

GYMNASIUM EDERTALSCHULE, FRANKENBERG-HESSEN

Jugend forscht 2020

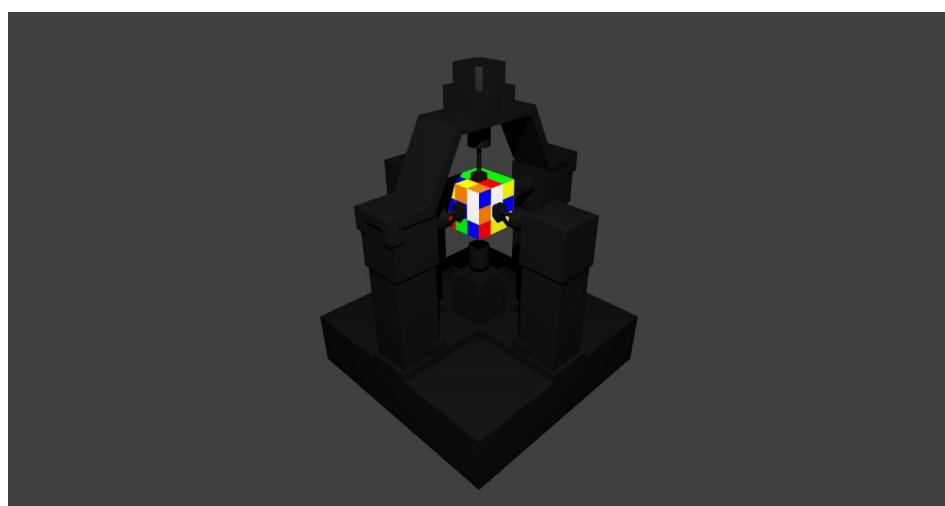
Philipp Geil, 19 Jahre

Tim Depner, 18 Jahre

Bastian Engel, 19 Jahre

Das Lösen eines Zauberwürfels - der Rubinator3000 -

Mathematik-Informatik



Projektbetreuer: Oliver Blinn, Edertalschule Frankenberg

Kurzfassung

Bei unserem Projekt *“Das Lösen eines Zauberwürfels - der Rubinator3000”* versuchten wir, einen verdrehten Zauberwürfel automatisch zu lösen.

Dazu haben wir Software und Hardware entwickelt, die einen Würfel erkennen, verarbeiten und einen Lösealgorithmus anwenden können.

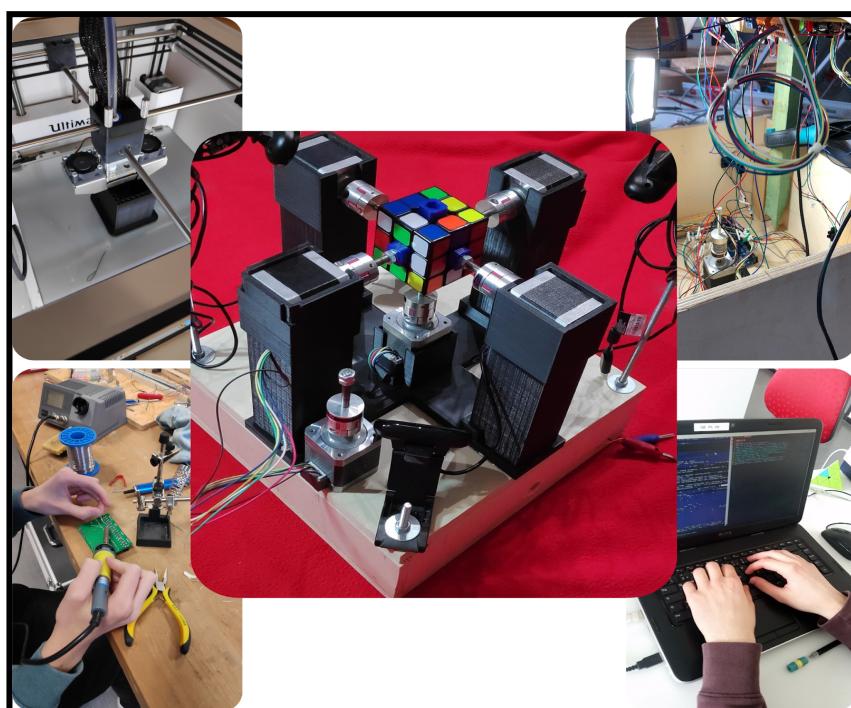
Anforderungen an die Hardware:

- Drehung jeder Seite des Würfels mit Schrittmotoren
- Elektronik zur Steuerung der Motoren
- Kameras zur Erkennung des verdrehten Würfels

Anforderungen an die Software:

- Auswertung der Kamerabilder und programmietechnische Darstellung des Würfels
- Lösealgorithmus entwickeln und Lösezüge berechnen
- Lösezüge am Würfel ausführen

...und natürlich das Vereinen aller Hardware-Komponenten in einer Maschine sowie dessen Kommunikation mit dem Rechner, auf welchem die Steuerungssoftware mit grafischer Benutzeroberfläche läuft.



Kurzfassung	2
1. Einleitung	4
1.1 Wieso einen Zauberwürfel lösen?	4
1.2 Gängige Lösemethoden	4
1.3 Das Ziel der Arbeit	5
2. Vorgehensweise, Materialien und Methode	6
2.1 Die Position der Farben auf dem Würfel	6
2.2 Drehungen der Seiten	6
2.3 Die Lösung des Zauberwürfels	7
2.3.1 Das weiße Kreuz	7
2.3.2 Das F2L	8
2.3.4 Die letzte Seite	8
2.4 Die Entwicklung der Software	8
2.5 Die Entwicklung der Hardware	9
2.6 Der Bau der Hardware	10
2.7 Die Digitalisierung des Würfels	11
3. Ergebnisse	13
3.1 Software	13
3.2 Hardware	15
4. Ergebnisdiskussion	15
5. Zusammenfassung	17
6. Danksagung	18
7. Quellen- und Literaturverzeichnis	18

1. Einleitung

1.1 Wieso einen Zauberwürfel lösen?

Unsere Faszination für Zauberwürfel hat vor ungefähr einem Jahr angefangen, als wir bei einer extrakurrikularen E-Technik-Vorlesung ein YouTube-Video (Q1) sahen, in dem jemand einen Zauberwürfel in 4.73 Sekunden löst. Da haben wir uns gefragt, wie so etwas überhaupt möglich ist. Wir haben es als Kinder selbst probiert und sogar bis zu drei Seiten gelöst bekommen, aber vollständig klappte es nie. Bei nachfolgender Recherche im Internet sind wir auf Methoden gestoßen, welche von sogenannten "Speedcubern" verwendet werden, um den Würfel in einer unglaublich kurzen Zeit zu lösen.

