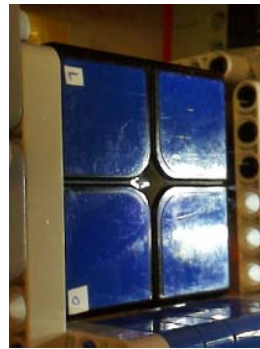
Links oben



rechts oben



starkes Licht bei dunklem Raum



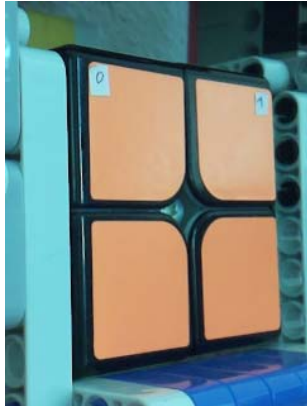
Beste Ergebnisse lassen mit der Smartphone Taschenlampe in etwa 0.5m Abstand erreichen

- Zu schwaches Umgebungslicht, lässt alles dunkler erscheinen



- Umgebungslicht mit anderer Spektraler Zusammensetzung lässt die Farben anders erscheinen (in Kombination mit dem AWB)  
Beispiele verschiedener Licht-Situationen:

Tageslicht



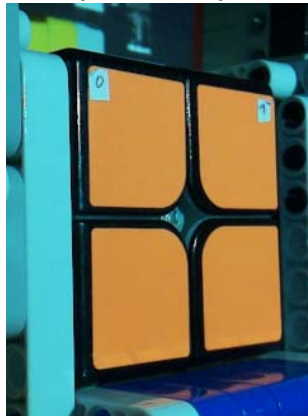
Led-Ringlicht (von Web Kamera) – 10cm Abstand



Deckenleuchte (LED)



Smartphone-Lampe 0.5m Abstand



**Smartphone Lampe wird verwendet, da stabilste reproduzierbare Ergebnisse unabhängig von der Tageszeit und sonstiger Lichtsituation (ausgenommen direktes Sonnenlicht)**

- Auto-White-Balance

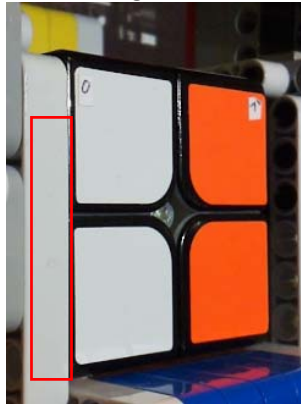
AWB ist standardmäßig aktiviert. Gut erkennbar ist der Weiss-abgleich an der Farbe umliegenden LEGO Bricks.

in diesen 2 Fällen klappt es auch ganz ordentlich.

Rot/weiß

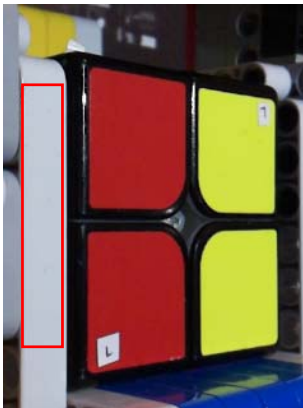


weiß/orange

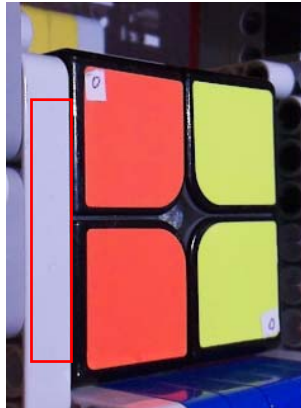


Betrachtet man die verbleibenden Bilder dieser Sequenz:

Rot/gelb



orange/gelb



Speziell beim orange/gelb Bild wird die Problematik deutlich, orange wird stärker Richtung rot verändert. AWB führt eine Anpassung durch indem der Gain des ROT und BLAU Kanales angepasst wird. In der Farbdekodierung wird es dann falsch erkannt, was auch mitunter daran liegt, dass orange, gelb und rot am hue Farbkreis recht eng aneinander liegen

Rot = 0°, orange = 30°, gelb = 60°

Die anderen Primärfarben Blau=240° und Grün=120° können immer richtig erkannt werden



Am real vorhanden Cube ist das orange leider schon mit einem rot-Anteil versehen, wodurch der Detektionsabstand von rot zu orange ohnehin sehr gering ist.

