Höhere Technische Bundeslehranstalt (HTL) Graz-Gösting
Ibererstraße 15-21
A - 8051 Graz

Abteilung für ELEKTROTECHNIK
Ausbildungszweig Fachspezifischer Informationstechnik (Abendschule)

Optischer Zauberwürfellöser

Diplomarbeit in Fachspezifische Informationstechnik

Betreuer: Prof. Dipl.-Ing. Johannes Steffan
Eingereicht von: Christoph Orgl
Martin Neuhold
Datum: 20.05.2022

Eidesstattliche Erklärung

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die aus anderen Quellen wörtlich und inhaltlich entnommenen Stellen als solche gekennzeichnet habe.

Graz, 20. Mai 2022 Christoph Orgl

Hiermit erkläre ich an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die aus anderen Quellen wörtlich und inhaltlich entnommenen Stellen als solche gekennzeichnet habe.

Martin Neuhold Graz, 20. Mai 2022

Kurzfassung

Diese Diplomarbeit beschreibt die Lösung eines Zauberwürfels mittels einer von den Studierenden entwickelten, zusammengebauten und programmierten Maschine.

Die Stellung des Zauberwürfels wird mithilfe von zwei fest montierten Kameras erfasst und an eine Software weitergegeben. Der Algorithmus, der auf einem Raspberry PI läuft, berechnet die benötigten Schritte, um den Zauberwürfel zu lösen. Die Schritte werden an sechs unabhängige Motorsteuerungen gesendet, welche die Schrittmotoren zur Lösung des Zauberwürfels, steuern.

Abstract

This diploma thesis describes the solution of a magic cube using a machine developed, assembled and programmed by the students.

The position of the Rubik's Cube is recorded using two fixed cameras and passed on to software. The algorithm, running on a Raspberry PI, calculates the steps needed to solve the Rubik's Cube. The steps are sent to 6 independent H-bridges which control the stepper motors to solve the Rubik's Cube.

Danksagung

An dieser Stelle möchten wir all jenen Personen recht herzlich bedanken, die uns ermöglicht haben, diese Diplomarbeit durchzuführen.
Als Erstes möchten wir unserem Betreuer, Herrn Prof. DI Johannes Steffan, für die Unterstützung in fachspezifischen Themen danken und dafür, dass er uns immer auf die richtige Lösung gebracht hat.
Ein besonderer Dank gilt unseren Familien und Freunden, die uns während der gesamten Studienzeit motiviert und unterstützt haben.

Aufgabenstellung für die Diplomarbeit

Im Haupttermin: 2021/2022 Jahrgang: 8IBET

Thema: **Zauberwürfellöser**

Kandidaten: Christoph Orgl
Martin Neuhold

Aufgabe: Funktionsfähiges mechanisches Zauberwürfellösesystem mit Farbeinlesesoftware. Motorische Lösung des Würfels durch 6 Stepper-Motoren. Berechnung des Algorithmus auf Raspberry PI und Ansteuern der Motoren.

Schwerpunktgegenstände:

Fachspezifische **I**nformatik (**FI**)
Computerunterstützte-**P**rojekt-**E**ntwicklung (**CPE**)

Teilaufgaben (Christoph Orgl)

Design und Herstellung des Gerüsts mittels 3D – Drucker. Schreiben der Software für die Farberkennung der einzelnen Felder je Seite und Weitergabe der daraus resultierenden Daten an den Algorithmus. Für die Farberkennung wird die Programmiersprache Python und die Software OpenCV verwendet.

Teilaufgaben (Martin Neuhold)

Entwickeln der Software des Lösungsalgorithmus und der Schrittmotoren. Den Raspberry PI konfigurieren und die nötige Software installieren. Für die gesamte Software wird die Programmiersprache Python verwendet.

Beteiligte Firmen: keine

Betreuer: Prof. Dipl.-Ing Johannes Steffan

INHALTSVERZEICHNIS

Seite

1 ALLGEMEINES 1

1.1 Projektentstehung und Planung 1

1.2 Aufgabenstellung 1

1.3 Aufgabenverteilung 1

1.4 Meilensteine der Diplomarbeit 2

1.5 Projektstrukturplan 2

1.6 Kostenaufstellung 6

1.7 Projekttagebuch 7

1.7.1 Christoph Orgl 8
1.7.2 Martin Neuhold 9

2 HARDWARE 10

2.1 Antriebe 10

2.1.1 Schrittmotor Allgemein 10
2.1.2 Unipolar oder bipolar 10
2.1.3 Bauformen 11
2.1.4 Kenngrößen eines Schrittmotors 12

2.2 Elektronik Bauteile 14

2.2.1 Raspberry Pi 4B 8GB 14
2.2.2 ULN2003 Treiber Board 16
2.2.3 Kamera 16

2.3 Maschinengerüst 17

2.4 3D-Drucker 27

2.5 Beleuchtung 28

2.6 Schaltplan 29

3 SOFTWARE 31

3.1 Python 31

3.2 Algorithmus 31

3.2.1 Die Notation 32
3.2.2 Die Lösungssoftware 32

3.3 Motorsteuerung 46
3.3.1 Stepper Modul 46
3.4 Farberkennung 50
3.4.1 Verwendete Bibliotheken 50
3.4.2 Herangehensweise 51
3.4.3 Code Farberkennung Initialisierung 52
3.4.4 Code Bildaufnahme 56
3.4.5 Code Farbfilter ganze Ecken 58
3.4.6 Code Farbfilter geteilte Steine 60
3.4.7 Code Auswertung 63
4 VERWENDETE FUNKTIONEN FARBERKENNUNG 69
5 ZUSAMMENFASSUNG UND AUSBLICK 74
6 ABBILDUNGSVERZEICHNISSE 75
7 LITERATURVERZEICHNIS 78
8 ANHANG (INHALT USB-STICK) 79
8.1 USB-Stick 79

1 ALLGEMEINES

1.1 Projektentstehung und Planung

Der erste Vorschlag von Teammitglied Christoph Orgl war ein Schachroboter, der die Stellung der Figuren erkennt und Züge vorgibt. Teammitglied Martin Neuhold schlug ein vertikales, automatisch sortierendes Regalsystem vor. Da keiner der Vorschläge den anderen angesprochen hat, wurde nach erneutem Brainstormen der Vorschlag des Rubik Cube Solver von beiden Teammitgliedern akzeptiert.

Nach der Ideenfindung ging es in die Machbarkeitsstudie, durch Recherchen im Internet, wurde die Idee immer realistischer und umsetzbarer. Durch das Design, Entwicklung, Herstellung und Programmierung des Solvers wurde der nötige Arbeitsaufwand von 180 Stunden pro Studierenden erreicht.

1.2 Aufgabenstellung

Das Grundgerüst wird von den Studierenden designt und mittels 3D – Druck hergestellt.

Die Stellung des Zauberwürfels wird mithilfe von zwei fest montierten Kameras erfasst und an eine Software weitergegeben. Der Algorithmus, der auf einem Raspberry PI läuft, berechnet die benötigten Schritte, um den Zauberwürfel zu lösen. Die Schritte werden an sechs unabhängige Motorsteuerungen gesendet, welche die Schrittmotoren, zur Lösung des Zauberwürfels, steuern.

1.3 Aufgabenverteilung

Die Aufgaben wurden unter den Studierenden wie folgt aufgeteilt:

Christoph Orgl:

- Auswahl der Komponenten
- Entwurf und Herstellung des Gerüsts
- Bildverarbeitung
- Übergabe der Bilddatei an den Algorithmus
- Dokumentieren
- Schreiben der Diplomarbeit

Martin Neuhold:

- Auswahl der Komponenten
- Programmieren des Algorithmus und der Motorsteuerung
- Lösungsschritte für den Zauberwürfel erlernen
- Übergabe der Bilddatei an den Algorithmus
- Dokumentieren
- Schreiben der Diplomarbeit

Trotz dieser Aufgabenverteilung unterstützten sich die Studierenden in sämtlichen Belangen.

1.4 Meilensteine der Diplomarbeit

Folgende Meilensteine wurden zeitlich festgelegt:

- 13.09.2021 Festlegen einer Idee
- 14.09.2021 Abschließen der Machbarkeitsstudie
- 18.09.2021 Auswahl einer Programmiersprache, Komponenten und der Kostenaufstellung
- 24.10.2021 Aneignung des benötigten Knowhows
- 31.10.2021 Planung des Prototyps
- 07.11.2021 Konstruktion ist funktionsfähig fertiggestellt
- 14.11.2021 Erfolgreicher Test der einzelnen Komponenten, unabhängig der Software
- 05.12.2021 Erster Erfolgreicher Test der Software mit der Hardware
- 09.01.2022 Erfolgreiches Testen des gesamten Rubik Cubes Solver

1.5 Projektstrukturplan

Der Projektstrukturplan soll einen Überblick über das Projekt verschaffen. Er ist in der obersten Ebene in überschaubare Teilbereiche und Abschnitte gegliedert. Diese Hauptthemen werden in kleiner Arbeitspakete geteilt und den beiden Teammitgliedern zugeordnet.

Durch die hierarchische Anordnung müssen immer die übergeordneten Aufgaben zuerst erledigt werden, um nachfolgende beginnen zu können. Der Fortschrittstatus dieser Arbeitspakete wird farblich markiert und ist dadurch immer aktuell ersichtlich.

Für eine bessere Übersicht ist die Darstellung des gesamten Projektstrukturplans nachfolgend auf mehrere Seiten aufgeteilt.

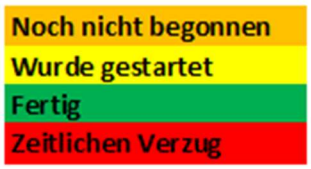


Abb.:1 Legende Projektstrukturplan

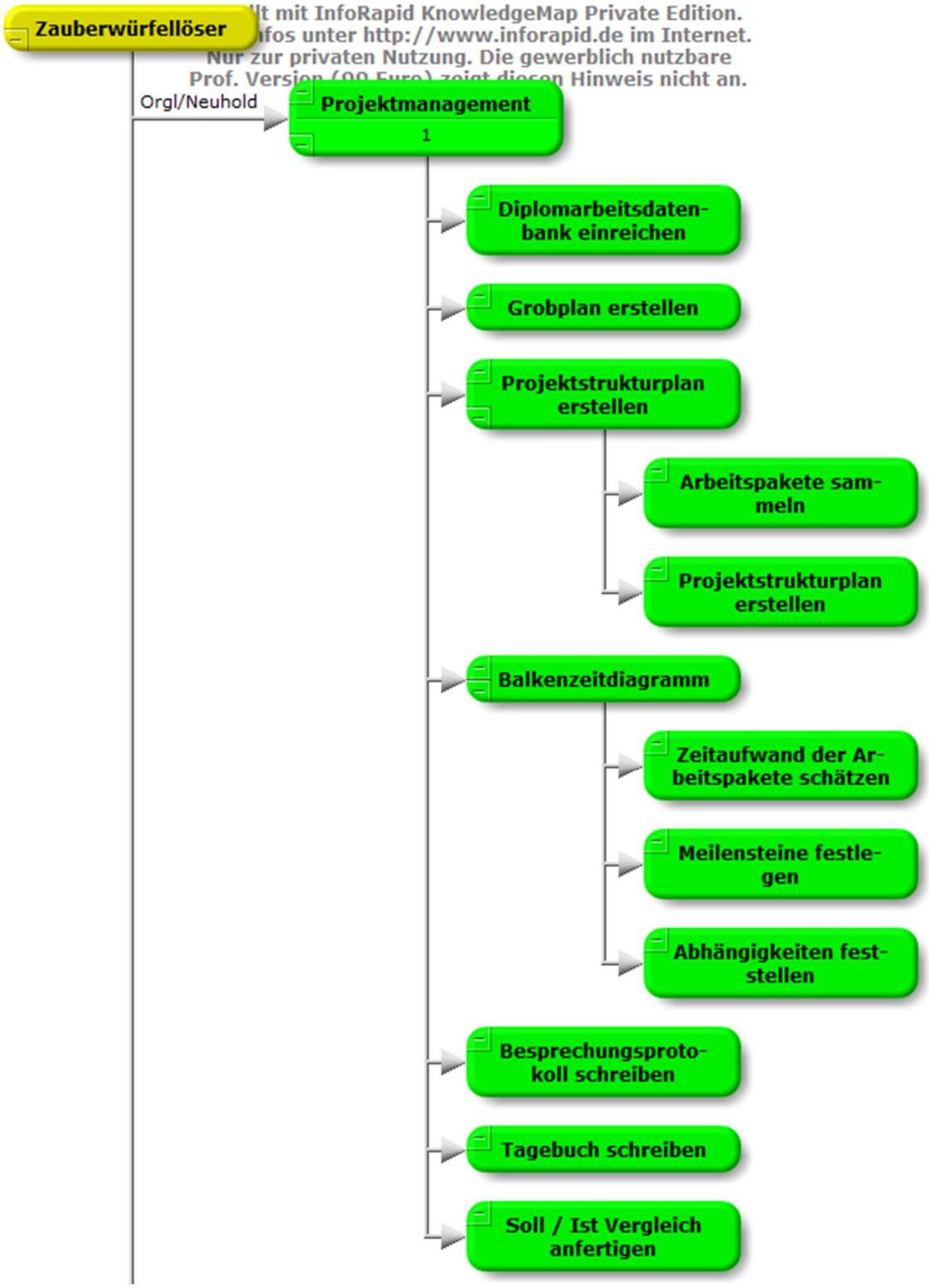


Abb.: 2 Projektstrukturplan Projektmanagement

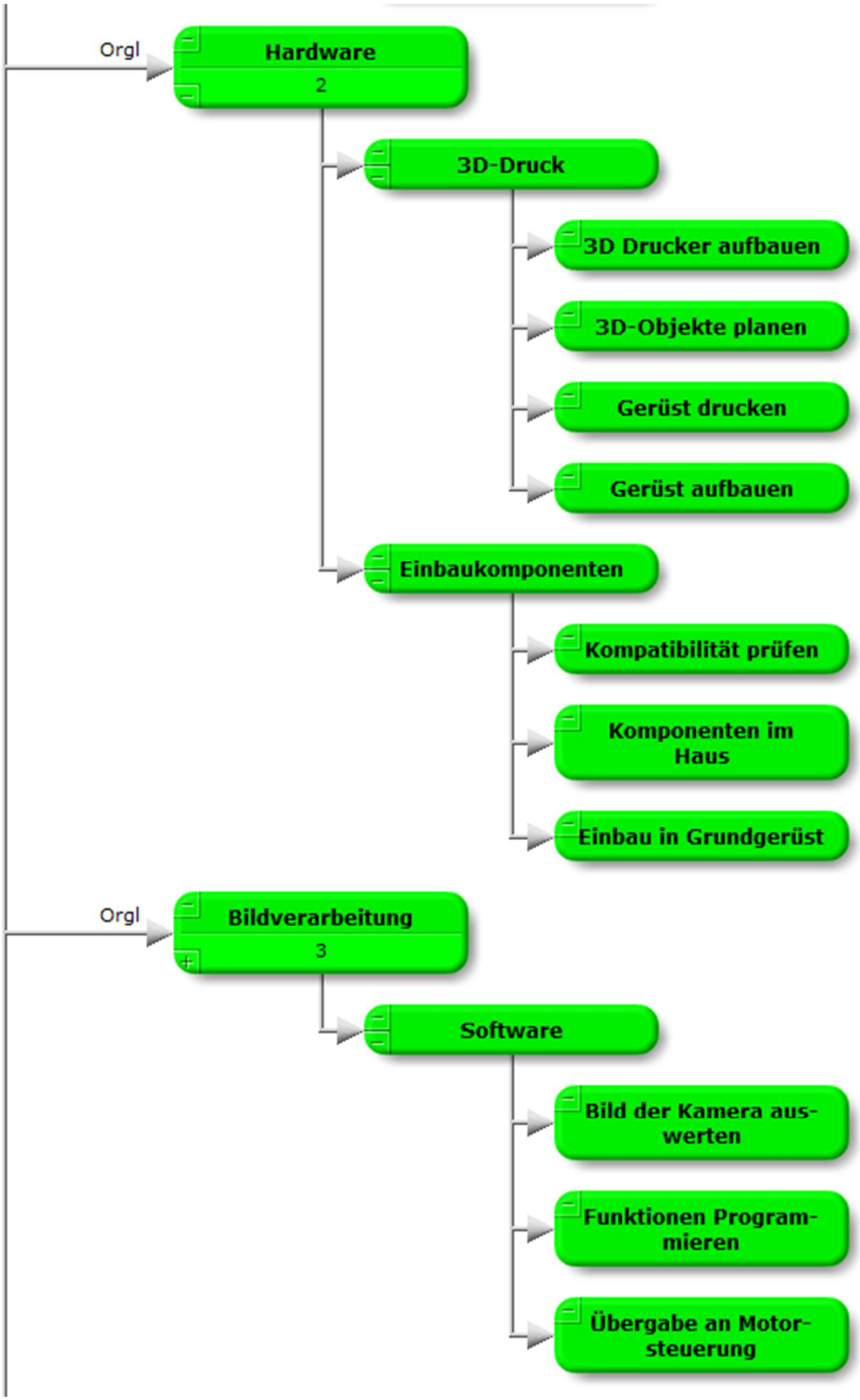


Abb.: 3 Projektstrukturplan Hardware und Bildverarbeitung

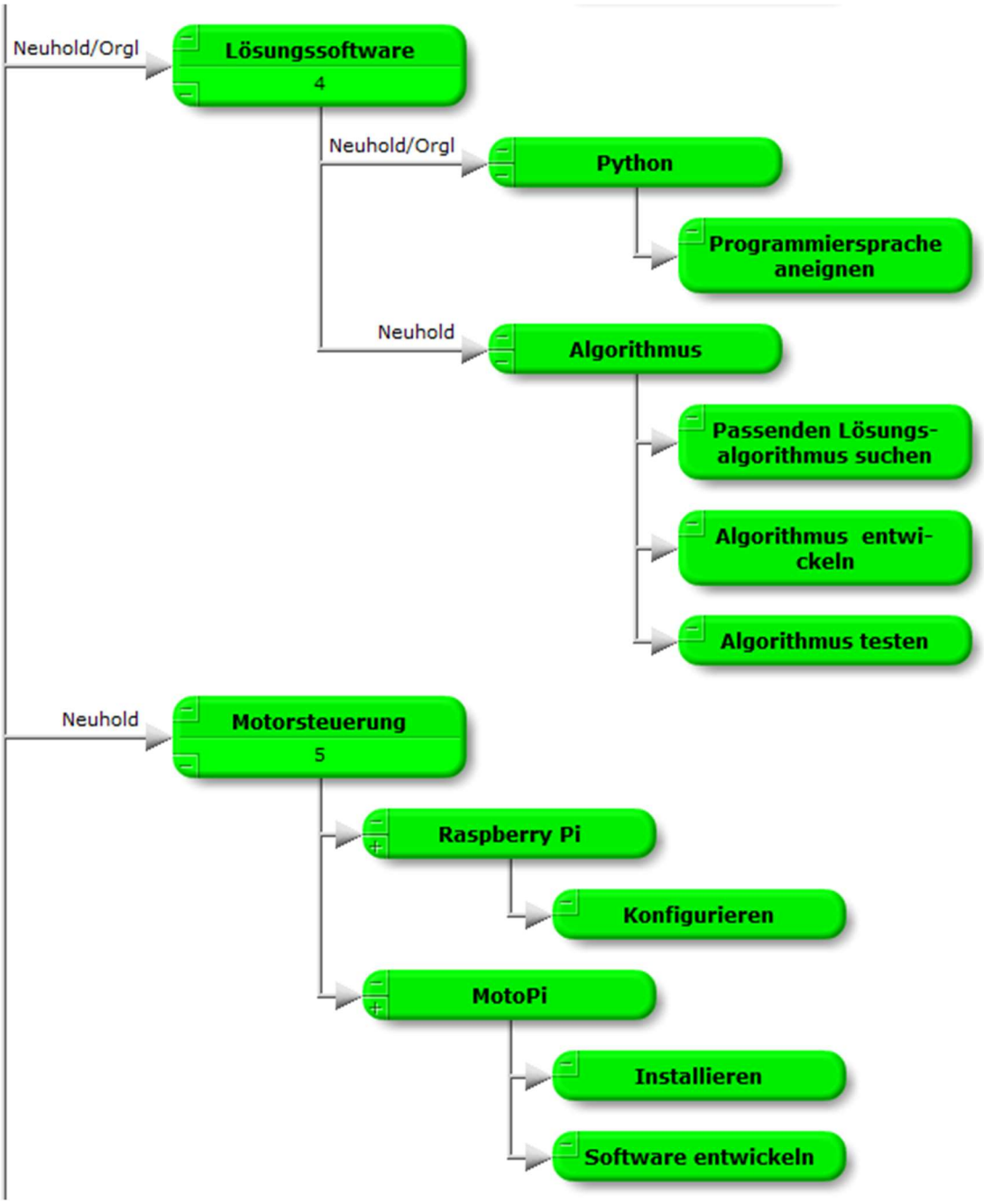


Abb.: 4 Projektstrukturplan Lösungssoftware und Motorsteuerung

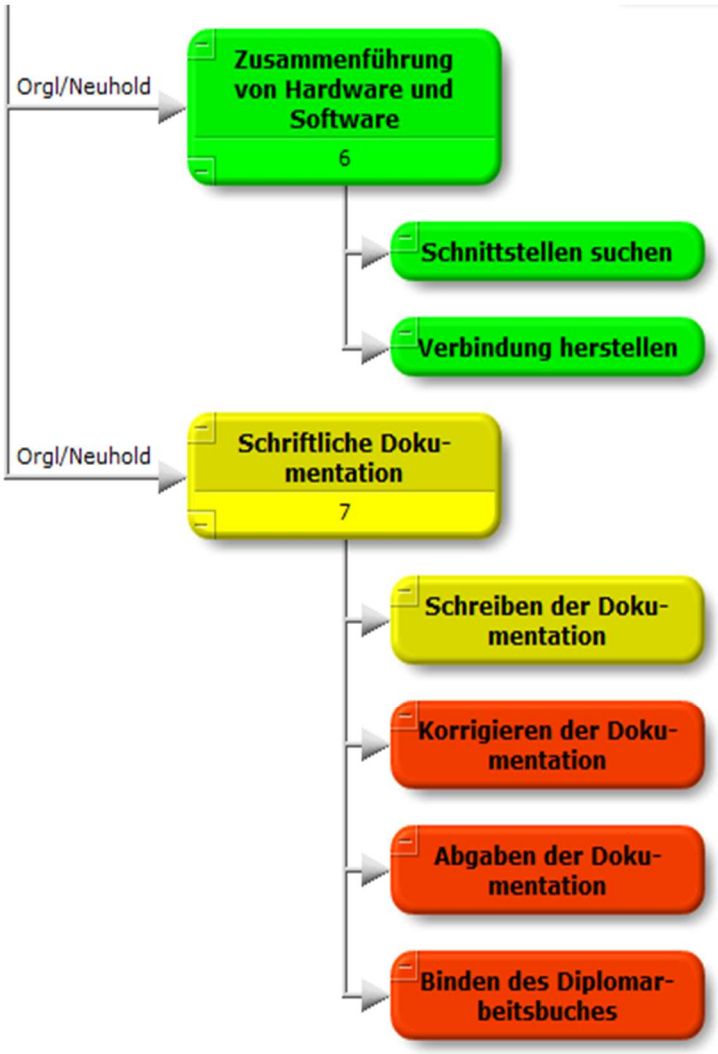


Abb.: 5 Projektstrukturplan Zusammenführung von HW, SW und schriftliche Dokumentation

1.6 Kostenaufstellung

Anzahl	Produkt	EH-Preis
1,00	Raspberry Pi 4B	€ 58,99
6,00	H-Brücke	€ 5,00
2,00	Webcam	€ 29,23
1,00	Rubic Cube	€ 7,99
6,00	Steppermotor	€ 18,96
1,00	Spannungsquelle	€ 14,24
1,00	Speicherkarte	€ 6,00
1,00	3D Drucker	€ 230,00
4,00	Druckerfilament	€ 20,00
6,00	Verlängerung	€ 10,00
1,00	LED-Band	€ 15,00
	Summe	**€ 674,44**

Abb.: 6 Gesamtkostenaufstellung

1.7 Projekttagebuch

Im Projekttagebuch wurde individuell festgehalten, welche Aufgaben und wann diese erledigt wurden. Das Projekttagebuch wird tabellarisch auf den folgenden Seiten dargestellt. Im Laufe der Diplomarbeit musste jedes Gruppenmitglied eine Mindeststundenanzahl von 180 Stunden in die Diplomarbeit investieren. Die im WLA investierten Stunden sind kein Bestandteil der Mindeststundenanzahl. Somit zählen nur Arbeitsstunden, die außerhalb der Schulzeit gearbeitet wurden zu der Mindeststundenanzahl. In dieser Gruppe hat jedes Mitglied die Mindeststundenanzahl überschritten und somit die Vorgabe eingehalten.

