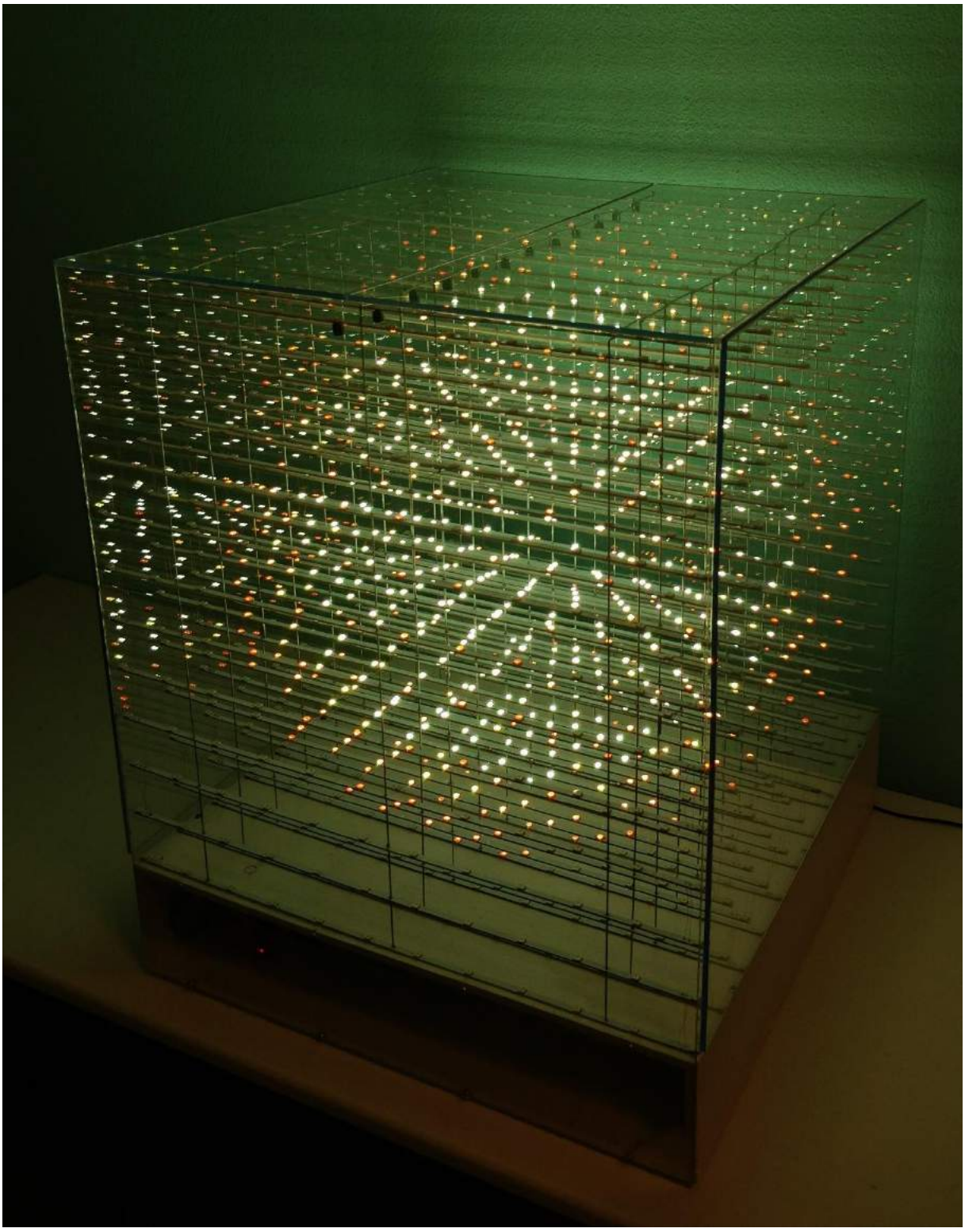
LED-Würfel



Maturitätsarbeit 2020 an der Kantonsschule Zürcher Oberland von Felix Roeck (C6a),
betreut von Oliver Seipel

Inhalt

1 Einleitung und Motivation

In meiner Maturitätsarbeit geht es um den Aufbau und die Programmierung eines LED-Würfels. In der Anordnung mit den Massen 50 x 50 x 50 cm^3, mit zwölf Ebenen à 144 RGB LEDs (5184 einzelne LEDs (light-emitting diodes)), sowie einem Raspberry Pi zur Ansteuerung der LEDs, können durch Anpassung der Farbe und Helligkeit der LEDs beliebige dreidimensionale Bilder erzeugt werden. Werden solche Bilder nacheinander in einem regelmässigen Zeitintervall generiert, entsteht eine Animation.

Das Bauen des LED-Würfels und das anschliessende Programmieren der Animation ermöglichte es mir, erste Erfahrungen im Bereich Elektronik und erweiterte Erkenntnisse in Python zu erwerben. Die monatelange Auseinandersetzung mit der Thematik der Maturitätsarbeit half mir bei meiner zukünftigen Studienwahl: So standen die Bereiche Hardware, Software und Design der Animation für die Studiengänge Elektrotechnik, Informatik und Game Design. Mein absoluter Favorit ist in der Zwischenzeit das Studienfach Elektrotechnik. Bei der Wahl der Maturitätsthematik ging es mir sowohl um die praktische Anwendung von Konzepten (wie z.B. um das Programmieren, die Vektorgeometrie, die Trigonometrie, die Kurvendiskussion und die Elektrizität) als auch um die Anwendung unterschiedlichster zuvor genannter Themenbereiche wie der Hardware, der Software und dem Design der Animation. Ausserdem konnte ich persönlich die Erfahrung machen wie grundlegend ausdauerndes, genaues und zuverlässiges Arbeiten ist.

Im ersten Teil der Arbeit wird der Bau des LED-Würfels und einzelne Bauteile beschrieben. Im zweiten Teil geht es um das Python-Programm, welches eine Animation auf dem LED-Würfel abgespielt.

2 Hardware

In diesem Kapitel wird detailliert die Struktur und der Bau des LED-Würfels beschrieben, wobei auf aufgetretene Probleme im Einzelnen eingegangen wird.

2.1 Raspberry Pi

Der Raspberry Pi wird zur Steuerung des LED-Würfels verwendet. Ein Raspberry Pi ist ein «Mini-Computer» mit einem Linux Betriebssystem. Der Vorteil gegenüber einem herkömmlichen Computer ist dessen Grösse, die ca. einer Zigarettenschachtel entspricht. Ausserdem weist das Betriebssystem sehr viele Freiheiten auf. Der Raspberry Pi bietet vielseitige Verwendungsmöglichkeiten für Einsteiger, um beispielsweise zu programmieren und mit dem Aufbau eines «einfachen» Computers Erfahrungen zu sammeln. Er wurde ursprünglich für die Lehre an Schulen und Universitäten entwickelt, erfreut sich heute aber auch einer grossen Beliebtheit bei Hobbyisten und kann handelsüblich erworben werden. Das handliche Format hat aber auch Nachteile: Im Vergleich zu herkömmlichen Computern hat der Raspberry Pi eine kleinere Rechenkapazität. Das führte in diesem Projekt zu Problemen. Verwendet wurde der Raspberry Pi 3b+. https://www.raspberrypi.org/

2.1.1 Setup

Der Raspberry Pi aus dem Starterpaket wird wie folgt in Betrieb genommen:

Micro SD-Karte mit NOOBS in SD-Slot einstecken
Raspberry Pi mit Maus, Tastatur und Bildschirm verbinden
Raspberry Pi an Strom anschliessen
Automatisch aufgerufener Installationsprozess durchführen (Raspbian, Wifi, Land, Sprache auswählen)
Terminal (Shell) aufrufen
sudo apt-get update; sudo apt-get upgrade (aktuellste SW-Versionen installieren)
Python 3 im Startmenu aufrufen

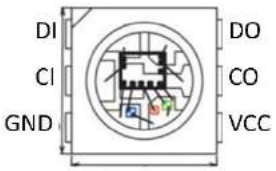
2.2 Architektur und Alternativen

Im Internet sind unzählige Bauanleitungen für LED-Würfel zu finden, diese benötigen aber meist eine komplizierte Elektronik, welche eine Matrize ansteuert. Um das ganze Farbspektrum darzustellen, sind drei LEDs: blaue, rote und grüne, notwendig. Mit einer grossen Anzahl von 5184 (=12^3*3) LEDs wäre der Bau der Elektronik ein Albtraum.