### 9.3.1 Auto AWB

Rot/gelb



orange/gelb



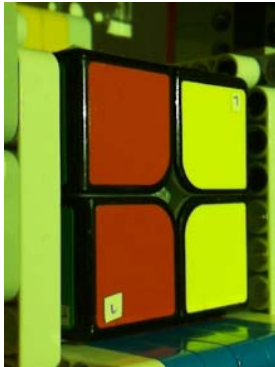
weiß/orange



### 9.3.2 Manueller AWB (R = 1, B = 1)

- Die Bilder werden gelbstichig

Rot/gelb



orange/gelb



weiß/orange



### 9.3.3 Manueller AWB (R=1, B=2)

Das wird vorerst beibehalten

Rot/gelb



orange/gelb

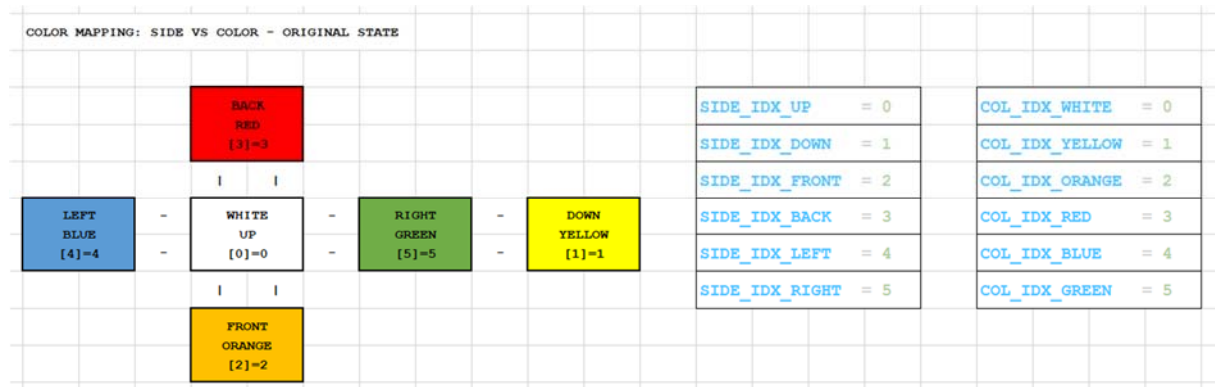


weiß/orange

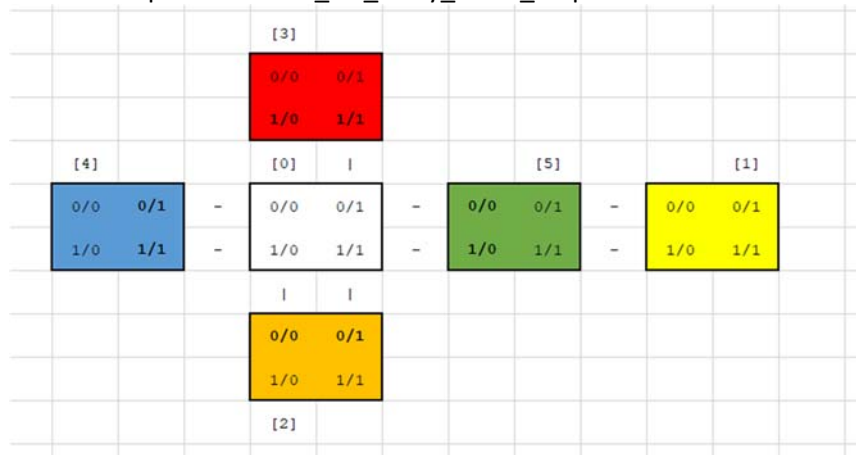


## 10 Rubiks Cube Facemap

Die Farb-Information wird als multidimensional Array erwartet mit Größe [6][2][2], der Wert der Farbe ist dabei ein Dezimal (0 bis 5) „color index“