1.7.1 Christoph Orgl

Datum	Zeit[h]	Tätigkeit
13.09.2021	4,0h	Auswählen der benötigten Komponenten und bestellen dieser
20.09.2021	3,0h	Aufbauen des benötigten 3D-Druckers, kalibrieren und einen Probedruck durchführen
21.09.2021	5,0h	Planen, konstruieren und drucken der ersten Prototypen für die Konstruktion
22.09.2021	6,0h	Planen, konstruieren und drucken der ersten Prototypen für die Konstruktion
23.09.2021	9,0h	Endgültigen Prototyp für Betaphase geplant und aufsetzen des Raspberry PI
24.09.2021	5,0h	Zeichnen des Gehäuses und der Grundstruktur
25.09.2021	8,5h	Vorbereiten RP4, OpenCV
28.09.2021	10,0h	Selbststudium Python, OpenCV
09.10.2021	3,5h	Selbststudium Python
02.12.2021	5,0h	Ausarbeiten Präsentation
12.01.2022	5,0h	Umplanen der Kamerahaltung
06.02.2022	2,0h	Motorsteuerung entwickeln
04.02.2022	5,0h	Motorsteuerung entwickeln
12.01.2022	1,0h	Umplanen der Motoren
16.12.2021	6,0h	Besprechung Prof. Tockner OpenCV
10.02.2022	8,0h	Fertigstellen der Hardware
09.02.2022	8,0h	Installieren der Beleuchtung
10.02.2022	5,0h	Installieren der Beleuchtung
16.02.2022	4,0h	Abstimmen der Servomotoren
16.02.2022	5,0h	Alternativ Konzept ausarbeiten
18.02.2022	9,0h	Motorsteuerung mit Schrittmotoren entwickeln
21.02.2022	5,0h	Umplanen und Drucken Halterung Motoren
22.02.2022	7,0h	Schreiben der Bilderkennungssoftware
23.02.2022	11,0h	Schreiben der Bilderkennungssoftware
24.02.2022	7,0h	Erster Testlauf der Hardware und Fehlerbehebung
25.02.2022	5,0h	Entwickeln der Bilderkennung
26.02.2022	3,0h	Entwickeln der Bilderkennung
27.02.2022	7,0h	Entwickeln der Bilderkennung
28.02.2022	3,0h	Entwickeln der Bilderkennung
08.03.2022	2,0h	Schreiben der Diplomarbeit
14.03.2022	3,0h	Schreiben der Diplomarbeit
15.03.2022	6,0h	Schreiben der Diplomarbeit
18.03.2022	4,0h	Schreiben der Diplomarbeit
19.03.2022	4,0h	Schreiben der Diplomarbeit
20.03.2022	7,0h	Schreiben der Diplomarbeit
30.04.2022	3,0h	Schreiben der Diplomarbeit

1.7.2 Martin Neuhold

Datum	Zeit[h]	Tätigkeit
23.09.2021	9,0h	Endgültigen Prototyp für Betaphase geplant und aufsetzen des Raspberry Pi
25.09.2021	8,5h	Vorbereiten RP4, OpenCV
26.09.2021	10,0h	Suche nach optimalem Lösungsweg für den Solver
27.09.2021	2,0h	Erlernen der Programmiersprache Python (Grundlagen)
28.09.2021	10,0h	Selbststudium Python, OpenCV
30.09.2021	1,0h	Remote Zugang für Raspberry einrichten
09.10.2021	3,5h	Selbststudium Python
16.10.2021 - 18.10.2021	7,0h	Selbststudium Python
23.10.2021 - 30.10.2021	29,5h	Algorithmus entwickeln
26.11.2021	2,0h	Algorithmus entwickeln
27.11.2021	4,0h	Algorithmus entwickeln
30.12.2021	4,5h	Algorithmus entwickeln
02.01.2022	6,0h	Algorithmus entwickeln
03.01.2022	2,5h	Algorithmus entwickeln
06.01.2022	3,0h	Algorithmus entwickeln
26.01.2022	3,0h	Motorsteuerung entwickeln
05.02.2022	3,0h	Motorsteuerung entwickeln
06.02.2022	2,0h	Motorsteuerung entwickeln
12.01.2022	1,0h	Umplanen der Motoren
16.02.2022	4,0h	Abstimmen der Servomotoren
16.02.2022	5,0h	Alternativ Konzept ausarbeiten
18.02.2022	9,0h	Motorsteuerung mit Schrittmotoren entwickeln
18.02.2022	3,0h	Hardware für Motorsteuerung entwerfen
19.02.2022	7,0h	Hardware für Motorsteuerung entwerfen
19.02.2022	2,0h	Lösungssoftware optimieren
21.02.2022	5,0h	Hardware für Motorsteuerung entwerfen
21.02.2022	3,0h	Software für Motorsteuerung entwerfen
24.02.2022	7,0h	Erster Testlauf der Hardware und Fehlerbehebung
24.02.2022	5,0h	Fehlerbehebung Algorithmus
08.04.2022	3,0h	Schreiben der Diplomarbeit
09.04.2022	8,0h	Schreiben der Diplomarbeit
22.04.2022	6,0h	Schreiben der Diplomarbeit
23.04.2022	6,0h	Schreiben der Diplomarbeit
30.04.2022	3,0h	Schreiben der Diplomarbeit

Summe Orgl Christoph	Summe Neuhold Martin	Gesamtstunden
194,0h	192,5h	**386,5h**

2 HARDWARE

2.1 Antriebe

2.1.1 Schrittmotor Allgemein

Für diese Diplomarbeit wurden sechs Stück Schrittmotoren, des Typs 28BYJ-48 verwendet. Sie sind sehr kostengünstig und für die Zwecke der Diplomarbeit, im Sinne von Genauigkeit und Drehmoment, völlig ausreichend.

Ein Schrittmotor ist eine Art elektromagnetisches Gerät. Das sich in diskreten Schritten bewegt. Dieser hat mehrere Spulen. Auf der Mittelwelle ist eine Reihe von Magneten montiert und die die Welle umgebenden Spulen werden abwechselnd mit Strom versorgt. Dadurch werden Magnetfelder erzeugt, die die Magnete auf der Welle abstoßen oder anziehen, wodurch der Motor rotiert.

Abb.: 7 Schrittmotor

2.1.2 Unipolar oder bipolar

Schrittmotoren werden ein- oder mehrphasig ausgeführt. Dabei kann die Polarität der Ständerpole auf zwei Arten geändert werden:

Von Unipolarbetrieb spricht man, wenn jede Erregerwicklung einen Mittelabgriff besitzt. Jeder Teil einer Spule erzeugt ein Magnetfeld in nur eine Richtung. Besteht die Erregerwicklung aus einer Spule, deren Stromrichtung zur Umpolung fortlaufend geändert wird, spricht man von Bipolarbetrieb.

[1]

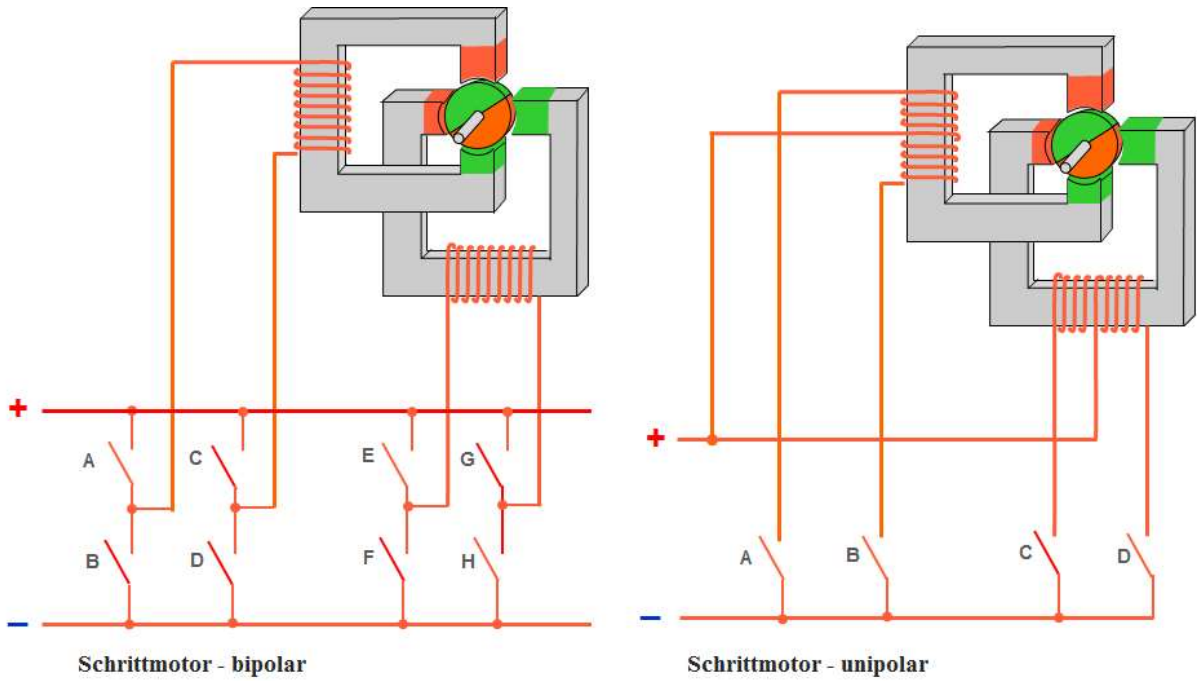


Abb.: 8 Bauformen Schrittmotor

Vorteil des unipolaren Aufbaus ist, dass nur vier Schalttransistoren benötigt werden. Mit der zunehmenden Integration von Halbleitern ist dieser Vorteil heute unbedeutsam. Der wesentliche Nachteil liegt darin, dass immer nur eine Hälfte der Wicklung bestromt wird, was zu einem niedrigeren Drehmoment bzw. zu einem schlechteren Verhältnis von Baugröße zu Drehmoment führt.

Bei bipolaren Schrittmotoren wird die gesamte Wicklung bestromt. Um die Wicklung umpolen zu können, sind an jedem Wicklungsende zwei Schalttransistoren nötig. Heutzutage erhält man zumeist Schrittmotoren mit bipolarem Aufbau.

[1]

2.1.3 Bauformen

Bei dem Reluktanzschrittmotor besteht der Rotor aus einem gezahnten Weicheisenkern. Bei diesem Material verschwindet nach dem Ausschalten des Statorstromes das Magnetfeld. Bei eingeschaltetem Strom fließt der magnetische Fluss durch den Weicheisenkern des Rotors. Die Drehbewegung des Rotors kommt zustande, weil vom gezahnten Stator (feststehendes, unbewegliches Teil des Motors) der nächstliegende Zahn des Rotors angezogen wird, da sich so der magnetische Widerstand verringert.

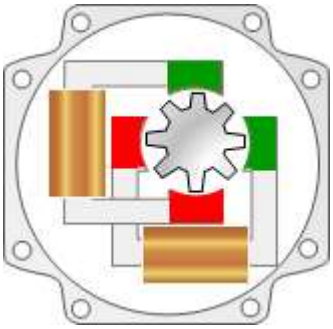


Abb.: 9 Reluktanzschrittmotor

[1]

Bei dem Permanentmagnetschrittmotor besteht der Stator aus Weicheisen und der Rotor aus Dauermagneten, die abwechselnd einen Nord- und einen Südpol aufweisen. Mit dem Stator-Magnetfeld richtet man den dauermagnetischen Rotor so aus, dass eine Drehbewegung entsteht.

[1]

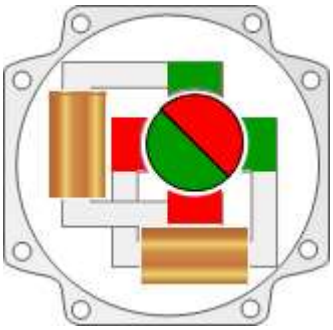


Abb.: 10 Permanentschrittmotor

Der Hybridschrittmotor vereint die Eigenschaften beider Bauformen, indem auf den Permanentmagneten noch ein gezahnter Weicheisenkranz eingefügt wird. Nahezu alle heute erhältlichen Schrittmotoren sind Hybridmotoren.

[1]

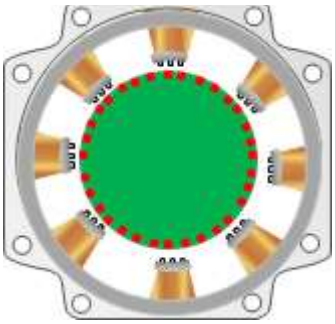


Abb.: 11 Hybridschrittmotor

2.1.4 Kenngrößen eines Schrittmotors

Phasenzahl:

Gebräuchlich sind Schrittmotoren mit ein bis fünf Phasen. Einphasige Motoren haben in der klassischen Antriebstechnik eine geringe Bedeutung. Die größte Verbreitung hat der 2-Phasenmotor mit seinen Varianten gefunden. Ein 4-Phasen Schrittmotor ist im Grunde ein 2-Phasenmotor mit getrennt herausgeführten Wicklungsanschlüssen. 5-Phasen Schrittmotoren bieten ein besseres Laufverhalten und eine von Natur aus höhere Schrittauflösung. Relativ neu auf dem Markt ist der 3-Phasen Schrittmotor.

[2]

Schrittwinkel:

Er besagt, welcher Drehwinkel mit dem Motor ohne elektronische Zusatzmaßnahmen aufzulösen ist. Die Angabe der Schrittzahl pro Umdrehung besagt sinngemäß dasselbe. Die Datenblätter der Motorhersteller beziehen sich auf Voll- oder Halbschritt. Um die gleiche Wellendrehzahl mit Halbschrittsteuerung zu erreichen, muss die doppelte Taktfrequenz - wie im Vollschritt - gefahren werden.

[2]

Phasenstrom:

Auf diesen Wert bezieht sich das Nennmoment des Motors. Mit diesem Strom kann der Motor im Allgemeinen auch dauernd betrieben werden, ohne ihn thermisch zu überlasten. In der Regel wird allerdings im Motorstillstand eine Stromabsenkung vorgenommen, um den Motor nicht unnötig thermisch zu belasten.

[2]

Haltemoment:

Das Haltemoment gibt an, welches Moment der Motor im Stillstand halten kann, ohne dass dieses eine kontinuierliche Drehung des Rotors hervorruft.

[2]

Drehmoment:

Das Drehmoment gibt an, welches maximale Moment der Motor bei unterschiedlichen Drehzahlen abgibt. Meist liegen diese Angaben in Form von Kennlinien vor.

[2]

Wicklungswiderstand

Dieser Wert ist Anhaltspunkt zur Berechnung der ohmschen Verluste im Motor. Ebenso hat dieser Wert ggf. Auswirkungen auf die Auslegung der Ansteuerung (Vorwiderstände etc.).

[2]

Wicklungsinduktivität:

Diese Angabe hat Bedeutung bei der Wahl der Betriebsspannung, da sie die Geschwindigkeit des Stromauf- und Abbaus mitbestimmt und damit die dynamischen Eigenschaften beeinflusst.

[2]

Nennspannung:

Die Nennspannung besagt, welche Spannung im stationären Fall an den Motor zu legen ist, um den Phasenstromnennwert zu erreichen. Sie ist bei Konstantstrom-Ansteuerung nicht mit der Betriebsspannung zu verwechseln.

[2]

Rotorträgheitsmoment:

Das Rotorträgheitsmoment addiert sich zum Trägheitsmoment der Last, und begrenzt damit die maximal mögliche Beschleunigung. Man benötigt diese Angabe zum Berechnen eines Antriebs.

[2]

Motortemperatur, Wicklungstemperatur:

Diese meist auf einen definierten Punkt am Motorgehäuse bezogene Temperatur darf während des Betriebes nicht überschritten werden, da ansonsten irreversible Schäden entstehen können. Die max. Motortemperatur ist abhängig von der verwendeten Temperaturbeständigkeit des Isolierlackes der Motorwicklung, dem Curietemperatur des Magnetmaterials, der Güte der Motorlager und weiteren konstruktiven Details des Motors.

[2]

Mechanische Maße, Gewicht, Wellenbelastung:

Zum mechanischen Ein- und Anbau von Motor und Last sind diese Angaben zu berücksichtigen.

[2]

Oft findet man auch Angaben über Schutzart, Isolationseigenschaften etc., welche bei speziellen Umgebungsbedingungen oder anderen Randbedingungen zu berücksichtigen sind.

2.2 Elektronik Bauteile

2.2.1 Raspberry Pi 4B 8GB

Der Einplatinen-Computer „Raspberry PI" dient dazu, über die zwei USB-Kameras, jeweils ein Bild des Würfels an den Raspberry PI zu senden und dieses auszuwerten. Die erkannten Felder werden an den Algorithmus weitergegeben, der die Lösungsschritte errechnet und damit die Motoren steuert.

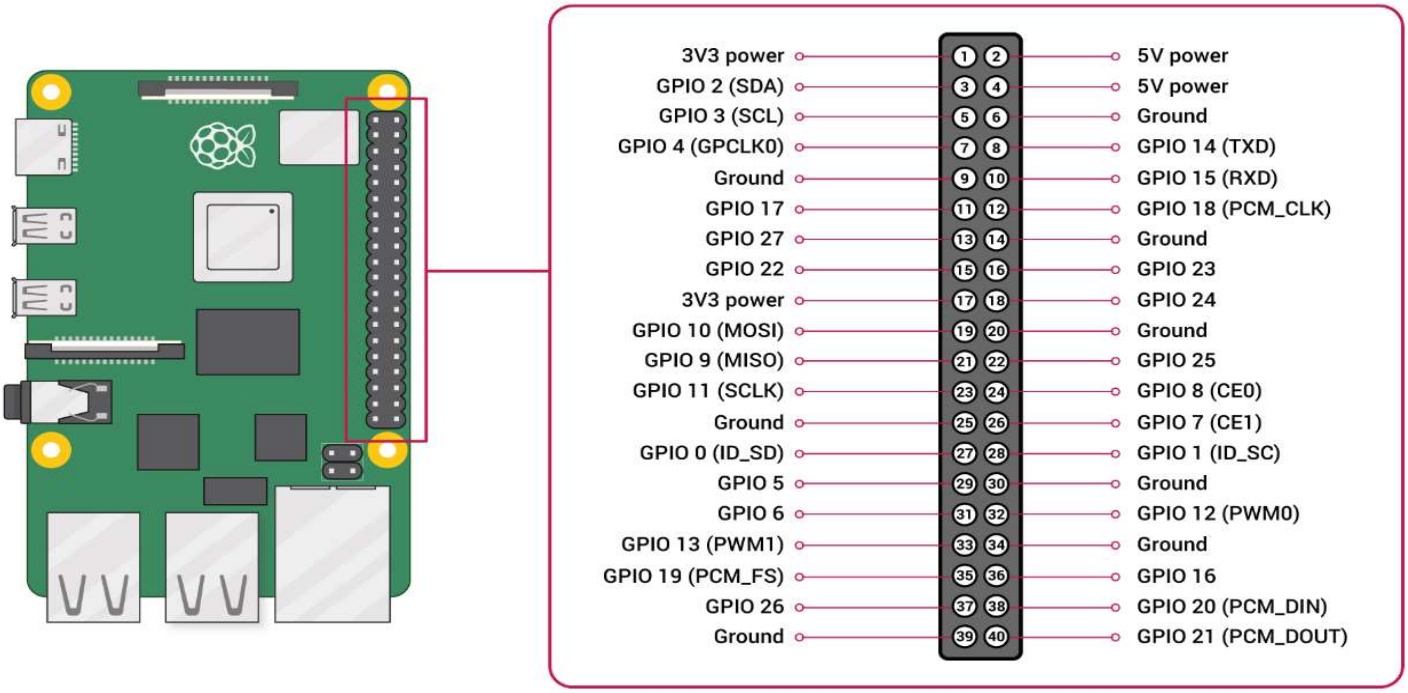


Abb.: 12 Pinbelegung RaspberryPi 4B

2.2.1.1 Hardwarespezifikation

- Quad core 64-bit ARM-Cortex A72 running at 1.5GHz
- 1, 2 and 4 Gigabyte LPDDR4 RAM options
- H.265 (HEVC) hardware decode (up to 4Kp60)
- H.264 hardware decode (up to 1080p60)
- VideoCore VI 3D Graphics
- Supports dual HDMI display output up to 4Kp60

2.2.1.2 Schnittstellen

- 802.11 b/g/n/ac Wireless LAN
- Bluetooth 5.0 with BLE
- 1x SD-Card
- 2x micro-HDMI ports supporting dual displays up to 4Kp60 resolution
- 2x USB2 ports
- 2x USB3 ports
- 1x Gigabit Ethernet port (supports PoE with add-on PoE HAT)
- 1x Raspberry Pi camera port (2-lane MIPI CSI)
- 1x Raspberry Pi display port (2-lane MIPI DSI)
- 28x user GPIO supporting various interface options

2.2.1.3 Software

Raspberry Pi OS (Raspbian) ist das offizielle Raspberry Pi Betriebssystem. Es ist kompatibel mit allen Einplatinencomputer-Boards der Raspberry Pi Foundation. Dieses Betriebssystem stammt von Debian OS ab, einer der beliebtesten Linux-Distributionen. Auf diesem Betriebssystem können die Studierenden alle Arten von allgemeinen Aufgaben ausführen. Dieses Betriebssystem verfügt über verschiedene Tools, mit denen die Studierenden sich mit Linux und der Programmierung vertraut machen können.

Das Hinzufügen dieser Tools ist für Schüler von Vorteil, die das Programmieren lernen. Es eignet sich speziell zum Erlernen von Python, Java, Scratch und Sonic Pi. Es beinhaltet auch andere Lernoptionen, die sehr hilfreich sein können.
[3]

2.2.2 ULN2003 Treiber Board

Die übliche Art, diesen 28BYJ-48 Motor anzusteuern, ist mit einem ULN2003 Quad – Darlington – Treiber, für den es viele billig verfügbare Breakout Boards (einige mit LEDs, die sehr bequem zu programmieren sind) gibt, die oft mit dem Motor verkauft werden.

Dieser Treiber wird über vier Drähte mit einem Controller wie einem Raspberry Pi verbunden, und die Ansteuerung der einzelnen Schritte wird durch die Software durchgeführt.

Abb.: 13 ULN2003 Treiber Board

2.2.3 Kamera

Die zwei Kameras dienen dazu, Bilder von allen sechs Seiten des Zauberwürfels aufnehmen zu können. Diese zwei Bilder werden an den Raspberry PI gesendet und dieser wertet mittels der Software OpenCV die Farben der Felder aus. Die Studierenden haben sich für die Kamera Logitech C270 entschieden, da sie mit dem Raspberry Pi kompatibel und sehr preiswert ist.

- Maximale Auflösung: 720p/30 FPS
- Kamera-Megapixel: 0.9
- Fokus: Fester Fokus
- Objektiv: Plastik
- Integriertes Mikrofon: Mono
- Mikrofon-Reichweite: Bis zu 1 m
- Diagonales Sichtfeld (FOV): 55°
- Universalhalterung für Notebooks, LCD- und Röhrenmonitore

Abb.: 14 Webcam Logitech C270

2.3 Maschinengerüst

Das Maschinengerüst wurde mit Fusion360 gezeichnet. Das CAD Programm ist nicht sehr intuitiv zu bedienen und benötigte einige Stunden an Eingewöhnungsphase.
Es wurden 18 einzelne Objekte gezeichnet, die nach dem Druck zusammengebaut wurden. Eine Herausforderung war es, das Maschinengerüst so zu designen, dass der Würfel ohne großen Aufwand aus- und wieder eingebaut werden kann.
Zusätzlich wurde versucht die Leitungsführung so weit wie möglich innen zu verlegen. Dieses Unterfangen wurde nicht umgesetzt, denn das Maschinengerüst wurde für Servomotoren mit kleineren Steckern ausgelegt und nicht, wie in der finalen Version, mit Schrittmotoren. Aus Zeitgründen wurde entschieden das Gerüst nicht anzupassen und die Leitungen frei zu verlegen.
Sollte es nicht anders erwähnt werden, sind sämtliche Objekte mit der Stückzahl Eins verbaut.

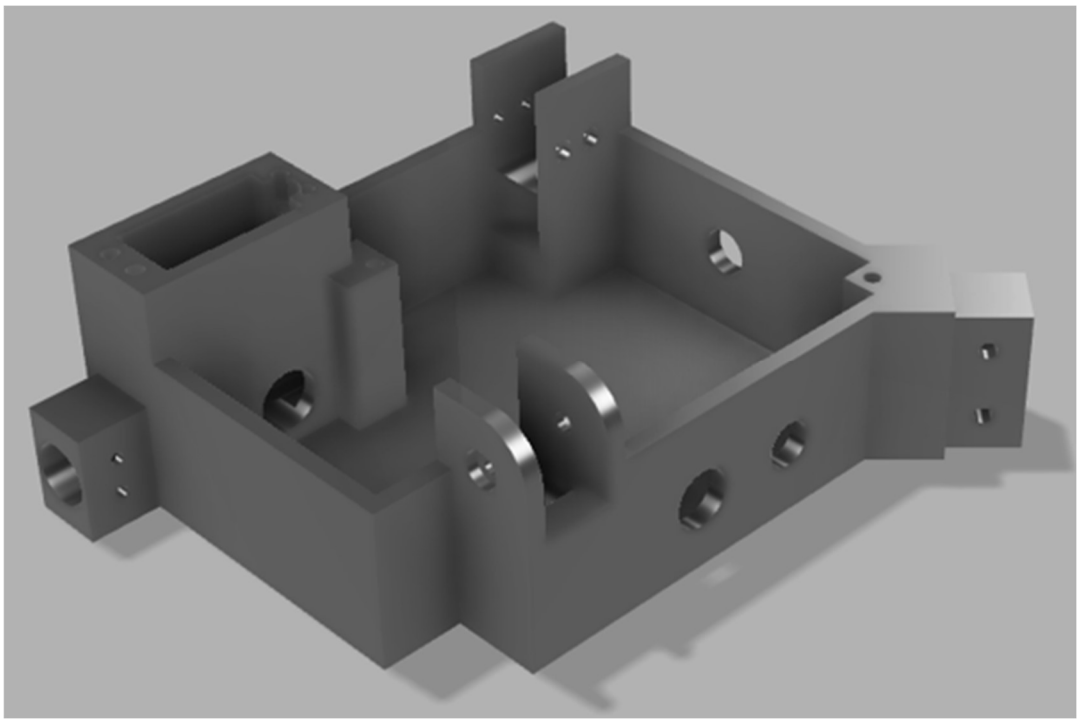


Abb.: 15 Grundplatte

In dieser Abbildung ist die Grundplatte ersichtlich, die ca. ¼ der gesamten Grundfläche einnimmt. Hier laufen alle Leitungen und Drähte zusammen und werden am Raspberry PI angeschlossen, der ebenso in der Grundplatte montiert wird. Die Öffnungen an den Seitenwänden sind Vorbereitungen für die Spannungsversorgung 5V, Spannungsversorgung 12V und START Taster. Auf der Grundplatte sind schon 2 Halterungen für die Säulen vorbereitet. Eine flach, da diese Säule fix stehen bleibt und eine nach außen abgerundet, um das Öffnen des Gerüsts zu ermöglichen.

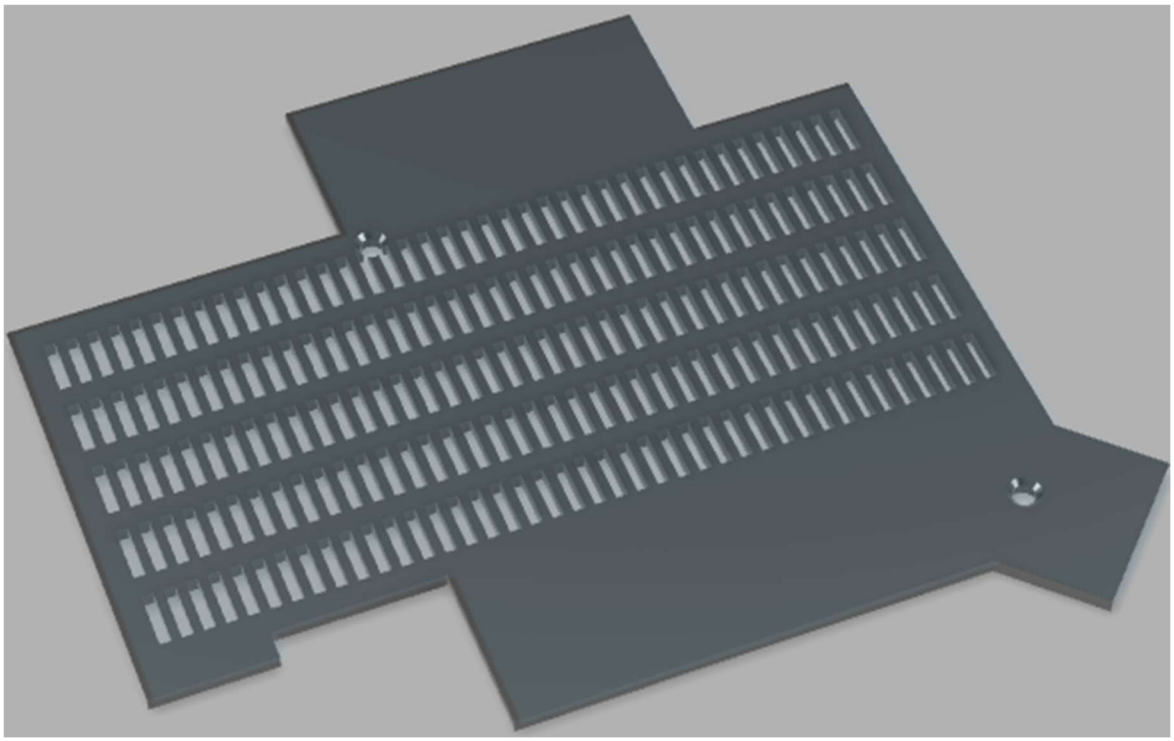
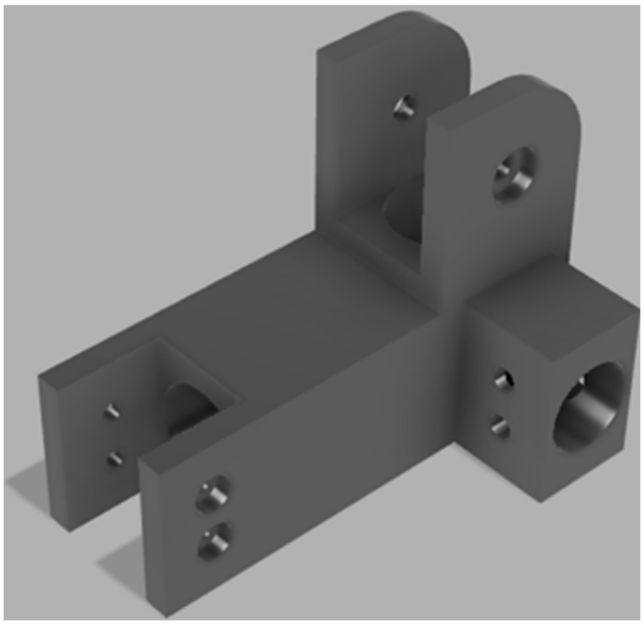


Abb.: 16 Deckel Grundplatte

Hier ersichtlich der Deckel der Grundplatte, versehen mit Lüftungsschlitzen.



