Städtisches Gymnasium Rheinbach

Facharbeit

Bau eines Zauberwürfel-Lösungsroboters und Implementierung eines Lösungsalgorithmus

Verfasser:
Keanu Fuchs

Betreuer:
R. Faßbender

Informatik LK1

April 2021

März 2020

Inhaltsverzeichnis

1. Einleitung

In dieser Facharbeit wird der Aufbau sowie der Bau eines selbstgebauten Lego Mindstorms-Roboters beschrieben und dargestellt, welcher einen Zauberwürfel lösen soll. Zusätzlich wird der Lösungsalgorithmus implementiert, welcher zum Lösen des Zauberwürfels angewendet wird. Dieser wird im weiteren Verlauf dieser Facharbeit näher erläutert und kommentiert.

1.1 Einstieg in das Thema Zauberwürfel

Den Zauberwürfel beziehungsweise „Rubik's Cube"[1] wie er im englischen genannt wird, kennt vermutlich jeder und auch so ziemlich jeder hat bereits versucht einen zu lösen. Das Problem ist nur, schaffen tut es ohne ein bestimmtes systematisches Vorgehen wohl keiner, beziehungsweise ist nahezu unmöglich. Einen Zauberwürfel zufällig zu lösen ist deshalb so unwahrscheinlich, weil es über mehr als 43 Trillionen (43.252.003.274.489.856.000) unterschiedliche Fälle gibt, wie ein Zauberwürfel vermischt sein kann.[2] Trotzdem schaffen es Menschen wie der Chinese „Yusheng Du (杜宇生)", den Zauberwürfel in 3,74 Sekunden zu lösen.[3] Dieses Fähigkeit einen Zauberwürfel in so einer Spitzenzeit zu lösen heißt „Speedcubing"[4]. Um einen Zauberwürfel in so einer Spitzenzeit so lösen, waren bestimmte Methoden eingesetzt.

Die meist vertretene Methode nennt sich „Fridrich" Methode oder auch „CFOP" Methode genannt.[5] Bei dieser durchläuft man vier Schritte um den Zauberwürfel zu Lösen. Dabei steht jeder Buchstabe bei „CFOP" für einen Schritt.[6] „C" steht für „White cross", bei welchem man, wie es der Name schon sagt, zuerst ein weißes Kreuz bildet.[7] Darauf folgt „F", was für „First two layers (F2L)" steht, auf Deutsch bedeutet das so viel wie „Die ersten beiden Ebenen". Zuletzt stehen „O" und „P" für jeweils „Orient last layer (OLL)" und „Permute last layer (PLL)", was auf Deutsch orientierung und tauschen der letzten Ebene bedeutet. Mithilfe dieser beiden Schritte ist es möglich die letzte Ebene, beziehungsweise den Zauberwürfels zu lösen.[8] Diese Schritte werden im weiteren Verlauf der Facharbeit noch näher erläutert und am Beispiel des Lösungsalgorithmus dargestellt.

1.2 Begründung der Themenwahl

An das Thema dieser Facharbeit stieß ich nun, da ich selbst sehr interessiert am Speedcubing bin. An das Hobby stieß ich das erste Mal im Jahr 2015, als ich mir aus Langeweile einen Zauberwürfel gekauft habe. Damals ging es mir noch nicht darum, den Zauberwürfel möglichst schnell zu lösen. Mein Ziel war es den Zauberwürfel überhaupt

---

[1] Geolino: „Fünfmal staunen über den Zauberwürfel".
[2] Vgl. ebd.
[3] Vgl. World Cube Association: „Rangliste".
[4] Speedcubing Schweiz: „Speedcubing".
[5] Vgl. Ruwix: „Advanced CFOP / Fridrich Method".
[6] Vgl. Ruwix: „Steps of the advanced method".
[7] Vgl. ebd.
[8] Vgl. ebd.