Nach weiteren Recherchen findet sich die Idee einer einfacheren Architektur, indem einzeln adressierbare LEDs in Serie mit einem SPI-Bus direkt vom Raspberry Pi angesteuert werden. Dazu eignet sich eine schmale, längliche Leiterplatte (=PCB), auf der zwölf LEDs angelötet sind, wodurch sie eine Reihe des Würfels bilden. Mehr zu den PCBs im Kapitel 2.3. Statt einen Draht zu verwenden, macht ein PCB den Würfel zusätzlich mechanisch stabiler. 144 PCBs bilden, elektrisch in Serie geschaltet, den gesamten Würfel.

2.2.1 Einzeln adressierbare LEDs

Um LEDs einzeln ansteuern zu können, also die Farbe und Helligkeit der individuellen LEDs zu kontrollieren, ist ein Chip nötig, welcher die digitalen Signale eines Kommunikationsbusses vom Raspberry Pi auswertet. Praktischerweise sind die LEDs als RGB-Set (RGB = Rot, Grün, Blau) in einem Gehäuse erhältlich, damit alle Farben im Farbspektrum dargestellt werden können. Der Kommunikationsbus ist ein sogenannter SPI-Bus und wird im nächsten Kapitel näher erläutert. Bei der Auswahl einer LED ist zu beachten, ob man eine 12V- oder 5V-LED verwendet. Vergleicht man 12V-LEDs mit 5V-LEDs weisen erstere einen halb so grossen Spannungsverlust bei gleicher Leistung auf. Bei diesem Projekt ist dies kein Problem, weshalb eine 5V-LED ausreicht. Die LED APA102 erfüllt alle diese Bedingungen und ist dazu relativ kostengünstig, deshalb wurde sie für dieses Projekt ausgewählt. Vergleiche solcher LEDs finden sich im Internet, z.B. hier: https://quinled.info/2019/06/02/what-digital-led-chip-to-choose/



5mm

DI	Data-Input
DO	Data-Output
CI	Clock-Input
CO	Clock-Output
GND	-
VCC	+5V

Quellenangabe Abb1[1]

2.2.2 SPI

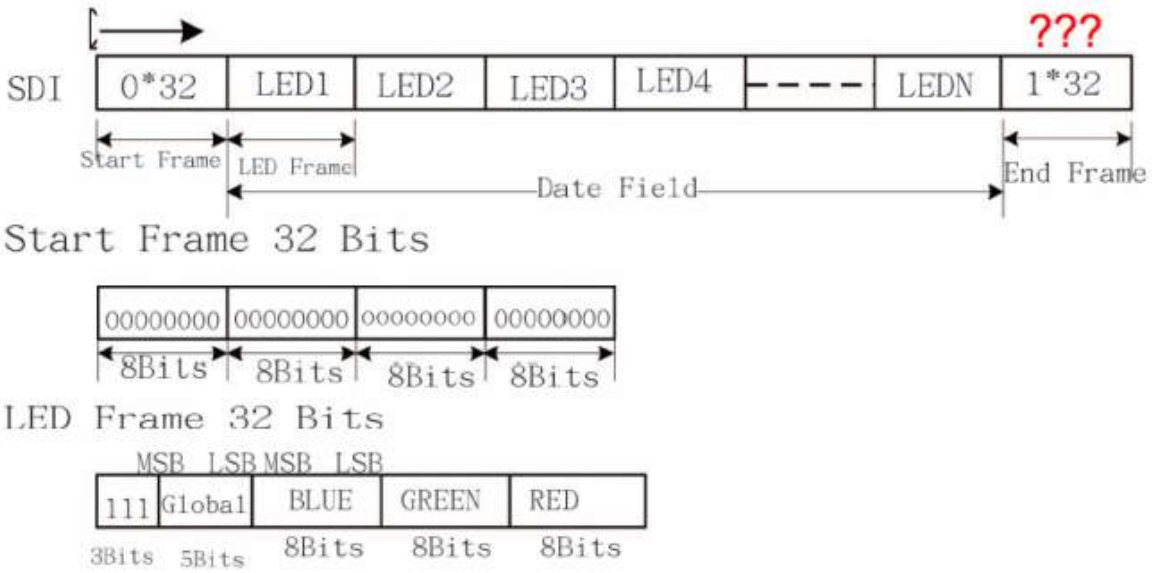
SPI ist ein Bus-System und steht für die Abkürzung «Serial Peripheral Interface». Sie ist eine Methode, Daten seriell zwischen verschiedenen, sogenannten Bus-Teilnehmern zu senden. Man kann sich dies folgendermassen vorstellen: Daten steigen an einer Haltestelle (=erster Chip) in einen Bus ein und bei einer anderen Haltestelle (=zweiter Chip) wieder aus. Daher auch der Begriff des Bus-Systems. Ein SPI funktioniert nach dem Master-Slave Prinzip, d.h. ein Gerät ist der Master und bestimmt das Timing (=den Fahrplan) und einer oder mehrere sind die Slaves. Der Master sendet ein Clock-Signal (=SCK) und Daten (MOSI, Master Out – Slave In) an die Slaves und erhält Daten via eine separate Leitung (MISO, Master In – Slave Out) zurück. Bei dem LED-Würfel ist der Raspberry Pi der Master, und die LEDs sind die Slaves. Da die LEDs keine Daten zurücksenden, wird die MISO-Leitung nicht verwendet.

Der Raspberry Pi sendet die Signale mit einer Spannung von 0 Volt bzw. 5 Volt aus, die Spannungen bezeichnen ein binäres Daten-Signal (0 oder 1) mit Anweisungen an die LEDs. Daneben wird ein Clock-Signal ausgesendet, das mit einem regelmässigen 0-1-0-1-0-1 …-Intervall von Spannungen die Zeit angibt. Die Daten werden nur bei einem 0-1 Übergang von der LED gelesen. Das Data-Signal beginnt mit einem Start-Frame von 32*0 Bits. Dieses ist notwendig, damit das Start-Frame, im Zusammenhang mit den drei führenden 1-Bits eines LED-Frames (siehe unten), der LED mitteilt, dass dieses nächste Daten-Frame ihr gehört. Gleichzeitig gibt diese LED während dem Lesen des Daten-Frames ein weiteres Start-Frame (alle Bit = 0) an die nächste LED weiter. Alle weiteren Daten-Frames werden danach normal weitergegeben. Als nächstes folgt das LED-Frame mit den Anweisungen an die LEDs: 32 Bits für jede LED. Am Schluss folgt das End-Frame mit (Anzahl LEDs/2) * 0 Bits.

[1]Quellenangabe Abb1: https://cdn-shop.adafruit.com/datasheets/APA102.pdf, Zugriff: 17.10.2019

Da die LED das Clock-Signal invertiert, um Zeitverzögerungen bei der Weitergabe der Daten an die nächste LED auszugleichen, fehlen nach allen Daten eine Anzahl Clock-Zyklen, welche mit dem End-Frame ausgeglichen werden. Details dazu findet man auf dem Datenblatt der APA102 sowie im Artikel: https://cpldcpu.wordpress.com/2014/11/30/understanding-the-apa102-superled/

Data-Signal an die LED APA102:



Quellenangabe Abb2[2]

2.3 Ausführung

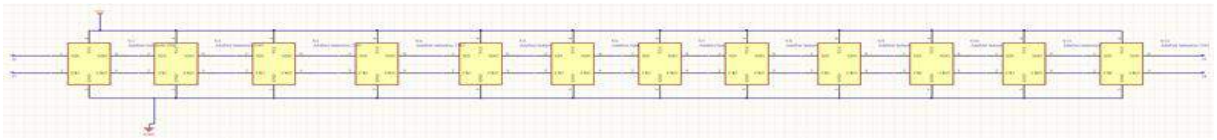
2.3.1 Design des PCBs

PCB ist die Abkürzung für «Printed Circuit Board». PCBs, auf Deutsch Leiterplatten, lassen sich in jedem elektronischen Gerät finden. Sie dienen dazu, verschiedene elektronische Bauteile elektrisch zu verbinden und diese mechanisch zu befestigen.