Dies ist einer der unteren Seitenarme, der für eine seitliche Montage der Halterung für die untere Kamera vorbereitet ist. Die Abrundung wird im Bild „Grundplatte" erklärt.

Abb.: 17 Seitenarm unten Kamera

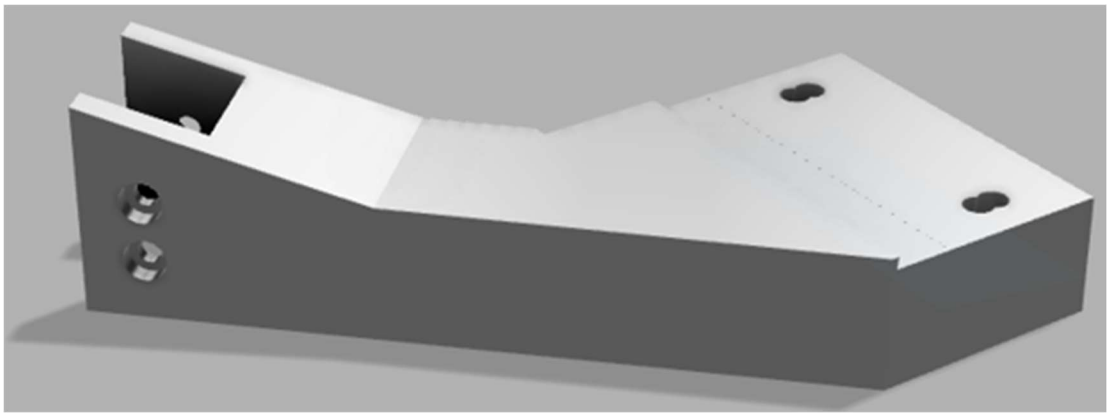
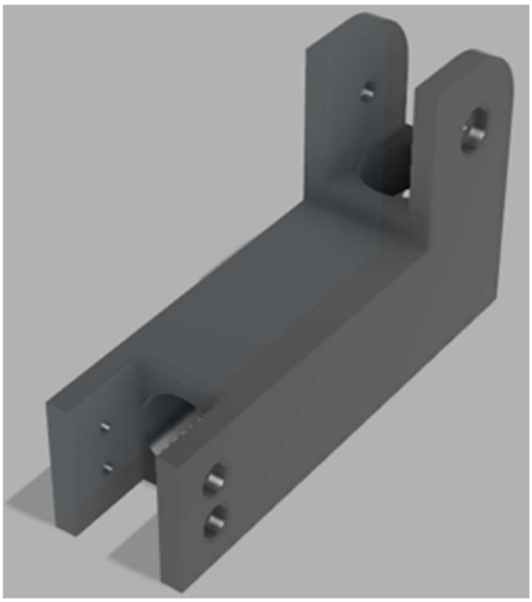


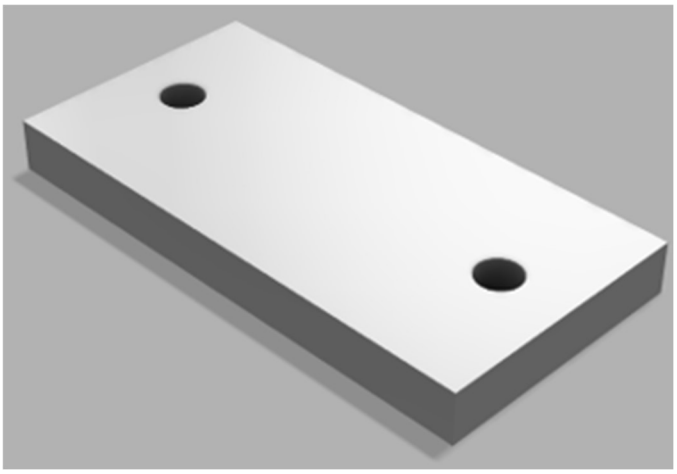
Abb.: 18 Halterung Kamera unten

Hier ersichtlich ist der Seitenarm für die untere Kamera, dieser wird an „*Seitenarm unten Kamera*" seitlich mittels Schrauben M4x30 und Mutter M4 befestigt.



Dies ist ein weiterer Unterarm, außer der Abrundung hat dieser keine weitere Funktion.

Abb.: 19 Seitenarm unten ohne Kamera



Dieses unscheinbare Objekt, wird dafür benötigt, die Kamera am Objekt „*Halterung Kamera unten*"zu befestigen und zu fixieren.
Außerdem wurde dieses Objekt benutzt, um die Position der oberen Kamera anzupassen.

Abb.: 20 Befestigung Kamera unten

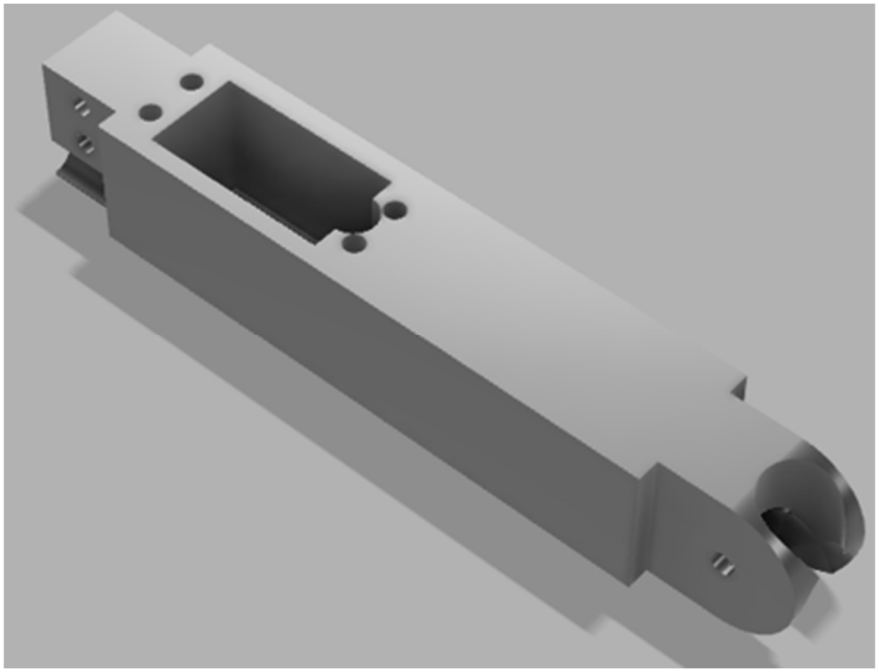


Abb.: 21 Säule beweglich

Diese Säule wird im Gerüst drei Mal benötigt- zwei Mal an den Seitenarmen und einmal an der Grundplatte. Sie bietet Platz für einen Motor und dessen Befestigung. Diese Säule ist am unteren Ende abgerundet, um sie seitlich wegklappen zu können.

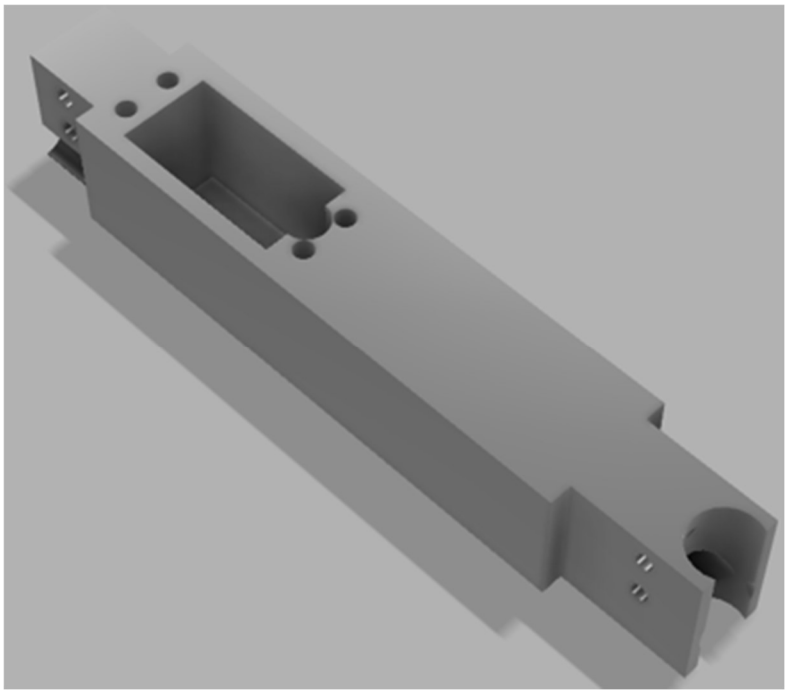


Abb.: 22 Säule fix

Diese Säule bietet Platz für einen Motor und dessen Befestigung. Sie wird an der Grundplatte befestigt.

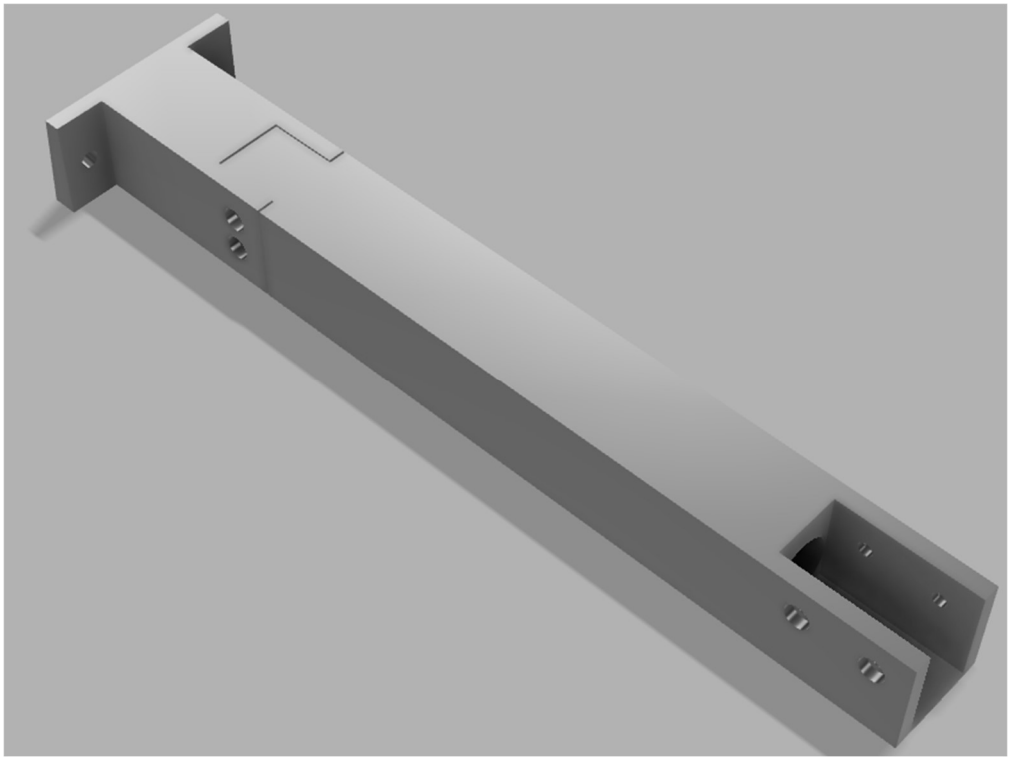
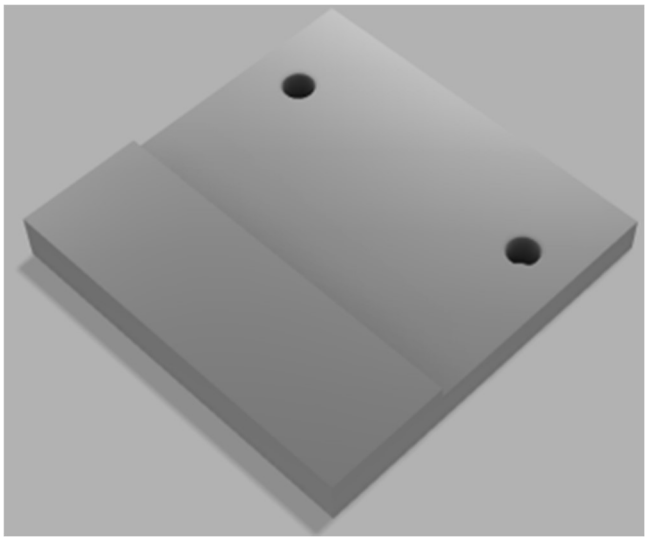


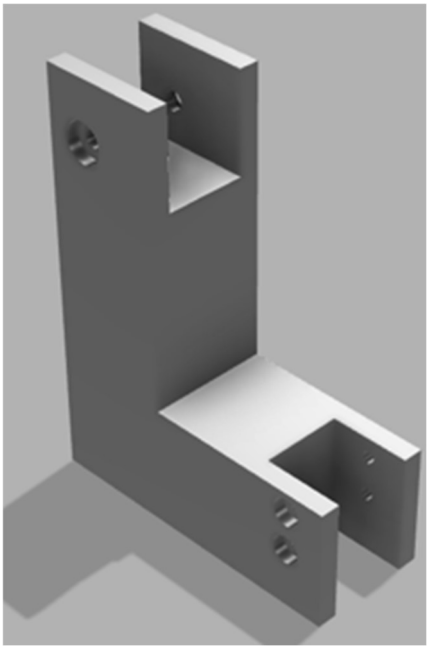
Abb.: 23 Halterung Kamera oben

Für diese Halterung waren 2 Objekte vonnöten, da die Druckfläche und -höhe nicht ausgereicht haben, um dieses Objekt in einem zu drucken.



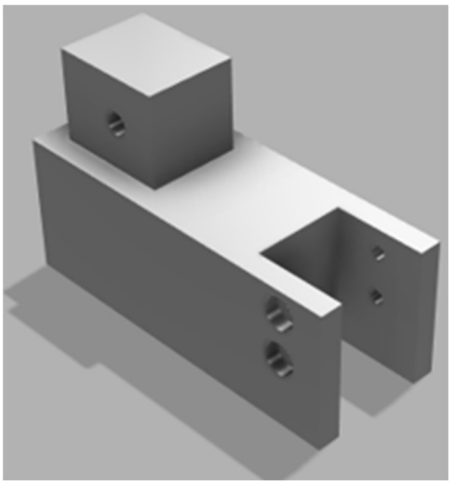
Dieses unscheinbare Stück, wird dafür benötigt, die Kamera am Objekt „*Halterung Kamera oben*"zu befestigen und fixieren .

Abb.: 24 Befestigung Kamera oben



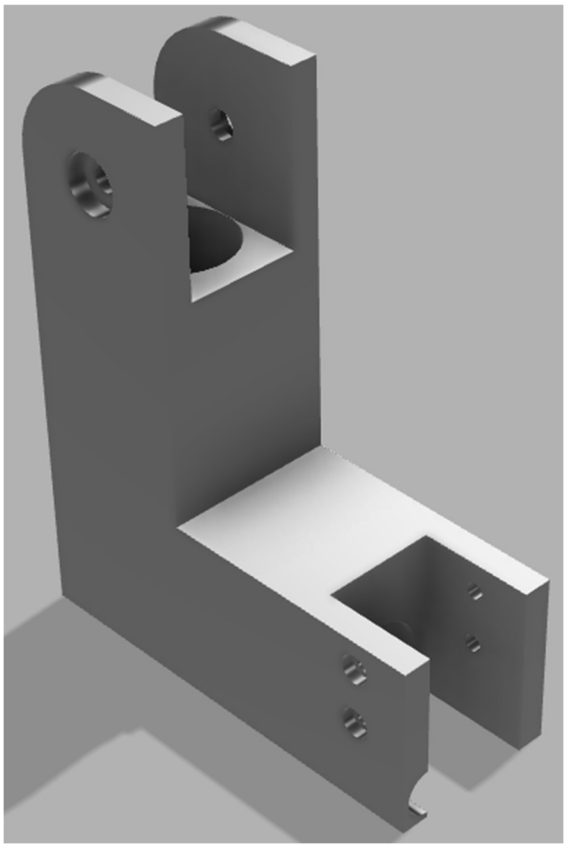
Der Knick dient als Übergang von den senkrechten Säulen zum waagrechten oberen Teil des Gerüsts. Dieser ist mit einer Einkerbung versehen, um in das obere Kreuz einzurasten und mit einer Schraube befestigt zu werden. Dieses Objekt wird zwei Mal verwendet.

Abb.: 25 Knick oben starr



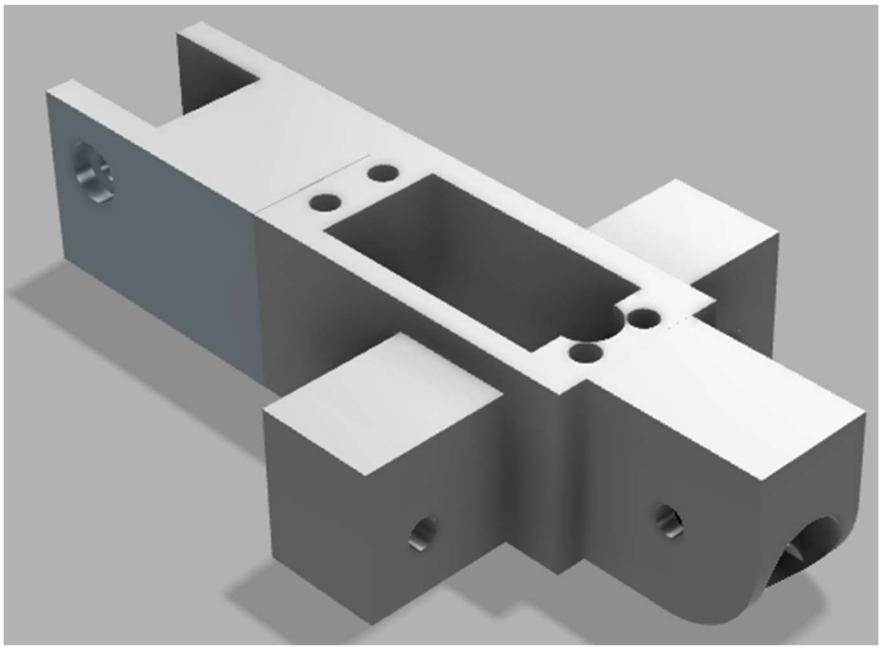
Diese Halterung sitzt gegenüber der starren Säule und hilft bei einer einfachen Montage des Würfels im Inneren, da das Kreuz hier einfach ohne Befestigung abgelegt werden kann.

Abb.: 26 Halterung oben



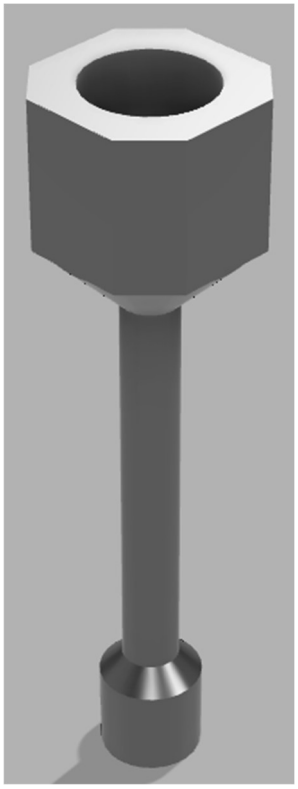
Dieses Objekt wird an der starren Säule montiert und bietet die Möglichkeit, das Kreuz hochzuklappen.

Abb.: 27 Halterung Kreuz



Das Kreuz stellt den „Schlussstein" des Gerüsts dar. Hier laufen alle 4 Säulen zusammen und es bietet Platz für einen Motor und dessen Befestigung. Das Kreuz ist beweglich und an der starren Säule befestigt.

Abb.: 28 Kreuz



Dieses Objekt wird sechs Mal benötigt, pro Motor einmal. Da bei dem verwendeten Zauberwürfel der Mittelstein entfernbar ist, um die Leichtgängigkeit einstellen zu können, wurde dieses Feature verwendet, um die Halterung zu befestigen.
Das breite Ende des Objekts wird in den Würfel gesteckt und das dünnere Ende wird über die Welle des Schrittmotors gesteckt.

Abb.: 29 Halterung Würfel

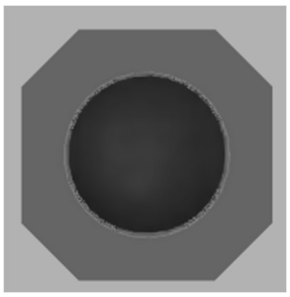


Abb.: 31 Ansicht oben

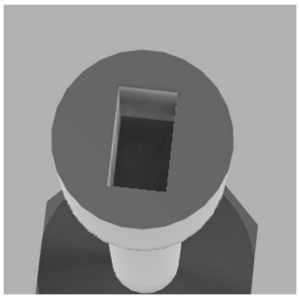
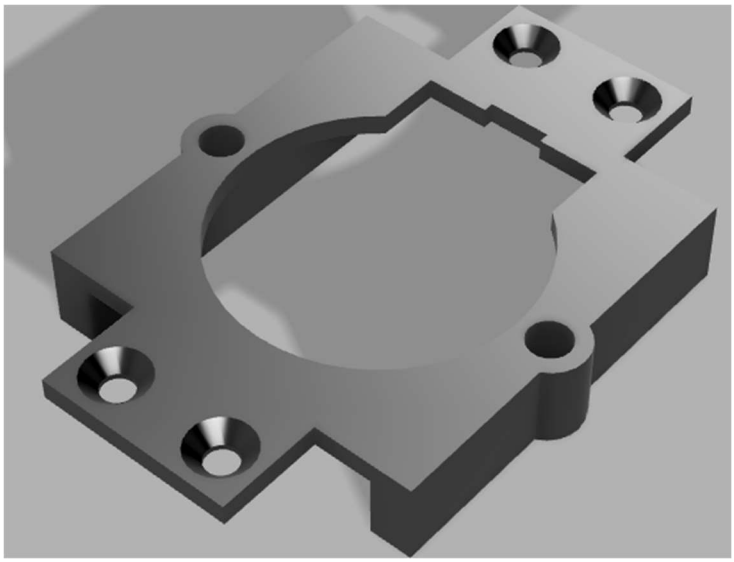


Abb.: 30 Ansicht unten



Diese Adapter waren notwendig, wie unter dem Punkt „2.3 Maschinengerüst" hingewiesen, da andere Motoren kamen als ursprünglich geplant und aus Zeitgründen nicht neu gedruckt wurde. Es werden sechs Stück dieser Adapter benötigt, pro Motor einer.

