Ruby der Zauberwürfellöser

# Einführung:

Wir wollen ein Programm schreiben, welches einen Rubik‘s Cube (auch Zauberwürfel genannt) selbständig löst. Dabei soll der verdrehte Würfel von einer Kamera eingescannt und danach von dem Programm gelöst werden. Die dafür benötigten Schritte sollen dann ausgegeben werden, sodass man den Würfel lösen kann.

### Gesetztes Ziel:

Einen Algorithmus zu programmieren, der den Rubik‘s Cube lösen kann, ohne ihn nach jedem Schritt erneut einzuscannen, also die Schritte „voraussehen“.

## Sollte:

Die Seiten des Würfels automatisch einzuscannen und die einzelnen Felder der jeweiligen Seite zu zuordnen. Nachdem der Algorithmus ausgeführt wurde, soll die Schrittreihenfolge ausgegeben werden.

## Nice to have:

Wir würden gerne einen Roboter bauen, der den Würfel selber lösen kann, jedoch nur wenn wir noch genug Zeit haben.

## Bewusst weggelassen:

Der Würfel sollte immer der gleiche Würfel sein, zumindest vom gleichen Fabrikanten, da die Farben sehr unterschiedlich seien können. Auch wollen wir keinen neuen Weltrekord aufstellen, somit soll es nicht auf Zeit sein.

Projektstrukturplan:

# Aufgabenbereiche:

## Lösungsalgorithmus

## Erkennen der Farben über einen Sensor

## Ausgabe der Schrittreihenfolge

## (ggf. Rubiksroboter)

# Teilaufgaben

## Lösungsalgorithmus

Der Algorithmus muss den verdrehten Würfel lösen können, ohne ihn dabei wirklich zu drehen. Deswegen müssen wir einen virtuellen Würfel programmieren, an dem die verschiedenen Drehungen von dem Programm ausgeführt werden. Dabei wird der Würfel nicht wirklich gedreht, sondern die Farbflächen den anderen Seiten nach einem bestimmten System zugeordnet.

### Aufgaben

* Einen Lösungsalgorithmus finden, der den Würfel lösen kann und dabei die Schritte voraussieht.
* Einen virtuellen Würfel zu programmieren.
* Den gefundenen Algorithmus in Processing einzuprogrammieren und mit dem virtuellen Würfel zu verbinden.

### Materialien

* Computer
* Fachliteratur
* Rubik’s Cube

### Wissen

* Algorithmen heraussuchen und ihre Umsetzbarkeit ausprobieren
* Programmierstrukturen heraussuchen, die uns das Programmieren erleichtern können

### Risiken

Das größte Risiko ist natürlich, dass wir keinen Algorithmus finden oder ihn nicht programmieren können.

## Erkennen der Farben über einen Sensor

Hier gilt es den Rubik‘s Cube zu digitalisieren: Um einen geeigneten, Würfel-spezifischen Lösungsalgorithmus zu finden, ist eine Digitalisierung des Würfels unabdingbar, sodass das Lösungsverfahren überhaupt angewendet werden kann. Um diese Digitalisierung nicht manuell durchführen zu müssen, ist ein Sensor, ob Webcam oder Ähnliches, notwendig, sodass die Struktur des zu lösenden Würfels gescannt wird.

## Sensor

Da sich der Sensor schließlich auf Farberkennung spezialisiert, liegt die Wahl zu einem RGB Sensor nahe, der die RGB-Werte einer gemessenen Fläche misst. Es gilt nun die genaue Beschaffenheit des Sensors zu spezifizieren:

Da Farbwerte vom Licht abhängig sind, besteht hier eine Fehlerquelle für den Würfel-Scan. Für den Lösungsalgorithmus ist es wichtig, dass die Würfelflächen der richtigen Farbe zugeordnet werden. Um dies zu gewährleisten muss entweder eine Toleranz in der Messung und Zuordnung implementiert werden oder eine genauere Messung erfolgen. Das Licht-Problem sei zum Beispiel gelöst, indem eine starke LED mit neutralem Licht jede einzelne Würfelfläche mit gleicher Belichtung ausleuchtet, sodass die Messung für jede Farbfläche gleiche Bedingungen hat.