Vertieft wurde unser Interesse weiter durch die unzähligen Variationen des Würfels. Abseits der bekannten 3x3-Version ist der Zauberwürfel auch in einigen anderen Größen von 2x2 bis zu 15x15 erhältlich. Dazu gibt es das Drehpuzzle nicht nur in Hexaeder-Form, sondern auch in anderen polyedrischen Formen, wie beispielsweise dem Tetraeder. Einige stellen sich sogar der Herausforderung, den Würfel mit nur einer Hand, mit den Füßen oder mit verbundenen Augen zu lösen.

Wir selbst schaffen es per Hand, den Zauberwürfel in einer Zeit von circa 45 Sekunden zu lösen. Schaffen wir es, diese Zeit mit unserer Maschine zu unterbieten?

1.2 Gängige Lösemethoden

Die Lösung des Zauberwürfels erfolgt nicht improvisiert, sondern läuft nach einem strikten Schema. Dies ist notwendig, um bereits gelöste Teile nicht erneut zu verdrehen. Manche Teile lassen sich zwar ohne Vorkenntnisse lösen, aber den Hauptteil löst man mit vorher entwickelten und auswendig gelernten Algorithmen und Methoden, die den Würfel auf eine bestimmte Art und Weise permutieren.

Die am häufigsten verwendeten Lösemethoden sind zum einen die "Fridrich"-Methode, welche von Jessica Fridrich entwickelt wurde, und zum anderen die "Layer-by-Layer"-Methode.

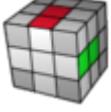
Die Fridrich-Methode enthält viele Algorithmen, die man auswendig lernen muss, welche jedoch eine kurze Lösezeit garantieren, weshalb sie auch von Speedcubern verwendet wird.

Die Layer-by-Layer-Methode ist einfacher zu erlernen, erfordert aber mehr Züge, was die Lösezeit natürlich verlängert.

Layer-by-Layer-Methode (Q3)

Ausgang	Weißes Kreuz	Weisse Ecksteine	Mittlere Schicht	Gelbes Kreuz	Gelbe Ecksteine	Ecksteine drehen
						

Fridrich-Methode (Q4)

Ausgang	Weißes Kreuz	“First two layers” (F2L)	“Orient last layer” (OLL)	“Permute last layer” (PLL)
				

Man erkennt, dass die Fridrich-Methode weniger Schritte benötigt. Dadurch wird sie jedoch auch komplizierter und schwieriger zu erlernen. Die Layer-by-Layer-Methode erfordert neun verschiedene Algorithmen, während es bei der Fridrich-Methode schon 78 sind.

1.3 Das Ziel der Arbeit

Wie schon erwähnt, brauchen wir ganze 45 Sekunden, um den Zauberwürfel per Hand zu lösen und vergessen häufig Algorithmen. Diesen Problemen wollen wir durch maschinelles Lösen entgegentreten. Motoren können die Drehungen am Würfel besser koordinieren als wir mit unseren Händen. Außerdem können Rechner Algorithmen ohne Probleme speichern und ausführen. Besonders beeindruckt hat uns dabei der Löseroboter von Ben Katz, welcher in der Lage ist, den Würfel in 0.38 Sekunden zu lösen (Q5).

Unser Ziel ist es, den Zauberwürfel innerhalb von 30 Sekunden lösen zu können, d.h. konkret, eine dafür nötige Farberkennung zu entwickeln, eine entsprechende Lösesoftware zu schreiben und eine Ansteuerung zu bauen, welche diese Anforderung erfüllen kann.

2. Vorgehensweise, Materialien und Methode

2.1 Die Position der Farben auf dem Würfel

Sei \mathbb{F} die Menge der Seiten des Würfels und \mathbb{T} die Menge der Flächen einer Seite. Damit sind die Positionen der einzelnen Flächen auf dem Würfel bestimmt durch:

$$\begin{aligned}\mathbb{P} &= \mathbb{F} \times \mathbb{T} \\ \mathbb{F} &= \{L, U, F, D, R, B\} \quad \mathbb{T} = \{0, 1, 2, 3, 4, 5, 6, 7, 8\} \\ \Rightarrow \mathbb{P} &= \{(L, 0), (L, 1), \dots, (B, 7), (B, 8)\} = \{(a, b) \mid a \in \mathbb{F} \wedge b \in \mathbb{T}\}\end{aligned}$$

Um den Würfel darstellen zu können, müssen die Positionen noch durch die Menge der Farben \mathbb{C} des Würfels ergänzt werden:

$$\mathbb{C} = \{ \text{Orange, Weiss, Gruen, Gelb, Rot, Blau} \}$$

Damit besteht die Darstellung des Würfels aus einem Array der Länge 6, in welchem die neun Farbwerte für jede Seite gespeichert sind. Das Programm geht davon aus, dass sich die orange Fläche links befindet und die weiße Fläche oben. Da die Implementation des Lösungsalgorithmus leichter fällt, wenn man nicht mit den einzelnen Flächen arbeitet, sondern jene Flächen zu Steinen zusammenfasst, welche über die Würfelkanten aneinander grenzen, haben wir uns dazu entschieden, eine Struktur für die Ecksteine (3 Flächen) und eine für die Kantensteine (2 Flächen) zu erstellen. In diesen Strukturen werden ein Tupel der Farben des Steines sowie ein Verweis auf den Würfel, zu dem dieser Stein gehört, gespeichert. Die Steine werden bei der Initialisierung der Cube-Klasse automatisch aus den vorgegebenen Positionen erstellt, die für jeden Stein festgelegt sind. Die Steine werden durch ihre Farben charakterisiert, da die Positionen zum einen durch die Indizes des Arrays festgelegt sind und es zum anderen für den Lösungsalgorithmus nötig ist, die Position gewisser Steine anhand der Farben zu identifizieren. Diese Positionen lassen sich durch einen Vergleich der Farben des Steines mit den vorgegebenen Steinpositionen bestimmen.

2.2 Drehungen der Seiten

Um den Zauberwürfel nun zu lösen oder zu verdrehen, mussten wir also noch eine Möglichkeit implementieren, um einzelne Seiten des Würfels im Programm zu drehen. Eine Drehung wird durch die zu drehende Seite und die Anzahl der Vierteldrehungen (90°) im Uhrzeigersinn festgelegt.

$$\mathbb{M} = \{(a, b) \mid a \in \mathbb{F} \wedge b \in \{0, 1, 2, 3\}\}$$

Jeder Move beschreibt eine bestimmte Permutation des Würfels, da zum einen die Farben auf der jeweiligen Seite, aber auch teilweise die Farben auf den angrenzenden Seiten vertauscht werden.