Abb.: 32 Adapter Motorhalterung

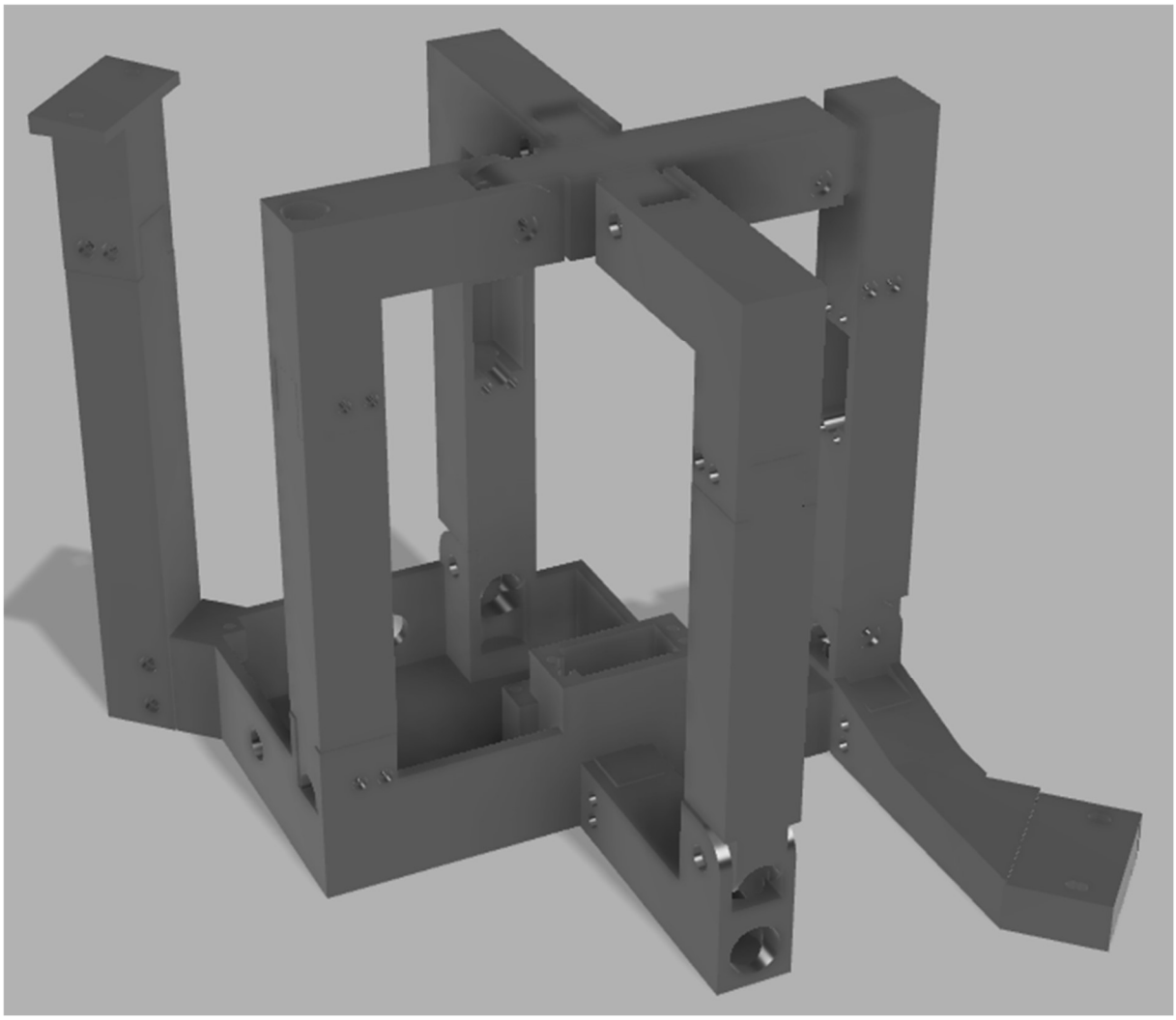


Abb.: 33 Maschinengerüst komplett

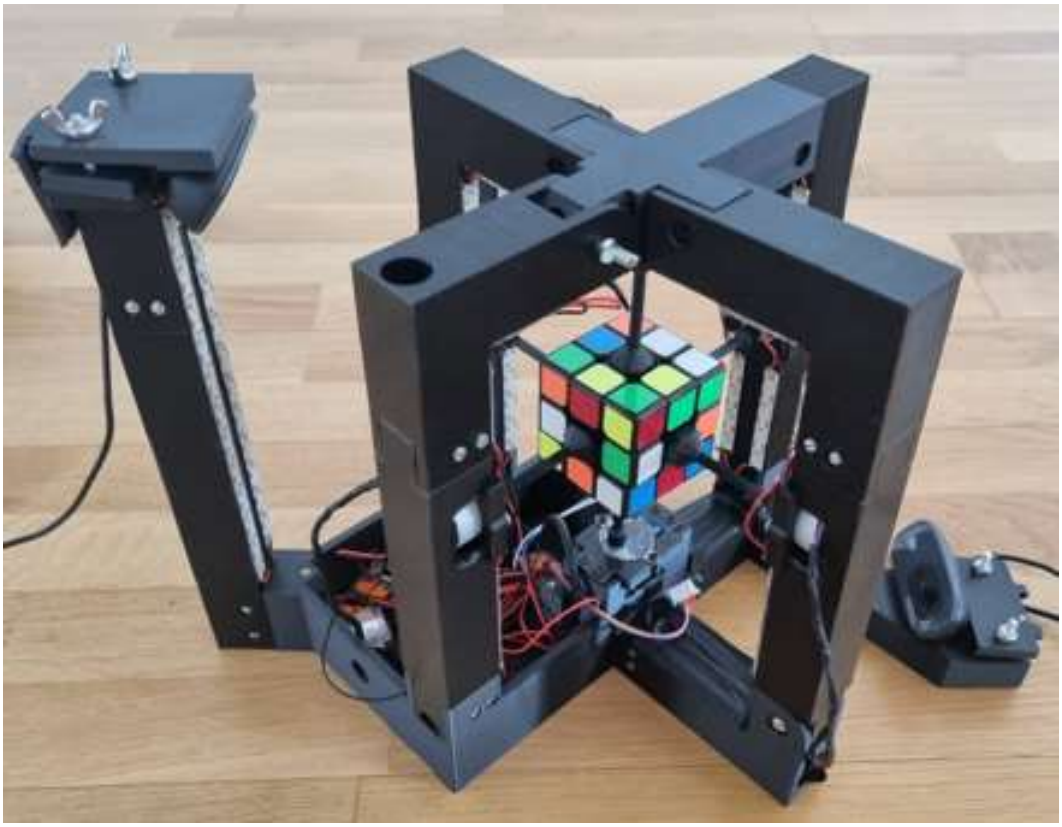


Abb.: 34 Optischer Zauberwürfellöser geschlossen

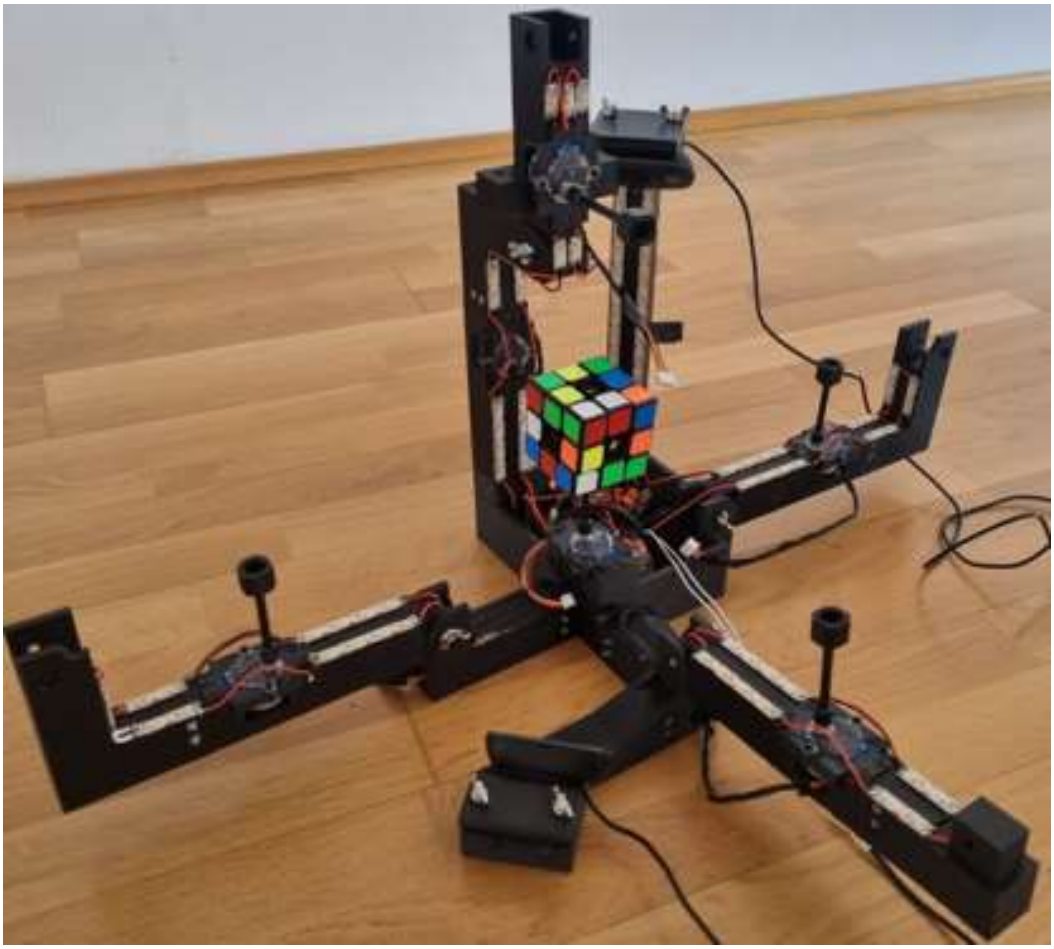


Abb.: 35 Optischer Zauberwürfellöser offen

2.4 3D-Drucker

Aus Kostengründen hat man sich dazu entschieden, dass das Gruppenmitglied, Christoph Orgl, diesen Drucker für private Zwecke kauft, da der Druck von 1cm³ bei diversen Anbietern über 50 Cent lag. Zum Herstellen der Objekte wurde ein 3D-Drucker verwendet, genauer der Creality Ender 3 V2. Dieser Drucker gilt als einer der einsteigerfreundlichsten 3D-Drucker und bietet, laut mehreren Foren, das beste Preis-Leistung Verhältnis.

Abb.: 36 Creality Ender 3 V2

Technische Daten:

- Düse 0,4 mm
- Extruder 1x
- Extruder (max.) 255 °C
- Filament ABS, PLA, PETG, TPU
- Filament Ø 1,75 mm
- Objektgröße (max.) 220 x 220 x 250 mm
- Schichtdicke 0,10 - 0,40 mm

2.5 Beleuchtung

Abb.: 37 LED Band

Es wurden mehrere LED-Bänder auf der Innenseite des Maschinengerüsts angebracht, um eine gleichmäßige Ausleuchtung des Würfels zu garantieren. Dadurch wurde die Fehlerquote der Farberkennung minimiert. Zusätzlich wurden auch Schatten auf dem Würfel verhindert.

Technische Daten:

- Spannung: DC 5V (USB-Netzteil)
- Wasserdicht: IP65 Wasserdicht
- LED-Typ: SMD 3528
- Streifenlänge: 2m/6.56ft (60leds/3.28ft)
- USB-Kabellänge: 1m (3.28ft)
- Lichtfarbe: Warmweiß
- Abstrahlwinkel: 120 Grad
- Lagertemperatur: -40 bis 80 Celsius
- Betriebstemperatur: -25 bis 60 Celsius

2.6 Schaltplan

Der Schaltplan wurden mittels der Software Fritzing gezeichnet. Die Software ist kostenlos im Internet downloadbar. Die Entwicklungsumgebung ist einfach gestaltet. Die ganzen elektronischen Bauteile sind grafisch abbildbar, was das Verstehen der Schaltung vereinfacht.

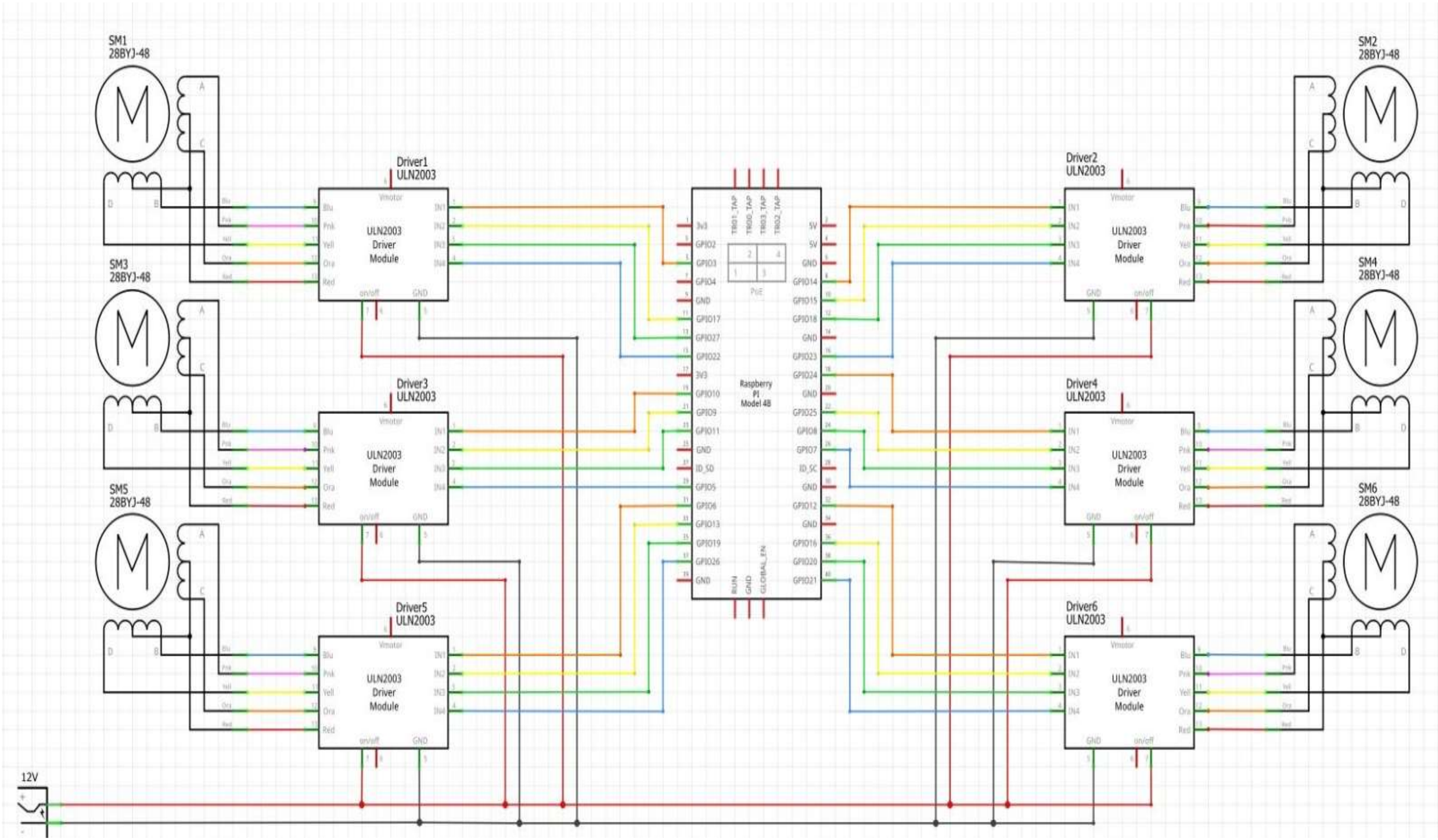


Abb.: 38 Schaltplan Motorsteuerung

3 SOFTWARE

3.1 Python

Python ist eine einfach zu erlernende, aber mächtige Programmiersprache mit effizienten abstrakten Datenstrukturen und einem einfachen, aber effektiven Ansatz zur objektorientierten Programmierung. Durch die elegante Syntax und die dynamische Typisierung ist Python als interpretierte Sprache sowohl für Skripte als auch für schnelle Anwendungsentwicklung hervorragend geeignet.

Der Python-Interpreter und die umfangreiche Standardbibliothek sind als Quelltext und in binärer Form für alle wichtigen Plattformen auf der Webseite http://www.python.org frei verfügbar und können frei weiterverbreitet werden. Auf der gleichen Seite finden sich Distributionen von Drittanbietern, Verweise auf weitere freie Module, Programme und Werkzeuge, sowie zusätzliche Dokumentation.

Der Python-Interpreter kann auf einfache Weise um neue Funktionen und Datentypen erweitert werden, die in C oder C++ (oder andere Sprachen, die sich von C aus ausführen lassen) implementiert sind. Auch als Erweiterungssprache für anpassbare Applikationen ist Python hervorragend geeignet.

3.2 Algorithmus

Der Algorithmus ist, vereinfacht gesagt, eine Vorgehensweise, um den Würfel lösen zu können. Um diesen Algorithmus zu entwickeln, ist es notwendig die Logik des Würfels zu verstehen und umzusetzen. Dazu wählt man, vor der Entwicklung der Software, eine passende Lösungsmethode. Die Entscheidung ist auf eine Mischung aus Anfängermethode und erweiterter Methode gefallen.
Die Wahl der Methoden ist so zu begründen, dass die Anschaulichkeit, die in diesem Projekt gewünscht ist, mit dieser Lösungsvariante gegeben ist. Notationen und Lösungsmethoden werden der Internetseite www.ruwix.com entnommen und auf das Projekt abgestimmt. Die Notation und die Lösung, bestehend aus sechs Schritten und zusätzlich benötigter Software, werden nachfolgend erklärt.

3.2.1 Die Notation

Um den Würfel zu lösen, muss man bestimmte Zugfolgen, sogenannte Algorithmen lernen. Dafür muss vorerst die Notation verstanden werden. Bei der Notation des Würfels handelt es sich um Buchstaben, die bestimmte Bewegungen vorgeben.
Ein Beispiel dafür ist: F, R, U, R‘, U‘, F‘

Denkt man sich die Anführungszeichen ‘ (welche für „inverted“ oder „inverse“ stehen) weg. Kann man erkennen, dass das Beispiel nur aus Buchstaben besteht. Welcher Buchstabe für welche Bewegung steht wird in dieser Tabelle erläutert:

Symbol:	F	B	R	L	U	D
Bedeutung:	Front	Back	Right	Left	Up	Down
Übersetzung:	Vorne	Hinten	Rechts	Links	Oben	Unten

Nach dieser Tabelle ist bei einer Drehung von F die vordere Seite angedacht. Das ist die sogenannte „Front Seite“, deshalb der leicht zu erklärende Buchstabe F. Jede Seite kann auf vier verschiedene Weisen gedreht werden, wobei 180° im Uhrzeigersinn und 180° gegen den Uhrzeigersinn keinen Unterschied bewirken. Deshalb wird in der nachstehenden Tabelle auf F‘2 (180° gegen den Uhrzeigersinn) verzichtet.
Um zu definieren, welche Drehung erfolgen soll, kommt zusätzlich zu den Buchstaben noch etwas hinzu. In der folgenden Tabelle werden die drei Möglichkeiten der Drehung dargestellt:

Symbol:	F	F‘	F2
Bedeutung:	Front	Front inverted	2x Front
Übersetzung:	Im Uhrzeigersinn	Gegen den Uhrzeigersinn	180° Im Uhrzeigersinn

Es wird nur auf projektrelevante Teile der Notation eingegangen.
In diesem Projekt werden die Symbole, die die Drehung gegen den Uhrzeigersinn beschreiben gegen Kleinbuchstaben ersetzt.
F‘ → f, U‘ → u usw.

3.2.2 Die Lösungssoftware

Die Lösungssoftware setzt sich aus insgesamt neun Teilen zusammen. Diese Teile sind in je eine Datei ausgelagert, um die Lesbarkeit des Codes zu vereinfachen. Die Dateien wiederum werden als Module benannt, wobei einige der Module auch Klassen mit demselben Namen beinhalten. Mittels „from – Importanweisung“, werden die Module wieder in die nötigen Dateien eingebunden. Auch benötigte Bibliotheken werden mit dieser Anweisung der Lösungssoftware hinzugefügt.
Die hier bezeichneten Module werden der Einfachheit halber nicht mit dem ganzen Namen, lokale Module, benannt. Der Unterschied zwischen lokalen Modulen und Bibliotheken liegt in der Anwendung dafür.
Bibliotheken stellen Datentypen oder Funktionen für sämtliche Python-Programme bereit.
Wobei lokale Module nur für ein Programm verfügbar sind.

Die neun Module werden in folgender Reihenfolge näher betrachtet:

1. Move
2. Set Surface
3. First Layer Edges
4. First Layer Corners
5. Second Layer
6. OLL (Orient the last layer)
7. PLL Edges (Permutate the last layer Kanten)
8. PLL Corners (Permutate the last layer Ecken)
9. Cube Solve

Bevor die Module näher erklärt werden, werden grundlegende Komponenten der Software erklärt, um das Programm besser zu verstehen.

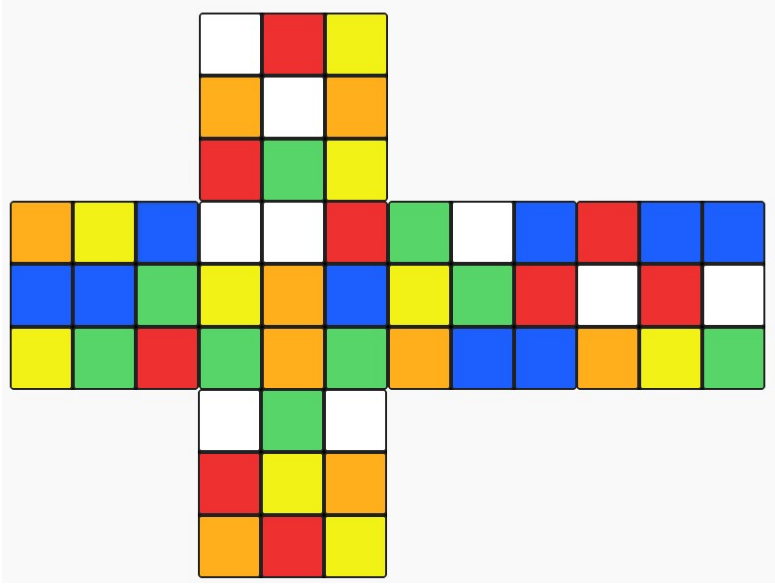


Abb.: 39 Mantelfläche Zauberwürfel

```
          [3, 1, 0]
          [1, 0, 2]
          [5, 2, 4]
[5, 2, 1] [4, 0, 0] [3, 3, 4] [1, 4, 2]
[1, 1, 4] [3, 2, 0] [3, 3, 4] [0, 4, 0]
[5, 1, 2] [0, 2, 4] [5, 3, 2] [1, 4, 2]
          [3, 5, 3]
          [5, 5, 5]
          [1, 5, 0]
```

Abb.: 40 Mantelfläche codiert

In der Software wird der Würfel als sechs 3x3 Listen dargestellt. Jede Liste beinhaltet neun Zahlen, die je einer Farbe zugewiesen sind. Zur Veranschaulichung dienen die Beispiel Bilder der Oberfläche des Würfels.
Die Farben, von null aufsteigend, sind:
0 → Gelb, 1 → Orange, 2 → Blau, 3 → Rot, 4 → Grün, 5 →Weiß

Bei der Lösung wird immer darauf geachtet, dass der Würfel an der Front betrachtet wird. Von dort ausgehend sind die Seiten wie folgt definiert:
Gelb → Oben, Orange → Links, Blau → Vorne, Rot → Rechts, Grün → Hinten,
Weiß → Unten
Auch zu beachten ist, dass die Mittelsteine einer jeden Seite, die nur eine Farbe haben, die Farbe der Seitenfläche definieren, Kantensteine 2 Farben und Ecksteine 3 Farben haben. In der Software werden Zahlen untereinander vertauscht, um den gewünschten Zustand zu erzielen. In dieser Dokumentation wird dieses Tauschen auch Drehung genannt.

3.2.2.1 Move

In diesem Modul wird die Drehung des Würfels bearbeitet. Es beinhaltet die gleichnamige Klasse „Move".
Um in der Klasse die Drehung des Würfels zu verwirklichen, müssen ein Modul und eine Bibliothek eingebunden werden.

```
1  from copy import deepcopy
2  from stepper import *
```

Abb.: 41 Code Move: Import

Das Modul „stepper" wird in dem Kapitel 3.3 Motorsteuerung genauer behandelt und aus der Bibliothek „copy" wird die Methode „deepcopy" importiert.
Die Methode „deepcopy" wird benutzt, um eine sogenannte tiefe Kopie zu erstellen. Dies wird verwendet, um sicherzustellen, dass wie in diesem Projekt eine Liste kopiert wird und die Kopie bei Veränderung der Ursprungsliste nicht betroffen ist. Würde man die Methode „copy" dafür verwenden, würde auch die Kopie mitverändert werden.

```
25  class Move:
26
27
28      def __init__(self, u, l, f, r, b, d):
29          """Initialisierung der einzelnen Seiten"""
30          self.u = u
31          self.f = f
32          self.l = l
33          self.r = r
34          self.b = b
35          self.d = d
```

Abb.: 42 Code Move: Initialisierungsschritt

In den Zeilen 28 – 35 werden die einzelnen Seiten initialisiert. Das bedeutet, dass die Seiten als Argument dem „__init__" übergeben werden und mit einem Objekt aufgerufen werden können. „self" steht für das Objekt selbst und wird nur in der dazugehörigen Klasse verwendet.

Die Klasse „Move" beinhaltet zusätzlich noch 24 Methoden, um die Drehung des Würfels zu verwirklichen. Sechs dieser Methoden sind der Vollständigkeit halber, falls der Code zukünftig verändert werden sollte, enthalten und werden in den Lösungsschritten nicht verwendet. Das sind die Methoden zur Drehung 180° gegen den Uhrzeigersinn. Die Methoden sind in sechs Gruppen aufgeteilt, die jeweils 4 Methoden enthalten.
Es wird nur eine Gruppe näher betrachtet, wobei auf die Drehungen gegen den Uhrzeigersinn nicht näher eingegangen wird, da diese selbsterklärend sind.

```
35      def do_U(self):
36          """obere Seite wird um 90° im Uhrzeigersinn gedreht"""
37
38          temp_u = deepcopy(self.u)
39          temp_l = deepcopy(self.l)
40          temp_f = deepcopy(self.f)
41          temp_r = deepcopy(self.r)
42          temp_b = deepcopy(self.b)
43          temp_d = deepcopy(self.d)
```

Abb.: 43 Code Move: Methode do_U (1)

Wie auf dem Bild beschrieben wird in der Zeile 35 die Methode zur Drehung der oberen Seite um 90° im Uhrzeigersinn beschrieben.
Es wird in den Zeilen 38 - 42 eine Kopie aus jeder Seite erstellt, wobei die Seite „d" bei dieser Drehung nicht verwendet wird.

```
45          self.u[0][2] = temp_u[0][0]
46          self.u[2][2] = temp_u[0][2]
47          self.u[2][0] = temp_u[2][2]
48          self.u[0][0] = temp_u[2][0]
49
50          self.u[0][1] = temp_u[1][0]
51          self.u[1][2] = temp_u[0][1]
52          self.u[2][1] = temp_u[1][2]
53          self.u[1][0] = temp_u[2][1]
54
55          self.l[0][0] = temp_f[0][0]
56          self.l[0][1] = temp_f[0][1]
57          self.l[0][2] = temp_f[0][2]
```

Abb.: 44 Code Move: Methode do_U (2)

Durch die Zuweisungen der Zeilen 45 – 48 werden die Ecksteine der oberen Seite vertauscht. Dazu muss man wissen, dass z. B. der Aufruf „self.u[0][2]" ein einzelnes Element über Indizes (Zahlen in der eckigen Klammer) in einer Liste anspricht. Wie bekannt, besteht eine Seite aus einer 3x3 Liste und [0][2] bedeutet, dass auf das zweite Element aus der nullten Liste zugegriffen wird. Wenn man die Seite von oben betrachtet, wird in Zeile 45 der linke obere Eckstein gegen den rechten oberen Eckstein ausgetauscht.
Die Kantensteine der oberen Seite werden in den Zeilen 50 – 53 ausgetauscht.
Von Zeile 55 – 57 werden alle linken Einzelflächen mit den dazugehörigen vorderen Einzelflächen ersetzt.

```
59          self.f[0][0] = temp_r[0][0]
60          self.f[0][1] = temp_r[0][1]
61          self.f[0][2] = temp_r[0][2]
62
63          self.r[0][0] = temp_b[0][0]
64          self.r[0][1] = temp_b[0][1]
65          self.r[0][2] = temp_b[0][2]
66
67          self.b[0][0] = temp_l[0][0]
68          self.b[0][1] = temp_l[0][1]
69          self.b[0][2] = temp_l[0][2]
70
71          RIGHT_TURN(90, 0)
```

Abb.: 45 Code Move: Methode do_U (3)

Die Zeilen 59 – 61, 63 -65 und 67 – 69 beschreiben den Austausch, wie oben erklärt, für die Seiten vorne, rechts und hinten.
Der Aufruf der Methode Right_Turn (90, 0) dient zur Ansteuerung eines Schrittmotors und wird im Kapitel 3.3 Motorsteuerung näher erläutert.

```
83      def do_2U(self):
84          """obere Seite wird um 180° im Uhrzeigersinn gedreht"""
85
86          Move.do_U(self)
87          Move.do_U(self)
```

Abb.: 46 Code Move: Methode do_2U

Die Methode „def do_2U(self)" ist die Methode, um die obere Seite 180° im Uhrzeigersinn zu drehen, indem die Methode zur Drehung von 90° im Uhrzeigersinn 2-mal aufgerufen wird.