Ein einfach zu bedienendes und lizenzfreies Programm, dass sich zum Designen eines PCBs eignet, ist das Programm «CircuitMaker». Es verfügt über ein Online-Archiv mit den Designs aller User für jeden öffentlich zugänglich. Deshalb lässt sich das PCB-Design dieser Arbeit auch dort finden: https://workspace.circuitmaker.com/User/Details/Felix-Roeck

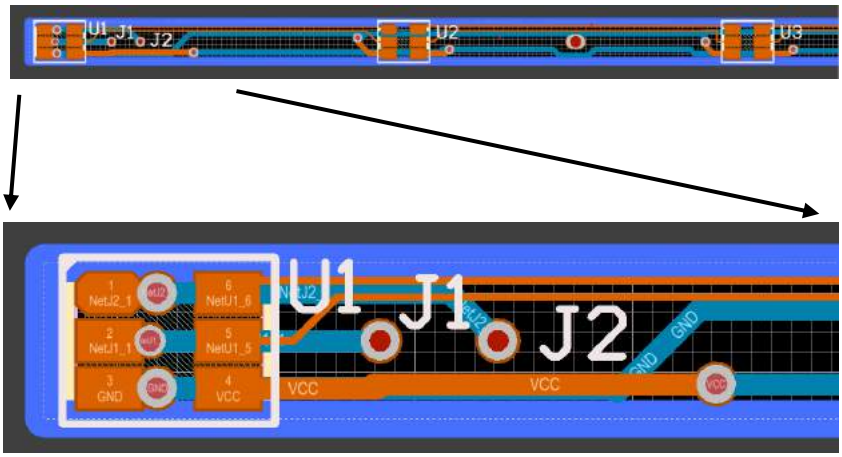
Der PCB hat die Aufgabe den Clock-Output mit dem Clock-Input der nächsten LED zu verbinden, indem Leiterbahnen, dünne Schichten von Kupfer, eine elektrische Verbindung zwischen zwei Lötflächen bereitstellen. Auf die Lötflächen kann eine LED angelötet werden. Der PCB verbindet ausserdem den Ground und die Supply-Voltage der LEDs jeweils parallel. Das unten zu sehende Schema veranschaulicht die oben genannten Vernetzungen der LEDs:

[2]Quellenangabe Abb2: https://cpldcpu.wordpress.com/2014/11/30/understanding-the-apa102-superled/, Zugriff: 17.10.2019



Um den Ground und die Supply-Voltage aller PCBs zu verbinden, braucht es in jedem PCB Löcher, welche an die Supply-Voltage oder den Ground aller LEDs des jeweiligen PCBs angeschlossen sind. Durch diese Löcher können Drähte durchgestossen werden, um alle LEDs an die Supply-Voltage mit 5V Spannung oder den Ground anzuschliessen. Das Loch für die Supply-Voltage befindet sich in der Mitte des PCBs und die zwei Löcher für den Ground befinden sich jeweils 18 cm von der Mitte entfernt. Dies, um eine genügende mechanische Stabilität der Konstruktion zu erreichen, ohne den Blick auf die LEDs zu stark einzuschränken.

Ansicht des PCBs für die Leiterbahnen der ersten drei LEDs:



U1	Erste LED
NetJ2_1	Data-Input
NetJ1_1	Clock-Input
NetU1_6	Data-Output
NetU1_5	Clock-Output
GND	Ground
VCC	5V Supply-Voltage
Blaue Leiterbahn	Befindet sich auf der Vorderseite des PCBs
Orange Leiterbahn	Befindet sich auf der Rückseite des PCBs
J2 	Loch, um den Data Input mit dem Data Output der letzten LED des vorherigen PCBs zu verbinden
	Leiterbahn geht von der Vorderseite zur Rückseite

Der fertig entworfene PCB kann anschliessend bei einem Leiterplattenhersteller bestellt werden. Dazu exportiert man mit CircuitMaker den Leiterplattenentwurf in ein Set von Dateien, welche dem ‘Gerber’-Format entsprechen. Diese Dateien übermittelt man einem Leiterplattenhersteller z.B. übers Internet bei www.pcbway.com. Nach ein paar Wochen erhält man die PCBs via Post und kann die Bauteile darauf anlöten.

2.3.2 Löten

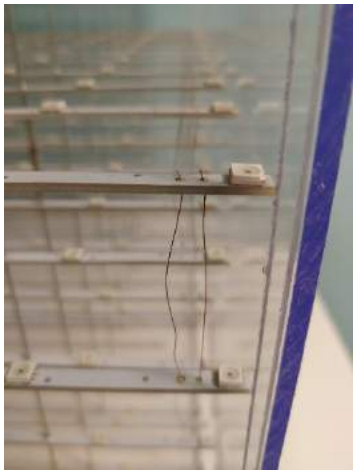
Nachdem alle Einzelteile ausgewählt und gekauft sind, wird der Würfel zusammengebaut. Entweder sind die LEDs beim Kauf der PCBs bereits angelötet oder müssen es noch werden. Bei diesem Projekt sind alle 1728 RGB-LEDs per Hand angelötet worden: für jede LED sind sechs elektrische Kontakte notwendig (Data-Input, Data-Output, Clock-Input, Clock-Output, Supply-Voltage, Ground). 40 Stunden Aufwand sind für diese sehr eintönige und grosses Durchhaltevermögen fordernde Arbeit notwendig.

Es empfiehlt sich mit einem Test vor der Weiterverwendung der LED-Streifen sicher zu stellen, dass es weder Kurzschlüsse, kalte Lötstellen - welche keinen Kontakt herstellen -, noch defekte PCBs oder LEDs gibt. Dazu werden die PCBs mit dem Raspberry Pi verbunden und ein einfaches Programm abgespielt, welches alle LEDs weiss aufleuchten lässt.

Nachdem die einzelnen Reihen des Würfels (=PCBs) gebaut wurden, ist der nächste Schritt die Ebenen herzustellen. Damit alle Ebenen exakt gleich gross sind und derselbe Abstand zwischen den Reihen besteht, wird ein Gerüst aus Holz gebaut, in den die PCBs hineingelegt werden.

Alle LEDs benötigen elektrischen Kontakt zum Ground und zur Supply-Voltage. Dafür weisen alle PCBs drei Löcher auf: einen für 5V und zwei für den Ground, welche mit allen zwölf LEDs parallelgeschaltet sind. Durch diese Löcher werden verzinkte Eisendrähte mit einer Länge von 60 cm geschoben und angelötet. Verzinkte Eisendrähte werden benutzt, weil sie mechanisch stabiler als Kupfer sind, aber trotzdem einen relativ kleinen spezifischen Widerstand aufweisen. Ein niedriger spezifischer Widerstand ist wichtig, damit die Drähte bei einem starken Strom nicht heiss werden bzw. einen zu hohen Spannungsabfall bei den LEDs verursachen. Dies wurde gerade noch so erreicht, d.h. bei sehr vielen sehr hellen LEDs können einzelne LEDs ‘zu spinnen’ beginnen.
Die Drähte müssen geradegebogen werden. Das Gerade-Biegen von Hand stellt sich als besonders schwierig heraus. Deshalb wird eine andere Methode aus dem Internet gewählt: Die Drähte werden in einem Schraubstock gespannt und das andere Ende in eine Bohrmaschine geklemmt, welche eingeschaltet nach wenigen Umdrehungen die Drähte perfekt geradebiegt.

Nachdem alle LEDs mit der Supply-Voltage und dem Ground elektrischen Kontakt haben, muss auch noch das Data- und Clock-Signal aller PCBs verbunden werden. Für diese Verbindungen werden dünne, kurze Kupferdrähte verwendet, diese sind vorher an den Kontaktpunkten mit einem Lötkolben abisoliert worden.