einmal zu lösen. Dann circa vier Jahre später habe ich das Hobby wieder aufgegriffen. Um auch schneller werden zu können, da ich mit Speedcubing anfangen wollte, habe ich mir einen professionellen Zauberwürfel gekauft. Das Problem an herkömmlichen Zauberwürfeln aus dem Einzelhandel ist nämlich, dass sich diese nur sehr schwer drehen lassen. Dies möchte man natürlich vermeiden, wenn man eine möglichst geringe Zeit erzielen möchte. In den darauffolgenden Monaten habe ich sehr viel Zeit damit verbracht den Zauberwürfel immer schneller zu lösen und neue Algorithmen zu lernen. Algorithmen werden benötigt, um bestimmte Farbkombinationen des Würfels so zu richten, dass man damit besser und schneller fortfahren kann.[9] Unteranderem habe ich dies auch in der Schule während des Unterrichtes gemacht. Zufälligerweise auch in der Informatikstunde bei Herrn Faßbender, als wir über die anstehenden Wahlen der Facharbeiten geredet haben. Auf die Frage von Herr Faßbender, wer es sich denn vorstellen könne, seine Facharbeit in Informatik zu schreiben, meinte Kevin zu mir: „Ey mach doch irgendwie ein Roboter der den Zauberwürfel löst oder so“. Die Idee fand ich super. In den folgenden Wochen habe ich zusammen mit der Hilfe von Herr Faßbender überprüft, ob die Umsetzung überhaupt möglich ist. Als sich gezeigt hat, dass dies der Fall ist, wurde dann das Thema der Facharbeit festgelegt.

2. Aufbau und Eigenschaften des Zauberwürfel-Lösungs-Roboters

2.1 Eigenschaften der Motoren

Vier, das ist die Anzahl der Motoren, welche in dem Roboter verbaut sind. Zwei dieser Motoren sind sogenannte „Große Servomotoren“ des Lego Mindstorms EV3 Modells.[10] Diese haben ein durchschnittliches Antriebsmoment von ca. 0,2 Newtonmeter und erreichen eine maximale Drehgeschwindigkeit mit bis zu 170 Umdrehungen in der Minute.[11] Es gibt zusätzlich noch einen dritten Motor dieser Art, jedoch von dem Lego Mindstorms NXT Modell, welcher Eigentum des Städtischen Gymnasium Rheinbachs ist. Die großen Servomotoren des EV3 und des NXT unterscheiden sich von ihren Eigenschaften nur minimal und sind beide mit dem Lego Mindstorms EV3 Modell kompatibel.[12] Der größte Unterschied, ist der optische. Der vierte Motor ist ein „Medium Servomotor“ welcher die Eigenschaft hat, dass dieser etwas kleiner ist als die großen Servomotoren und mit einem durchschnittlichen Antriebsmoment von 0,08Nm auch etwas weniger als halb so stark ist.[13] Jedoch hat er den Vorteil, dass er circa doppelt so viele Umdrehungen in der Minute schafft.[14]

2.2 Eigenschaften des Farbsensors

In dem Roboter befindet sich kein normaler „Lego EV3-Farbsensor“[15], stattdessen ist ein „HiTechnic NXT Color Sensor V2“[16] Farbsensor verbaut. Dieser kann 17 Farben

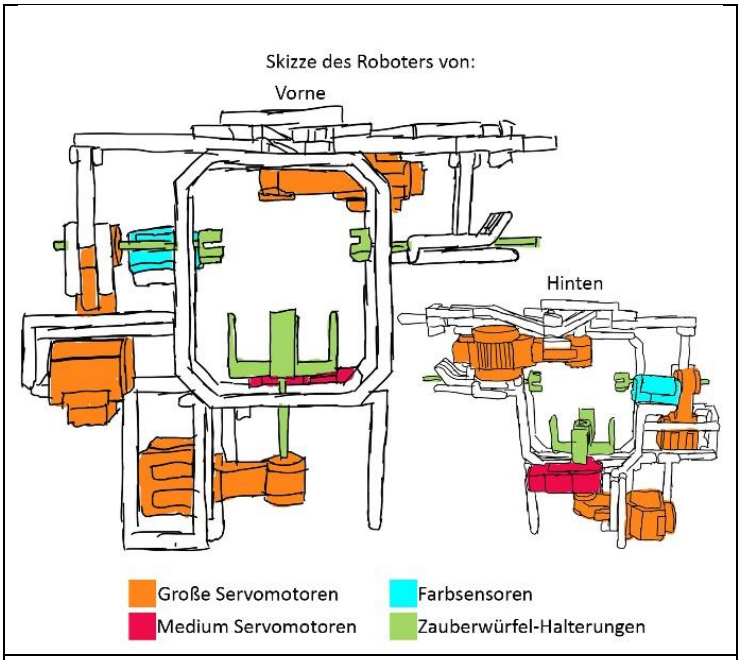
---

[9] Vgl. Frank Klautke: „Lösungsbuch für Rubik´s Zauberwürfel“.
[10] Vgl. Lego: „Großer EV3 Servomotor Produktdetails“.
[11] Vgl. ebd.
[12] Vgl. Philohome: „LEGO® 9V Technic Motors compared characteristics“.
[13] Vgl. Lego: „EV3 Servomotor medium Produktdetails“.
[14] Vgl. ebd.
[15] Lego: „EV3-Farbsensor Produktdetails“.
[16] HiTechnic Sensors: „NXT Color Sensor V2“.