3.2.2.2 Set_Surface

```
1   class SetSurface:
2       def __init__(self, p00, p01, p02, p10, p11, p12, p20, p21, p22):
3           """Die Oberfläche der Seiten wird, in einer 3x3 Liste, initialisiert"""
4           self.p = [[p00, p01, p02],
5                     [p10, p11, p12],
6                     [p20, p21, p22]]
```

Abb.: 47 Code Set_Surface: Initialisierung

Das Modul mit gleichnamiger Klasse Set Surface dient zum Erstellen eines Objektes für die 3x3 Liste je Würfelseite. Dieses Modul besteht nur aus der Klasse und dem dazugehörigen Initiator. In der Zeile 2 ist die besondere Methode __init__ mit den Übergabeparametern definiert.
In den Zeilen 4 – 6 wird die 3x3 Liste, bestehend aus den übergebenen Argumenten, dem Objekt zugeordnet. Die Objekte werden im Schritt „First Layer Edges" angelegt.

3.2.2.3 First_Layer_Edges

Der erste Schritt „First Layer Edges" bedeutet, dass die Kantensteine der weißen Fläche gelöst werden. Der Würfel wird immer von vorne betrachtet. Aus der Notation ist bekannt, dass die vordere Fläche die Farbe Blau hat. Ausgehend dieser Situation werden die Schritte mit den einzelnen Bewegungen beschrieben. In diesem Schritt werden auch die Objekte von Set Surface und Move angelegt.

```
1   from Move import Move
2   from Set_Surface import SetSurface
3   from Kamera_Unten import *
4   from Kamera_Oben import *
5
```

Abb.: 48 Code First_Layer_Edges: Import

In den Zeilen 1- 4 werden alle benötigten Klassen und Module importiert. Die zwei Module Kamera unten und Kamera oben werden im Abschnitt Farberkennung näher erläutert.

```
33  f = SetSurface(allcolorsnew[19][2], allcolorsnew[16][2], allcolorsnew[11][2],
34                 allcolorsnew[20][2], BLUE, allcolorsnew[12][2],
35                 hiddenF, allcolorsnew[17][2], allcolorsnew[13][2])
36  r = SetSurface(allcolorsnew[7][2], allcolorsnew[5][2], allcolorsnew[1][2],
37                 allcolorsnew[8][2], GREEN, allcolorsnew[2][2],
38                 allcolorsnew[9][2], allcolorsnew[6][2], hiddenR)
```

Abb.: 49 Code First_Layer_Edges: anlegen der Objekte r und f

Der Ausschnitt der Zeilen 33 – 38 zeigt das Anlegen von Objekten der Klasse SetSurface mit den dazugehörigen Argumenten. Die Argumente sind Methoden, mit denen die Zahlen für die Farben ermittelt und aus den Modulen Kamera unten und Kamera oben übergeben werden. Die Objekte werden nach den Seiten benannt, um die Lesbarkeit des Codes zu vereinfachen.

```
47  M = Move(u.p, l.p, f.p, r.p, b.p, d.p)
```

Abb.: 50 Code First_Layer_Edges: anlegen des Objektes M

Es wird in der Zeile 47 ein Objekt aus der Klasse Move mit dem Namen M angelegt. Dieses Objekt dient dazu, um in allen weiteren Lösungsschritten die einzelnen Drehungen auszuführen.

```
53  class FirstLayerEdges:
54
55      def solve_blue_white_front(self):
56          """Lösung blau/weißer Kantenstein vorne"""
57          if M.f[1][2] == 5 and M.r[1][0] == 2:  # blue/white, right edge, white on frontside
58              M.do_R()
59              M.do_U()
60              M.do_2F()
```

Abb.: 51 Code First_Layer_Edges: Methode solve_blue_white_front

Die Zeile 55 definiert eine Methode, die eine Reihenfolge von Drehungen ausführen soll, um den weiß/blauen Kantenstein in die richtige Position zu drehen. Mit der if Abfrage in der Zeile 57 wird überprüft, ob sich ein weißer Kantenstein an der rechten vorderen Seite befindet. Ist das der

Fall werden mit dem Objekt M, in den Zeilen 58 – 60, die Methoden der Klasse Move aufgerufen. Sollte sich der Stein nicht an dieser Stelle befinden, wird die nächste mögliche Stelle überprüft. Die Methode in Zeile 55 wiederholt sich in abgeänderter Form für alle möglichen Positionen der vier weißen Kantensteine. Es wird lediglich eine andere Position abgefragt und es sind andere Verdrehungen nötig, um die Kantensteine an ihre richtige Position zu setzen.

```
160     def blue_white(self):
161         """zusammenfassen aller Methoden zu nur einer Methode"""
162         FirstLayerEdges.solve_blue_white_front(self)
163         FirstLayerEdges.solve_blue_white_left(self)
164         FirstLayerEdges.solve_blue_white_right(self)
165         FirstLayerEdges.solve_blue_white_back(self)
166         FirstLayerEdges.solve_blue_white_up(self)
167         FirstLayerEdges.solve_blue_white_down(self)
```

Abb.: 52 Code First_Layer_Edges: Methode blue_white

Die Methoden für die Drehungen des blau/weißen Kantensteines werden hier zu einer Methode zusammengefasst. Dies wird auch mit den Methoden der verbliebenen drei weißen Kantensteinen gemacht.

```
504 def first_layer_edge_solve():
505     """zusammenfassen aller Methoden, zur Lösung der einzelnen Steine,
506      zu nur der Methode die im Cube_Solve aufgerufen wird"""
507     edge = FirstLayerEdges()
508 
509     edge.blue_white()
510     edge.orange_white()
511     edge.red_white()
512     edge.green_white()
```

Abb.: 53 Code First_Layer_Edges: Methode first_layer_edge_solve

In der Zeile 504 wird die Lösungsmethode definiert, die im Modul „Cube_Solve", aufgerufen wird. Diese Methode befindet sich außerhalb der Klasse „FirstLayerEdges", deshalb muss ein Objekt der Klasse angelegt werden, um die darin enthaltenen Methoden zur Lösungsmethode „first_layer_edge_solve()" zusammenzuführen.

```
514  def print_result():
515      """Methode um die Lösung im Terminal, als Mantel des Würfels, auszugeben"""
516
517      print("         ", M.u[0]), print("         ", M.u[1]), print("         ", M.u[2])
518      print(M.l[0], M.f[0], M.r[0], M.b[0])
519      print(M.l[1], M.f[1], M.r[1], M.b[1])
520      print(M.l[2], M.f[2], M.r[2], M.b[2])
521      print("         ", M.d[0])
522      print("         ", M.d[1])
523      print("         ", M.d[2])
```

Abb.: 54 Code First_Layer_Edges: Methode print_result

Die Methode „print_result()" dient zur softwareseitigen Ausgabe der Lösung des Würfels. Dies wird auch zur Überprüfung der einzelnen Schritte verwendet

3.2.2.4 First_Layer_Corners

In diesem Schritt werden die weißen Ecksteine in ihre richtige Position gebracht. Die Ecksteine besitzen drei Farben und es muss bei diesem Schritt darauf geachtet werden, dass die davor gelösten Kantensteine nicht aus ihrer derzeitigen Position gedreht werden. Ist diese Aktion ausgeführt, sollte die weiße Fläche gelöst sein.

```
1  from First_Layer_Edges import *
```

Abb.: 55 Code First_Layer_Corners: Import

In diesem Modul wird nur mehr der letzte Schritt importiert, denn in den übrigen Modulen werden alle zuvor importierten Schritte auch mitübergeben. Auch in den noch zu verbleibenden Schritten wird immer nur der davor benötigte Schritt importiert.

```
10     def blue_white_orange_front(self):
11         """Lösung blau/weiß/oranger Eckstein vorne"""
12
13         if M.f[0][0] == 5 and M.u[2][0] == 2 and M.l[0][2] == 1:
14             M.do_u()
15             M.do_l()
16             M.do_U()
17             M.do_L()
18
19         elif M.f[2][0] == 5 and M.l[2][2] == 2 and M.d[0][0] == 1:...
```

Abb.: 56 Code First_Layer_Corners: blue_white_orange_front

Ähnlich wie bei „First_Layer_Edges" werden auch hier Methoden ausgeführt, die überprüfen, ob sich der gewünschte Eckstein an einer bestimmten Position befindet.
Der einzige Unterschied ist, dass bei der if Abfrage drei Positionen überprüft werden müssen, da es sich um einen Eckstein mit drei Farben handelt.

Auch in diesem Schritt werden die Methoden zu einer Lösungsmethode zusammengefasst.
Alle nachfolgenden Schritte, außer der Lösungsschritt „Cube_Solve" sind ähnlich aufgebaut und werden deshalb nur bei gravierenden Unterschieden näher erklärt.

3.2.2.5 Second_Layer

Bei diesem Vorgang werden alle Kantensteine, die kein gelb oder weiß enthalten, in ihre Position gebracht. Da sich der Code nur durch die Angaben der if Abfrage zu „First_Layer_Edges" unterscheidet wird nicht näher darauf eingegangen.

3.2.2.6 OLL

Dieser Schritt wird verwendet, um die gelbe Fläche zu lösen. Dazu müssen die Positionen der gelben Einzelflächen überprüft werden und mit allen möglichen Mustern, die sich aus den vorhergehenden Schritten ergeben, verglichen werden.

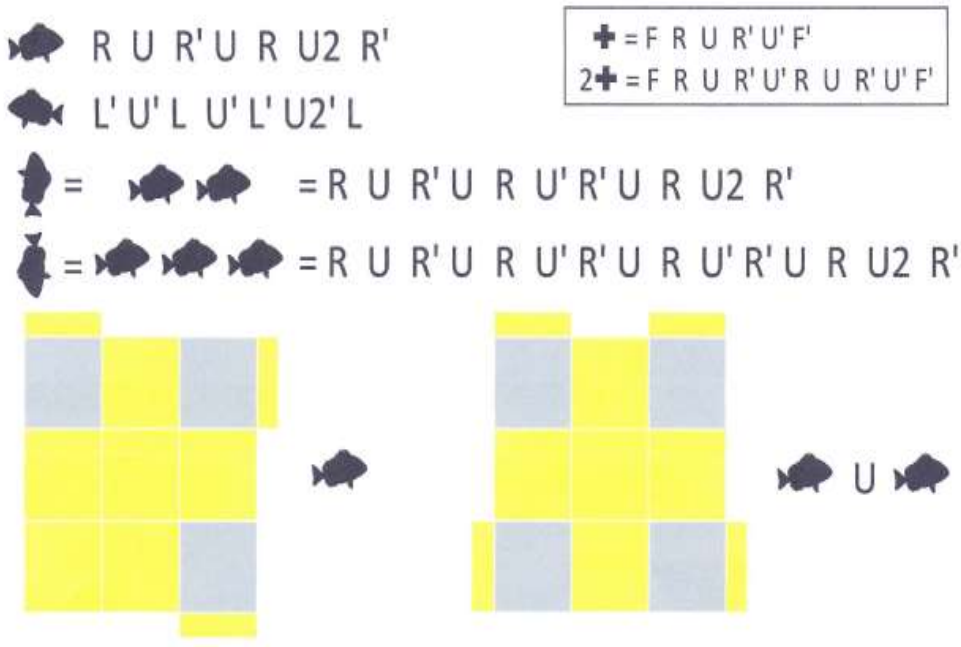


Abb.: 57 Lösungsmuster OLL

Das Bild zeigt zwei dieser möglichen Muster und die dazugehörigen Zugfolgen.

```
378     def solve_fish_left(self):...
427
428     def solve_fish_right(self):...
479
480     def solve_cross(self):...
```

Abb.: 58 Code OLL: Ausschnitt von Methoden

Die Methoden sind passend zu den Mustern benannt. Insgesamt gibt es neun Möglichkeiten. Dadurch, dass der Würfel im Programm immer von vorne betrachtet wird, muss beim Code auch die Drehung des Musters beachtet werden. Dies führt zu Erweiterung des Codes.

```
845 def oll_solve():
846     oll = OLL()
847
848     solved = False
849
850     while not solved:
851         oll.solve_point()
852         oll.solve_corner()
853         oll.solve_line_horizontal()
854         oll.solve_line_vertical()
855         oll.solve_fish_left()
856         oll.solve_fish_right()
857         oll.solve_cross()
858         oll.solve_eight()
859         oll.solve_spaceinvader()
860
861
862         if M.u[0][0] == 0 and M.u[0][1] == 0 and M.u[0][2] == 0 \
863                 and M.u[1][0] == 0 and M.u[1][1] == 0 and M.u[1][2] == 0 \
864                 and M.u[2][0] == 0 and M.u[2][1] == 0 and M.u[2][2] == 0:
865             solved = True
```

Abb.: 59 Code OLL: Methode oll_solve

Da bei „OLL" die Möglichkeit besteht, dass die Methoden mehrfach durchlaufen werden müssen, ist in der Zeile 850 eine while Schleife angelegt, die so lange durchläuft, bis die Variable solved True ist. Die boolsche Variable solved wird in der Zeile 848 angelegt und auf False gesetzt. Erst wenn die gelbe Fläche gelöst ist, wird die if Abfrage, die sich in der Schleife bei Zeile 862 befindet, abgearbeitet und solved wird auf True gesetzt und das Programm beendet die Schleife.

3.2.2.7 PLL_Edges

Der Vorgang dient dazu die Kantensteine mit gelber Teilfläche in ihre Position zu drehen.

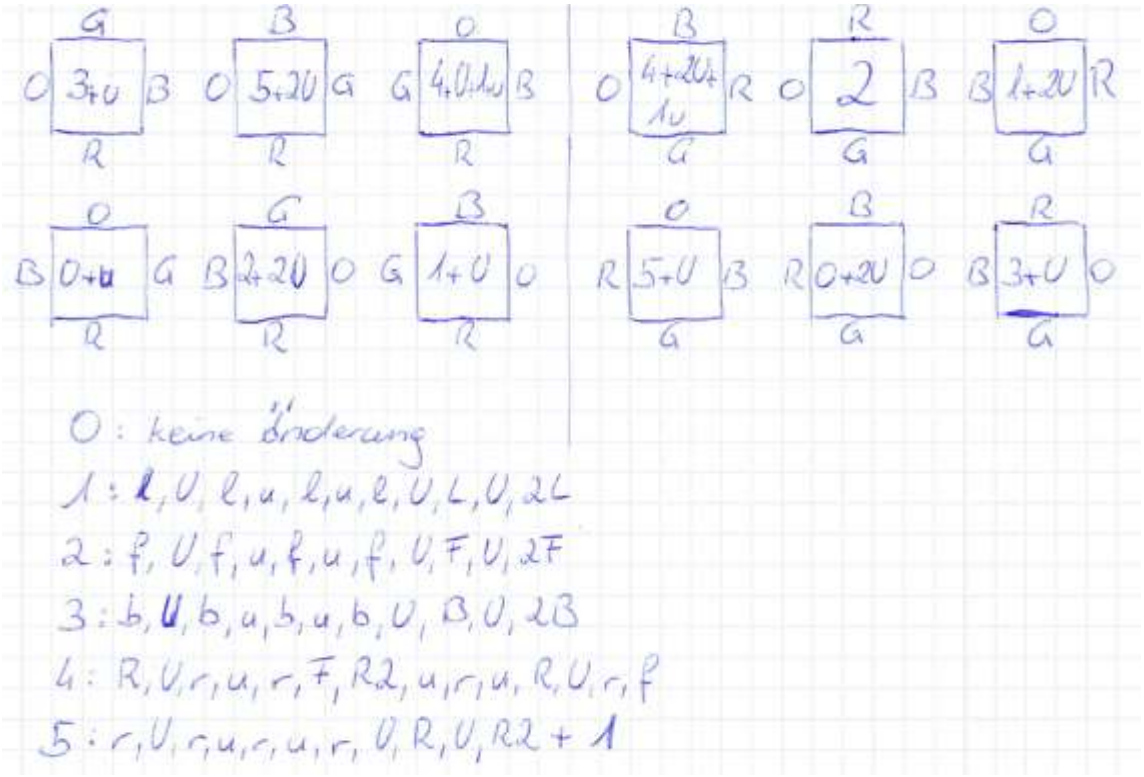


Abb.: 60 Lösungsmuster PLL_Edges

Auf dem Bild ist die händische Dokumentation dieses Vorgangs zu sehen. Es wurde eine Skizze mit allen möglichen Stellungen des Würfels und die dazugehörigen Zugreihenfolge, um diesen Schritt zu lösen, erarbeitet. Mit dieser Skizze, als Hilfestellung, wurde das Modul „PLL" geschrieben.

```
176     def red_front_solve(self):
177         """Lösung wenn rot/gelber Kantenstein vorne"""
178         if M.f[0][1] == 3 and M.l[0][1] == 1 and M.b[0][1] == 4 and M.r[0][1] == 2:
179             M.do_u()
180             M.do_b()
181             M.do_U()
182             M.do_b()
183             M.do_u()
184             M.do_b()
185             M.do_u()
186             M.do_b()
187             M.do_U()
188             M.do_B()
189             M.do_U()
190             M.do_2B()
```

Abb.: 61 Code PLL_Edges: Methode red_front_solve

In den Zeilen 176 – 190 ist ein Teil der händischen Dokumentation als Code programmiert.

3.2.2.8 PLL_Corners

„PLL_Corners" beschreibt den letzten Lösungsschritt des Würfels. Hier werden alle gelben Ecksteine in ihre richtige Lage gebracht. Auch hier wurde eine händische Skizze erarbeitet.

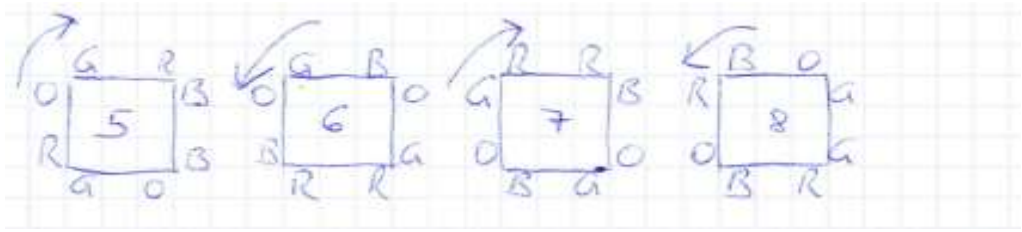


Abb.: 62 Lösungsmuster PLL_Corners(1)

Das Bild zeigt einen Ausschnitt von vier möglichen Würfelzuständen. Die Buchstaben an den Seiten geben die Farbe vor und der Pfeil die Richtung, in die gedreht werden muss.

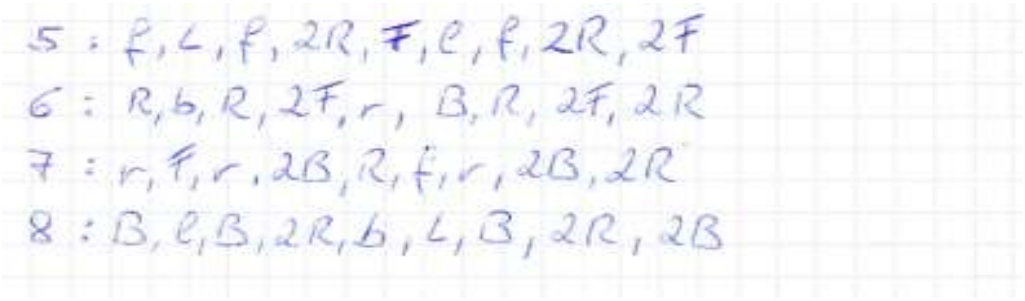


Abb.: 63 Lösungsmuster PLL_Corners(2)

Hier sind die nötigen Züge aufgeschrieben, um den Würfel zu lösen.
Die Zahlen passen mit den Zahlen des letzten Bild zusammen. Mit dieser Dokumentation wurde die Programmierung erleichtert.

```
6   class PLLCorners:
7
8       def solve_blue_red(self):
9           """Lösung wenn blau/rot/gelber Eckstein gelöst"""
10
11          if M.f[0][2] == 2 and M.r[0][0] == 3 and M.r[0][2] == 1 and M.b[0][0] == 2:...
21
22          elif M.f[0][2] == 2 and M.r[0][0] == 3 and M.r[0][2] == 4 and M.b[0][0] == 1:...
```

Abb.: 64 Code PLL_Corners: Methode solve_blue_red

Der Code ist fast ident zu den vorhergegangenen Programmzeilen der anderen Schritte und wird deshalb nicht näher erklärt.
Der Würfel sollte nach diesem Schritt gelöst sein.

3.2.2.9 Cube_Solve

Das Modul „Cube_Solve" wurde erstellt, um die Lösungsmethoden aller Module, die Lösungsschritte enthalten, auszuführen.

```
1    from PLL_Corners import *
```

Abb.: 65 Code Cube_Solve: Import

In Zeile 1 wird der letzte Lösungsschritt importiert und somit auch alle vorhergehenden Module und Klassen.

```
4    def solve_cube():
5        """Methode um die Methoden, zum Lösen des Würfels ,aufzurufen"""
```

Abb.: 66 Code Cube_Solve: Methode solve_cube

Es ist eine Methode erstellt worden, um die einzelnen Lösungsmethoden aufzurufen und eine Kontrollabfrage zu machen, ob in den Einzelschritten Logikfehler auftreten.

```
17           if M.d == [[5, 5, 5], [5, 5, 5], [5, 5, 5]]:
18               print("white face solved!")
19               print("----------------------------------------")
20               second_layer_solve()
21               print_result()
```

Abb.: 67 Code Cube_Solve: Abfrage weiße Seite (1)

Die Zeilen 17 – 21 zeigen einen Ausschnitt der Fehlerabfrage. Ist der Lösungsschritt erfolgreich, wird in der Konsole ein Text ausgegeben, der Mantel des Würfels mit print_result() ausgegeben, und es wird der nächste Schritt gestartet.

```
71       else:
72           print("stop: there is something wrong in first layer edge!!!")
```

Abb.: 68 Code Cube_Solve: Abfrage weiße Seite (2)

Ist jedoch ein Fehler aufgetreten so wird, wie in Zeile 71 und 72 ersichtlich, eine Fehlermeldung ausgegeben, um zu erkennen in welchem Schritt der Fehler auftritt.

```
75   solve_cube()
76   GPIO.cleanup()
77   """Alle GPIO´s des Raspberry werden freigegeben"""
```

Abb.: 69 Code Cube_Solve: Methoden solve_cube und GPIO_cleanup

In Zeile 75 wird die zuvor beschriebene Methode ausgeführt.

Die Methode „GPIO.cleanup()“ muss am Ende des Programmes ausgeführt werden, um die, von den Motoren verwendeten, Pin´s am Raspberry Pi freizugeben, damit es zu keiner unerwünschten Erwärmung der Motoren kommt.

3.3 Motorsteuerung

Die Motorsteuerung ist eine Software, um die Schrittmotoren mittels Raspberry Pi zu steuern. Das Prinzip der Schrittmotoren ist unter dem Kapitel Hardware näher erläutert. Die in diesem Projekt verwendeten Motoren benötigen, durch ihre Übersetzung, 512 Halbschritte für eine gesamte Umdrehung. Mit dieser Information werden in diesem Modul die Schritte in Grad umgerechnet. Das ermöglicht eine präzise Steuerung der Motoren für den optischen „Zauberwürfellöser“.

3.3.1 Stepper Modul

Die Software der Motorsteuerung besteht aus dem Modul „stepper“, das in die Lösungssoftware importiert wird.

```
1    import RPi.GPIO as GPIO
2    import time
```

Abb.: 70 Code Stepper Modul: Import

Es werden 2 Bibliotheken benötigt, um den Code zu schreiben. Die Bibliothek „RPi.GPIO“ in Zeile 1 wird importiert und mit dem Objekt GPIO in der Software verwendet. Die Methoden dieser Bibliothek dienen der Benutzung der Pin´s am Raspberry Pi. In Zeile 2 wird die Bibliothek „time“ hinzugefügt, um die Drehzahl der Motoren einzustellen.

```
4    GPIO.setmode(GPIO.BOARD)
```

Abb.: 71 Code Stepper Modul: ausführen der Methode GPIO.setmode

Die Methode setmode() wird benutzt um den Modus der Pin´s festzulegen. Das in Zeile 4 verwendete Argument der Methode setmode() bedeutet, dass die festgelegte Pin-Nummerierung des Raspberry Pi verwendet wird. Die zweite Möglichkeit wäre die GPIO Nummerierung zu verwenden, die von der Pinbelegung des Raspberry abweicht. Zur Veranschaulichung zeigt die nachfolgende Abb.: 72 Pinbelegung RaspberryPi 4B die Pinbelegung des Raspberry Pi 4B.