Diese beiden Permutationen können jeweils durch Funktionen beschrieben werden:

$$f_1: t \rightarrow s, \quad t \in \mathbb{T} \wedge s \in \mathbb{T}$$

$$f_2: p \rightarrow q, \quad t \in \mathbb{P} \wedge q \in \mathbb{P}$$

Betrachtet man die Drehung einer Seite im Uhrzeigersinn, so lässt sich f_1 bestimmen:

$$\begin{array}{cccccc} 0 & 1 & 2 & 6 & 3 & 0 \\ 3 & 4 & 5 & \rightarrow & 7 & 4 & 1 \\ 6 & 7 & 8 & & 8 & 5 & 2 \end{array}$$

Der Funktionswert von f_1 gibt hierbei die neue Position nach der Drehung an:

$$f_1 = \{(0, 2), (1, 5), (2, 8), (3, 1), (4, 4), (5, 7), (6, 0), (7, 3), (8, 6)\}$$

$$f_1 = 3(t \bmod 3) + 2 - \left\lfloor \frac{t}{3} \right\rfloor = 3(t - \left\lfloor \frac{t}{3} \right\rfloor \cdot 3) + 2 - \left\lfloor \frac{t}{3} \right\rfloor = 2 + 3t - 10 \left\lfloor \frac{t}{3} \right\rfloor$$

Die Funktionswerte von f_2 lassen sich nicht allgemein formulieren, da f_2 abhängig von der gedrehten Seite ist. Die Relationen von f_2 sind im Programm hinterlegt und werden auf die entsprechenden Seiten angewendet.

2.3 Die Lösung des Zauberwürfels

2.3.1 Das weiße Kreuz

Das weiße Kreuz wird durch ein Objekt der "CrossSolver"-Klasse gelöst. Dazu wird zuerst ein Pivotelement festgelegt. Bei diesem Element handelt es sich entweder um einen weißen Kantenstein, der sich bereits richtig auf der weißen Seite befindet und bei dem sich die meisten Steine in der gewünschten Position befinden, oder um einen weißen Kantenstein, der sich mit den wenigsten Zügen in die richtige Position bringen lässt. Um die Anzahl der Züge zu bestimmen, die benötigt werden, um den Stein relativ zum Pivotelement richtig auszurichten, gibt es eine Methode, die jeden Stein bewertet und die Anzahl der Züge zurückgibt. Beim Lösen des weißen Kreuzes bewertet der "CrossSolver" zuerst für jeden Kantenstein, wie viele Züge jeweils benötigt werden und löst dann den Kantenstein, für den die wenigsten benötigt werden. Zum Lösen der einzelnen Steine unterscheidet der "CrossSolver" zwischen vier verschiedenen Fällen:

1. Der Stein befindet sich mit der weißen Fläche auf der weißen Seite, aber die seitliche Farbe stimmt nicht.
2. Der Stein befindet sich mit der weißen Fläche auf der unteren Seite.
3. Der Stein befindet sich auf der mittleren Ebene, also mit keiner Fläche auf der oberen oder unteren Seite.
4. Der Stein befindet sich mit der weißen Fläche auf einer der seitlichen Seiten (orange, grün, rot, blau) und mit der anderen Fläche auf der oberen oder unteren Seite.

Die einzelnen Fälle werden dann durch vorgegebene Algorithmen gelöst.

2.3.2 Das F2L

Das Lösen des F2L funktioniert analog zum Lösen des weißen Kreuzes. Das Programm orientiert sich hier nun nicht mehr an nur einem Stein, sondern an jeweils zwei Steinen, die gelöst werden sollen. Dazu bewertet es wieder jedes Steinpaar und löst jenes, welches die wenigsten Züge benötigt. Diese Berechnung erfolgt asynchron, um Rechenzeit zu sparen. Danach wird das Paar mit Hilfe von 13 Algorithmen gelöst.

2.3.4 Die letzte Seite

Für das Lösen der letzten Seite sind im Programm Muster und die dazugehörigen Zugfolgen hinterlegt (57 OLL und 21 PLL). Die Muster für das OLL beziehen sich nur auf die Anordnung der gelben Flächen. Dazu wird eine Folge von 9 Bits verwendet, welche die gelbe Seite repräsentiert, und jeweils vier Folgen von je 3 Bits, welche die 3 Flächen der über die Würfelkanten angrenzenden Seiten repräsentieren. Das Programm vergleicht dann jegliche Muster mit dem Würfel und kann somit die Zugfolge angeben. Die PLL-Muster sind etwas komplexer als die OLL-Muster, da das PLL die einzelnen Farben einbezieht. Dazu wird jeder der vier seitlichen Farben eine Zahl zugeordnet ((orange, 0), (grün, 1), (rot, 2), (blau, 3)). Im Programm sind dann die Differenzen zwischen den drei Steinen für alle vier Seiten hinterlegt. Mit diesen Mustern kann daraufhin die Zugfolge bestimmt werden, die zur Lösung des Zauberwürfels führt.

2.4 Die Entwicklung der Software

Wir haben mithilfe von Visual Studio Community 2019, einer von Microsoft entwickelten IDE, programmiert. Das Projektmanagement wird dadurch vereinfacht und der integrierte Designer erlaubt einfache und gute GUI-Erstellung. Da jeder von uns schon im Voraus Erfahrung mit dieser Entwicklungsumgebung hatte, mussten wir uns nicht umstellen und der Entwicklungsprozess konnte sofort starten.

Als Programmiersprache entschieden wir uns für C#, um WPF (Windows Presentation Foundation) nutzen zu können. Das ist eine GUI-Library, mit der viele Windows-Programme geschrieben werden. Ein großer Vorteil davon ist, dass das GUI-Design extern in XAML-Dateien definiert und damit vollständig von der Logik des Programms abgekoppelt ist. Dies ermöglichte uns, die Arbeit besser aufzuteilen. So kann ein Programmierer am GUI arbeiten, während ein anderer die Logik dazu schreibt. Die 2D-und 3D-Ansicht des Würfels erstellten wir mit OpenGL, einer Graphics-Library, die auch in vielen Computerspielen verwendet wird und sehr weitreichende Funktionen besitzt.