Das facemap ist laut cube\_col\_array\_index\_map.xls definiert:



### „color facemap“

Der gelöste cube sieht im color facemap also so aus

[0][0][0] = 0	weiß
[0][0][1] = 0	weiß
[0][1][0] = 0	weiß
[0][1][1] = 0	weiß
[1][0][0] = 1	gelb
[1][0][1] = 1	gelb
[1][1][0] = 1	gelb
[1][1][1] = 1	gelb

## 10.1 Remapping scan-pics > rubikscube-facemap

Es wurden kleine Sticker angebracht um das remapping der Bilddaten auf das vollständige RubiksCube facemap zu erleichtern

Zusammenhang der Sticker mit der Position im Facemap:

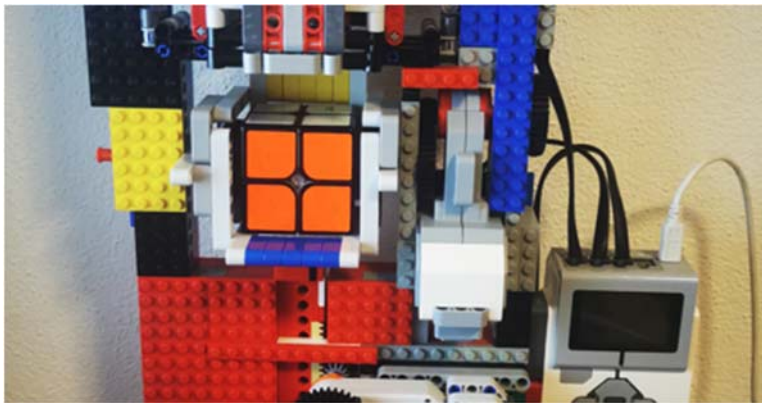
Sticker „0“ =  $[FACE\_IDX][0][0]$

Sticker „1“ =  $[FACE\_IDX][0][1]$

Es sind also immer die linke sowie die rechte obere Ecke definiert durch die Sticker

Der gelöste Würfel wurde so in die Maschine gelegt, dass seine Lage der standard Betrachtung des facemap entspricht:

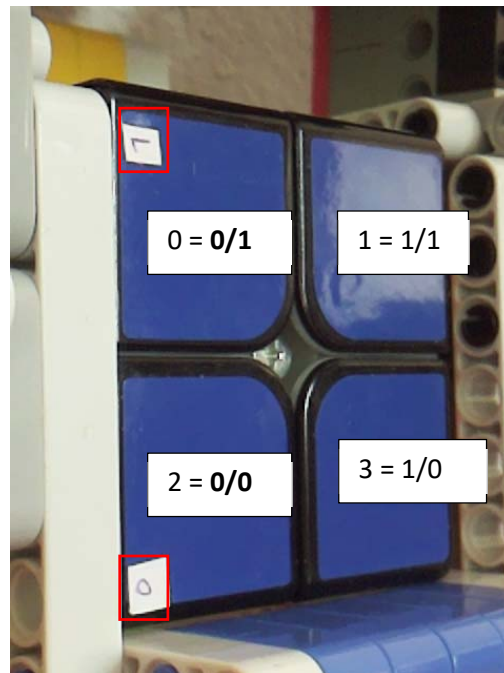
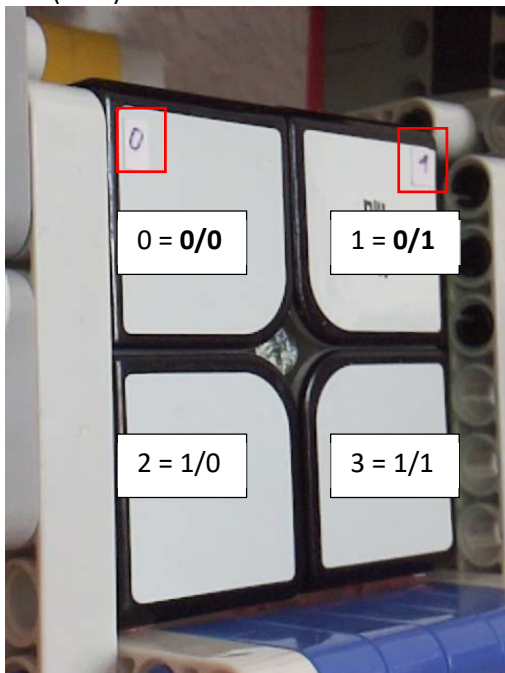
- up = weiß
- front = orange
- ..