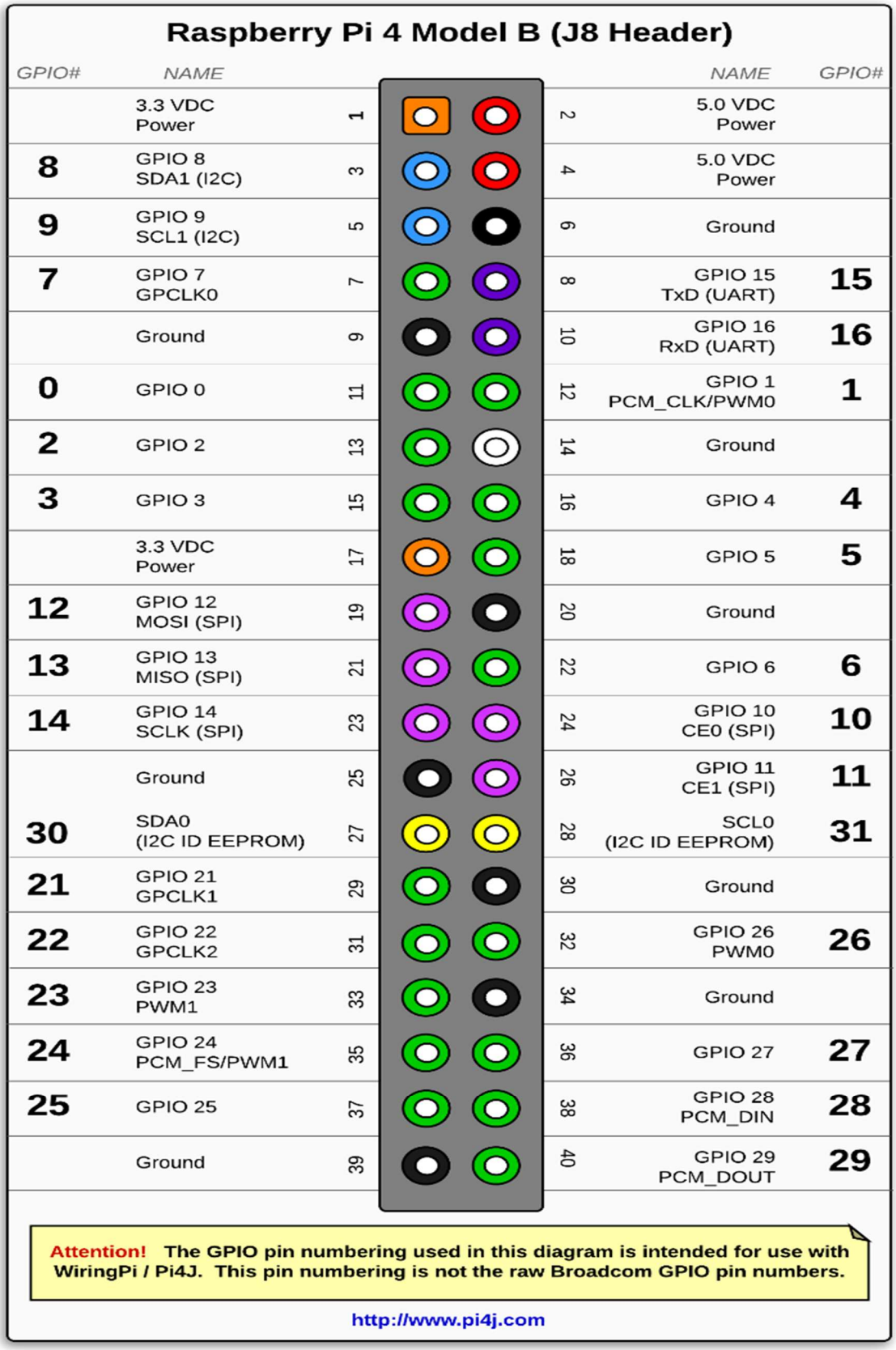


Abb.: 72 Pinbelegung RaspberryPi 4B

Die in der Abb.: 72 Pinbelegung RaspberryPi 4B dick gedruckten Ziffern ist die Nummerierung der GPIO´s und diese werden mit dem Mode GPIO.BCM verwendet. Da bei diesem Projekt fast alle GPIO´s verwendet werden ist es einfacher die Nummerierung der Steckleiste, mit dem zuvor genannten Modus GPIO.Board, zu verwenden.

```
7   control_pins_m = [
8       [11, 12, 13, 15],
9       [16, 18, 22, 5],
10      [10, 40, 24, 26],
11      [19, 21, 23, 8],
12      [29, 31, 33, 35],
13      [32, 36, 37, 38]
14      ]
```

Abb.: 73 Code Stepper Modul: Liste control_pins_m

In den Zeilen 7 – 14 ist eine 6x4 Liste angelegt, um den sechs Motoren die benötigten Pin´s zuzuordnen. Je Motor werden vier Steuerpin´s benötigt.

```
16  for i in range(6):
17      for pin in control_pins_m:
18          GPIO.setup(pin, GPIO.OUT)
19          GPIO.output(pin, 0)
```

Abb.: 74 Code Stepper Modul: GPIO Zuweisung

Die for – Schleife in Zeile 16 wird sechs Mal durchlaufen, je Motor einmal. In Zeile 17 ist eine zweite for – Schleife programmiert, die sich innerhalb der zuvor genannten Schleife befindet, die jeden der 4 Pin´s pro Liste mit der Methode setup() als Ausgang deklariert. Da es sich um eine Liste in einer Liste handelt, benötigt man 2 indizes um diese anzusprechen. Ein Beispiel für eine Programmzeile wäre: control_pins_m[0][1]. Das würde den Motor 0 und den Pin 12 betreffen. Da man zwei indizes benötigt, sind auch die verschachtelten Schleifen nötig, um für jeden Motor alle Pin´s zu beschalten. Durch die Methode output() wird beim zuvor verwendeten Pin, der als output deklariert wurde, eine Spannung von 0 Volt angelegt. Würde man output(pin, 1) schreiben, so würde die benötigte Spannung von 12 Volt angelegt werden.

```
42  step_seq_right = [
43      [1, 0, 0, 1],
44      [0, 0, 0, 1],
45      [0, 0, 1, 1],
46      [0, 0, 1, 0],
47      [0, 1, 1, 0],
48      [0, 1, 0, 0],
49      [1, 1, 0, 0],
50      [1, 0, 0, 0]
51  ]
```

Abb.: 75 Code Stepper Modul: Liste step_seq_right

In den Zeilen 42 – 51 wird eine 8x4 Liste angelegt, um die Drehrichtung der Motoren zu bestimmen. Hier werden für jede Listenebene die vier Steuerpin´s ein oder ausgeschalten (0 Volt oder 12 Volt). Um das nötige Drehmoment mit den verwendeten Motoren aufzubringen, werden Halbschritte anstatt Vollschritte verwendet. Da es sich um Halbschritte handelt, sind es acht Listenebenen und in jeder zweiten Ebene müssen zwei Pin´s eingeschalten sein. Die Drehrichtung bestimmen die Pin´s, die nacheinander eingeschalten werden. In dieser Abbildung sieht man eine Sequenz, um den Motor im Uhrzeigersinn zu drehen. In der Sequenz, um den Motor gegen den Uhrzeigersinn zu drehen, schreibt man die Listen von unten nach oben in das Programm und benennt die Liste, wie in diesem Programm, auf „step_seq_left" um.

```
52  def LEFT_TURN(deg, mot):
53      GPIO.setmode(GPIO.BOARD)
54      full_circle = 512
55      degree = int(full_circle / 360 * deg)
56
57      for i in range(degree):
58          for step in range(8):
59              for pin in range(4):
60                  GPIO.output(control_pins_m[mot][pin], step_seq_left[step][pin])
61              time.sleep(0.001)
```

Abb.: 76 Code Stepper Modul: Methode LEFT_TURN

Die Methode, die in der Zeile 52 definiert ist, wird in der Lösungssoftware, im Modul „Move", ausgeführt. Der Methode besitzt zwei Übergabeparamete. Die zwei Parameter sind „deg" und „mot". Der Parameter „deg" steht für die Grade, die sich der Motor drehen soll und wird zur Berechnung der Grade in Zeile 55 verwendet und „mot" für den Motor, der betrieben werden soll und wird in der Liste „control_pins_m[][] als erster Indize verwendet. Die Werte werden in dem Modul „Move", in der Methode „LEFT_TURN()", als Argumente übergeben. Die Funktion beinhaltet die Berechnung der Grade sowie auch das Setzen der richtigen Pin´s für die Drehrichtung. Da die

Methoden der GPIO Bibliothek erneut verwendet werden, muss auch der Modus in Zeile 53 neu gesetzt werden.
In Zeile 54 ist der Variable „full_circle" ein Wert für eine gesamte Umdrehung zugewiesen. Diese Variable wird in der Berechnung für die Grade, in Zeile 55 verwendet. Da nur 90 und 180 Grad benötigt werden, wird der Variable „degree" ein integer Wert zugewiesen, dafür steht das „int" vor der Berechnung.

Die verschachtelten for – Schleifen in den Zeilen 57 – 61 sind zur richtigen Beschaltung der Steuerpin´s, um die Drehrichtung zu bestimmen.
Die erste Schleife wird so oft durchlaufen, wie es der Variable „degree" zugewiesen wurde.
Die zweite Schleife wird acht Mal durchlaufen. Das ist die Anzahl der Halbschritte und bestimmt mit der Variable step den ersten indize in der eckigen Klammer der Liste step_seq_left[][], in der Zeile 60.
In der dritten Schleife werden die Anzahl der Pin´s durchlaufen und der Variable pin zugewiesen. In dieser Schleife werden, mit der Methode „output()", die richtigen Seteuerpin´s gesetzt. Mit „control_pins_m[][]" wird die benötigte Pinnummer aufgerufen und mit „step_seq_left[][]" auf 0 oder 1 gesetzt, je nachdem welcher Wert in den eckigen Klammern als Indizes steht.
Man benötigt nach jedem Halbschritt eine Verzögerung, um die nötigen Pin´s zu beschalten. Mit dieser Verzögerung wird auch die Drehzahl bestimmt. In Zeile 61 ist die Methode sleep() aus der Bibliothek time programmiert. Die Funktion befindet sich in der zweiten Schleife da nur die Halbschritte verzögert werden müssen. Die hier verwendete Verzögerung beträgt eine tausendstel Sekunde, um das benötigte Verhältnis aus Drehmoment und Drehzahl zu bekommen.

3.4 Farberkennung

Die Aufgabe der Farberkennung ist es, sämtliche Farben zu erkennen und den Feldern in richtiger Reihenfolge zuzuordnen. Diese Zuordnung wird an den Algorithmus übergeben.
Um die Lesbarkeit und das Verständnis des Codes so angenehm wie möglich zu halten, werden sämtliche verwendete Funktionen genauer unter dem Punkt „4 Verwendete Funktionen Farberkennung" erklärt.

3.4.1 Verwendete Bibliotheken

3.4.1.1 OpenCV

OpenCV ist eine freie Programmbibliothek mit Algorithmen für die Bildverarbeitung und maschinelles Sehen. Sie ist für die Programmiersprachen C, C++, Python und Java geschrieben und steht als freie Software zu Verfügung. Das „CV" im Namen steht für „Computer

Vision". OpenCV ist ein von BSD lizenziertes Produkt, das es Unternehmen leicht macht, den Code zu verwenden und zu modifizieren.

3.4.1.2 NumPy

NumPy ist eine Programmbibliothek, die eine einfache Handhabung von Vektoren, Matrizen oder generell großen mehrdimensionalen Arrays ermöglicht. Neben den Datenstrukturen bietet NumPy auch effizient implementierte Funktionen für numerische Berechnungen an.

3.4.2 Herangehensweise

Für die Farberkennung wurden zwei Klassen erstellt, eine für die obere Kamera und eine für die untere Kamera. Die zwei Klassen unterschieden sich in zwei Punkten – zum einen im Port der Kameras und zum anderen wurden die Werte der HSV-Filter angepasst. Es handelt sich hierbei um eine variable Lösung, das heißt, der Würfel und die Kameras müssen keine absoluten Positionen haben. Solange pro Kamera jeweils drei Seiten zur Gänze erkennbar sind, erkennt das Programm 24 Felder und weist die Farben zu. Für die Auswertung der Farben wird der HSV-Farbraum verwendet.

3.4.2.1 HSV – Farbraum

Der HSV-Farbraum definiert eine Farbe über den Farbton (engl.: hue), die Sättigung (engl.: saturation) und die Helligkeit, beziehungsweise die Dunkelstufe (engl.: value) und ähnelt damit der menschlichen Farbwahrnehmung mehr als die additiven und subtraktiven Farbmodelle. So kann schnell eine Farbe bestimmt werden, deren Sättigung und Helligkeit, beziehungsweise Dunkelheit angepasst wird.

Wegen dieser Vorteile findet die Farbwahl mittels des HSV-Farbraums zum Beispiel in vielen gängigen Grafikprogrammen ihre Anwendung. Auch der Standard-Farbauswahl-Dialog, zum Beispiel vom Betriebssystem Windows, beruht auf dem HSV-Farbmodell: Hier gibt es ein Farbfeld, in dem die Farben nach Farbton und Sättigung angeordnet und ausgewählt werden können, sowie einen zusätzlichen Regler für die Helligkeit von weiß bis schwarz, mit dem die ausgewählte Farbe verändert werden kann.

Der Farbton (H) wird als Winkel auf dem Farbkreis angegeben und kann daher die Werte zwischen 0° und 359° annehmen. 0° entspricht der Farbe Rot, 120° entspricht der Farbe Grün und 240° entspricht der Farbe Blau. Die Sättigung (S) wird in Prozent angegeben und kann demnach die Werte von 0% bis 100% (bzw. 0 bis 1) annehmen. Eine Sättigung von 100% bedeutet eine vollständig gesättigte und damit reine Farbe.

Umso kleiner die Sättigung wird, desto mehr wandelt sich die Farbe in ein neutrales Grau. Auch die Helligkeit (V) wird in Prozent angegeben, wobei 0% keine Helligkeit bedeutet, also Schwarz, und 100% die volle Helligkeit, also ein Spektrum zwischen der reinen Farbe (Sättigung 100%) und Weiß (Sättigung 0%).

Wenn sowohl die Sättigung als auch der Helligkeitswert 100% betragen, kommt es zur reinen Farbe. Ist die Sättigung 0% und der Helligkeitswert 100% ergibt sich Weiß und für alle Fälle, in denen der Helligkeitswert 0% beträgt, ergibt sich Schwarz.
[4]

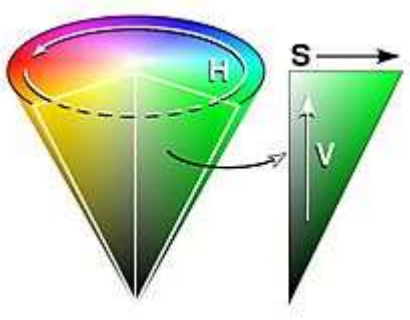


Abb.: 77 HSV-Farbraum

3.4.3 Code Farberkennung Initialisierung

Unter diesem Punkt werden die Parameter erklärt, die in der Initialisierung gesetzt werden.

```
8   """Simulation oder Realität"""
9   cammode = False
```

Abb.: 78 Code Farberkennung Initialisierung

In der Zeile 9 wird mit dem booleschen Wert „False" ein Simulationsmodus gewählt. Dieser Simulationsmodus nimmt ein zuvor aufgenommenes Bild, das im Speicher liegt, wertet dieses aus und dient zu Vorführzwecken. Mit dem booleschen Wert „True" wird in den Kameramodus gewechselt, der das auszuwertende Bild über die Kamera bezieht.

```
11  """Kallibrierung der HSV-Filter: 1. Wert = HUE(Farbwert),
12  | 2.Wert = Saturation(Farbsättigung), 3.Wert = Value(Hellwert)"""
13  HSV_RED_UPPER = [174,255,255]
14  HSV_RED_LOWER = [133, 140, 0]
15  HSV_YELLOW_UPPER = [38, 255, 255]
16  HSV_YELLOW_LOWER = [21, 70, 41]
17  HSV_GREEN_UPPER = [79, 135, 255]
18  HSV_GREEN_LOWER = [56, 53, 67]
19  HSV_BLUE_UPPER = [146, 226, 255]
20  HSV_BLUE_LOWER = [106, 152, 140]
21  HSV_WHITE_UPPER = [153, 35,255]
22  HSV_WHITE_LOWER = [63, 0, 159]
23  HSV_ORANGE_UPPER = [179, 188, 255]
24  HSV_ORANGE_LOWER = [170, 97, 165]
```

Abb.: 79 Code Farberkennung Initialisierung

Von Zeile 13 bis Zeile 24 werden die Ober- und Untergrenzen der HSV-Filter eingegeben. Die gewünschte Farbe befindet sich zwischen der Ober- und Untergrenze. Die einzelnen Listen sind wie folgt zu verstehen: der erste Wert entspricht dem Farbwert (engl.: hue), der zweite Wert entspricht der Farbsättigung (engl.: Saturation) und der dritte Wert entspricht der Helligkeit (engl.: value).
Python arbeitet, nicht wie andere Programme, mit Graden beim Farbwert und Prozenten bei der Farbsättigung sowie der Helligkeit. Der Wertebereich sieht wie folgt aus:
Wertebereich Python:

- Farbwert (engl.: hue): 0 – 180
- Farbsättigung (engl.: saturation): 0 – 255
- Helligkeit (engl.: value): 0 – 255

Genaueres zum HSV-Farbraum kann unter dem Punkt „3.5.2.1 HSV – Farbraum" nachgelesen werden.
Die Werte der Filter wurden mittels einer selbstgeschrieben Kalibrierungssoftware festgestellt.

3.4.3.1 Code Kalibrierung

Um den Filter passend und schnell einzustellen, wurde ein kurzes Kalibrierungsprogramm geschrieben, das den Studierenden helfen sollte, schnell bei extremen Lichtverhältnissen die Filter anzupassen.

```
7   cap = cv2.VideoCapture(1) #Videoaufnahme
8
9   while True:
10      ret, img = cap.read()
11
12      #img = cv2.imread('../Pictures/opencv_frame_17.png')
13      cv2.namedWindow("TrackBars")
14      cv2.resizeWindow("TrackBars", 640,240)
15      cv2.createTrackbar("Hue Min", "TrackBars", 0,179,empty)
16      cv2.createTrackbar("Hue Max", "TrackBars", 179,179,empty)
17      cv2.createTrackbar("Sat Min", "TrackBars", 0,255,empty)
18      cv2.createTrackbar("Sat Max", "TrackBars", 255,255,empty)
19      cv2.createTrackbar("Val Min", "TrackBars", 0,255,empty)
20      cv2.createTrackbar("Val Max", "TrackBars", 255,255,empty)
```

Abb.: 80 Code Kalibrierung

Mit der Methode „cv2.VideoCapture(#)", wird das Livebild der Variable „cap" zugewiesen. Um dieses Livebild sichtbar zu machen und es weiter zu verwenden, wird in einer While-Schleife Frame für Frame auf die Variable „img" zugewiesen. Die boolesche Variable „ret" gibt „True" zurück, wenn ein Frame Verfügbar ist.

Bei der Methode „cv2.VideoCapture(#)" gilt zu beachten, dass die richtige Kamera in der Klammer angegeben wird; wenn mehrere Kameras verwendet werden: Sollte nur eine Kamera verwendet werden, ist in der Klammer stets die „0" einzutragen, jede weitere Kamera wird mit +1, addiert.

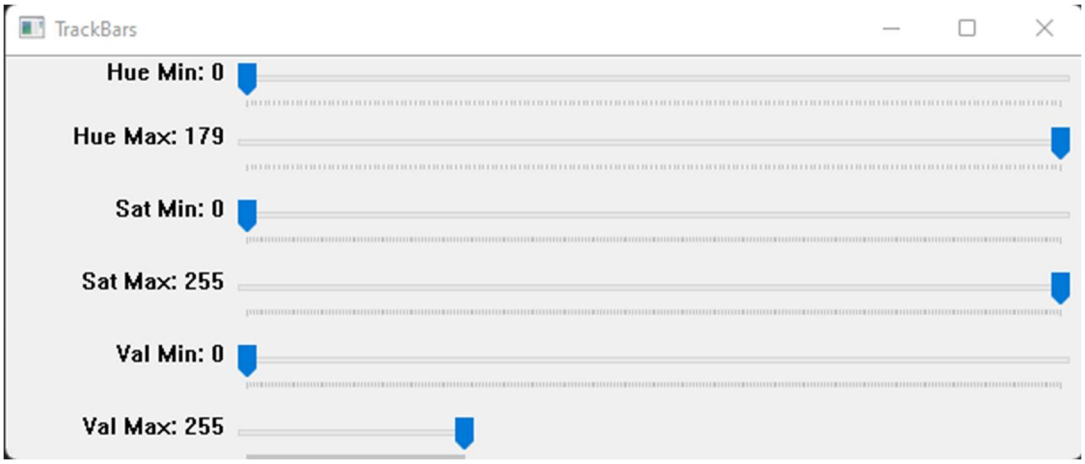


Abb.: 81 Trackbars Kalibrierung, zum Einstellen der HSV-Werte für die Filter

Von den Zeilen 13 – 20 wird ein Fenster mit sechs Trackbars erstellt. Diese dienen dazu, die einzelnen HSV-Werte einzustellen.

```
22      while True:
23          imgHSV = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
24          h_min = cv2.getTrackbarPos("Hue Min", "TrackBars")
25          h_max = cv2.getTrackbarPos("Hue Max", "TrackBars")
26          s_min = cv2.getTrackbarPos("Sat Min", "TrackBars")
27          s_max = cv2.getTrackbarPos("Sat Max", "TrackBars")
28          v_min = cv2.getTrackbarPos("Val Min", "TrackBars")
29          v_max = cv2.getTrackbarPos("Val Max", "TrackBars")
30          print(h_min, h_max, s_min, s_max, v_min, v_max)
31          lower = np.array([h_min, s_min, v_min])
32          upper = np.array([h_max, s_max, v_max])
33          mask = cv2.inRange(imgHSV, lower, upper)
34          imgResult = cv2.bitwise_and(img, img,mask = mask)
35
36          cv2.imshow("mask", mask)
37          cv2.imshow("result", imgResult)
38          cv2.waitKey(1)
```

Abb.: 82 Code Kalibrierung

In der Zeile 23 werden die einzelnen Originalframes von BGR-Farbraum in den HSV-Farbraum umgewandelt. BGR ist dasselbe wie RGB, nur Rot- und Blau-Wert sind vertauscht. Python verwendet nur BGR.

Die Werte der Trackbars werden in den Zeilen 24 bis 29 ausgelesen und einem NumPy Array zugewiesen. Da Python selber keine Arrays, sondern nur Listen hat, wird hier auf eine Funktion der NumPy Bibliothek zurückgegriffen.

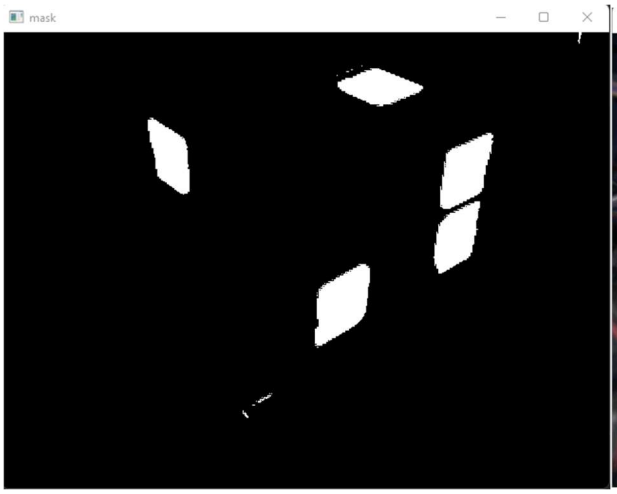


Abb.: 84 HSV-Filter Farbe blau

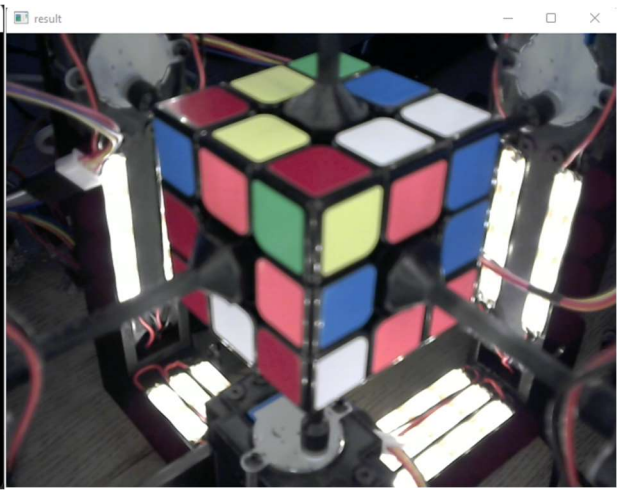


Abb.: 83 Originalbild ohne Filter

In der Zeile 33 wird jeder einzelne Pixel geprüft, ob er in den Grenzwerten liegt und bekommt, wenn er im Grenzwert ist, „1", andernfalls „0". Das Ergebnis ist ein Binärbild.

3.4.4 Code Bildaufnahme

```
45  #get upper Cam picture-------------------------------------------------------------------
46  if cammode == True:
47      def getCam(SPACEBAR):
48          """Hier wird das Bild der Kamera angezeigt, mit Betätigung der Leertaste
49          wird ein Bild aufgenommen und an das Programm übergeben"""
50          """Auswahl der USB Kamera 0 oder 1 und einstellen der Bildbreite und Bildhöhe"""
51          Cam = cv2.VideoCapture(0)
52          Cam.set(cv2.CAP_PROP_FRAME_HEIGHT, 640)
53          Cam.set(cv2.CAP_PROP_FRAME_WIDTH,480)
54          cv2.namedWindow("Adjustierung Oben")
55          img_counter = 0
```

Abb.: 85 Code Bildaufnahme

In diesem Ausschnitt wird das Kamerabild aufgenommen und die Größe des Bildes gesetzt. Wie die Aufnahme der Kamera abläuft, wird unter dem Punkt „3.5.3.1 Code Kalibrierung" näher erklärt.

```
56          while True:
57              ret, img = Cam.read()
58              cv2.imshow("Adjustierung Oben", img)
59              if not ret:
60                  break
61              k = cv2.waitKey(1)
62              if k%256 == 27:
63                  # ESC pressed
64                  print("Escape hit, closing...")
65                  break
66              elif k%256 == 32:
67                  # SPACE pressed
68                  frame1 = "opencv_frame_{}.png".format(img_counter)
69                  cv2.imwrite(frame1, img)
70                  imgOrg  = cv2.imread(frame1)
71                  print("{} written!".format(frame1))
72                  img_counter += 1
73                  return imgOrg
74              elif SPACEBAR == 32:
75                  frame1 = "opencv_frame_{}.png".format(img_counter)
76                  cv2.imwrite(frame1, img)
77                  imgRetry  = cv2.imread(frame1)
78                  print("{} written!".format(frame1))
79                  img_counter += 1
80                  return imgRetry
```

Abb.: 86 Code Bildaufnahme

In Zeile 58 wird ein Livebild der Kamera angezeigt, um zu sehen, ob der Würfel noch adjustiert werden muss und alle 24 farblichen Felder zu erkennen sind.
In Zeile 59 wird abgefragt, ob ein Frame vorhanden ist, andernfalls werden die Fenster geschlossen.

Sollte es ein gültiges Bild geben, kann entweder mit „ESC" abgebrochen (Zeile 62) oder mit „SPACEBAR" ein Bild aufgenommen werden (Zeile 66).
Das zweite „elif" in der Zeile 74 dient dazu, dass bis zu zehn Mal versucht wird, ein neues Bild aufzunehmen, wenn weniger als 21 Felder erkannt werden. Dazu genaueres unter dem Punkt „3.5.6 Auswertung".

```
83    if cammode == True:
84        ImgOrig = getCam(0)
85    elif cammode == False:
86        ImgOrig = cv2.imread('../Pictures/opencv_frame_17.png')
87
88    cv2.imshow("Original", ImgOrig)
89    HSVimg = cv2.cvtColor(ImgOrig, cv2.COLOR_BGR2HSV)
```

Abb.: 87 Code Bildaufnahme

In den Zeilen 83 bis 86 wird abgefragt, welcher Modus aktiv ist. Ob nun das Livebild aus der Funktion (Zeile 84) übernommen werden soll oder das abgespeicherte, oder aber auch unbearbeitete, Bild (Zeile 86).
Mit der Funktion „cv2.imshow()" wird der verwendete Frame in seinem Urzustand dargestellt.
In der Zeile 89 wird das Originalbild in ein HSV-Bild umgewandelt.