Auch existieren arduino-freundliche Messsensoren. Hier möchten wir beispielsweise „Adafruit‘s RGB Color Sensor with IR filter and White LED – TCS34725“ nennen, der zusätzlich die genannte weiße LED hat, sowie IR filtert.

https://www.adafruit.com/product/1334

Der „Rolls-Royce“ unter den Sensoren ist wohl ein „TCS3200 Color Sensor Module“, was mit vier LEDs ausleuchtet und die Farben direkt sortiert und auch auf weite Entfernung gut misst.

http://www.miniinthebox.com/de/tcs3200-color-sensor-module\_p406981.html?currency=EUR&litb\_from=paid\_adwords\_shopping&utm\_source=google\_shopping&utm\_medium=cpc&adword\_mt=&adword\_ct=151070323334&adword\_kw=&adword\_pos=1o5&adword\_pl=&adword\_net=g&adword\_tar=&adw\_src\_id=9772115911\_681685170\_37354518680\_pla-256352631726&gclid=CMOAhp-hi9QCFSEz0wod1VgFEw

Wir heißen jedoch ersteren für unser Projekt gut, da dieser unserer Messung wohl am angepasstesten ist. Wir benötigen keine großen Entfernungen, sodass eine LED zur Beleuchtung wohl ausreichen wird. Dazu ist die Platine deutlich kleiner, was für eine optionale spätere Umsetzung in einer Konstruktion nur von Vorteil sein kann.

## Farbwerte definieren

Farbwerte müssen definiert werden, um ihnen Namen geben zu können und sie zuordnen zu können.

Hier kann man natürlich von einem festen Rubik‘s Cube ausgehen, ihn vorher mehrmals messen und schließlich diese Messwerte als absolute Norm für die einzelnen Farben nehmen. Natürlich müsste man noch Toleranzwerte festlegen.

Deutlich besser ist natürlich eine anpassungsfähigere Methode. Vorzuschlagen wäre hier ein Farbwerte Vergleich: Da ein Farbsensor die RGB-Werte filtert, müssen diese R-, B-, G-Werte nur bezüglich ihrer Größe untereinander verglichen werden und können dementsprechend dann einer Farbe zugeordnet werden. Diese Zuordnung ist situationsangepasster, wenn nicht gar würfel-unabhängig (solange die einzelnen Farbflächen monochrom sind).

Nachdem man nun ein Feld gemessen hat und ihm ein Farbwert zugewiesen hat, kommt die Zuordnung.

## Farben den einzelnen Seiten zuordnen

Die Farben können schließlich den einzelnen Seiten zugeordnet werden. Hier gibt es verschiedene Möglichkeiten, deren Qualität zu beurteilen und vergleichen ist.

Man kann z.B. von vorne herein festlegen, welche Seite zuerst gescannt werden soll, denn am mittleren Feld ist die vorausgesetzte Farbe zu erkennen, da das mittlere Feld unbeweglich ist. So hat man eine festgelegte Voreinstellung, was Code und Berechnung spart.

Schöner wäre hier jedoch, sich gerade die Festlegung des Mittelsteins zu Nutze zu machen und gemäß eines (später genauer beschriebenen) Scan-Musters den Mittelstein durch Farbmessung festzustellen und die Platzierung der anderen Felder logisch zu ergänzen oder nach demselben Verfahren festzustellen. Dies erzeugt zwar mehr Messung und Code, jedoch voraussichtlich nur geringfügig, sodass der Nutzen, die höhere Flexibilität, hier heraus sticht.