Nach dem Zusammenbau der Ebenen empfiehlt es sich wieder alle Ebenen optisch auf Kurzschlüsse und kalte Lötstellen zu überprüfen, um dann alle LEDs einer Ebene zum Testen mit dem Raspberry Pi zu verbinden.

Zu guter Letzt wird der Würfel aufgebaut. Dazu ist ein Gehäuse notwendig, auf dem der Würfel später stehen kann. Die Basis ist ein Quader mit einer Deck- und einer Grundplatte, welche verbunden sind mit zwei länglichen, rechteckigen Hölzern. Der Abstand zwischen den zwei Platten beträgt ca. 10 cm, genug Platz, um den Raspberry Pi und ein Netzteil unterzubringen. In die obere Platte werden Löcher gebohrt, durch welche die Drähte für die Stromversorgung der Ebenen gestossen werden.

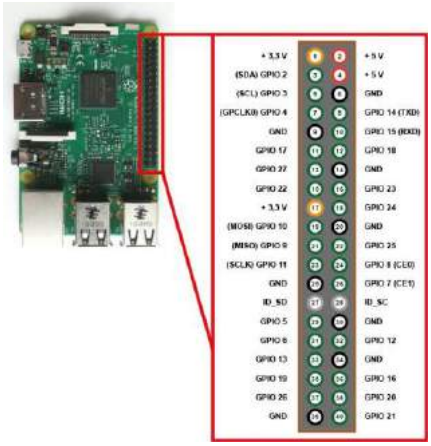
Mit einer dicken Kupferlitze werden jeweils die Ground- und 5V-Drähte der einzelnen Ebenen verbunden und an das Netzteil angeschlossen. Die dicke Kupferlitze ist notwendig, da dort die grössten Ströme fliessen. Für die Signal- und Clock-Verbindungen zwischen den Ebenen verwendet man ebenfalls dünne, kurze Kupferdrähte, wie bei den PCBs.

2.3.3 Material: Quellenangaben

Anzahl	Material	Quelle
1728	APA102	Aliexpress.com
144	PCB	PCBway.com
1	10m Rolle verzinkter Eisendraht mit 1.2mm Durchmesser	Baumarkt, z.B. Hornbach
1	Verzinnte Kupferlitze mit 3mm Durchmesser	Elektronikladen, Conrad in Dietlikon
1	Kupferlackdraht mit 0.2mm Durchmesser	Elektronikladen, Conrad in Dietlikon
1	Netzteil (60A, 5V)	Aliexpress.com
4	Holzplatten	Baumarkt z.B. Hornbach

2.3.4 Anschluss an den Raspberry Pi

Um den LED-Würfel an den Raspberry Pi anzuschliessen müssen die Daten an den Pin 19(MOSI) und das Clock-Signal and den Pin 23(SCLK) angelötet werden. Die Supply-Voltage und der Ground der LEDs werden an ein separates Netzteil angeschlossen.



Quellenangabe: Abb3[3]

2.3.5 Acrylglas Haube

Um dem LED-Würfel Schutz zu bieten und damit er nicht einstaubt wurde eine Acrylglas Haube um ihn herum gebaut. Damit der Würfel nicht überhitzt, wurden drei Luftschlitze vom Acrylglas freigelassen. Schliesslich müssen bei voller Helligkeit der LEDs mehrere hundert Watt an Wärme abgeführt werden.

Zudem sorgt die Haube dafür, dass auch Kinder nicht die gefährlichen 230V-Netzspannungsanschlüsse berühren können.

Damit ist der Bau des Würfels abgeschlossen.

[3]Quellenangabe Abb3: https://www.elektronik-kompendium.de/sites/raspberry-pi/1907101.htm, Zugriff: 17.10.2019

3 Software

3.1 Einführung

Die Software zur Ansteuerung der LEDs wurde in Python 3 geschrieben. Python ist eine beliebte Programmiersprache für mathematische Berechnungen. Praktischerweise ist sie auf einem Raspberry Pi gleich vorinstalliert. Das heisst man kann gleich loslegen.

Python eignet sich sehr, um objektorientierte Software zu schreiben. Was aber ist objektorientiertes Programmieren?

«Object-oriented programming is a method of implementation in which programs are organized as cooperative collections of objects, each of which represents an instance of some class, and whose classes are all members of a hierarchy of classes united via inheritance relationship»[4]

Was soll man sich darunter vorstellen?

- Klassen: Klassen dienen als Baupläne für Objekte, sie beschreiben ihre statischen und dynamischen Eigenschaften, das heisst ihre Attribute aber auch ihr Verhalten.
- Objekte: Wenn eine Klasse instanziiert wird entsteht ein Objekt.
- Die dynamischen Eigenschaften werden in Methoden beschrieben, das heisst Software code, welche die Attribute verändern.
- Vererbung: ist ein Konzept, bei dem grundlegende Eigenschaften, welche für mehrere Klassen gelten, zusammengefasst werden in einer Basisklasse. Zum Beispiel, dass alle Autos vier Räder, einen Motor sowie ein Steuerrad besitzen gilt für ein Sportauto von Porsche genauso wie für ein Lastwagen von Mercedes. Solche Eigenschaften können dann für eine Klasse ‘Limousine’ weiterverwendet werden. Dabei übernimmt die neue Klasse alle Methoden und Attribute der Basisklasse und kann folgend erweitert werden. Der 5-er BMW meines Nachbarn entspricht dann einem Objekt, das heisst sie ist eine spezifische Instanz der Klasse ‘Limousine’.
- Attribute: Jede Klasse wird unter anderem von ihren Attributen definiert. Sie sind Variablen die speziell nur zu ihrer Klasse gehören und werden immer als self.NameVariable definiert. Wenn einer Methode in einer Klasse self übergegeben wird, übernimmt sie alle objekteigenen Attribute.

3.2 Software Struktur

Das Programm für den LED-Würfel besteht aus mehreren Ebenen von Klassen:

Main ist der Software-Einstiegspunkt und beinhaltet die zeitliche Ablaufsteuerung der Animation in einer Endlosschlaufe.

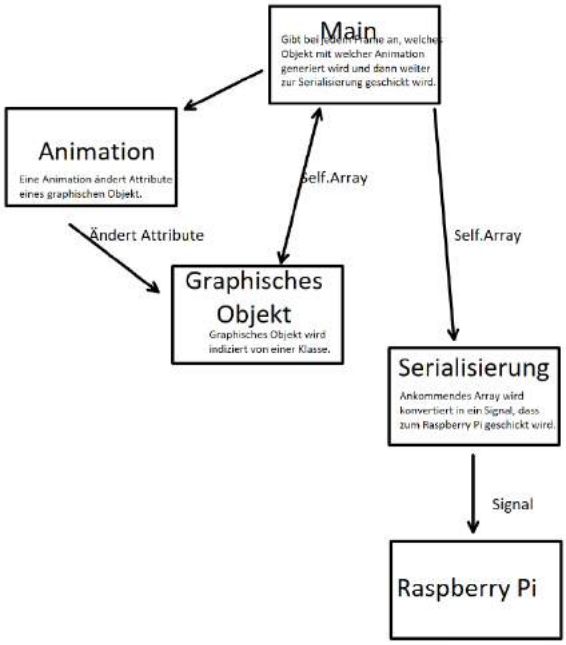
In dieser Schlaufe werden Methoden des Animations-Layers aufgerufen, welche Attribute der verschiedenen graphischen Objekte über die Zeit verändern.

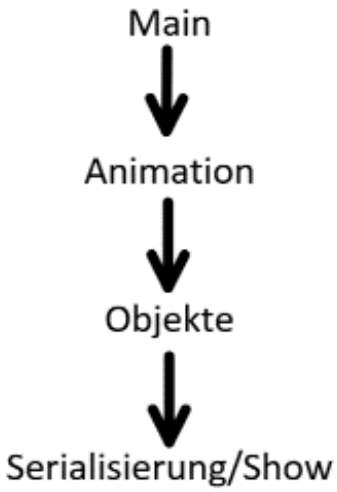
Im Objekte-Layer sind diese grafischen Objekte definiert, welche auch eine Vererbung der gemeinsamen Eigenschaften von einer Basisklasse enthält.