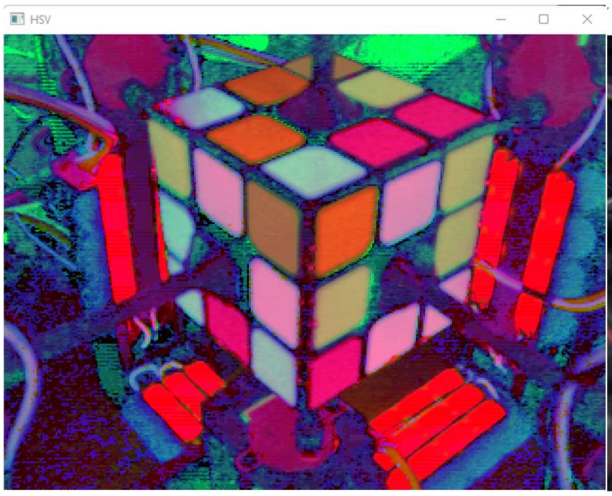


Abb.: 89 HSV-Bild

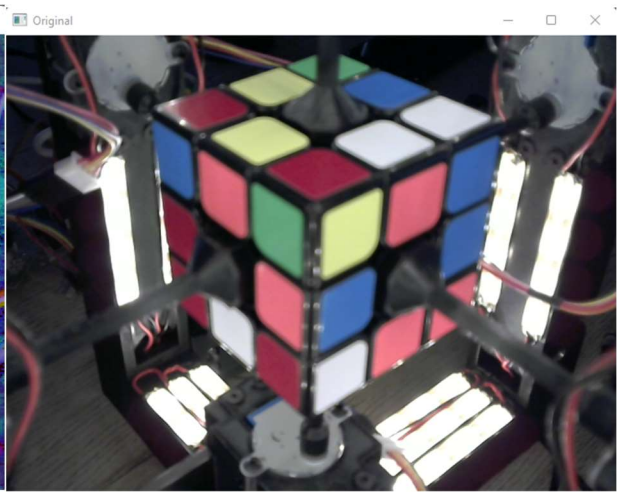


Abb.: 88 Originalbild

3.4.5 Code Farbfilter ganze Ecken

Es gibt für jede Farbe eine eigene Funktion, die, wenn sie passende Konturen findet, diese als Liste zurückgibt. Diese Liste beinhaltet X – Y – Koordinaten, Farbe, Breite und Höhe der gefundenen Felder.

```
 92 > def RedFilter(HSVimg, hsvCrop, origCrop):···
142
143 > def BlueFilter(HSVimg, hsvCrop, origCrop):···
184
185 > def GreenFilter(HSVimg, hsvCrop, origCrop):···
224
225 > def YellowFilter(HSVimg, hsvCrop, origCrop):···
263
264 > def OrangeFilter(HSVimg, hsvCrop, origCrop):···
302
303 > def WhiteFilter(HSVimg, hsvCrop, origCrop):···
```

Abb.: 90 Code Farbfilter ganze Ecken

```
93  def RedFilter(HSVimg, hsvCrop, origCrop):
94      """Zeigt an ob es Farben in dem Bild gibt das in die Range des Filters passt"""
95      """Array wird erzeugt, mit den Grenzwerten des Filters"""
96      lower_RED = np.array([HSV_RED_LOWER[0], HSV_RED_LOWER[1], HSV_RED_LOWER[2]])
97      upper_RED = np.array([HSV_RED_UPPER[0], HSV_RED_UPPER[1], HSV_RED_UPPER[2]])
98      """Hier ensteht ein schwarz-weiß Bild, seiß sind sämtliche Pixel die in den Filter fallen,
99       andere Pixel sind schwarz"""
100     mask_RED = cv2.inRange(HSVimg, lower_RED, upper_RED)
101     """Hier wird die "Maske" auf Konturen untersucht."""
102     mask_contours, hierarchy  = cv2.findContours(mask_RED, cv2.RETR_EXTERNAL,
103     cv2.CHAIN_APPROX_SIMPLE)
```

Abb.: 91 Code Farbfilter ganze Ecken

Da Python kein Array verwendet, wurden mittels der Bibliothek NumPy zwei Arrays erzeugt. Ein Array beinhaltet den oberen Schwellenwert und das andere Array den unteren Schwellenwert des Filters.
Mit der Funktion „cv2.inRange()" wird das HSV-Bild Pixel für Pixel abgeglichen. Wenn der Pixel innerhalb der Schwellenwerte liegt, bekommt dieser Pixel den Wert „1". Das Ergebnis, ein Binärbild, wird auf die Variable „mask_RED" geschrieben.

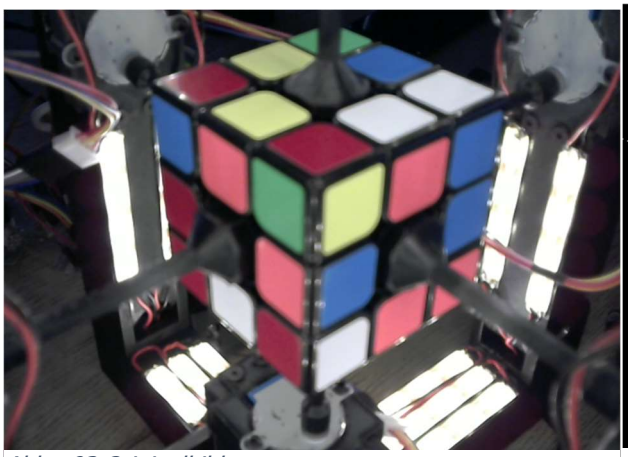


Abb.: 93 Originalbild

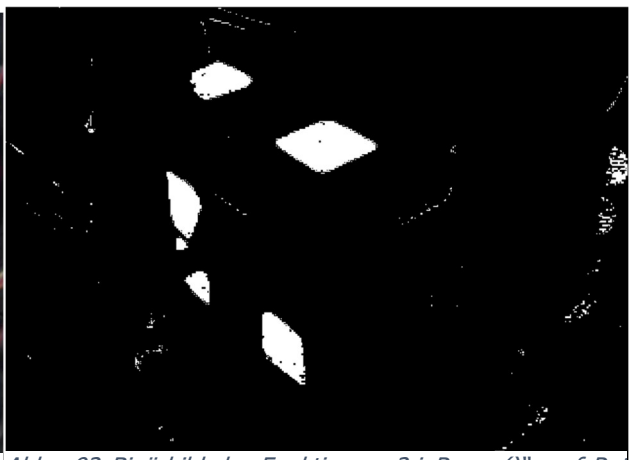


Abb.: 92 Binärbild der Funktion „cv2.inRange()", auf Rot gefiltert

Dieses Binärbild wird mit der Funktion „cv2.findContours()" untersucht. Sollten sich in diesem Bild Konturen finden, werden diese Konturen in einer Liste abgespeichert. Zum Abrufen der Konturen wurde der Modus „cv2.RETR_EXTERNAL" verwendet. Dieser Modus extrahiert nur die stark ausgeprägten Außengrenzen der Konturen. Für das Konturenannäherungsverfahren wurde das Verfahren „cv2.CHAIN_APPROX_SIMPLE" verwendet. Dieses Verfahren komprimiert alle horizontalen, vertikalen und diagonalen Segmente und lässt nur mehr dessen Endpunkte übrig. Ein aufrechtstehendes Rechteck wäre zum Beispiel mittels 4 Punkten codiert. Die Funktion gibt eine Liste oder einen Baum von Listen zurück und die Hierarchie zeigt an, wie die Formen miteinander verbunden sind.

```
105     """Sollte die Anzahl der Konturen >1 sein und vom Originalbild stammen"""
106     if len(mask_contours) != 0 and hsvCrop !=0:
107         cnt = 0
108         """Jede gefundene Konture wird auf ihre Größe kontrolliert"""
109         for mask_contour in mask_contours:
110             if cv2.contourArea(mask_contour) > CONTOUR_AREA:
111                 """Diese Funktion kalkuliert und retouniert das kleinste
112                 Rechteck über die gefunden Konturen."""
113                 x, y, w, h = cv2.boundingRect(mask_contour)
114                 """Zeichnen des Rechtecks, übergeben der Werte des gefunden
115                 Rechtecks an die Liste"""
116                 cv2.rectangle(ImgOrig, (x, y), (x + w, y + h), (0, 0, 255), 3)
117                 red[cnt] = x,y,RED,w,h
118                 cnt += 1
```

Abb.: 94 Code Farbfilter ganze Ecken

In einer „if-Abfrage" wird geprüft, ob Konturen gefunden wurden und ob das zu untersuchende Bild vom Originalbild stammt. Jede einzelne Kontur läuft durch eine „for-Schleife" und wird auf eine Mindestgröße kontrolliert. Somit werden kleinere Farbflächen herausgefiltert.

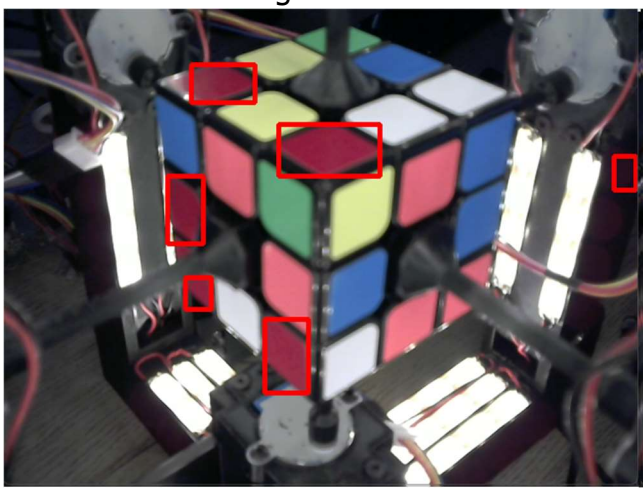


Abb.: 96 ohne Filterung

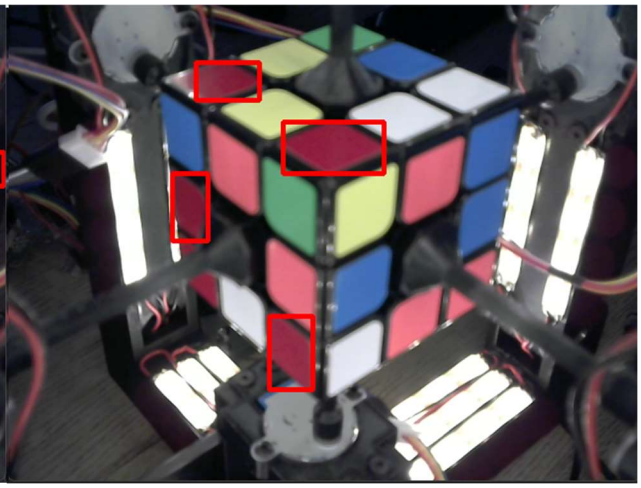


Abb.: 95 mit Filterung

Sollte die Kontur größer sein als die Mindestgröße, wird mit der Funktion „cv2.boundingRect" die Position und Größe des zu zeichnenden Rechtecks berechnet.

Diese Funktion schreibt die X-Y Koordinaten des Startpunktes (links oben), die Breite und die Höhe des Rechtecks auf Variablen.
Diese Variablen werden dafür verwendet, um das Rechteck über die erkannten Felder in der passenden Farbe zu zeichnen.
Zeitgleich werden die Variablen in eine Liste gespeichert. Dieser Liste wird je nach Filter noch eine Farbe hinzugefügt. Ersichtlich in der Zeile 117 „red[cnt] = x, y, RED, w, h". Die Liste „red[]" wird am Ende der Funktion retourniert.
Diese Vorgehensweise betrifft auch die restlichen Filter, somit wird auf die anderen Farben nicht weiter eingegangen.

3.4.6 Code Farbfilter geteilte Steine

Da aufgrund des Aufbaus des Maschinengerüsts die sechs Halterungen des Würfels jeweils einen Stein in zwei Teile teilen, bestand die Herausforderung darin, die Farbe des Steines zu extrahieren, ohne eine zu große Fehlertoleranz einzubauen.

```
439   """Ausschneiden der Steine hinter der Halterung und dieses Bild erneut
440    durch den Filter laufen lassen mit angepassten Areal, gibt die größte gefundene Fläche Retour"""
441
442   img_cropped_r = ImgOrig[allcolorsnew[6][1]-30:allcolorsnew[6][1]+30,
443    allcolorsnew[2][0]-20:allcolorsnew[2][0]+allcolorsnew[2][3]]
444
445   img_cropped_r_hsv = cv2.cvtColor(img_cropped_r, cv2.COLOR_BGR2HSV)
446
447   img_cropped_f = ImgOrig[allcolorsnew[20][1]+50:allcolorsnew[17][1]+10,
448   allcolorsnew[20][0]:allcolorsnew[17][0]]
449
450   img_cropped_f_hsv = cv2.cvtColor(img_cropped_f, cv2.COLOR_BGR2HSV)
451
452   img_cropped_u = ImgOrig[allcolorsnew[14][1]-20:allcolorsnew[14][1]+10,
453   allcolorsnew[14][0]+60:allcolorsnew[14][0]+130 ]
454
455   img_cropped_u_hsv = cv2.cvtColor(img_cropped_u, cv2.COLOR_BGR2HSV)
```

Abb.: 97 Code Farbfilter geteilte Steine

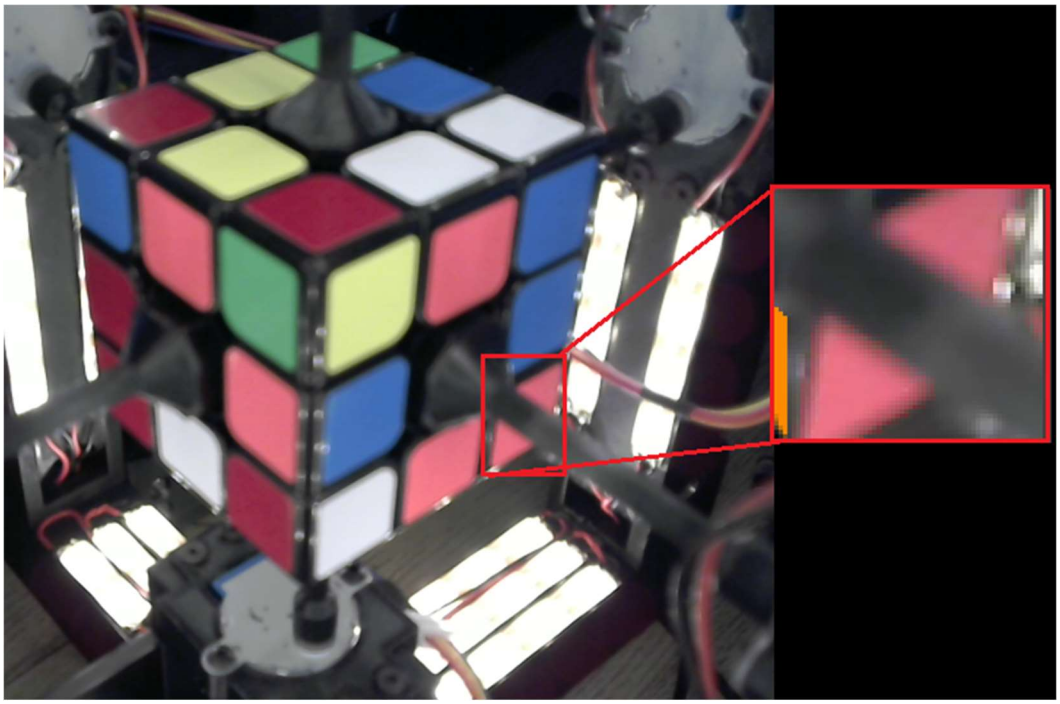


Abb.: 98 Code Farbfilter geteilte Steine

In Abhängigkeit von zwei Nachbarsteinen wurde pro verdeckten Stein ein Bild herausgeschnitten, das circa die Größe und Position des verdeckten Steines hatte.

```
457   RedCroppedR =    RedFilter   (img_cropped_r_hsv,0, img_cropped_r)
458   BlueCroppedR =   BlueFilter  (img_cropped_r_hsv,0, img_cropped_r)
459   WhiteCroppedR =  WhiteFilter (img_cropped_r_hsv,0, img_cropped_r)
460   OrangeCroppedR=  OrangeFilter(img_cropped_r_hsv,0, img_cropped_r)
461   GreenCroppedR =  GreenFilter (img_cropped_r_hsv,0, img_cropped_r)
462   YellowCroppedR=  YellowFilter(img_cropped_r_hsv,0, img_cropped_r)
```

Abb.: 99 Code Farbfilter geteilte Steine

So konnte das Areal des Farbfilters verkleinert und erneut durch alle Farbfilter ausgewertet und automatisch Variablen zugeordnet werden.

```
119         """Sollte die Anzahl der Konturen >1 sein und vom abgeschnitte
120         Bild der rechten Seite stammen"""
121     elif len(mask_contours) != 0 and hsvCrop == 0:
122         for mask_contour in mask_contours:
123             if cv2.contourArea(mask_contour) > CONTOUR_AREA_CROPPED:
124 
125                 x, y, w, h = cv2.boundingRect(mask_contour)
126                 cv2.rectangle(origCrop, (x, y), (x + w, y + h), (0, 0, 255), 3)
127                 red[8] = x,y,RED,w,h
128         """Sollte die Anzahl der Konturen >1 sein und vom abgeschnitte
129         Bild der vorderen Seite stammen"""
130     elif len(mask_contours) != 0 and hsvCrop == 2:
131         for mask_contour in mask_contours:
132             if cv2.contourArea(mask_contour) > CONTOUR_AREA_CROPPED:
133                 x, y, w, h = cv2.boundingRect(mask_contour)
134                 cv2.rectangle(origCrop, (x, y), (x + w, y + h), (0, 0, 255), 3)
135                 red[8] = x,y,RED,w,h
136         """Sollte die Anzahl der Konturen >1 sein und vom abgeschnitte
137         Bild der oberen Seite stammen"""
138     elif len(mask_contours) != 0 and hsvCrop == 3:
139         for mask_contour in mask_contours:
140             if cv2.contourArea(mask_contour) > CONTOUR_AREA_CROPPED:
141                 x, y, w, h = cv2.boundingRect(mask_contour)
142                 cv2.rectangle(origCrop, (x, y), (x + w, y + h), (0, 0, 255), 3)
143                 red[8] = x,y,RED,w,h
144     else:
145         """Sollte keine Kontur gefunden werden wird die letzte
146         Stelle der Liste auf 0 gesetzt um eine Fehlauswertung zu verhindern."""
147         red[8] = 0,0,0,0,0
148 
149     return red
```

Abb.: 100 Code Farbfilter geteilte Steine

Die Auswertung läuft gleich ab wie bei dem Originalbild, nur die Konturenfläche ist kleiner. Wird etwas gefunden, werden auf die letzte Stelle der Liste Werte eingetragen. Ist dies nicht der Fall, wird die letzte Stelle immer auf [0,0,0,0,0] gesetzt, um Fehlauswertungen zu vermeiden. Jeder verdeckte Stein hat seine eigene Auswertung, um eine klare Trennung der Steine zu gewährleisten.

```
468    """Wurde eine Kontur gefunden, wird ein Wert
469     auf die letzte Stelle der Liste geschrieben
470    und in dieser If Abfrage abgefragt, sollte ein Wert
471     gefunden werden der ungleich 0 ist wird der Variable
472     diese Farbe zugewiesen."""
473
474    if RedCroppedR[8][0] != 0:
475        hiddenR = RED
476    elif BlueCroppedR[8][0] !=0:
477        hiddenR = BLUE
478    elif WhiteCroppedR[8][0] !=0:
479        hiddenR = WHITE
480    elif OrangeCroppedR[8][0] !=0:
481        hiddenR = ORANGE
482    elif GreenCroppedR[8][0] !=0:
483        hiddenR = GREEN
484    elif YellowCroppedR[8][0] !=0:
485        hiddenR = YELLOW
```

Abb.: 101 Code Farbfilter geteilte Steine

Da jeder verdeckte Stein einzeln für sich ausgewertet wird, kann dieser Stein nur eine Farbe besitzen. In der hier gezeigten „if-Abfrage" wird die letzte Stelle der Liste abgefragt. Da dieser eine Stein nur eine Farbe haben kann, muss der Filter, der ungleich Null zurückgibt, die Farbe des Steines sein und die Variable wird gesetzt.
Diese Vorgehensweise betrifft auch die restlichen fünf verdeckten Steine, somit wird auf die anderen Farben nicht weiter eingegangen.

3.4.7 Code Auswertung

Um die Felder der einzelnen Seiten richtig zuordnen zu können, wurden mehrere Sortieralgorithmen verwendet.

```
352    """Zuweisen der erkannten Farben und unsortierten Felder an eine Liste."""
353    red    =     RedFilter    (HSVimg,1, 1)
354    blue   =     BlueFilter   (HSVimg,1, 1)
355    white  =     WhiteFilter  (HSVimg,1, 1)
356    orange =     OrangeFilter (HSVimg,1, 1)
357    green  =     GreenFilter  (HSVimg,1, 1)
358    yellow =     YellowFilter (HSVimg,1, 1)
359
360    allcolorsUnsort = red + blue + white + orange + green + yellow
```

Abb.: 102 Code Auswertung

Zu Beginn wurden alle Listen einer großen unsortierten Liste zugeordnet.

```
363   """Kontrolliert ob genügend Felder gefunden wurden"""
364   cntfield = 0
365   if(len(allcolors) < 21 and cntfield <=10 and cammode == True):
366       getCam(32)
367       cntfield +=1
368   elif(cntfield>10):
369       print("Fehler, nicht genügend Konturen gefunden")
```

Abb.: 103 Code Auswertung

Danach wird die Liste darauf kontrolliert, ob 21 Einträge vorhanden sind. Sollte dies nicht der Fall sein, wird bis zu zehn Mal versucht ein Kamerabild aufzunehmen. Wenn nach zehn Versuchen kein gültiges Bild aufgenommen wird, gibt das Programm eine Fehlermeldung aus.

```
371   """Sortieren der X-Werte von kleinsten zu größten Wert,
372   größter X-Wert = rechtestes Feld des Würfel"""
373
374   for i in range(54):
375       for j in range(0, 53-1-i):
376           if allcolors[j][0] < allcolors[j+1][0]:
377               allcolors[j], allcolors[j+1] = allcolors[j+1], allcolors[j]
```

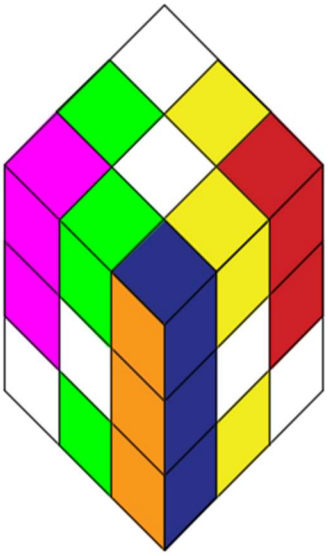
Abb.: 104 Code Auswertung

Mittels des „Bubblesort" Sortieralgorithmus wird die X-Koordinaten von links aufsteigend sortiert. Somit sind alle rechten Felder rechts und alle linken Felder links. Aber die Felder sind nicht nach Höhe, also Y-Koordinaten, sortiert.

```
379   """Zerlegen der sortierten X-Werte in einzelne Spalten."""
380
381   rechtsrechts = allcolorsUnsort[0:3]
382   rechtsmitte = allcolorsUnsort[3:7]
383   rechts = allcolorsUnsort[7:10]
384   links = allcolorsUnsort[10:14]
385   linksmitte = allcolorsUnsort[14:18]
386   linkslinks = allcolorsUnsort[18:21]
```

Abb.: 105 Code Auswertung

Um die Felder auch nach Y-Koordinaten sortieren zu können, wurden die X-Koordinaten in sechs Spalten unterteilt. In der nachfolgenden Skizze dürfte die Aufteilung der Spalten klarer werden.



- Rot = rechtsrechts
- Gelb = rechtsmitte
- Blau = rechts
- Orange = links
- Grün = linksmitte
- Pink = linkslinks

Die weißen Felder sind für diese Sortierung irrelevant, denn die Mittelsteine einer Seite sind fix vorgegeben und die äußeren Ecksteine wurden im Kapitel, „3.5.6 Code Farbfilter geteilte Steine", behandelt.

Abb.: 106 Sortierung nach Y-Koordinaten

```
388  """Sortieren der Y-Werte von Oben nach Unten """
389  for i in range(len(rechtsrechts)):
390      for j in range(len(rechtsrechts)-1-i):
391          if rechtsrechts[j][1] > rechtsrechts[j+1][1]:
392              rechtsrechts[j], rechtsrechts[j+1] = rechtsrechts[j+1], rechtsrechts[j]
393
394  for i in range(len(rechtsmitte)):
395      for j in range(len(rechtsmitte)-1-i):
396          if rechtsmitte[j][1] > rechtsmitte[j+1][1]:
397              rechtsmitte[j], rechtsmitte[j+1] = rechtsmitte[j+1], rechtsmitte[j]
398
399  for i in range(len(rechts)):
400      for j in range(len(rechts)-1-i):
401          if rechts[j][1] > rechts[j+1][1]:
402              rechts[j], rechts[j+1] = rechts[j+1], rechts[j]
403
404  for i in range(len(links)):
405      for j in range(len(links)-1-i):
406          if links[j][1] > links[j+1][1]:
407              links[j], links[j+1] = links[j+1], links[j]
408
409  for i in range(len(linksmitte)):
410      for j in range(len(linkslinks)-i):
411          if linksmitte[j][1] > linksmitte[j+1][1]:
412              linksmitte[j], linksmitte[j+1] = linksmitte[j+1], linksmitte[j]
413
414  for i in range(len(linkslinks)):
415      for j in range(len(linkslinks)-1-i):
416          if linkslinks[j][1] > linkslinks[j+1][1]:
417              linkslinks[j], linkslinks[j+1] = linkslinks[j+1], linkslinks[j]
418
419  allcolorsSort = rechtsrechts + rechtsmitte + rechts + links + linksmitte + linkslinks
```

Abb.: 107 Code Auswertung

Nun wird jede einzelne Spalte für sich mittels des „Bubblesort" Sortieralgorithmus in den Y-Koordinaten von unten aufsteigend sortiert und einer neuen sortierten Liste zugewiesen.

```
425   """Beschriftet sämtliche gefunden Konturen die über geblieben sind"""
426   for i in range(len(allcolorsSort)):
427               cv2.circle(ImgOrig,(allcolorsUnsort[i][0],allcolorsUnsort[i][1] ), 5, (0,0,0),3)
428               cv2.putText(ImgOrig, str(i), (allcolorsSort[i][0],allcolorsSort[i][1]), 0, 1, (0,0,0),2)
```

Abb.: 108 Code Auswertung

In dieser Schleife bekommen sämtliche Starpunkte eines gefundenen Feldes eine fortlaufende Nummer sowie eine Markierung mittels eines Kreises. Dies dient zur Kontrolle, ob der Algorithmus richtig gearbeitet hat.

```
438   """Zuweisen der Felder an die jeweiligen Seiten in der richtigen Reihenfolge,
439    ohne Steine hinter Halterung"""
440
441   r = (allcolorsSort[7][2], allcolorsSort[5][2], allcolorsSort[1][2], allcolorsSort[8][2],
442    GREEN, allcolorsSort[2][2], allcolorsSort[9][2], allcolorsSort[6][2], hiddenR)
443
444   f = (allcolorsSort[19][2], allcolorsSort[16][2], allcolorsSort[11][2], allcolorsSort[20][2],
445    BLUE, allcolorsSort[12][2], hiddenF, allcolorsSort[17][2], allcolorsSort[13][2])
446
447   u = (hiddenU, allcolorsSort[3][2], allcolorsSort[0][2], allcolorsSort[14][2], WHITE,
448    allcolorsSort[4][2], allcolorsSort[18][2], allcolorsSort[15][2], allcolorsSort[10][2])
```

Abb.: 109 Code Auswertung

Sämtliche Felder werden nun der jeweiligen Seite und Position zugeordnet. In dieser Zuordnung haben bereits 24 der 27 Felder die richtige Farbe. Diese Zuordnung passiert, damit die verdeckten Steine extrahiert werden können, da diese Filterung von den Nachbarsteinen abhängt.
Der genauere Ablauf der Filterung der verdeckten Steine wird im Kapitel, „3.5.6 Code Farbfilter geteilte Steine" behandelt.

```
538   """Zuweisen der Felder an die jeweiligen Seiten in der richtigen Reihenfolge,
539    mit allen Farben aller Felder"""
540   r = (allcolorsSort[7][2], allcolorsSort[5][2], allcolorsSort[1][2], allcolorsSort[8][2],
541    GREEN, allcolorsSort[2][2], allcolorsSort[9][2], allcolorsSort[6][2], hiddenR)
542   f = (allcolorsSort[19][2], allcolorsSort[16][2], allcolorsSort[11][2], allcolorsSort[20][2],
543    BLUE, allcolorsSort[12][2], hiddenF, allcolorsSort[17][2], allcolorsSort[13][2])
544   u = (hiddenU, allcolorsSort[3][2], allcolorsSort[0][2], allcolorsSort[14][2], WHITE,
545    allcolorsSort[4][2], allcolorsSort[18][2], allcolorsSort[15][2], allcolorsSort[10][2])
546
547   print("rechts", r)
548   print("vorne", f)
549   print("unten", u)
550
551   cv2.imshow("Ecken",   ImgOrig)
552   cv2.waitKey(0)
```

Abb.: 110 Code Auswertung

Nach der erneuten Filterung der verdeckten Steine werden die Felder endgültig ihren Variablen zugewiesen und ausgegeben, zusätzlich wird das finale Bild der Farberkennung ausgegeben.

```
rechts (0, 1, 2, 2, 4, 2, 5, 1, 1)
vorne (2, 1, 4, 3, 2, 1, 3, 5, 3)
unten (4, 2, 5, 0, 5, 5, 3, 0, 3)
```

Abb.: 111 Code Auswertung

Die Ausgabe wird an den Lösungsalgorithmus in dieser Form weitergegeben.