erkennen und aktualisiert die Farbwerte etwa 100-mal in der Sekunde.[17] Der EV3-Farbsensor von Lego hat zwar mit 1kHz eine höhere Abtastrate, doch kann nur acht Farben erkennen und ist somit insgesamt ungenauer.[18] Um die Farben zu erkennen, beleuchtet der HiTechnic NXT Color Sensor V2 die Oberfläche mit einer weißen LED Lampe.[19] Den zurückreflektierten Farbkomponenten wird dann ein Farbwert zugeordnet.[20]

2.3 Aufbau und Funktion des Gestells

Das Gestell des Lösungs-Roboters besteht aus mehreren Teilen und Bereichen. Der wichtigste Bereich ist der große Zwischenraum in der Mitte des Gestells. In diesen Zwischenraum wird der Zauberwürfel eingelegt. Dort wird er von drei 3D gedruckten Zauberwürfel-Halterungen gehalten (vgl. M1). Zwei davon befinden sich jeweils an der rechten beziehungsweise linken Innenseite des Gestells. Die dritte und letzte Halterung ist unten innerhalb des Gestells angebracht. Diese unterscheidet sich stark von den seitlichen Halterungen. Die beiden seitlichen Halterungen greifen in den Zauberwürfel hinein, um ihn zu halten. Anders aber die untere Halterung diese greift zwar auch in den Zauberwürfel hinein, jedoch zusätzlich noch um den Zauberwürfel herum (vgl. M4). Dadurch wird mehr Stabilität gegeben. Der Zauberwürfel steht nicht schief und fällt auch nicht herunter. Die seitlichen Zauberwürfel-Halterungen greifen deshalb nicht, wie die untere Zauberwürfel-Halterung, um den Zauberwürfel herum, da sich diese ansonsten beim Drehen ineinander verkannten würden (vgl. M3). Das Ende der linken und der unteren Zauberwürfel-Halterung befinden sich jeweils in einem großen EV3 Servomotor (vgl. M2). Im linken und unteren Teil des Gestells befinden sich zwei Container. In jeden dieser Container passt genau einer dieser großen EV3 Servomotor hineinpasst. Die Motoren, welche sich dort befinden, sind gleichzeitig auch die Motoren, in welche das Ende der Zauberwürfel-Halterung steckt (vgl. M2). Durch diese Container gibt es einen Bereich, in dem sich die beiden Motoren frei bewegen können, das wäre durch eine normale Halterung nicht möglich. Der untere Motor kann sich dadurch vertikal bewegen. Dass selbe bei dem linken Motor. Dieser kann sich nun horizontal bewegen. Werden die Motoren nun horizontal, beziehungsweise



M2: Skizze des Zauberwürfel-Lösungs-Roboters.

[17] Vgl. ebd.
[18] Lego: „EV3-Farbsensor Produktdetails“.
[19] Vgl. HiTechnic Sensors: „NXT Color Sensor V2“.
[20] Vgl. ebd.

vertikal bewegt, dann werden die Zauberwürfel-Halterungen aus dem Zauberwürfel hinausgezogen oder hineingeschoben. Dadurch können nun alle seitlichen Seiten des Zauberwürfel im Gestell gedreht werden und nicht nur die linke Seite. Dies funktioniert wie folgt. Wird nun die seitliche Zauberwürfel-Halterung hinausgezogen und danach die untere Zauberwürfel-Halterung gedreht, so wird nicht die untere Seite des Zauberwürfels gedreht. Stattdessen wird der gesamte Zauberwürfel entlang der Y-Achse gedreht. Nun können zwar alle seitlichen Seiten des Zauberwürfels gedreht werden, allerdings nicht die obere Seite. Eine Möglichkeit war zu finden, welche es dem Roboter ermöglicht alle sechs Seiten des Zauberwürfel zu erreichen. Die Lösung war es, die dritte Halterung an der rechten Innenseite des Gestells zu platzieren. (Vgl. M2) Verbunden mit einem Motor, wie die anderen beiden Zauberwürfel-Halterungen, ist diese zwar nicht, allerdings ist das auch nicht notwendig. Wird die rechte Zauberwürfel-Halterung nun zusammen mit der linken Zauberwürfel-Halterung in den Würfel hineingeschoben und die untere Zauberwürfel-Halterung hinausgezogen, dann ist eine Rotation entlang der X-Achse möglich. Nun kann also auch die obere Seite des Zauberwürfels nach unten bewegt werden und dann mithilfe der unteren Zauberwürfel-Halterung gedreht werden. So sind nun alle Sech Seiten erreichbar für den Roboter. Doch das nur in der Theorie. Damit der Roboter diese Bewegungen auch selbstständig ausführen kann muss er auch die Zauberwürfel-Halterungen eigenständig bewegen können beziehungsweise die Motoren, in welchen diese stecken.