Durch die SCAN Abfolge der Maschine erscheinen manche Seiten verdreht. Die Sticker helfen zu erkennen wo sich die Ecken befinden

In den Bildern sind die Sticker nun entsprechend zu erkennen und zeigen wo die beiden Ecken zu finden sind

up (white) „richtig“ dargestellt, d.h. wie es am facemap auch ist  
left (blue) seite ist verdreht 90° CCW





Das vollständige Remapping sieht so aus:

```
#remaps the 4 faces of a cube_pic
def face_remap(side_str, face):
    #number of rotations in ccw direction
    rot_ccw = 0
    if(side_str is "up"):    rot_ccw = 0
    if(side_str is "down"):  rot_ccw = 2    #180° CCW = 180° CW
    if(side_str is "left"):  rot_ccw = 3    #270° CCW = 90° CW
    if(side_str is "right"): rot_ccw = 1    #90° CCW = 270° CW
    if(side_str is "front"): rot_ccw = 0
    if(side_str is "back"):  rot_ccw = 2
```

Aufgenommene Bilder der Scan-Sequenz:



So würde es aussehen wenn man die Bild-Dateien entsprechend dreht.

Das remapping passiert aber mit den extrahierten HSV Werten der einzelnen cubies und nicht auf den Bild Rohdaten.





## 11 C-Solver

Der Solver wird in C geschrieben als CMD-Line Anwendung. Das wird gemacht da eine compiled App (binary - Maschinencode) eine schnellere Ausführungszeit hat als Python Skript Code, welcher zuerst interpretiert werden muss.

### 11.1 Data Representation

Es wird ein Facemap genutzt wie in Python.

Anstelle eines 3-Dimensionalen Arrays [6][2][2] erfolgt die Darstellung in einem u32 array der Größe [6], also in 6x4Byte. Jedes u32 repräsentiert eine Seite.

#### Python Darstellung

[0]	
0/0	0/1
1/0	1/1

Datenreihung:

0,1

2,3

#### C-Darstellung (mit den Bitmasken)

u32face[0]	
0xFF	0xFF<<8
0xFF<<24	0xFF<<16

Datenreihung: (byte order)

0,1

3,2

#### ACHTUNG:

Die Datenreihung wurde für die C-Darstellung geändert. Dadurch kann die „face rotation“ mit einem einzelnen CPU Kommando (ROTR bzw. ROTL) durchgeführt werden.

ROTR (rotate right) ... rotate\_face\_CCW

Es ist zu prüfen wie der maschinencode aussieht, da man die rotation ja nicht direkt beschreiben kann in C.

Der Source-Code zur Rotation. u8num ist in diesem Fall 8, 16 oder 24.

Wie man sieht muss man die Rotation durch Schiebe Operation und OR realisieren

```
ROTR = ((u32face >> u8num) | ((u32face << (32-u8num))));  
ROTL = ((u32face << u8num) | ((u32face >> (32-u8num))));
```

Ein ROTR mit 24Bits entspricht natürlich gleichermaßen einem ROTL mit 8 Bits in diesem Beispiel.