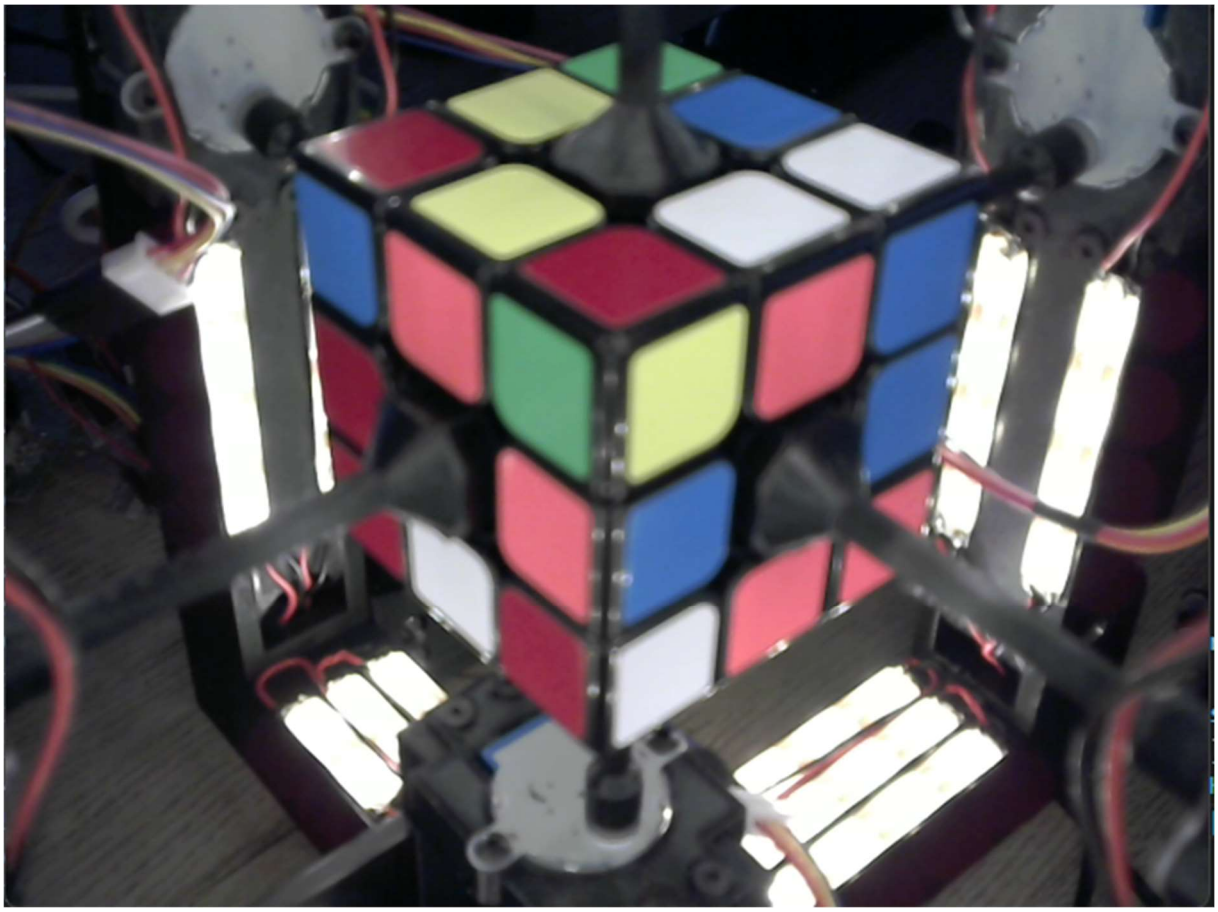


Abb.: 112 Originalbild Kamera

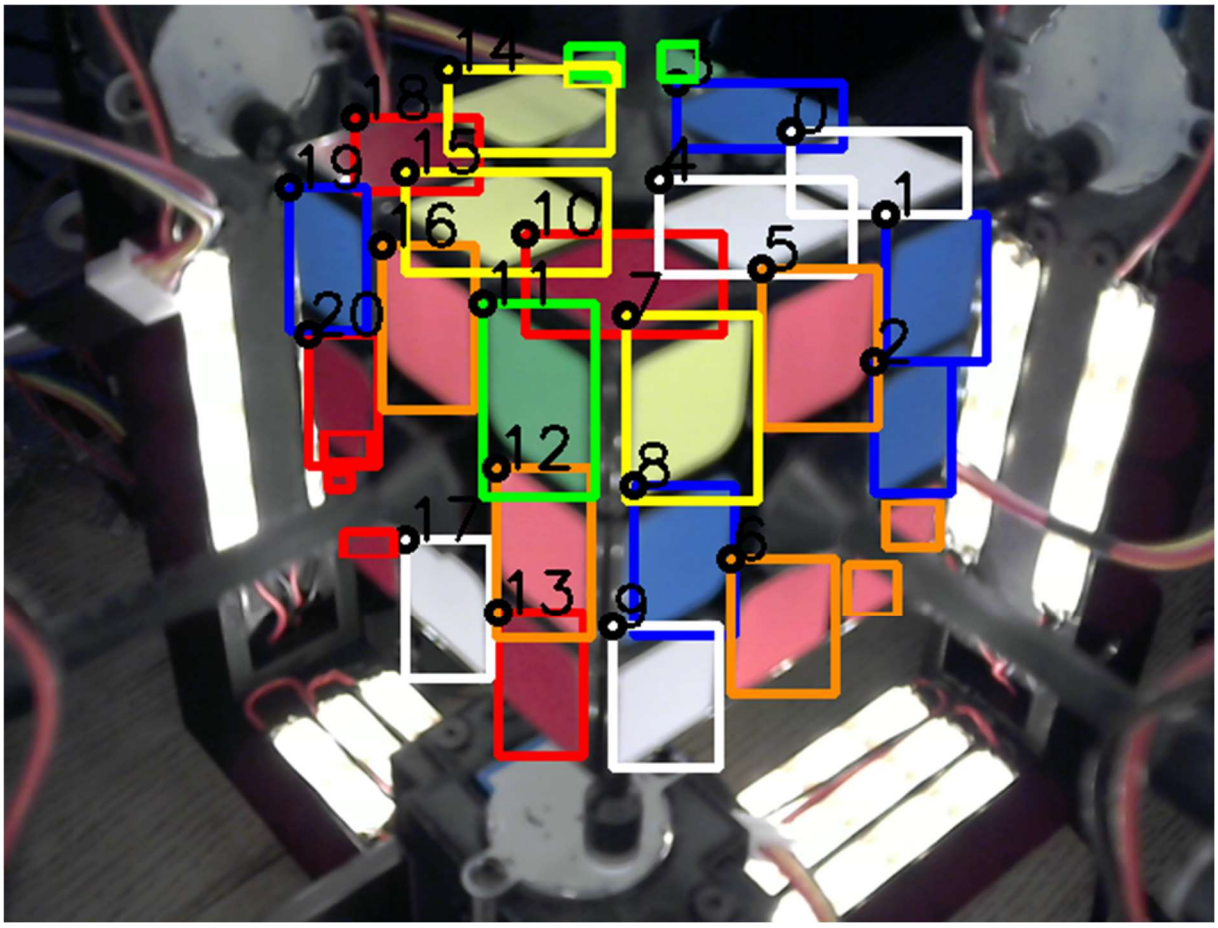


Abb.: 113 Ausgewertetes Bild der Farberkennung

Diese Vorgehensweise betrifft auch die untere Kamera, somit wird auf diese nicht weiter eingegangen.

4 VERWENDETE FUNKTIONEN FARBERKENNUNG

cv2.boundingRect():	wird verwendet, um ein ungefähres Rechteck zu zeichnen.
Syntax:	cv2.boundingRect(points)
Parameter:	
points:	2D-Punktsatz, gespeichert in einem Vektor
Rückgabewert:	X, Y Koordinate des linken oberen Ecks des Rechtecks, sowie Breite und Höhe des Rechtecks

cv2.circle():	wird verwendet, um einen Kreis zu zeichnen
Syntax:	cv2.circle(image, center_coordinates, radius, color, thickness)
Parameter:	
image:	das zu bearbeitende Bild
center_coordinates:	die Koordinaten geben das Zentrum des Kreises an, müssen in X und Y Werten angegeben werden
radius:	gibt den gewünschten Radius des Kreises an
color:	gibt die Farbe des Kreises mittels BGR-Farbraums an. Zum Beispiel: (255,0,0) für die Farbe Blau
thickness:	gibt die Stärke der Linie in Pixel an. Bei -1 wird das Rechteck mit der gewählten Farbe ausgefüllt
Rückgabewert:	gibt ein Bild zurück

cv2.contourArea():	wird dazu verwendet, die Größe der gefundenen Konturen festzustellen
Syntax:	cv2.contourArea(contours)
Parameter:	
contours:	eine Liste oder Baum von Listen, von Punkten

cv2.cvtColor():	wird verwendet, um das Bild von einem Farbraum in einen anderen zu konvertieren
Syntax:	cv2.cvtColor(src, code[, dst[, dstCn]])
Parameter:	
src:	das Bild das verändert werden soll
code:	ist der Farbraumänderungscode
dst:	ob das Bild, das zurückkommt, dieselbe Größe und Tiefe wie das Originalbild hat (optional)

dstCn:	es ist die Nummer der Kanäle des Bildes. Bei 0 werden die Kanäle des Originalbildes verwendet
Rückgabewert:	gibt ein Bild zurück

cv2.findContours():	Hilft dabei, Konturen in einem Bild zu erkennen. Konturen sind alle Punkte, die ununterbrochen an einer Linie oder an Flächen anschließen, die dieselbe Intensität und Farbe haben. Am besten funktioniert die Erkennung bei Binärbildern. Zum Finden der Konturen wird der Algorithmus „Satoshi Suzuki and others. Topological structural analysis of digitized binary images by border following. Computer Vision, Graphics, and Image Processing, 30(1):32–46, 1985“ angewendet.
Syntax:	cv.findContours(image, mode, method[, contours[, hierarchy[, offset]]])
Parameter:	
images:	das zu untersuchende Binärbild
contours:	erkannte Konturen, jede Kontur ist als Vektor von Punkten gespeichert
hierarchy:	beinhaltet Information zur Topologie des Bildes (optional).
mode:	Modus zum Abrufen von Konturen
method:	Konturnäherungsverfahren
offset:	offset, um den jeder Konturpunkt verschoben wird(optional)
Rückgabewert:	eine Liste oder ein Baum von Listen und die Hierarchie zeigt an, wie die Formen miteinander verbunden sind

cv2.imread():	Lädt ein Bild von einem bestimmten Pfad. Wenn das Bild nicht gelesen werden kann, gibt diese Methode eine leere Matrix zurück.
Syntax:	cv2.imread(path, flag)
Parameter:	
path:	ein String der den Pfad zum lesenden Bild angibt
flag:	gibt an in welcher Art das Bild eingelesen wird. Der Standardwert ist cv2.IMREAD_COLOR
Rückgabewert:	gibt das Bild von dem Pfad zurück

cv2.imshow():	Wird verwendet, um ein Bild in einem Fenster darzustellen. Das Fenster passt sich automatisch der Größe des Bildes an.
Syntax:	cv2.imshow(window_name, image)
Parameter:	
window_name:	ein String, der den Namen des Fensters darstellt
image:	ist das Bild, das dargestellt werden soll
Rückgabewert:	gibt nichts zurück

cv2.imwrite():	Wird benutzt, um ein Bild abzuspeichern. Das Bild wird im zurzeit arbeiteten Ordner abgespeichert.
Syntax:	cv2.imwrite(filename, image)
Parameter:	
filename:	ein String, der den Dateiennamen angibt. Der Dateienname muss das Dateienformat beinhalten. (.jpg, .png, etc.
image:	ist das zu speichernde Bild
Rückgabewert:	gibt bei erfolgreicher Speicherung „True" zurück

cv2.inRange():	Wird verwendet, um in einem HSV-Bild, jeden einzelnen Pixel darauf zu kontrollieren, ob es innerhalb des oberen und unteren Schwellenwertes liegt. Sollte es innerhalb des Wertebereichs liegen, wird der Pixel auf „1" gesetzt, anderenfalls auf „0".
Syntax:	inRange(sourcearray, upperboundarray, lowerboundarray)
Parameter:	
sourcearray:	das zu untersuchende Bild
upperboundarray:	ist ein Array, das die Werte der oberen Grenze beinhaltet
lowerboundarray:	ist ein Array, das die Werte der unteren Grenze beinhaltet
Rückgabewert:	gibt ein Binärbild zurück

cv2.namedWindow():	Wird verwendet, um ein Fenster zu erzeugen, um Bilder und Videos anzuzeigen. Die Größe des Fensters passt sich automatisch der Größe der Quelle an.
Syntax:	cv2.namedWindow(window_name, flag)
Parameter:	
window_name:	gibt den Namen des Fensters an
flag:	gibt an, ob die Größe des Fensters automatisch bezogen werden soll oder fix voreingestellt ist
Rückgabewert:	gibt nichts zurück

cv2.putText():	schreibt einen Text auf das Bild
Syntax:	cv2.putText(image, text, org, font, fontScale, color[, thickness[, lineType[, bottomLeftOrigin]]])
Parameter:	
image:	das zu bearbeitende Bild
text:	den gewünschten Text eingeben
org:	gibt die Position des Textes an. Die Koordinaten müssen in X und Y Werten angegeben werden (Startpunkt: links unten)
font:	den gewünschten Fontstyle angeben. Zum Beispiel: FONT_HERSHEY_SIMPLEX
fontScale:	Faktor für die Schriftgröße
color:	Schriftfarbe
thickness:	Dicke der Schrift
lineType:	gibt den Typ der Linie vor (optional)
bottomLeftOrigin:	wenn „True", ist der Ursprung links unten, anderenfalls links oben (optional)
Rückgabewert:	gibt ein Bild zurück

cv2.rectangle(): zeichnet Rechtecke

Syntax:	cv2.rectangle(image, start_point, end_point, color, thickness)
Parameter:	
image:	das zu bearbeitende Bild
start_point:	gibt Startpunkt des Rechtecks an (links oben)
end_point:	gibt Endpunkt des Rechtecks an (rechts unten)
color:	gibt die Farbe des Rechteckes mittels BGR-Farbraums an. Zum Beispiel: (255,0,0) für die Farbe Blau
thickness:	gibt die Stärke der Linie in Pixel an. Bei -1 wird das Rechteck mit der gewählten Farbe ausgefüllt
Rückgabewert:	gibt ein Bild zurück

cv2.waitKey():	Erlaubt es den Nutzern ein Fenster anzuzeigen. Entweder mittels einem Zeitlimit, in Millisekunden, oder bis eine Taste gedrückt wird. Bei „0" wird auf einen Tastendruck gewartet.
cv2.waitKey(5000):	das Fenster bleibt für 5 Sekunden geöffnet und schließt sich nach Ablauf der Zeit automatisch
cv2.waitKey(0)	wartet auf einen Tastendruck, es ist irrelevant, welche Taste gedrückt wird

numpy.array():	wird verwendet, da Python keine Arrays hat, sondern nur Listen. Gewisse Funktionen von OpenCV benötigen aber ein Array, z.B. cv2.inRange()

5 ZUSAMMENFASSUNG UND AUSBLICK

Diese Projektarbeit hatte ein sehr ambitioniertes Ziel und daher stellte es die Studierenden vor eine große Herausforderung. Das Ziel der Diplomarbeit wurde erfüllt, jedoch unter größten Bemühungen und einiger Rückschläge.

Im Zuge dieses Projekts, wurde die Wichtigkeit des V-Modells klar, denn durch die selbständige Planung und Auslegung der Komponenten wurden, von den Studierenden, einige Fehler begangen. Diese Fehler hätten auf Industrieebene zu gravierenden Mehrkosten geführt. Da die bereits bestellten Servomotoren nicht die benötigte und gewünschte Genauigkeit aufzeigten, mussten diese zur Gänze ausgetauscht und durch Schrittmotoren ersetzt werden. Dadurch musste zusätzlich der Code, der Motorsteuerung, und die bestehende Hardware überarbeitet werden.

Damit der Lösungsweg ausprogrammiert werden konnte, musste der Hauptverantwortliche, für den Lösungsalgorithmus, die Logik des Würfels erlernen. Dies erforderte einen großen Rechercheaufwand im Internet.

Die Entscheidung fiel auf die Programmiersprache Python, da sie eine intuitiv und relativ leicht zu erlernende Sprache ist. Der Vorteil ist, dass das Programm zur Bearbeitung der Bilder, OpenCV, komplett mit Python kompatibel ist und es so zu keinem Mehraufwand durch Schnittstellen kommt.

Sämtliche Störfaktoren wie schulische Aufgaben wurden beim Festlegen der Meilensteine nicht berücksichtigt. Somit konnten manche Meilensteine nicht eingehalten werden.

Was die Studierenden in ihr Berufsleben mitnehmen werden:

- dass es von sehr großem Vorteil ist vor dem Projektstart eine genaue und ausführliche Planung zu haben, um Mehraufwand und Mehrkosten zu vermeiden,
- eine genaue Definition der Schnittstellen unter Abteilungen geben muss,
- um eine Zeitplanung richtig erstellen zu können, müssen im Voraus so viele Störfaktoren wie möglich bekannt sein und berücksichtigt werden,

Verbesserungen, die noch gemacht werden könnten:

- Stärkere Schrittmotoren
- Sicherheitsschleifen
- Kompaktere Bauweise
- Erweiterung durch ein Display mit Benutzeroberfläche

6 ABBILDUNGSVERZEICHNISSE

Abb.:1 Legende Projektstrukturplan 3
Abb.: 2 Projektstrukturplan Projektmanagement 3
Abb.: 3 Projektstrukturplan Hardware und Bildverarbeitung 4
Abb.: 4 Projektstrukturplan Lösungssoftware und Motorsteuerung 5
Abb.: 5 Projektstrukturplan Zusammenführung von HW, SW und schriftliche Dokumentation 6
Abb.: 6 Gesamtkostenaufstellung 7
Abb.: 7 Schrittmotor 10
Abb.: 8 Bauformen Schrittmotor 10
Abb.: 9 Reluktanzschrittmotor 11
Abb.: 10 Permanentschrittmotor 11
Abb.: 11 Hybridschrittmotor 11
Abb.: 12 Pinbelegung RaspberryPi 4B 14
Abb.: 13 ULN2003 Treiber Board 16
Abb.: 14 Webcam Logitech C270 16
Abb.: 15 Grundplatte 17
Abb.: 16 Deckel Grundplatte 18
Abb.: 17 Seitenarm unten Kamera 18
Abb.: 18 Halterung Kamera unten 19
Abb.: 19 Seitenarm unten ohne Kamera 19
Abb.: 20 Befestigung Kamera unten 20
Abb.: 21 Säule beweglich 20
Abb.: 22 Säule fix 21
Abb.: 23 Halterung Kamera oben 21
Abb.: 24 Befestigung Kamera oben 22
Abb.: 25 Knick oben starr 22
Abb.: 26 Halterung oben 23
Abb.: 27 Halterung Kreuz 23
Abb.: 28 Kreuz 24
Abb.: 29 Halterung Würfel 24
Abb.: 30 Ansicht unten 24
Abb.: 31 Ansicht oben 24
Abb.: 32 Adapter Motorhalterung 25
Abb.: 33 Maschinengerüst komplett 25
Abb.: 34 Optischer Zauberwürfellöser geschlossen 26
Abb.: 35 Optischer Zauberwürfellöser offen 26
Abb.: 36 Creality Ender 3 V2 27
Abb.: 37 LED Band 28

Abb.: 38 Schaltplan Motorsteuerung 30
Abb.: 39 Mantelfläche Zauberwürfel 33
Abb.: 40 Mantelfläche codiert 33
Abb.: 41 Code Move: Import 34
Abb.: 42 Code Move: Initialisierungsschritt 35
Abb.: 43 Code Move: Methode do_U (1) 35
Abb.: 44 Code Move: Methode do_U (2) 36
Abb.: 45 Code Move: Methode do_U (3) 36
Abb.: 46 Code Move: Methode do_2U 37
Abb.: 47 Code Set_Surface: Initialisierung 37
Abb.: 48 Code First_Layer_Edges: Import 38
Abb.: 49 Code First_Layer_Edges: anlegen der Objekte r und f 38
Abb.: 50 Code First_Layer_Edges: anlegen des Objektes M 38
Abb.: 51 Code First_Layer_Edges: Methode solve_blue_white_front 38
Abb.: 52 Code First_Layer_Edges: Methode blue_white 39
Abb.: 53 Code First_Layer_Edges: Methode first_layer_edge_solve 39
Abb.: 54 Code First_Layer_Edges: Methode print_result 40
Abb.: 55 Code First_Layer_Corners: Import 40
Abb.: 56 Code First_Layer_Corners: blue_white_orange_front 40
Abb.: 57 Lösungsmuster OLL 41
Abb.: 58 Code OLL: Ausschnitt von Methoden 42
Abb.: 59 Code OLL: Methode oll_solve 42
Abb.: 60 Lösungsmuster PLL_Edges 43
Abb.: 61 Code PLL_Edges: Methode red_front_solve 43
Abb.: 62 Lösungsmuster PLL_Corners(1) 44
Abb.: 63 Lösungsmuster PLL_Corners(2) 44
Abb.: 64 Code PLL_Corners: Methode solve_blue_red 44
Abb.: 65 Code Cube_Solve: Import 45
Abb.: 66 Code Cube_Solve: Methode solve_cube 45
Abb.: 67 Code Cube_Solve: Abfrage weiße Seite (1) 45
Abb.: 68 Code Cube_Solve: Abfrage weiße Seite (2) 45
Abb.: 69 Code Cube_Solve: Methoden solve_cube und GPIO_cleanup 45
Abb.: 70 Code Stepper Modul: Import 46
Abb.: 71 Code Stepper Modul: ausführen der Methode GPIO.setmode 46
Abb.: 72 Pinbelegung RaspberryPi 4B 47
Abb.: 73 Code Stepper Modul: Liste control_pins_m 48
Abb.: 74 Code Stepper Modul: GPIO Zuweisung 48
Abb.: 75 Code Stepper Modul: Liste step_seq_right 49
Abb.: 76 Code Stepper Modul: Methode LEFT_TURN 49
Abb.: 77 HSV-Farbraum 52
Abb.: 78 Code Farberkennung Initialisierung 52

Abb.: 79 Code Farberkennung Initialisierung 52
Abb.: 80 Code Kalibrierung 53
Abb.: 81 Trackbars Kalibrierung 54
Abb.: 82 Code Kalibrierung 54
Abb.: 83 Originalbild ohne Filter 55
Abb.: 84 HSV-Filter Farbe blau 55
Abb.: 85 Code Bildaufnahme 56
Abb.: 86 Code Bildaufnahme 56
Abb.: 87 Code Bildaufnahme 57
Abb.: 88 Originalbild 57
Abb.: 89 HSV-Bild 57
Abb.: 90 Code Farbfilter ganze Ecken 58
Abb.: 91 Code Farbfilter ganze Ecken 58
Abb.: 92 Binärbild der Funktion „cv2.inRange()", auf Rot gefiltert 58
Abb.: 93 Originalbild 58
Abb.: 94 Code Farbfilter ganze Ecken 59
Abb.: 95 mit Filterung 59
Abb.: 96 ohne Filterung 59
Abb.: 97 Code Farbfilter geteilte Steine 60
Abb.: 98 Code Farbfilter geteilte Steine 61
Abb.: 99 Code Farbfilter geteilte Steine 61
Abb.: 100 Code Farbfilter geteilte Steine 62
Abb.: 101 Code Farbfilter geteilte Steine 63
Abb.: 102 Code Auswertung 63
Abb.: 103 Code Auswertung 64
Abb.: 104 Code Auswertung 64
Abb.: 105 Code Auswertung 64
Abb.: 106 Sortierung nach Y-Koordinaten 65
Abb.: 107 Code Auswertung 65
Abb.: 108 Code Auswertung 66
Abb.: 109 Code Auswertung 66
Abb.: 110 Code Auswertung 66
Abb.: 111 Code Auswertung 67
Abb.: 112 Originalbild Kamera 67
Abb.: 113 Ausgewertetes Bild der Farberkennung 68

7 LITERATURVERZEICHNIS

[1] „learnchannel-tv.com," 13 März 2022. [Online]. Available: https://learnchannel-tv.com/de/drives/stepper-motor/.

[2] „goetz-automation.de," 13 März 2022. [Online]. Available: http://www.goetz-automation.de/Schrittmotor/KenngroessenSM.htm.

[3] „tutorials-raspberrypi.de," 13 März 2022. [Online]. Available: https://tutorials-raspberrypi.de/top-21-raspberry-pi-betriebssysteme-fuer-dein-projekt/.

[4] „wikipedia.org," 13 März 2022. [Online]. Available: https://de.wikipedia.org/wiki/HSV-Farbraum.

8 ANHANG (INHALT USB-STICK)

8.1 USB-Stick