Dafür wurden zwei zusätzliche Motoren verbaut. Ein Medium EV3 Servomotor und ein großer NXT Servomotor. Der Medium EV3 Servomotor befindet sich im unteren Teil des Gestells (vgl. M2). Mit seiner Hilfe kann der große EV3 Servomotor, welcher sich unten befindet, vertikal bewegt werden. Die Rotationsbewegung des Medium EV3 Servomotors wird dazu genutzt. Wie dies genau funktioniert, wird im späteren Verlauf dieser Facharbeit noch näher erläutert. Der große NXT Servomotor hat in etwa die gleiche Aufgabe. Er wurde oben auf das Gestell draufgebaut (vgl. M2). Durch seine Drehbewegung kann er den linken großen EV3 Servomotor horizontal bewegen, somit wird die linke Zauberwürfel-Halterung hineingezogen oder hinausgezogen. Zusätzlich wird auch die rechte Zauberwürfel-Halterung hineingeschoben, beziehungsweise herausgezogen, diese ist nämlich auch mit dem großen NXT Servomotor verbunden. Somit passieren die Bewegungen immer auf beiden Seiten gleichzeitig. Der Grund dafür, dass beide seitlichen Zauberwürfel-Halterungen mithilfe eines großen NXT Servomotors bewegt werden, ist der gleiche Grund, weshalb die rechte Zauberwürfel-Halterung nicht in einem Motor steckt. Es liegt daran, dass der Lego Mindstorms EV3 nur vier Motoranschlüsse hat. Somit war die Bauweise sehr begrenzt, weshalb dieser Aufbau erfolgte.

Um die Farben des Zauberwürfel zuerkennen, verfügt der Roboter über einen fest angebrachten Farbsensor. Der Farbsensor ist ein „HiTechnic NXT Color Sensor V2“. Dieser ist an der Rückseite des Gestells angebracht und auf den hinteren Rand der linken Seite des Zauberwürfels ausgerichtet. Dadurch können die Farben an den Kanten erkannt werden (vgl. M40). Wenn aber die linke Seite des Zauberwürfel um 45 Grad gedreht wird, wird eine Ecke vor den Sensor geschoben und es kann die Farbe der Ecke erkannt werden (vgl. M41). Dadurch kann eine gesamte Seite des Zauberwürfels gescannt werden. Durch bestimmtes Rotieren des Zauberwürfels entlang der X-Achse und der Y-Ache ist es somit auch möglich die Farben des ganzen Zauberwürfels zu erfassen.

3. Interaktion zwischen Roboter und Zauberwürfel

3.1 Einstieg in leJOS

Um überhaupt die Motoren ansteuern zu könne, benötigt man einen Lego Mindstorms. Diesen kann man mit der mitgelieferten Software von Lego programmieren.[21] Es gibt aber auch eine andere Möglichkeit. Mithilfe von leJOS kann der Lego Mindstorms mit Java programmiert werden.[22] Jedoch muss leJOS zuerst auf dem Lego Mindstorms installiert werden. Dazu wird das Programm auf eine SD-Karte installiert, welche dann in den Lego Mindstorms gesteckt wird.[23] Nun muss noch die leJOS-Bibliothek in Eclipse installiert werden, damit Packages, wie beispielsweise „lejos.hardware.lcd“[24], importiert werden können.[25] Jetzt hat man Zugriff auf die Klassen und Methoden von leJOS, allerdings muss der Lego Mindstorms noch mit dem Computer verbunden werden, damit das leJOS Programm auf den Lego Mindstorms gespielt werden kann. Zum Verbinden mit einem Computer gibt es drei verschiedene Möglichkeiten. Es ist möglich eine Verbindung über Kabel, Bluetooth oder WLAN herzustellen.[26] Da für diese Facharbeit aber nur eine Verbindung über Bluetooth verwendet wurde, wird auch nur diese beschrieben. Dazu wird zu allererst der Lego Mindstorms mit dem Computer per Bluetooth gekoppelt. Dies funktioniert wie bei üblichen Bluetooth-fähigen Geräten auch. Um die leJOS Programme aber auch überspielen zu können, muss der Lego Mindstorms unter „Bluetooth-PAN-Geräte“ als „Zugriffspunkt“ ausgewählt werden.[27] Nun können die leJOS Programme kabellos auf den Lego Mindstorms geladen und gestartet werden.