## 11.2 Kodierung der Farb-Information

```
enum COLOR_BITMAP{
    COL_IDX_WHITE   = 0,      //0b00000000
    COL_IDX_YELLOW  = 1,      //0b00000001
    COL_IDX_ORANGE   = 2,      //0b00000010
    COL_IDX_RED      = 3,      //0b00000011
    COL_IDX_BLUE     = 4,      //0b00000100
    COL_IDX_GREEN    = 5,      //0b00000101
};
```

Die Farbinformation in Dezimal kodierung passt in 3 Bits. Je face wird also  $4*3=12$  Bits Information benötigt, womit man auch mit einem u16 auskommen würde für den 2x2 cube.

Es wird aber ein u32 genommen da dieser auch für den 3x3 cube passen wird ( $8*3 = 24$  Bits, der center-piece ist nicht relevant für den solver da er beim 3x3 nicht veränderbar ist)

Beispiel:



Gemäß Datenreihung:

„Green-Yellow-Red-Orange“ > „5-1-3-2“

0xFF	0xFF<<8
0xFF<<24	0xFF<<16

u32face = 2<<24 | 3<<16 | 1<<8 | 5  
 u32face = TL (top-left) | TR | BR | BL (bot-left)

u32face = 0x 02 03 01 05 = 0b 00000010 00000011 00000001 00000101  
 ROTR (CCW) = 0x 05 02 03 01 = 0b 00000101 00000010 00000011 00000001  
 ROTL (CW) = 0x 03 01 05 02 = 0b 00000011 00000001 00000101 00000010

Hier sieht man ganz gut die ungenutzten Bits und dass es in den u16 reinpasst.

### 11.3 Umgang mit den Angrenzenden faces – Var 1

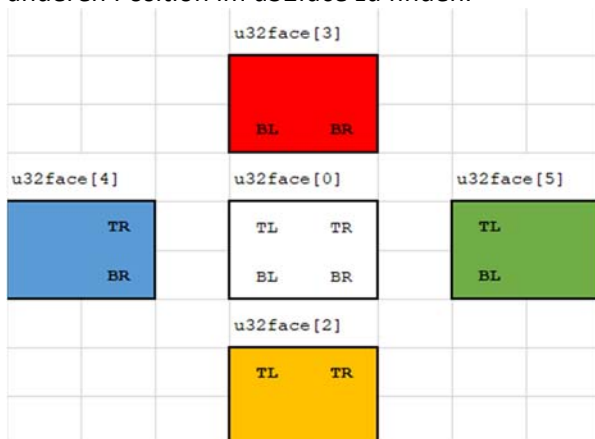
Austausch auf Elemente-Basis

TL...TOP LEFT

TR...TOP RIGHT

usw.

Hier sieht man die Beteiligten Datenelement bei einer Rotation der weißen Seite. Wie man sieht ist von den 4 angrenzenden Seiten bei gewählter Daten-Repräsentation die Information immer an einer anderen Position im u32face zu finden.



Für dieses Beispiel kann die rotation der angrenzenden Seite wie folgt beschrieben werden für ein CW rotation:

```
u32_mem = u32face[5]
```

```
TL_SET(u32face[5], BL_GET(u32face[3]));
```

```
BL_SET(u32face[5], BR_GET(u32face[3]));
```

```
BL_SET(u32face[3], BR_GET(u32face[4]));
```

```
BR_SET(u32face[3], TR_GET(u32face[4]));
```