[4]Booch Jacobson Rumbaugh: Object-oriented analysis and design, S. 38: ISBN 0-8053-5340-2

Nachdem die Attribute aller graphischen Objekte angepasst sind, werden für jedes solches Objekt diejenigen LEDs in einem LED-Array bestimmt, welche mit einer jeweiligen Farbe und Intensität aufleuchten sollen.

Schlussendlich wird das LED-Array ausgegeben auf den physischen Würfel. Diese Ausgabe beinhaltet die Serialisierung des dreidimensionalen LED-Arrays.





3.2.1 Serialisierung/Show

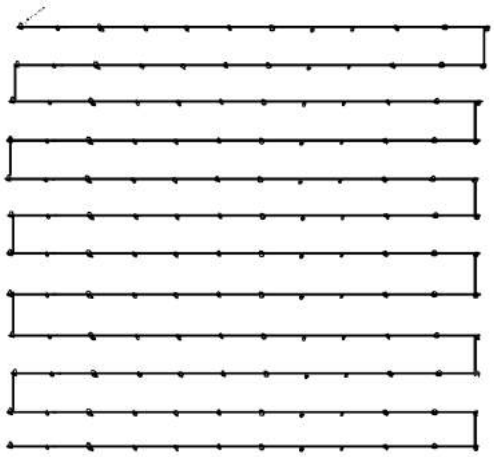
Vom Raspberry Pi wird ein eindimensionales Signal mit allen Anweisungen an die LEDs ausgesendet. Die LEDs sind in einem Würfel angeordnet, jeder LED kann eine Koordinate der x-, y- und z-Achse zugeordnet werden. Mathematisch gesehen bilden die LEDs ein dreidimensionales Array.

$$ledarray[x][y][z] \;\; ; with\ x, y, z := \{0, 1, 2, \dots 11\}$$

Da die Ausgabe über die SPI-Schnittstelle aber eindimensional ist, muss in der «Serialisierung» das dreidimensionale Array in einen eindimensionalen Datenstrom umgewandelt werden.

```
for y in range(self.ledarray_length):
    for z in range(self.ledarray_length):
        for x in range(self.ledarray_length):
            self.ser.append(self.ledarray[x][y][z])
```

Nebenan ist der Weg auf einer Ebene, welches ein SPI-Bussignal beim LED-Würfel durchläuft, aufgezeigt. Unten links startet das Signal vom Raspberry Pi aus und läuft oben links weiter zur nächsten Ebene. Jeder Punkt stellt eine LED dar.



```
for t in range(0,(self.numLED-1),2*self.ledarray_length):
    t+=self.ledarray_length
    self.ser[t:(t+self.ledarray_length)] = reversed(self.ser[t:(t+self.ledarray_length)])

for u in range(0,(self.numLED-1),2*self.ledarray_plane):
    u+=self.ledarray_plane
    self.ser[u:(u+self.ledarray_plane)] = reversed(self.ser[u:(u+self.ledarray_plane)])
```

Beim eindimensionalen Signal werden zusätzlich die Signale jeder zweiten Reihe und jeder zweiten Ebene gekehrt. Der source code befindet sich im File «HAL.py».

3.2.2 Objekte

Software-Objekte sind einerseits konkrete Datenstrukturen instanziierter Klassen, in diesem Projekt werden sie auch für die Repräsentation der realen graphischen Objekte in Form aufleuchtender LEDs verwendet. Immer, wenn eine Methode einer solchen Klasse aufgerufen wird, wird ihr ein Pointer auf die Datenstruktur des Objektes (self) und damit auch auf self.Array, einer Beschreibung des LED-Würfels, der Klasse übergeben. Self.Array wird in der «Main» Klasse generiert. Sie hat vier Dimensionen, eine für die x-Achse, die y-Achse, die z-Achse (sie geben die Position jeder LED an) und eine letzte Dimension, die die Helligkeit und Farbe der LED angibt.

Vererbung: Alle Klassen, welche eine geometrische Form generieren, erben Eigenschaften von der Klasse «Template». Diese Klasse beinhaltet Attribute und Methoden, welche in allen diesen Klassen vorhanden sein sollen. Einfachheitshalber müssen sie nur einmal in der Klasse «Template» (im File «template.py») definiert werden.

Zu diesen Attributen gehören:

self.center (stellt das Zentrum des graphischen Objektes dar)

self.rotationsvector (wird genau im Kapitel Rotationsmatrix erklärt)

self.brightness (Helligkeit von 0 bis 31)

self.color (rot, grün, blau von 0 bis 255)

self.gravity (m/s^2)

self.velocity(m/s)

self.Array_length (Anzahl LEDs auf einer Achse: 12, weil der Würfel 12*12*12 LEDs hat)

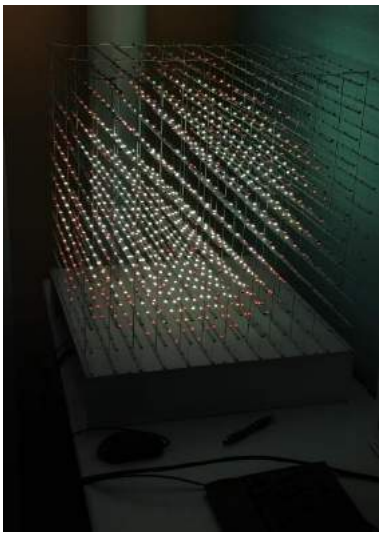
Methoden:

Zu allen Attributen gibt es eine get- und eine set-Funktion. Beim Ausführen einer get-Funktion bekommt man den Wert eines Attributs. Diese Funktion wird gebraucht, um ausserhalb der Klasse auf lokale Variablen zuzugreifen.

```
def getcenter(self):
    return self.center

def setcenter(self, center):
    self.center = center
```

Set-Funktionen werden verwendet um den Wert lokaler Variablen (hier Attribute) ausserhalb der Klasse zu ändern.

Kugel/Sonne: Ein einfaches Beispiel eines graphischen Objektes ist die instanziierte Klasse «Sphere». Eine Kugel besitzt unter anderem einen Radius, eine Farbe und eine bestimmte Position (Koordinate des Zentrums). Farbe und Position können als Attribute der Klasse «Sphere» von der «Template» Klasse geerbt werden.
In der Methode «Malen» wird dann die Farbe der Kugel verändert, d.h. dem Attribut self.color einen neuen Wert zugewiesen. Mit einer weiteren Methode wird die Position der Kugel verändert.



In der Klasse «Sphere» wird für jede LED der Abstand zum Kugelzentrum berechnet. Wenn der Abstand in einem bestimmten Bereich ist, wird die Helligkeit der entsprechenden LED im self.LEDarray auf einen entsprechenden Wert gesetzt und die LED leuchtet auf.

Die Klasse «Sphere» ist im File «sphere.py» zu finden. Im File «sun.py» ist eine leicht abgeänderte Version von der Klasse «Sphere».

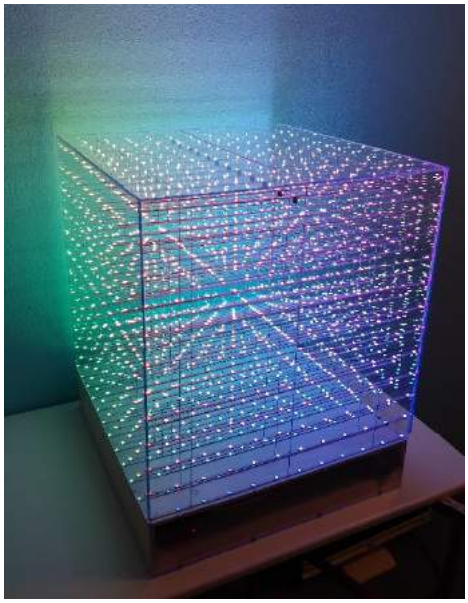
Attribute, welche nur in «Sphere» und «Sun» vorkommen sind:

self.radius(Radius der Kugel)

self.distance(Die Kugel nimmt ab einem bestimmten Radius an Helligkeit ab, self.distance gibt die Distanz an, ab welcher die Helligkeit abnimmt)

Colorgradient: Die Klasse «Colorgradient» erzeugt eine einfache Visualisierung des ganzes Farbspektrums. Je grösser die x-Koordinate desto blauer wird es, das gleiche passiert bei der y-Koordinate mit grün und der z-Koordinate mit rot.