3.1 Klassen und Methoden in leJOS

Im obigen Textabschnitt wurde bereits erläutert, dass durch das Importieren von leJOS Packages die leJOS Klassen und Methoden genutzt werden können. Eine dieser Klassen ist unteranderem die Klasse „LCD“ diese Klasse ist in dem Package „lejos.hardware.lcd“ untergebracht.[28] Eine beispielhafte Methode wäre passend dazu „drawString(String str, int x, int y)“. Diese zeigt einen String, das könnte beispielsweise ein Wort oder eine Zahl sein, an angegebener X und Y Stelle auf dem LCD des Lego Mindstorms an (vgl. M5).[29]

3.2 Interaktion zwischen Methoden und Zauberwürfel

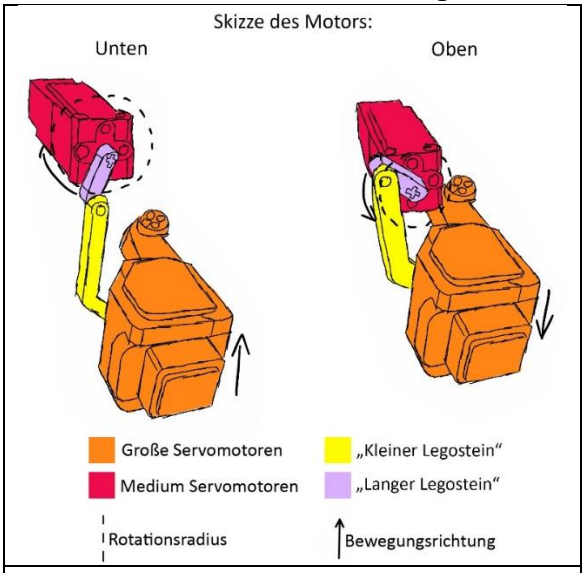
Damit der Roboter nun selbständig in der Lage ist die Seiten des Zauberwürfels zu drehen, wurden dafür spezielle Methoden erstellt. Diese greifen unteranderem auch aufeinander zurück. Der Grundstein aller anderen Methoden, welche zum Interagieren mit dem Zauberwürfel genutzt werden, sind diese Methoden: „SeitenBewegen()“, „UntenBewegen()“, „UntenDrehen(int i)“ und „LinksDrehen(int i)“.

---

[21] Vgl. Lego: „Mindstorms Education“.
[22] Vgl. leJOS: „Java“.
[23] Vgl. SourceForge: „Installiere leJOS“.
[24] leJOS: „Packages“.
[25] Vgl. SourceForge: „Installieren des leJOS Eclipse Plugins“.
[26] Vgl. Coding-unicorn: „Verbindung mit dem EV3“.
[27] Vgl. ebd.
[28] Vgl. Roberta: „10 Klassen -und Methodenübersicht“.
[29] Vgl. ebd.

Mit „SeitenBewegen()“ und „UntenBewegen()“ werden die untere beziehungsweise die seitlichen Zauberwürfel-Halterungen in den Zauberwürfel hineingeschoben oder herausgezogen. Dies passiert wie folgt. Bei der Methode „UntenBewegen()“ wird zuerst geprüft, welchen Zustand der „boolean“ namens „Oben“ hat. Dieser kann den Zustand „wahr“ und „falsch“ annehmen.[30] Ist die untere Zauberwürfel-Halterung nicht im Würfel, so hat „Oben“ den Zustand falsch. Andernfalls nimmt „Oben“ den Zustand „wahr“ an (vgl. M7).