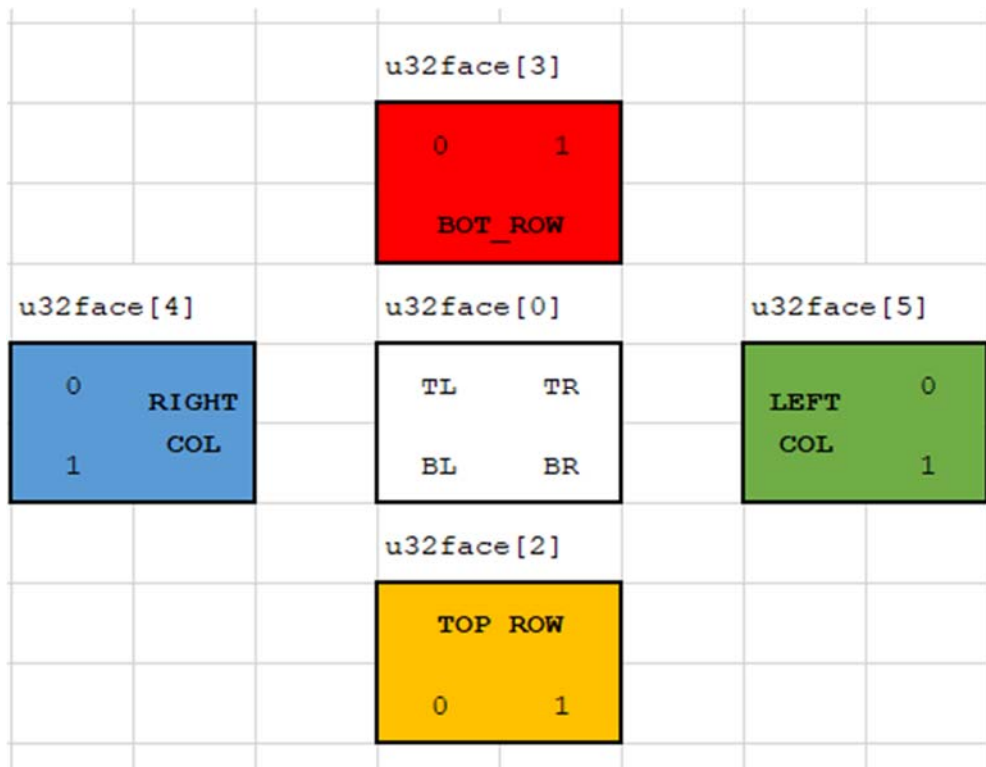
usw.

das lässt sich nicht sehr elegant umsetzen und ist kaum leserlich. Auch ist es ein großer Aufwand für die CPU, da jeweils nur 3 Bit in einer Zeile verschoben werden.

Auch die denkbare Erweiterung auf einen 3x3 / NxN cube explodiert im Aufwand des codings sowie CPU.

## 11.4 Umgang mit den Angrenzenden faces – Var 2

Austausch auf Zeilen / Spalten Basis



Diese Variante erscheint in der Darstellung bereits einfacher, wie man an der vollständigen Beschreibung für die CW-Rotation sieht:

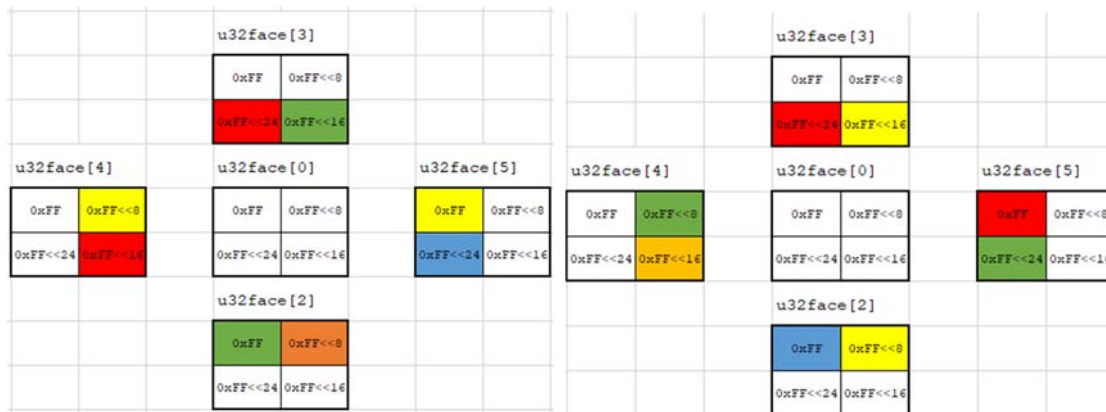
```
u32_mem = u32face[5]
LEFT_COL_SET(u32face[5], BOT_ROW_GET(u32face[3]));
BOT_ROW_SET(u32face[3], RIGHT_COL_GET(u32face[4]));
RIGHT_COL_SET(u32face[4], TOP_ROW_GET(u32face[2]));
TOP_ROW_SET(u32face[2], LEFT_COL_GET (u32_mem));
```