## Würfel nach einem bestimmten Muster scannen

Hier sei zu überlegen, ob man den Würfel linear oder gedreht scannt, jedes Quadrat des Würfels einzeln oder den Achsen entlang. Hier kann man abwägen bezüglich Scan-Geschwindigkeit und Code-Länge, da verschieden Muster verschieden lange brauchen und verschiedene komplex sind bezüglich ihrer digitalen Verarbeitung. Dies möchten wir uns aber derzeitig noch offen lassen und wollten nur ein paar Möglichkeiten und deren Auswahlkriterien erwähnen.

## Ausgabe der Schritte / Drehungen

Die Ausgabe wird wohl über Processing erfolgen, da die IDE von Processing einem die Erstellung eines GUIs, beziehungsweise eines Fensters, sowie die Visualisierung von Flächen (somit auch Körpern) einem dermaßen vereinfacht, dass wir hier die Vorteile dieser Software nutzen wollen.

Die Ausgabe der Schritte erfolgt nach der Lösung des Würfels durch den Algorithmus. Also muss man die einzelnen Schritte irgendwie abspeichern und nach voller Berechnung ausgeben. Man kann hier überlegen, ob man die Schritte sich gleich als „Anleitung“ ausgeben lässt oder jeden Schritt einzeln anzeigt und auf Knopfdruck den nächsten Schritt angezeigt bekommt.

Wie sehr wir die Schritte Visualisieren bleibt offen. Mindestanforderung ist auf jeden Fall eine Textausgabe mit den Schritten. Von hier an könnte man weitergehen und beispielsweise via Processing die Schritte „zeichnen“, was wahrscheinlich, insofern man es anschaulich gestalten will, eine Heraklesaufgabe ist, da trotz allem eine Körper-Erstellung in Processing wohl sehr fisselig ist. Lieber wäre es uns, wenn wir Bilder in Processing importieren, die die Bewegung mit Pfeilrichtung zeigt. Hier kann man Fotos oder Zeichnungen (Clip-Art ähnlich) verwenden.

## Ggf. Roboter, der den Würfel lösen kann

## Konstruktion des Roboters

### Aufgaben

Wir möchten einen Roboter konstruieren und programmieren, der den vorher berechneten Lösungsalgorithmus selbst durchführen und den Würfel somit entwirren kann.

Der Roboter muss einzelne Seiten des Würfels gegeneinander verdrehen können. Dies muss sehr akkurat sein, da sich der Würfel sonst verhaken kann. Es gibt verschiedene Herangehensweisen, solch eine Konstruktion zu bauen, zum Beispiel:

Fixierung aller sechs Mittelsteine mit Stangen: <https://www.youtube.com/watch?v=by1yz7Toick>

Die drehbaren Stangen können jede einzelne Seite des Würfels drehen, der komplette Würfel kann jedoch seine Position nicht verändern. Diese Konstruktion ist vor allem auf Schnelligkeit aus. Sie ist vermutlich sehr schwer zu bauen und man muss den Würfel fest in den Roboter einbauen, was auch das ***Scannen der Farben mit einer Webcam*** erschweren würde.

Mit vier 180/360°drehbaren Greifarmen: <https://www.youtube.com/watch?v=Q8BYKwbwZSM>

Ein einzelner Greifarm bzw. die beiden gegenüberliegenden Greifarme gemeinsam können den gesamten Würfel kippen. Um einzelne Seiten zu drehen, müssen ein oder mehrere Greifarme den Würfel fixieren, während ein aktiver Arm die betreffende Seite greift und dreht. Die Schwierigkeit dieser Konstruktion ist, dass alle Greifarme präsize zusammenarbeiten müssen um den Würfel passgenau einspannen und drehen zu können. Außerdem benötigt man voraussichtlich zwei Motoren pro Greifarm, was einen relativ hohen Materialaufwand bedeutet.