Ist nun, zum Zeitpunkt des Methodenaufrufs, der Zustand von „Oben“ „wahr“, so wird „MotorC“, welcher der medium EV3 Servomotor an der Hinterseite des Gestells ist, um -135 Grad gedreht (vgl. M8). Durch diese Drehbewegung, welche auf Grund des Minus gegen den Uhrzeigersinn verläuft, wird der darunterliegende große EV3 Servomotor heruntergelassen (vgl. M6). Somit wird auch die Zauberwürfel-Halterung aus dem Zauberwürfel hinausgezogen. Ist der Zustand von „Oben“ aber „wahr“, so wird „MotorC“ um 135 Grad gedreht. Somit wird der darunterliegende Motor zusammen mit der unteren Zauberwürfel-Halterung hochgezogen. Dieser Mechanismus funktioniert wie folgt. Ein an den Motor fest angebrachten kleiner Legostein, vergrößert den Radius der Drehbewegung (vgl. M6). An dem kleinen Legostein ist am Ende ein langer Legostein angebracht, dieser kann sich an der Schnittstelle mit dem kleinen Legostein frei bewegen. So kann die Drehbewegung zu einer Zugbewegung umwandelt werden, da der große Legostein durch die Drehbewegung hoch oder runter bewegt wird (vgl. M6). Das andere Ende des langen Legosteins ist an dem darunterliegenden großen EV3 Servomotor welcher den Namen „MotorD“ hat angebracht (vgl. M8). Somit kann dieser zusammen mit dem langen Legostein hochgezogen werden (vgl. M6).



M6: Skizze der vertikalen Bewegung von „MotorD“.

Das gleiche Prinzip wird auch bei „SeitenBewegen()“ angewendet. Hier ist der boolean namens „Zusammen“ je nachdem mit dem Zustand „wahr“ oder „falsch“ versehen. Ist dieser „wahr“, so wird „MotorA“ um -45 Grad gedreht. Dadurch wird die rechte Zauberwürfel-Halterung herausgezogen und „MotorB“ wird nach links bewegt. Durch das nach links Bewegen von „MotorB“ wird auch die linke Zauberwürfel-Halterung herausgezogen. Ist der Zustand von „Zusammen“ aber „falsch“, so wird „MotorA“ um 45 Grad gedreht. Somit wird die rechte Zauberwürfel-Halterung herausgezogen und „MotorB“ wird nach links verschoben, beziehungweise wird auch die linke Zauberwürfel-Halterung herausgezogen. Mechanisch funktioniert dies wie bei der unteren Zauberwürfel-Halterung. Nur in diesem Fall sind an beiden Seiten des Legosteins, welcher den Rotationsradius vergrößert, ein weiterer Legostein angebracht.

[30] Vgl. Panjutorials: „Boolsche Variable“.

Diese werden dann weiter auseinandergeschoben beziehungsweise zusammengezogen (vgl. M9).

Nun gibt es noch die Methoden „UntenDrehen(int i)“ und „LinksDrehen(int i)“ (vgl. M10). Diese sind zum tatsächlichen Interagieren mit dem Zauberwürfel notwendig. Mithilfe der anderen zwei Methoden können zwar schon die Zauberwürfel-Halterungen in den Würfel hineingeschoben und herausgezogen werden, allerdings kann er noch nicht gedreht werden. Dafür sind ebendiese Methoden notwendig. Sie sagen dem jeweiligen Motor, welcher mit der Zauberwürfel-Halterung verbunden ist, wie sich dieser zu drehen hat. Damit dieser auch weiß, um wie viel Grad er sich drehen muss, muss immer eine Zahl in Form eines Integer angegeben werden.[31] Die Implementation von „UntenDrehen(int i)“ ist an sich die gleiche Methode, nur wird „MotorD“ angesteuert und nicht „MotorB“.

Es gibt allerdings noch eine weitere essenzielle Methode. Die Methode zum Erkennen der Farben namens „RGBWerte()“. Diese liest die RGB-Werte aus, welche der HiTechnic NXT Color Sensor V2 scannt (vgl. M12). Die gescannten Werte werden dann mithilfe der Methode „runden(double l)“ auf eine Nachkommastelle gerundet und in einer weiteren Methode namens „FarbWert()“ einer bestimmten Farbe zugewiesen (vgl. M11). Diese Methode prüft, ob die gerundeten RGB-Wert einem Wert übereinstimmt, welcher in der Methode „Farbwert()“ angegeben ist. Ist dies der Fall, so wird eine von sechs möglichen Zahlen zurückgegeben, wobei jede Zahl einer Farbe des Zauberwürfels entspricht (vgl. M13). Somit können die Farben des Zauberwürfels bestimmt werden.