Es bleibt die Unschönheit dass die Rohinformation jeweils an einer anderen Bit-Position steht und nicht direkt verarbeitet werden können

### 11.4.1 Beispiel – Rotation „U“

Bild: VORHER

BILD: NACH ROTATE\_CW



Die entsprechenden Bitmasken für Reihen & Spalten:

BOT\_ROW (face[3]) MASK: 0xFFFF 0000  
 TOP\_ROW (face[2]) MASK: 0x0000 FFFF  
 RIGHT\_COL (face[4]) MASK: 0x00FF FF00  
 LEFT\_COL (face[5]) MASK: 0xFF00 00FF

Und den entsprechenden Dateninhalten:

BOT\_ROW (face[3]) DATA: 0x0305 0000  
 TOP\_ROW (face[2]) DATA: 0x0000 0205  
 RIGHT\_COL (face[4]) DATA: 0x0003 0100  
 LEFT\_COL (face[5]) DATA: 0x0400 0001

Drehung CW – Daten-Transformation:

Face[3] > Face[5] MASK: 0xFFFF 0000 > 0xFF00 00FF  
 Face[3] > Face[5] DATA: 0x0305 0000 > 0x0500 0003 (ROTL)  
  
 Face[5] > Face[2] MASK: 0xFF00 00FF > 0x0000 FFFF  
 Face[5] > Face[2] DATA: 0x0400 0001 > 0x0000 0104 (ROTL)  
  
 Face[2] > Face[4] MASK: 0x0000 FFFF > 0x00FF FF00  
 Face[2] > Face[4] DATA: 0x0000 0205 > 0x0002 0500 (ROTL)  
  
 Face[4] > Face[3] MASK: 0x00FF FF00 > 0xFFFF 0000  
 Face[4] > Face[3] DATA: 0x0003 0100 > 0x0301 0000 (ROTL)

Damit ist der Transformations-Algorithmus wie folgt zu formulieren:

Face[3] > Face[5]  
 Face[5] &= ~LEFT\_COL\_MASK; //zu ändernde Bits löschen = 0  
 Face[5] |= (LEFT\_COL\_MASK & ROTL(Face[3], 8)); //korrekt Bits setzen = 1

Selbiges lässt sich 1:1 Anwenden für ROTATE\_CCW, jedoch dann mit ROTR.