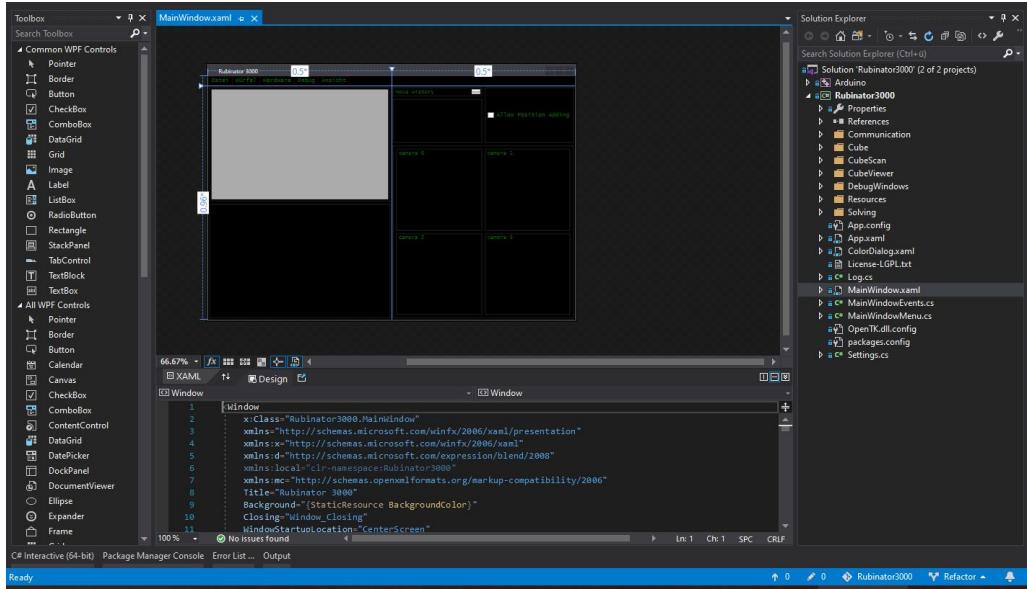


Abbildung 1, GUI-Entwicklung mit Visual Studio, Screenshot 04.01.2020

Da wir zu dritt waren und viele Änderungen parallel zusammenkamen, benötigten wir ein Versionskontrollsystem, um diese zu verwalten und gegebenenfalls bei Bugs frühere Versionen wiederherstellen zu können. Entschieden haben wir uns für Git, welches auch industriell verwendet wird. Damit war es kein Problem mehr, Aufgaben aufzuteilen und zu synchronisieren.

2.5 Die Entwicklung der Hardware

Das CAD-Modell unseres „Rubinator3000“ haben wir mit dem kostenlosen Programm „FreeCAD“ entworfen, da dieses für unsere Anforderungen genügt und wir unser Projekt kostengünstig gestalten wollten. Zudem war es uns mit FreeCAD möglich, die Bauteile in das STL-Format zu exportieren, um sie mit einem 3D-Drucker drucken zu können. Da unsere Hardware fast ausschließlich aus 3D-Druck Bauteilen besteht, war diese Funktion sehr hilfreich. Die Schaltpläne für die Ansteuerung der Schrittmotoren erstellten wir mit Eagle. Eagle bietet zudem die Möglichkeit, Leiterplatten zu designen, um diese bei einem Leiterplattenhersteller anfertigen zu lassen.

Die Ansteuerung der Schrittmotoren musste über eine Verstärkerschaltung getätigter werden. Das lag zum einen daran, dass die Schrittmotoren eine Spannung von 12V benötigen, während der Arduino nur 5V liefert. Zum anderen wird ein sehr großer Strom benötigt, um die Magnetfelder innerhalb der Schrittmotoren mit Hilfe von Spulen zu erzeugen.

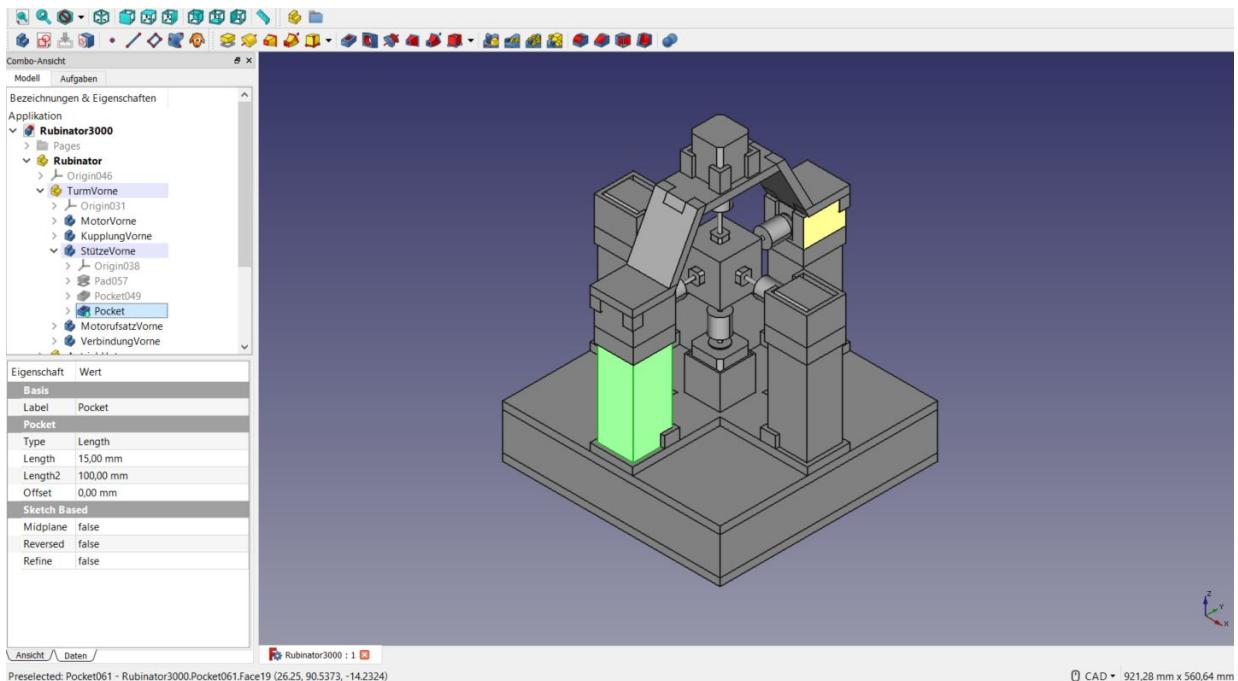


Abbildung 2: 3D-Design mit FreeCAD, Screenshot 04.01.2020

2.6 Der Bau der Hardware

Wir haben uns für 3D-gedruckte Bauteile entschieden, da diese eine höhere Präzision der Mechanik ermöglichen. Unser erster Prototyp bestand ausschließlich aus Holz und es kam aufgrund mangelhafter Genauigkeit der Mechanik zum sogenannten “Verkantungsproblem”: Eine Seite des Würfels verkantete sich nach einer unvollständigen Drehung. Dieses Problem lösten wir letztendlich durch die zweite Version der Hardware und einen anderen Zauberwürfel. Zudem ließen die verwendeten Miniaturbalg-Kupplungen zu viel Spielraum zu, was zu Ungenauigkeiten bei den Würfeldrehungen führte. Deshalb ersetzten wir sie durch Elastomerkupplungen, da diese deutlich genauer sind, wodurch sich der Würfel viel besser drehen lässt.

Zudem haben wir uns Platinen zur Ansteuerung der Schrittmotoren anfertigen lassen, da die anfangs selbst gelöteten Platinen teilweise unzuverlässig waren und sehr viel Platz im Gehäuse in Anspruch nahmen. Hinzu kam noch, dass die selbst gelöteten Platinen viel Wärme entwickelten, da jede einzelne von ihnen einen

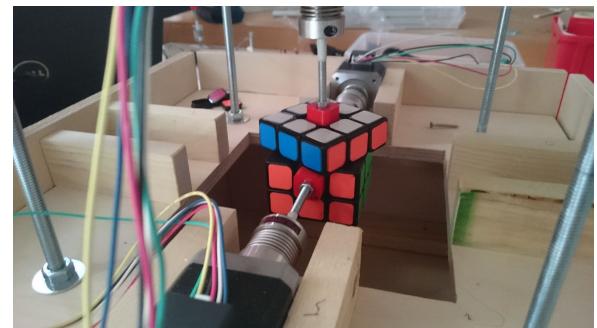


Abbildung 3: Der erste Prototyp, Bild 14.05.2019

separaten Spannungsregler besaß, der für den Anschluss eines Mikrocontrollers gedacht war.