Mit einer Plattform und einem Arm: <https://www.youtube.com/watch?v=dreTvumjNyw>

Der Würfel befindet sich auf einer drehbaren Plattform. Mithilfe des Arms kann der Würfel um 90° gekippt werden- je nach Konstruktion zieht der Arm die hintere Kante zu sich oder drückt die vordere Kante von sich weg. Die Drehung der einzelnen Seiten erfolgt durch die Plattform. Hierzu wird der Würfel mit der entsprechenden Seite nach unten gekippt, anschließend fixiert der Arm den Würfel in seiner Position, sodass nur die untere Seite beweglich ist. Besonders anspruchsvoll bei dieser Konstruktion stellen wir uns den Bau des Arms vor, der sehr präzise arbeiten muss, um den Würfel im richtigen Winkel zu treffen und ihn somit zu kippen. Dadurch, dass nur die Plattform wirklich Drehungen durchführen kann, dauert es außerdem vermutlich länger als die anderen beiden Versionen. Es werden jeoch voraussichtlich weniger Motoren benötigt und das ***Scannen der Farben mit einer Webcam*** ließe sich einfach realisieren, da der Würfel recht frei steht.

Wir tendieren für unsere Konstruktion zur zweiten oder dritten Variante, da diese unseren Mitteln und Zielen besser entsprechen.

### Materialien

Bis zu **8 Motoren**, am besten 360° drehbar

**Arduino(s)** zum Steuern

**Batterien**, um die Motoren anzutreiben

**Haushaltsgegenstände, LEGO, Holz, Schnüre,…** zum Bau der Mechanik

### Wissen

**Optimale, präsize Konstruktion herausfinden** (Videos von Beispielen anschauen, Online-Anleitungen, z.B. von LEGO lesen, Try-and-Error-Methode)

### Risiken

Wir könnten am Bau einer guten Konstruktion scheitern, wenn wir es nicht schaffen, die Teile präzise zusammenzubauen, oder das ***Programmieren des Drehroboters*** uns überfordert. Da wir recht viele Motoren benötigen werden, könnte auch die Stromversorgung zum Problem werden. Wenn einzelne Motoren aufgrund magelnden Stroms ins Stocken geraten, führt das schnell zur Fehlfunktion des Roboters. Eine wichtige Komponente ist außerdem die begrenzte Zeit, die uns einen Strich durch die Rechnung machen könnte.

Falls wir es nicht schaffen, den Roboter zu konstruieren, können wir uns auf eine Bild-/ Textausgabe des ***Lösungsalgorithmus***‘ beschränken. Die mechanischen Drehungen müssten dann von einem Menschen ausgeführt werden, Ruby würde jedoch alle Anweisungen geben.

## Programmieren des Drehroboters

### Aufgaben

Der schönste konstruierte Roboter nützt uns nichts, wenn wir kein Programm schreiben können, welches ihn steuern kann.

Das Programm muss den errechneten ***Lösungsalgorithmus*** aufnehmen und in entsprechende Drehbewegungen umrechnen. Diese müssen and die Steuerungselemente der Greifarme/ Plattform geschickt werden und ausgeführt werden. Auch hier ist ein präzises Arbeiten notwendig, um den Würfel nicht zu verhaken oder versehentlich in eine falsche Position zu bringen.

### Materialien

**Arduino(s)**

**Processing**

### Wissen

**Verknüpfung von Arduino und Processing für das Programm** (Corvin fragen, online recherchieren)

Evtl. **Verknüpfung mehrerer Arduinos** (Corvin/ Darya fragen, online recherchieren)

**Hilfreiche Programmierbefehle** (Corvin/ Darya fragen, online recherchieren, Try-and-Error)

### Risiken

Wie schon bei der ***Konstruktion des Roboters*** erwähnt, könnten wir schlicht und einfach am Schreiben des Programms scheitern, weil uns die notwendigen Fähigkeiten fehlen. Hinzu kommen Zeitdruck und die begrenzten Kapazitäten eines Arduinos. Letzteres könnten wir durch die Nutzung mehrerer Arduino-Elemente zu umgehen versuchen, dies würde das Schreiben des Programms aber vermutlich noch verkomplizieren.

Auch hier wäre die Alternative, eine digitale Ausgabe des ***Lösungsalgorithmus***‘ anzustreben.

Kein scannen: Grafische Eingabe auf dem PC

Mehrere Scanvorgänge