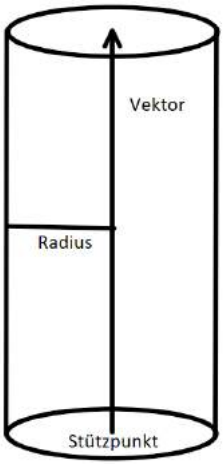
```
self.Array_length = 12
quotient=int(256/12)
for z in range(self.Array_length):
    for y in range(self.Array_length):
        for x in range(self.Array_length):
            Array[x][y][z][0] = self.brightness
            Array[x][y][z][1] = quotient*x
            Array[x][y][z][2] = quotient*y
            Array[x][y][z][3] = quotient*z
```



Letter: Diese Klasse beinhaltet das ganze Alphabet, nach Belieben kann mit ihr ein Buchstabe dargestellt werden.

Self.buchstabe ist ein Attribut der Klasse «Letter», es enthält ein Array von Koordinaten jedes Pixels für alle Buchstaben im Alphabet. Das File mit der Klasse «Letter» heisst «letter.py».

Vector: Mit dieser Klasse lässt sich ein beliebiger Vektor erzeugen. LEDs, welche nicht direkt auf dem Vektor liegen, nehmen, je weiter sie vom Vektor entfernt sind, an Helligkeit ab. Wird der Radius der LEDs, welche um den Vektor leuchten, erhöht, sieht der Vektor wie ein Zylinder aus.



In der Klasse «Vector» ist der Stützpunkt des Vektors die Variable self.center, die Richtung gibt die Variable self.vektor an. Ein weiteres Attribut definiert den Radius, den maximalen Abstand der LEDs vom Vektor, welche noch aufleuchten, an.
Das File, dass die Klasse «Vector» enthält, heisst «vector.py».

Plane: In der Klasse «plane» wird eine Vektorebene mit dem Stützpunt self.center und dem Normalen Vektor (Vektor der einen Winkel von 90 Grad zur Ebene hat) definiert. Um herauszufinden welche LEDs auf oder fast auf der Ebene liegen, benötigt man den Abstand jeder LED zur Ebene. Dies wird wie folgt gemacht:

Die Hessische Normalform der Ebenengleichung lautet $E: \frac{\mathrm{ax+by+cz+k}}{\sqrt{a^2+b^2+c^2}} = 0$, wobei $\vec{n} = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$ ein Normalenvektor der Ebene E ist, Punkt P als Vektor $\vec{P} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$ ist ein beliebiger Vektor auf der Ebene E.

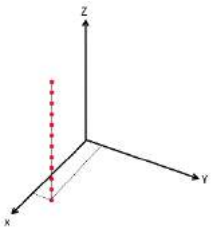
k definiert die Höhe der Ebene, wenn k verändert wird verschiebt sich die Ebene um $\frac{k}{|\vec{n}|}$ entlang der Normalen $\vec{n}$. k lässt sich auflösen, wenn man den Punkt P und Vektor n hat.

Wenn der Punkt P nicht auf der Ebene E liegt, bekommt man den Abstand d von Punkt P zur Ebene E. $d = |\frac{ax+by+cz+k}{\sqrt{a^2+b^2+c^2}}|$. Die gewünschte Helligkeit lässt sich mit dem Abstand d zur Ebene danach einstellen.

Das File, dass die Klasse «Plane» enthält, heisst «plane.py».

Wave: Diese Klasse lässt eine Welle auf einem Koordinatensystem abbilden. Die Funktion einer Welle ist $f(x,y) = a * \sin(\sqrt{x^2+y^2}+b)$. Beim Verändern der Variable a lässt sich die Amplitude der Welle verändern. Wenn b sich ändert, entsteht der Eindruck als ob sich die Welle bewegen würde.

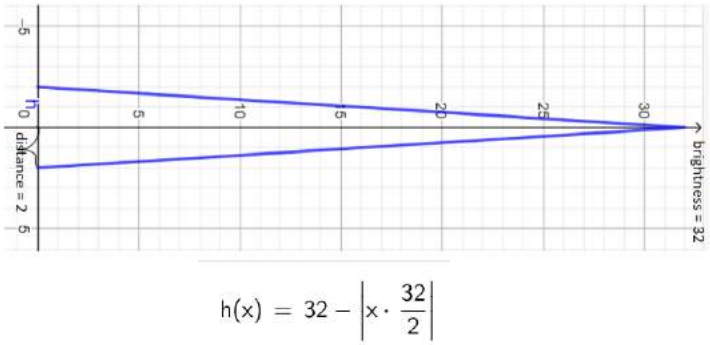
Im Programm wird das Koordinatensystem wie folgt berechnet: Für jede der 144 Reihen von jeweils 12 LEDs in der xy-Ebene (siehe in Bild rechts ein Beispiel einer Reihe) wird mit der Funktion f(x,y) die Höhe(z) der LED, welche aufleuchten sollte, berechnet. Die Höhe z der aufleuchtenden LEDs ist ein auf integer gerundeter Wert. Um diese Ungenauigkeit auszugleichen leuchten LEDs mit einem grösseren Rundungsfehler weniger hell auf. Wenn die Funktion f(x,y) verändert wird, können auch andere Graphen dargestellt werden.



Helligkeitsabnahme: Im Objekt Vektor wurde bereits vorgestellt, dass LEDs, welche nicht direkt auf dem Vektor liegen, bis zu einer gewissen Distanz vom Vektor entfernt, an Helligkeit gemäss dem Abstand zum Vektor abnehmen. Das gleiche Prinzip wird auch beim Objekt Kugel und Objekt Sonne genutzt. Anstatt, dass die Helligkeit linear abnimmt, nimmt sie jedoch mit einer Potenzfunktion ab. Dies ergibt einen schöneren Helligkeitsverlauf, da die Helligkeit der LEDs für das menschliche Auge auch nicht linear zunimmt.

Lineare Abnahme der Helligkeit:

```
int(a) Funktion rundet die Zahl a auf einen int
abs(a) Funktion gibt den Betrag von a an
max(a,b) Funktion gibt den grösseren der beiden Werte a oder b aus
distance = Wert der die Distanz angibt, in welcher die Helligkeit abnimmt
brightness = maximale Helligkeit
dis = Abweichung der LED vom idealen Wert (je grösser dis desto dünkler wird die LED)
scale1 = brightness/distance
Helligkeit einer LED = self.Array[x][y][z][0] = int(max(0,brightness-abs(dis*scale1)))
```



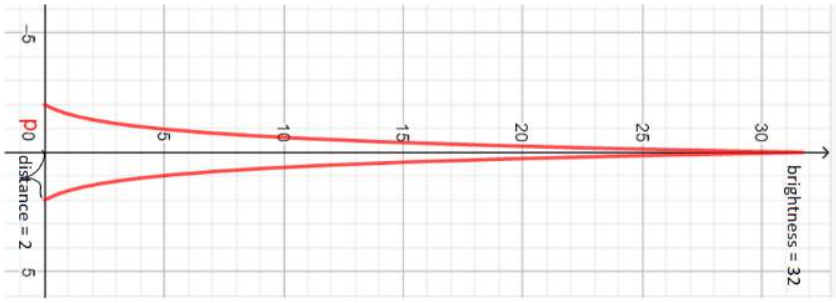
$$h(x) = 32 - \left|x \cdot \frac{32}{2}\right|$$

Die beiden Graphen für die lineare und Potenz Abnahme der Helligkeit wurden mit «GeoGebra» erstellt. Die y-Achse stellt die Helligkeit einer LED dar, die x-Achse stellt «dis» dar. «dis» ist die Abweichung der LED von ihrer idealen Position, z.B. wenn eine LED nicht genau auf einem Kreis liegt.