3.3 Methoden für die Bewegungen des Zauberwürfels

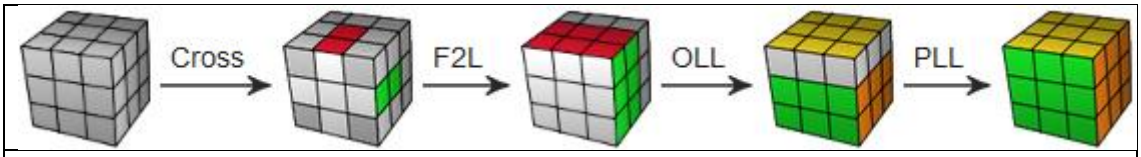
Ein Zauberwürfel besitzt sechs Seiten. Somit besitzt er auch sechs drehbare Seiten. Da es bei sechs Seiten aber schnell zu einer Verwirrung kommen kann, welche Seite nun gemeint ist, die gedreht werden soll, hat jede Drehung einer bestimmten Seite seine eigene Bezeichnung. „D“ Steht für die untere Seite, „U“ für die obere Seite, „L“ und „R“ für jeweils rechts und links und zuletzt „F“ und „B“, welche für die Vorder- und Hinterseite stehen (vgl. M14).[32] Diese Bewegungen werden immer in Richtung des Uhrzeigersinns ausgeführt. Befindet sich aber ein „ ‘ “ dahinter, so wird die Drehung gegen den Uhrzeigersinn ausgeführt.[33] Es gibt auch Fälle, bei denen eine „2“ dahinter steht. Dann wird die Seite zwei Mal gedreht.[34] In der Implementierung des Zauberwürfel-Lösungs-Programms wurden diese Bezeichnungen auch verwendet. Zur Vereinfachung wurden Methoden implementiert, welche beispielsweise „Rs()“ heißen und in diesem Beispiel die rechte Seite des Zauberwürfels gegen den Uhrzeigersinn drehen (vgl. M15). Zusätzlich zu diesen sechs Methoden gibt es aber noch zwei weitere Methoden. Diese drehen den gesamten Zauberwürfel. Die Methode „X()“ dreht den Zauberwürfel entlang der x-Achse und die Methode „Y()“ tut dies entlang der y-Achse.

---

[31] Vgl. Fu-berlin: „Die Klasse »java.lang.Integer« in Java“.
[32] Vgl. Wikibooks: „Zauberwürfel/ 3x3x3/ Notation“.
[33] Vgl. ebd.
[34] Vgl. ebd.

4. Darstellung des Lösungsalgorithmus

Der Lösungsalgorithmus basiert auf die „CFOP“ Methode, welche zu Beginn schon angesprochen wurde. Wie beschrieben besteht diese aus vier Schritten (vgl. M16).[35] Um genauer zu sein werden die „Ebenen“ des Zauberwürfels nacheinander gelöst (vgl. M19). Der Lösungsalgorithmus des Roboters besteht allerdings aus fünf, da er sich leicht unterscheidet. Wie bei der normalen „CFOP“ Methode wird zu Beginn das weiße Kreuz gelöst. Dabei ist zu beachten, dass nicht nur ein weißes Kreuz zu lösen ist. Dies kann nämlich zu Verwirrung führen. Denn jedes weiße Kantenteil hat nämlich auch noch eine zweite Farbe (vgl. M18). Diese Farbe muss mit der mittleren Farbe der zweiten Ebene übereinstimmen (vgl. M16).



M16: Visualisierung der „CFOP“ Methode.