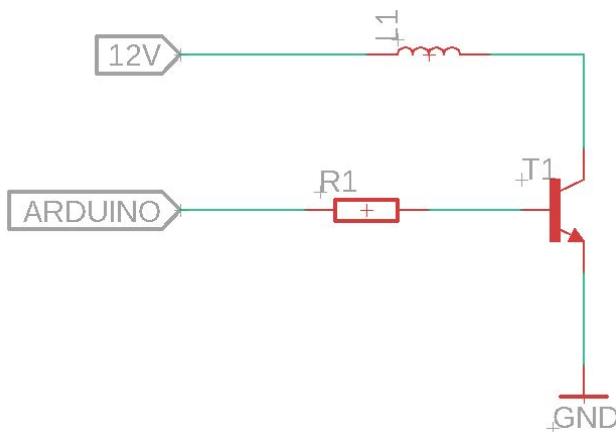


Abbildung 4: Ansteuerung Schrittmotor,
gezeichnet mit EAGLE 14.01.2020

Im nebenstehenden Bild ist eine Schaltung abgebildet, welche zeigt, wie mithilfe des Transistors (T1) der Stromkreis der Spule des Schrittmotors (L1) geschlossen wird¹. Dadurch wird es möglich, dass Strom fließen kann, welcher das magnetische Feld im Schrittmotor erzeugt. Der Basiswiderstand (R1) dient zur Begrenzung des Basisstroms des Transistors, da der Ausgang des Arduino einen Strom von maximal 40mA zur Verfügung stellt.

Wir ließen die Bodenplatte bei der Firma Viessmann drucken, weil diese für unseren 3D-Drucker zu groß war. Sie stellte uns für diesen Druck einen besseren 3D-Drucker zur Verfügung. Nach dem Druck haben wir sie auf eine solide Holzbox gesetzt, welche auch die Elektronik enthält. Dadurch wird Stabilität garantiert und die Motoren können passend positioniert werden.

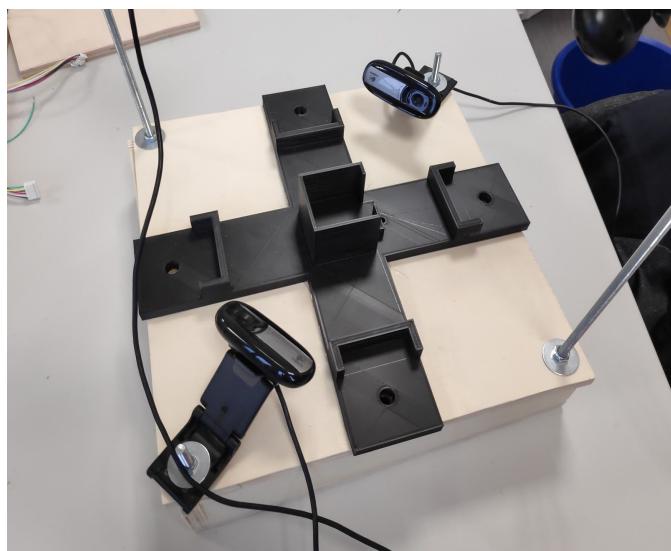


Abbildung 5, Bodenplatte, Bild 06.12.2019

2.7 Die Digitalisierung des Würfels

Unsere erste Methode zur Digitalisierung des Würfels sah folgendermaßen aus:

¹ Die Freilaufdiode wurde zur Vereinfachung der Schaltung weggelassen

Wir lasen jede einzelne Seite mit der Kamera eines Smartphones ein, um daraus den Würfel digital zusammensetzen zu können. Dafür haben wir eine Android-App in der Programmiersprache Java geschrieben.

Ein Computer – also auch das Smartphone, welches wir benutzen – stellt Farben im RGB-Format dar. Anhand dieser Werte werden die einzelnen Würfel-Farben (Orange, Weiß, Grün, Gelb, Rot und Blau) auch unterschieden.

Um sich auch wechselnden Lichtverhältnissen anpassen zu können, war es nötig zu Beginn Referenz-RGB-Werte festzulegen, anhand welcher die App die Farben unterschied.

Im nächsten Schritt hielten wir die Handykamera vor jede Würfelseite. Die App las die RGB-Werte der einzelnen Flächen aus und ordnete ihnen anhand der anfangs eingelesenen Referenz-RGB-Werten die entsprechende Farbe zu.

Sobald die App eine Würfelseite eingelesen hatte, schickten wir dessen Farben über ein WLAN-Netzwerk an jenen Laptop, auf welchem das Hauptprogramm lief.

Nachdem wir auf diese Weise alle sechs Seiten eingelesen und an den Laptop geschickt hatten, könnte dieser daraus den Würfel digital zusammensetzen.

Die zweite Version unseres Projekts brachte auch ein Upgrade in der Farberkennung mit sich. Anstatt mit einer Handy-App jede Seite einzeln einzulesen, brachten wir vier stationäre USB-Kameras an der Maschine an, mit welchen wir die RGB-Werte der einzelnen Flächen des Würfels automatisch einlesen. Dies beschleunigt den Prozess der Digitalisierung des Würfels natürlich enorm.

Auch an der Erfassung und Verarbeitung der eingelesenen RGB-Werte haben wir Änderungen vorgenommen:

Für jede einzelne Fläche des Würfels legen wir eine Position auf einem der 4 Kamera-Streams fest, an welcher die RGB-Werte der jeweiligen Fläche ausgelesen werden sollen. Da die 8 Mittelsteine des Würfels ihre Position während des Drehen nie verändern, werden deren RGB-Werte gar nicht mehr eingelesen.

So lesen wir alle RGB-Werte an den 48 Positionen ein und berechnen die jeweilige Wahrscheinlichkeit dafür, dass die jeweiligen RGB-Werte zu einer orangen, weißen, grünen, gelben, roten oder blauen Fläche gehören.

Die 8 Flächen an den Positionen mit den höchsten Orange-Wahrscheinlichkeiten bestimmen wir als orange Flächen. So gehen wir die restlichen 5 Farben ebenfalls durch und haben zum Schluss 8 Flächen von jeder Farbe, mit welchen wir wiederum den Würfel digital darstellen.