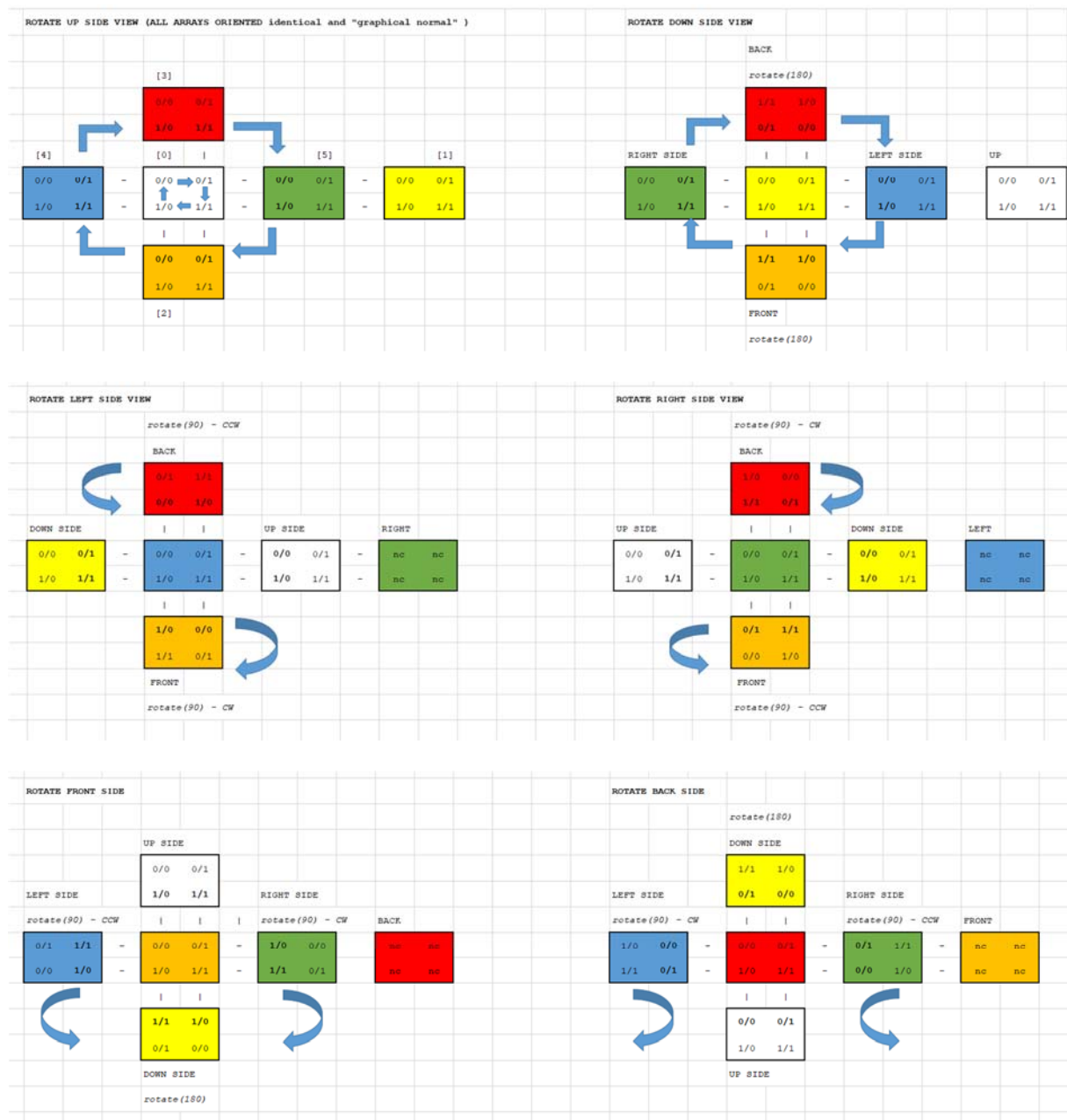
Solange das facemap einer Seite in ein integer passt, der durch CPU-instruction rotierbar ist, ist dies eine sehr elegante Lösung.

## 11.5 Umsetzung für ALLE Möglichen Seiten Rotationen

(Anhand der Python Umsetzung des 3x3 cube bekannt)

Es ergibt sich folgendes Schema für pre-processing Aktionen wenn man eine Seite drehen will:

UP-SIDE:	no pre-processing		
DOWN-SIDE:	BACK: 180°	FRONT: 180°	
LEFT-SIDE:	BACK: 90° CCW	FRONT: 90° CW	
RIGHT-SIDE:	BACK: 90° CW	FRONT: 90° CCW	
FRONT-SIDE:	LEFT: 90° CCW	RIGHT: 90° CW	DOWN: 180°
BACK-SIDE:	LEFT: 90° CW	RIGHT: 90° CCW	DOWN: 180°



Es braucht es also bei den Aktionen noch zusätzliche rotationen an faces, die jedoch allesamt wieder auf bitweise rotation zurückzuführen sind. Lediglich die Rotation an der UP-SIDE (U, U', U2) erfordert kein Preprocessing. Diese Schritte sollten die CPU-Zeit jedoch nicht verändern, da lediglich die Anzahl in der Bit-Rotation geändert wird und keine zusätzlichen Instructions hinzukommen

Im Python Skript-Code ist aktuell die Variante #1 umgesetzt mit einem remap der Einzelnen-Elemente.

Für den C-Solver soll die Variant #2 umgesetzt werden.

Die Python Implementierung kann dabei als Test-Environment genutzt werden zur Validierung der C-Implementation.