Potenz Abnahme der Helligkeit:

```
math.pow(a,b) Funktion gibt a^b zurück
math.log2(a) Funktion gibt den Logarithmus von a zur Basis 2 zurück
bright = math.log2(brightness+1)
scale2 = bright/distance
Helligkeit einer LED = Array[x][y][z][0] = int(max(0,math.pow(2,(self.distance-abs(dis))*scale2)-1))
```

$$p(x) = 2^{(2-|x|)\frac{\log_2(33)}{2}} - 1$$



3.2.3 Main

Die Klasse «Main» ist die zentrale Komponente des Programms, sie steuert unter anderem die zeitliche Regelung, wann welche Animation und welches graphische Objekt ausgeführt werden.

Beim Start des Programmes wird die Funktion «main» aufgerufen, sie instanziiert das Objekt der Klasse «LEDWürfel». Darauffolgend wird die Methode «clock» in der Klasse «LEDWürfel» ausgeführt.

```
def main():
    obj = LEDWürfel()
    obj.clock()

main()
```

Diese Methode ruft wiederum die Methode «generate» auf, welche self.Array generiert, die wichtigste Variable im Programm, welche eine mathematische Repräsentation des LED-Würfels darstellt. Sie hat vier Dimensionen, eine für die x-Achse, die y-Achse, die z-Achse (sie geben die Position jeder LED an) und eine letzte Dimension, die die Helligkeit und Farbe der LEDs angibt.

```
self.Array_length = 12 #LED Würfel hat in jeder Achse 12 LEDs
def generate(self):
    self.Array = [[[[Helligkeit,blau,grün,rot]for q in range(self.Array_length)] for w in range(self.Array_length)] for e in range(self.Array_length)]
    #self.Array[x-Achse][y-Achse][z-Achse][0] = 31    -> Helligkeit wird in dieser LED auf 31 gesetzt
```

Als nächstes werden Objekte aller Klassen, welche wir für unser Programm brauchen, instanziiert. Danach startet die eigentliche Animation des Programms. In einer while-Schleife wird ein zeitlicher Ablauf erzeugt, welcher ca. 10 Frames pro Sekunde (FPS) erzeugt. In jedem Frame wird festgelegt, welche Animationen und graphischen Objekte ausgeführt bzw. dargestellt werden. Eine Methode aus der «Animation» Klasse verändert die Attribute eines graphischen Objektes. Beim Ausführen der «Rendering» Methode eines graphischen Objektes (immer nach der Klasse benannt), gibt die Methode die veränderte Variable self.Array zurück, mit dem graphischen Objekt auf ihr abgebildet. Sollen mehrere graphische Objekte gleichzeitig aufleuchten, so wird der self.Array Output des vorhergehenden Objekts als Eingang des nächsten Objekts verwendet und damit evtl. LEDs geschrieben. Am Ende jedes Frames wird die Variable self.Array der «Serialisierung» Klasse übergeben, sie konvertiert das Array in ein Signal, dass dem LED-Würfel übergeben wird. Die LEDs leuchten auf.

Beispiel aus der Methode «clock» einer Animation:

```
objektxy = filexy.Klassexy() #Objekt von einer Klasse wird instanziiert
animation1 = animation.Animation() #Objekt, das alle Animationen beinhaltet
show1 =  HAL.Show() #Objekt für die Serialisierung

self.frame = 0 #Anzahl Frames, welche seit dem Start des Programmes vergangen sind
tmp = 0
timemod = 10 # frames per second
animationlength_in_sec = 30


while(self.frame<(timemod*animationlength_in_sec)): #bis animationlength_in_sec Zeit umgegangen ist laufen die Animationen
    now = math.floor(time.time()*timemod)
    if(now != tmp): # Jede 1/timemod Sekunden ist das if statement True, es gibt ein neues Frame
        tmp = now


        if self.frame in range(0,100): #während der angebenen Zeitspanne läuft diese Animation

            animation1.animation_xy(objektxy,self.frame,andere Parameter)#Methode animation_xy verändert jedes Frame Attribute in objektxy
            self.Array = objektxy.Methodexy(self.Array) #objektxy wird gerendert, das graphische objektxy ist in self.Array abgebildet

        show1.show(self.Array) #Am Ende des Frames wird self.Array der Serialisierung übergeben, die LEDs leuchten auf
        self.frame += 1 #Anzahl Frames wird hinaufgezählt
```

3.2.4 Animation

Unter Animation wird hier eine Methode in der Klasse «Animation» verstanden, welche einerseits die Aufgabe hat Attribute eines angegebenen graphischen Objektes in jedem Frame zu verändern. Andererseits hat sie die Aufgabe, wenn die Methode das erste Mal aufgerufen wird, die Attribute des graphischen Objektes auf einen angegeben Startwert zu setzen. In den folgenden Abschnitten werden einige Animations-Methoden vorgestellt.

animation_brick:

Bei der Animation_brick wird das Attribut self.center verändert, also das Zentrums eines Objekts. Da self.center die Position des graphischen Objektes definiert, bewegt sich das graphische Objekt.

```
#objekt = graphisches Objekt bei dem die Animation laufen soll
#frame = Zeit welche seit dem Start des Programms vergangen ist
#time = Zeitpunkt an dem die Animation das erste Mal ausgeführt wird
#brightness/center = auf diesen Wert wird self.brightness/self.center
#in objekt gesetzt beim ersten Aufruf der Methode.


def animation_brick(self, objekt, frame, time, brightness, center):
    if frame == time: #ist nur beim ersten Aufruf der Methode True
        objekt.setbrightness(brightness)
        objekt.setcenter(center)

    if frame in range(time,time+9):
        objekt.move_x(1) #verschiebt self.center in der x-Koordinate um 1
        objekt.change_blue(-25) #Die Farbe blau nimmt um 25 ab
```

animation_radius:

Bei dieser Animation wird der Radius einer Kugel um cos(t) verändert. In anderen Worten vergrössert und verkleinert sich die Kugel sinusförmig.

```
def animation_radius(self, objekt, frame, time, brightness, center ,radius):
    if frame == time:
        objekt.setbrightness(brightness)
        objekt.setcenter(center)
        objekt.setradius(radius) #self.radius in objekt wird auf radius gesetzt

    gr=math.pi/16
    radius = radius + math.cos(frame*gr)*1.5
    objekt.setradius(radius)
```

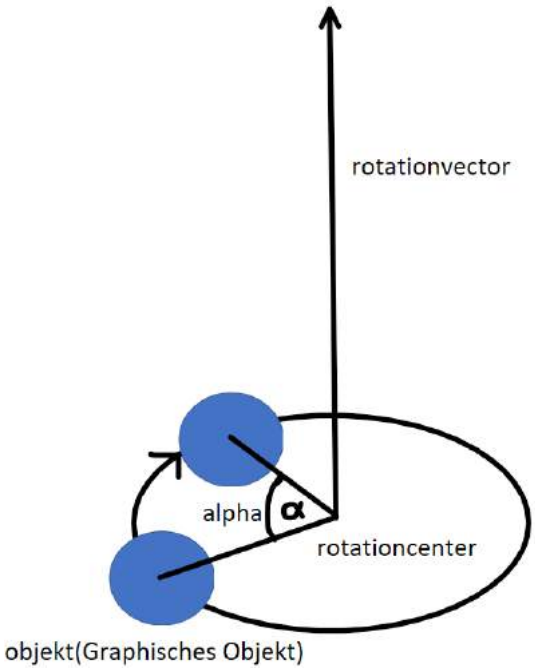
animation_rotation:

Diese Animation lässt ein graphisches Objekt(objekt) alpha Grad um einen beliebigen Vektor(rotationvector) drehen.

```
#rotationvector = Vektor um den objekt kreist
#rotationcenter = Stützupunkt des rotationvector
#alpha = Winkel der angibt um wie viel sich objekt um den rotationsvector dreht
def animation_rotation(self,objekt, frame,time, brightness, center, rotationvector, rotationcenter, alpha):
    if frame == time:
        objekt.setbrightness(brightness)
        objekt.setcenter(center)
    objekt.setrotation(alpha,rotationcenter,rotationvector) #ändert self.center in objekt
```