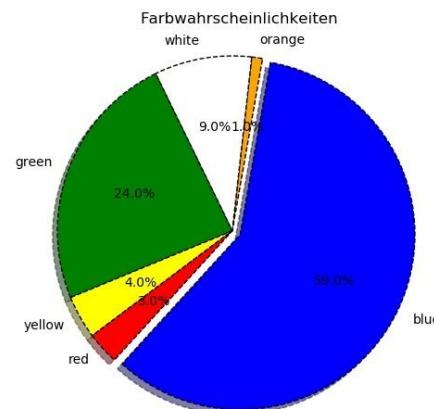


Abbildung 6, Farb-Wahrscheinlichkeiten für blaue Fläche,
erstellt mit pyplot am 16.01.2020

Durch dieses Verfahren wird die Farberkennung viel präziser, da wir davon ausgehen, dass wir genau 8 Flächen von allen Farben haben.

Ein Problem, welches wir bei der Farberkennung hatten war Folgendes:

Aufgrund der sehr glatten Oberfläche der Würfelflächen, spiegelten diese das Umgebungslicht teilweise gebündelt direkt in eine der Kameralinsen. Dadurch war im Kamerabild nicht die entsprechende Farbe der Seite zu sehen, sondern ein helles, weißliches Leuchten. Dies verfälschte die RGB-Werte, welche für die dazugehörige Fläche eingelesen wurden. Eine eigentlich blaue Fläche erschien somit unerwünschterweise weiß. Wir haben dieses Problem gelöst, indem wir einen Vorhang über die ganze Maschine gehängt haben, um den Spiegelungseffekt zu minimieren. Damit wir jedoch überhaupt noch etwas mit den Kamera sehen können, haben wir zusätzlich LEDs direkt in die Maschine gebaut, welche den Würfel mit diffusem Licht bescheinen.

Dies soll jedoch nur vorübergehend die Lösung des Problems sein, da das Abdecken und das Entfernen der Abdeckung recht viel Zeit braucht. Für die Zukunft planen wir, nicht absolute Positionen, an welchen die Farben des Würfels ausgelesen werden, zu benutzen. Stattdessen soll das Programm mittels Konturerkennung selbst herausfinden, wo sich auf dem entsprechenden Kamerabild die einzelnen Würfel-Flächen befinden, von welchen das Programm daraufhin die entsprechenden Farben ausliest. Hierbei könnten wir zu helle Pixel aussortieren, wodurch das "Spiegelungs-Problem" gelöst würde. Erste Tests in separat von uns programmierten Anwendungen liefern vielversprechende Ergebnisse. Jetzt gilt es noch, die neue Farberkennung in das Hauptprogramm zu implementieren.

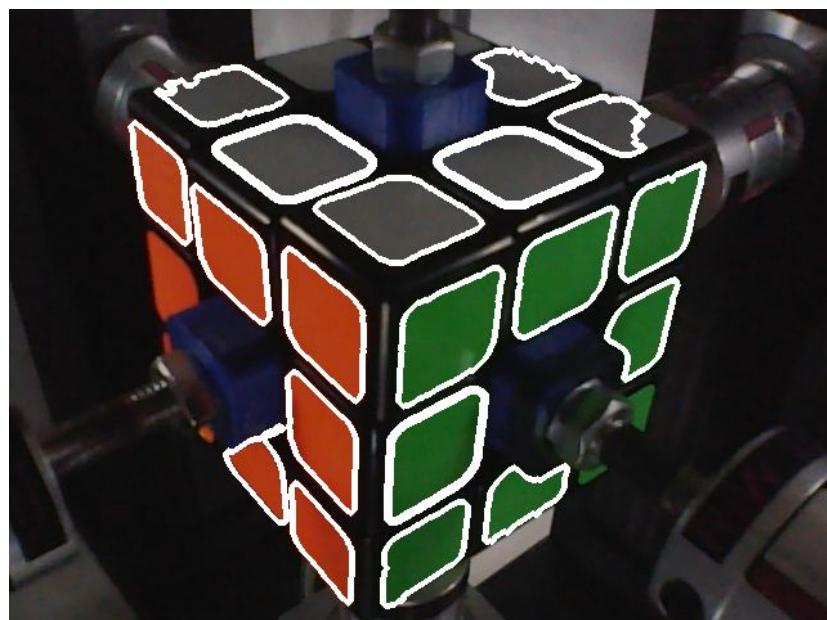


Abbildung 7, Kantenerkennung der Würfelflächen, Bild 22.02.2020

3. Ergebnisse

3.1 Software

Wir haben mithilfe der vorgestellten Vorüberlegungen und Methoden das Hauptprogramm fertiggestellt, welches den Würfel lösen und die Lösealgorithmen über eine serielle Schnittstelle an den Arduino schicken kann.

Die Effizienz der Algorithmen lässt sich an folgendem Graph ablesen:

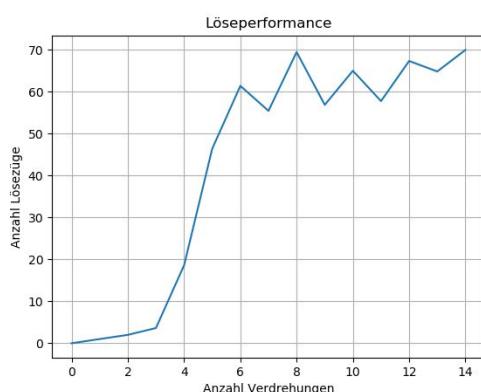


Abbildung 8, Lösperformance, erstellt mit pyplot, 13.01.2020

Für jede Verdrehungsanzahl machten wir 30 Testdurchgänge, woraufhin daraus der Mittelwert gebildet wurde. Die Schwankungen lassen sich dadurch erklären, dass nur ein einziger zusätzlich ausgeführter Algorithmus mehrere Züge enthält und dadurch das Ergebnis stark beeinflusst werden kann.

Es ist zu erkennen, dass unsere Lösung ab vier Verdrehungen erheblich mehr Züge benötigt. Ab sechs Verdrehungen pendelt sich die Anzahl der Lösezüge auf ungefähr 65 ein.

Die Schrittmotoren lassen sich mit 360° pro Sekunde drehen, was zwei Zügen entspricht. Daher ist eine theoretische Lösezeit von ca. 15 Sekunden erreichbar, die aber durch Ausnahmefälle wie zum Beispiel die Doppeldrehung einer Seite oder dem *Multiturning* Schwankungen unterliegt.

Außerdem ist zu erkennen, dass der Zugzuwachs am stärksten vom FTL beeinflusst wird, wobei die OLL- und PLL-Algorithmen gegen Ende weniger beitragen. Das Kreuz hat kaum Einfluss.

Die grafische Oberfläche des Programms enthält alle nötigen Kontrollelemente, so zum Beispiel

das Verdrehen oder Lösen des Würfels oder Debug-Fenster zum Beheben von fehlerhaften Algorithmen und Mustern. Dadurch können wir leicht und ohne

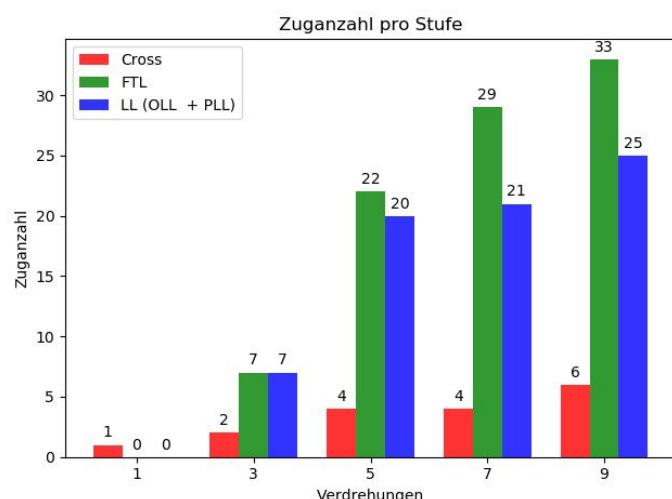


Abbildung 9, Stufenanalyse, erstellt mit pyplot, 13.01.2020

ständiges Überlegen Einstellungen ändern und Tests durchführen. Das ist zum Beispiel besonders relevant für die Verbindung zum Mikrocontroller, da diese über einen sich ändernden COM-Port stattfindet, welchen man anfangs auswählen muss.

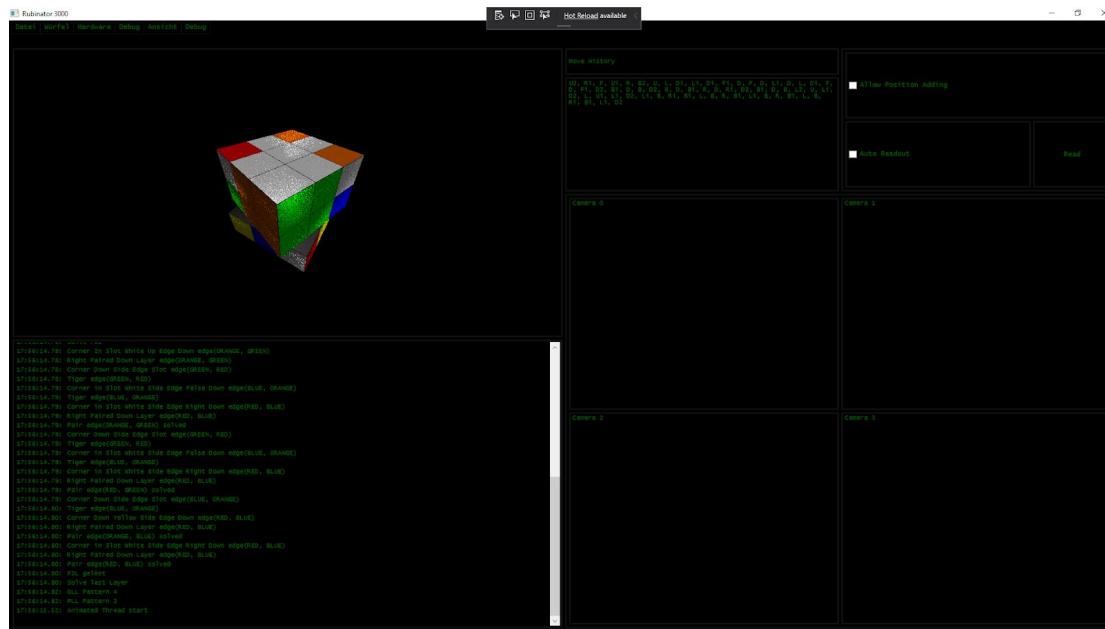


Abbildung 10, Hauptprogramm ohne Kameras, Screenshot 14.01.2020

3.2 Hardware

Der Arduino wartet nach dem Verbinden mit dem Programm auf die Befehle und führt diese aus. Er ist nötig, da die Schrittmotoren nicht direkt an den Computer angeschlossen

werden können. Zudem benötigen die Schrittmotoren sehr viel Strom. Deshalb ist es notwendig, für jeden Schrittmotor eine Schaltung zu bauen, die genug Strom liefern kann, um den Würfel zu drehen. Anfangs hatten wir die Motoren nach einer Drehung nicht abgeschaltet und so entstand das Problem, dass alle inaktiven Motoren zu

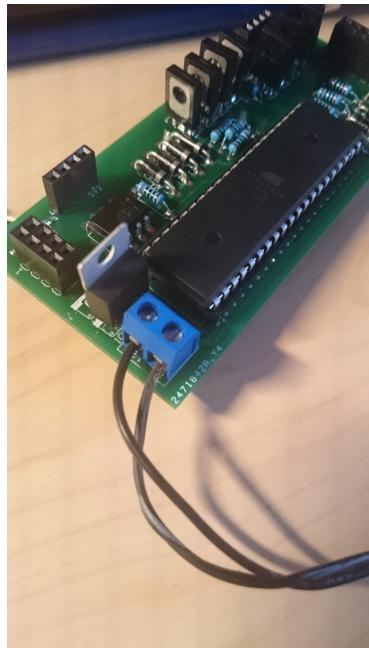


Abbildung 11.1, Platine zur Ansteuerung, Bild 13.12.2019

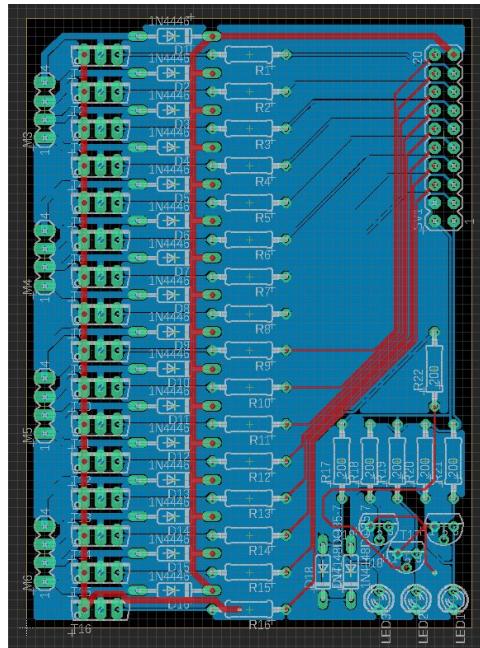


Abbildung 11.2, Platine zur Verstärkung der Arduinosignale, Screenshot EAGLE, 14.01.2020

viel Strom verbrauchten und somit nicht genug Drehmoment bereitstand, um den Würfel zu drehen. Dieses Problem haben wir softwaretechnisch gelöst, indem wir den Strom der Motoren nach jeder Drehung abschalteten und so der komplette Strom des Netzgerätes für einen oder bei Multiturns zwei Motoren bereitsteht.

4. Ergebnisdiskussion

Unser „*Rubinator3000*“ ist noch nicht das perfekte Ergebnis jahrelanger Forschung, aber es kann sich trotzdem sehen lassen.