In der «animation_rotation» Methode ist nicht zu erkennen um wie viel sich self.center schlussendlich verändert, dies wird in der Methode setrotation im File «template» berechnet.

```
def setrotation(self, alpha, rotationcenter,rotationvector):
    xroce = rotationcenter[0]
    yroce = rotationcenter[1]
    zroce = rotationcenter[2]

    x = self.center[0]- xroce
    y = self.center[1]- yroce
    z = self.center[2]- zroce

    n1 = rotationvector[0]
    n2 = rotationvector[1]
    n3 = rotationvector[2]

    s = math.sin(alpha)
    c = math.cos(alpha)

    x1 = x*(n1*n1*(1-c)+c) + y*(n1*n2*(1-c)-n3*s) + z*(n1*n3*(1-c)+n2*s)
    y1 = x*(n2*n1*(1-c)+n3*s) + y*(n2*n2*(1-c)+c) + z*(n2*n3*(1-c)-n1*s)
    z1 = x*(n3*n1*(1-c)-n2*s) + y*(n3*n2*(1-c)+n1*s) + z*(n3*n3*(1-c)+c)

    self.center = np.array([x1+xroce,y1+yroce,z1+zroce])
```



animation_rotationplane :

Diese Animation macht praktisch das gleiche wie «animation_rotation». Anstatt self.center um einen Vektor zu drehen, lässt diese Methode die Normale einer Ebene und somit die Ebene um einen Vektor rotieren.

animation_ballbounce

Die Methode simuliert Bälle, welche von einer bestimmten Höhe fallen und am Boden wieder hochspringen.

```
    #velocity = Dieser Wert gibt die Geschwindigkeit von objekt an (m/s)
    #gravity =  Um diesen Wert nimmt velocity jedes Frame zu (m/s^2)
def animation_ballbounce(self, objekt, frame, time, brightness, center, gravity, velocity):
    if frame == time:
        objekt.setbrightness(brightness)
        objekt.setcenter([random.randrange(1,10),random.randrange(1,10),11])
        objekt.setgravity(gravity)
        objekt.setvelocity(velocity)


    center = objekt.getcenter()
    resistance = 0.9 #(1-resistance) verliert velocity beim Aufprall

    #Wenn objekt am Boden ankommt wird velocity zu -velocity
    if 0 > (center[2] - objekt.getradius()-(1-resistance)*objekt.getvelocity()):
        objekt.setvelocity(-objekt.getvelocity()*resistance)
        objekt.setcolor([random.randint(0,255),random.randint(0,255),random.randint(0,255)])
    else: #velocity wird um gravity grösser
        objekt.setvelocity(objekt.getvelocity()+objekt.getgravity())

    objekt.move_z(-objekt.getvelocity()) #Objekt fällt um velocity
```

3.3 Snek3D

«Snek3D» ist ein Game inspiriert von «Snake». Im Spiel bewegt der Spieler eine Schlange welche, wenn sie isst, länger wird. Die Schlange wechselt beim Drücken der Tasten w/a/s/d auf der xy-Ebene die Richtung und beim Drücken der Pfeiltaste ‘oben’ und ‘unten’ kriecht die Schlange die z-Achse entlang. Das Ziel des Spieles ist es, die Schlange so lang wie möglich werden zu lassen. Das Spiel endet, wenn die Schlange sich selbst beisst. Am Ende des Spieles gibt es eine «GAME OVER» Animation.

Um die Richtung der Schlange zu wechseln, muss der Tastatur-Input gelesen werden, um zu erkennen, ob eine Taste gedrückt wird. Um den Tastatur-Input zu erhalten wird die «pygame» library verwendet.

```
for event in pygame.event.get():
    if event.type == pygame.KEYDOWN: #ist eine Taste gedrückt?

        if event.key == pygame.K_UP: #Pfeiltaste oben gedrückt?
            snek1.setdirection(3) #ändert Richtung
```

Das Game «Snek3D» startet beim Ausführen des Files «snek3D.py».

3.4 Automatischer Start

Für die Präsentation der Animation möchten wir das Python Skript «LEDmain» direkt beim Aufstarten des Raspberry ohne manuellen Eingriff starten. Dafür stellt Raspian, die Linuxversion für den Raspberry Pi, mehrere Varianten zu Verfügung. Die hier verwendete Variante geht folgenderweise:

Als erstes muss der Befehl «sudo nano /etc/rc.local» im Terminal ausgeführt werden. Dieser Befehl öffnet mit Adminrechten (sudo) das File mit dem Dateipfad «/etc/rc.local» mit dem Editor «nano».

In diesem File wird über der Zeile «exit 0» das Kommando «python3 Dateipfad_und_Name_des _Skripts &» eingefügt, also z.B. python3 /home/pi/python/LEDwuerfel/ledmain.py &. Wichtig zu beachten ist, dass der Dateipfad keine Leerzeichen beinhaltet. Diese Methode ist, via Google gesucht, hier näher beschrieben:

http://raspberry.tips/raspberrypi-einsteiger/raspberry-pi-autostart-von-skripten-und-programmen-einrichten

4 Abschluss / Nächste Schritte

Insgesamt bin ich mit dem erreichten Stand meiner Arbeit zufrieden. Mit der Programmierung der Animation und eines Games kam ich weiter als ich zunächst geplant hatte. Gerade im Bereich Software gäbe es weitere mögliche Schritte zu bearbeiten: Die FPS-Rate der Animation beträgt nur etwa 10 FPS, wobei die Serialisierung ca. 0.05 Sekunden und die Berechnungen der Animation und des graphischen Objektes nochmals weiter ca. 0.05 Sekunden benötigen. Mit der Optimierung des Codes, dem Gebrauch der anderen Cores des Prozessors und dem Gebrauch eines leistungsstärkeren Computers würde sich die FPS-Rate erhöhen lassen. Am liebsten hätte ich mich noch intensiver mit dem Programmieren beschäftigt. Interessiert hätte mich die Entwicklung von Simulationen von fluid dynamics bzw. ein Game im Style von Super Mario Bros.

Auch im Bereich der Hardware, also beim LED-Würfel, gab es Probleme. So ist die Spannung an den LEDs bei hoher Helligkeit instabil auf Grund der Eisendrähte. Es kommt bei hoher Helligkeit mehrerer LEDs vor, dass eine LED deshalb temporär ihren Dienst versagt. Da die LEDs alle in Serie geschaltet sind, reagieren die hinteren LEDs damit nicht mehr. Um dieses Problem zu lösen, müssten Kupferdrähte verwendet sowie Kondensatoren in die PCBs eingebaut werden. Zurzeit muss in einem solchen Fall der LED-Würfel kurzfristig aus- und wieder eingeschaltet werden.

Abschliessend möchte ich noch einmal meinen besonderen Dank all denjenigen aussprechen die mir das Projekt LED-Würfel ermöglicht haben. Dabei handelt es sich um meinen Physiklehrer Herrn Seipel und meinem Vater, denen ich viele wichtige Tipps und Hilfestellungen verdanke. In Zukunft werde ich mich weiter mit der Thematik befassen und konnte insgesamt sehr von der intensiven Auseinandersetzung der umfassenden Thematik profitieren.

5 Weitere Quellenangaben:

LEDs:

https://cpldcpu.wordpress.com/2014/11/30/understanding-the-apa102-superled/

https://www.gree-leds.com/industry-information/digital-addressable-led-strip-wiring.html

https://quinled.info/2019/06/02/what-digital-led-chip-to-choose/

https://cdn-shop.adafruit.com/datasheets/APA102.pdf

https://www.mikrocontroller.net/topic/448775

https://www.espruino.com/WS2811

https://www.espruino.com/Individually+Addressable+LEDs

Leiterplatte:

https://de.wikipedia.org/wiki/Leiterplatte

Raspberry Pi:

Kofler, Kühnast, Scherbeck: Raspberry Pi: ISBN 978-3-8362-4220-2

Programmieren (Programmiertipps):

https://stackoverflow.com/

Johannes Ernesti, Peter Kaiser: Python 3: ISBN 978-3-8362-3633-1

Snek Game:

https://www.pygame.org/docs/ref/key.html