Wie anfangs bereits erwähnt, kam uns die Idee zum Bau einer solchen Maschine, bei einer extrakurrikularen E-Technik-Vorlesung im Rahmen der Young Engineer Academy (YEA). Wir haben dort überlegt, was wir als unser Abschlussprojekt bei der YEA realisieren könnten. Da wir noch viel Zeit hatten, bis das Projekt vorgestellt werden sollte, entschieden wir uns nach einigen Suchanfragen im Internet für den Bau eines doch recht aufwendigen Zauberwürfel-Lösers. Einige weitere Kursteilnehmer, welche sich ebenfalls für den Zauberwürfel begeistern, unterstützten unsere Idee.

Anfangs schien unsere Aufgabe recht simpel, da man lediglich 6 Schrittmotoren benötigte, die die Seiten des Zauberwürfels drehen sollten.

Bereits in den nächsten Wochen (Januar 2019) starteten wir mit der Programmierung. Der eigentliche Bau der ersten Maschine begann dann Ende April 2019 und pünktlich zum Präsentationstermin am 28. Juni vollendeten wir die erste Version der Hardware. Bis dahin hatten wir während der Schulzeit schon dutzende Freistunden in unser Projekt investiert, da die Umsetzung des Projekts dann doch komplizierter war, als wir zunächst vermutet hatten. Vor allem die Drehung des Würfels machte uns zu schaffen, da die Hardware zu ungenau war und der Würfel sich deshalb immer wieder verkantete. Dies war aber letztendlich nicht das größte Problem, welches wir bei der Präsentation hatten.

Auf der Bühne scheiterte die Vorführung unserer Maschine am Lösen des Zauberwürfels durch das Programm.

Also entschlossen wir uns, in den Sommerferien das Programm zu überarbeiten. Da der bestehende Code mittlerweile sehr unübersichtlich war und wir nicht genau wussten, wo die einzelnen Fehler lagen, entschlossen wir uns dazu, das Programm von Grund auf neu zu entwickeln. Wir besaßen bereits das Wissen aus dem ersten Versuch, weshalb uns die Entwicklung der zweiten Maschine leichter fiel.

Aufgrund der Verkantungen des Würfels kam uns die Idee, die Teile der Hardware, welche eine hohe Genauigkeit erforderten, mit dem 3D-Drucker zu fertigen, da dieser viel genauere Ergebnisse liefert als das Aussägen aus Holz. Zudem haben wir neue Kupplungen eingebaut, die zur Erhöhung der Genauigkeit beitrugen.

Das Projekt erschien uns auf den ersten Blick leicht, aber es gab viele Faktoren, die wir nicht berücksichtigt hatten. Darunter fiel zum Beispiel das Einlesen des Würfels, das Bauen der Hardware mit einer hohen Genauigkeit und vieles andere.

Die Software könnte noch verbessert werden, indem man beispielsweise eine andere Methode zum Lösen des Würfels verwendet, die weniger Züge benötigt. Diese Verbesserung würde das maschinelle Lösen auch erheblich beschleunigen, da sich die Motoren dadurch weniger drehen müssten.

Zudem könnte man auch Motoren verwenden, die sich schneller ansteuern lassen, welche ebenfalls das Lösen beschleunigen würden.

Wir hatten während der Entwicklung die Idee das Hauptprogramm auf einem Raspberry Pi laufen zu lassen, da dieser den Computer ersetzen würde, der die Lösung berechnet. Diese Idee verworfen wir aber schnell, da wir schon angefangen hatten, die Software auf dem .NET Framework von Microsoft für Windows zu entwickeln, während der Raspberry auf einem linuxbasierten Betriebssystem läuft. Der Ansatz könnte aber in der Zukunft realisiert werden.

Auch die Ansteuerung der Motoren hat uns Probleme bereitet, da wir eine eigene Platine entwickelt und einen Mikrocontroller programmiert haben. Dieser ist jedoch beim Debuggen der Software durch eine Überspannung kaputt gegangen und wir mussten ihn durch einen Arduino ersetzen.

5. Zusammenfassung

Zusammenfassend lässt sich sagen, dass unser Projekt noch viele Erweiterungsmöglichkeiten bietet. Es könnten beispielsweise noch LEDs eingebaut werden, die für eine bessere Beleuchtung des Würfels bei schlechten Lichtverhältnissen sorgen.

Unser Ziel, dass unsere Maschine weniger Zeit zum Lösen des Würfels benötigt als wir selbst per Hand, hat das Projekt noch nicht ganz erreicht. Es befindet sich aber auf einem guten Weg zu einer Lösung, die unseren ursprünglichen Anforderungen gerecht wird. Dies könnte durch weitere Verbesserungen der Hard- und Software erreicht werden.

Das Projekt „Rubinator3000“ hat zwar nur einen kleinen Nutzen für die Allgemeinheit, aber es hat uns geholfen, unsere Kompetenzen im Bereich der Programmierung und der Entwicklung von Hardware zu erweitern. Zudem haben wir die Erfahrung gemacht, dass Dinge und Ideen oft einfacher erscheinen, als sie es eigentlich sind. Es lohnt sich also, zuerst darüber nachzudenken, wie man sie angeht und realisiert. Zuletzt durften wir die Erfahrung machen, was man mit Teamwork alles erreichen kann. Während des Entwicklungsprozesses haben sich alle Team-Mitglieder nur auf

jeweils ein Teilgebiet spezialisiert. Trotzdem können wir durch unsere gelungene Zusammenarbeit nun einen voll funktionsfähigen Zauberwürfellöser präsentieren.

6. Danksagung

Abschließend bedanken wir uns für jegliche Unterstützung während der Umsetzung unseres Projekts.

Exemplarisch möchten wir hier die Fachschaft Physik der Edertalschule nennen, welche uns bei Fragen in der Umsetzung beriet und uns kostenlos über einen längeren Zeitraum den Zugang zu ihrem Werkraum gewährte, in welchem wir unsere Maschine entwickeln und anfertigen konnten. Zudem danken wir unseren Mitschülern der Young Engineer Academy, die uns beim Bau der ersten Version des "Rubinator3000" unterstützt haben.

Darüber hinaus geht ein riesiges Dankeschön an die Viessmann Group, welche uns finanziell bei der Anfertigung der Hardware unterstützte. Zusätzlich ermöglichte sie uns die Teilnahme an der Young Engineer Academy, durch welche wir essentielle Grundkenntnisse und letztendlich auch die Idee für dieses Projekt bekommen haben.

7. Quellen- und Literaturverzeichnis

Q1: 04.01.2020, 16:57

<https://www.youtube.com/watch?v=KfpJZHP6DLA>

Q2: 04.01.2020 17:03

<https://speedcube.de/>

Q3: 04.01.2020 17:41

<https://ruwix.com/the-rubiks-cube/how-to-solve-the-rubiks-cube-beginners-method/>

Q4: 04.01.2020 17:41

<https://ruwix.com/the-rubiks-cube/advanced-cfop-fridrich/>

Q5: 04.01.2020 17:49

<https://www.youtube.com/watch?v=nt00QzKuNVY>

Q6: 14.01.2020 19:14

<https://www.arduino.cc/en/Tutorial/DigitalPins>