Der Roboter tut löst das weiße Kreuz wie folgt. Zu Beginn wird der Zauberwürfel einmal komplett gescannt. Die gescannten Farben werden während dessen an entsprechender Stelle in ein zweidimensionales Array namens „Wurfel“ eingetragen. Dieses stellt den Zauberwürfel in 2D dar (vgl. M17). Man kann sich ein zweidimensionales Array wie eine Tabelle vorstellen, welche aus Zeilen und Spalten besteht (vgl. M17).[36] Nun muss die Passende Kante gefunden werden (vgl. M16) (vgl. M18). Dies geschieht Mithilfe der Methode „FindePassendeKante(int farbe1, int farbe2)“ (vgl. M23). Beim ersten Durchlauf wird die rot-weiße Kante gesucht. Um diese zu finden, werden zuerst alle weißen Felder des Zauberwürfels gesucht und abgespeichert (vgl. M18). Dafür ist die Methode „FindeFarben(int farbe)“ zuständig (vgl. M22). Diese vergleicht jeden Wert des Arrays „Wurfel“ mit dem Wert der gesuchten Farbe. In diesem Fall Weiß. Gibt es einen Treffer, so wird sich mithilfe des Arrays „GefundenFarben“ gemerkt, wo dieser ist (vgl. M22). Insgesamt wird die gesuchte Farbe neun Mal gefunden, da alle Farben neun Mal auf dem Zauberwürfel vertreten sind. Nun muss allerding die rot-weiße Kante unter den neun weißen Feldern gefunden werden. Allerdings ist es dafür vorerst notwendig zu wissen, welche Felder zusammen eine Kante bilden. Deshalb wird die Methode „KantenPaare(int z, int s)“ aufgerufen (vgl. M25). Dabei werden die Ecken ignoriert, da diese in diesem Schritt nicht relevant sind. Diese Methode gibt dann das jeweils andere zugehörige Feld der gesuchten Kante zurück. Nun überprüft die Methode „FindePassendeKante(int farbe1, int farbe2)“, ob das zweite Feld der Kante, welches im Schritt zuvor gefunden wurde, den Farbwert der gesuchten Farbe hat. Dieser sollte in diesem Fall Rot sein. Ist dies der Fall wird das weiße Feld dieser Kante in dem Array „RichtigeKante“ gespeichert. Andernfalls wird das nächste von den neun weißen Feldern überprüft. Der letzte Schritt ist das Platzieren der Kanten an die richtige Stelle. Die Methode „KreuzTeilPlatzieren(int z, int s)“ führt nun die benötigten Drehungen am Zauberwürfel aus, welche die Kante an die richtigen Positionen bringen. All diese

[35] Vgl. Ruwix: „Steps of the advanced method“.
[36] Vgl. Journaldev: „Two Dimensional Array in Java“.

Methoden werden insgesamt vier Mal ausgeführt, damit das weiße Kreuz vollständig zusammengesetzt ist. Bei jeder neuen Ausführung wird aber eine andere Seite nach links geschoben und somit auch die weiße Kante einer anderen Farbe gesucht.

Der nächste Schritt weicht ein wenig von der „CFOP“ Methode ab. Anstatt wie der Name des Schritts schon sagt, beiden ersten Ebenen gleichzeitig zu lösen, tut der Roboter dies in zwei Schritten.[37] Zuerst werden neben den Kanten, welche das weiße Kreuz bilden noch die passenden Ecken eingefügt. Dies funktioniert wie bei dem weißen Kreuz. Der einzige Unterschied liegt darin, dass eine Ecke drei unterschiedliche Farben hat (vgl. M18). Deshalb wird nicht die Methode „FindePassendeKante(int farbe1, int farbe2)“ verwendet, da diese nur auf 2 Farben überprüft. Die Methode „FindePassendeEcke(int farbe1, int farbe2, int farbe3)“ hingegen überprüft, ob alle drei Felder den gesuchten Farbwert haben (vgl. M27). Wird eine Richtige Kante gefunden, so wird auch hier das dazugehörige weiße Feld in einem Array namens „RichtigeEcke“ gespeichert (vgl. M27). Nun wird noch die Methode zum Positionieren der Ecken ausgeführt. Diese funktioniert wie die Methode „KreuzTeilPlatzieren(int z, int s)“. Der einzige Unterschied liegt darin, dass der Zauberwürfel anders gedreht wird und dass die Ecken woanders sein können als die Kanten im vorherigen Schritt. Darauffolgend werden die fehlenden Kanten der zweiten Ebene gelöst auch dieser Schritt ist sehr ähnlich mit dem des weißen Kreuzes. Ein Unterschied ist, dass nicht nach weißen Feldern gesucht wird, denn diese sind schon alle richtig positioniert worden, stattdessen wird nach den Kanten gesucht, welche an die jeweilige Stelle der zweiten Ebene gehören (vgl. M16) (vgl. M30). Ist die entsprechende Kante gefunden, soll diese an die richtige Position gebracht werden. Dafür ist ein Algorithmus notwendig. Die zu positionierenden Kanten müssen allerdings entsprechend positioniert werden, damit der Algorithmus ausgeführt werden kann. (vgl. M30). Dafür ist die Methode „InPositionBringen(int[] Kante)“ zuständig (vgl. M31). Durch den Algorithmus ist es möglich, Kanten aus der dritten Ebene in die zweite Ebene zu bewegen, ohne dabei die erste und zweite Ebene zu verändern (vgl. M30). Befindet sich die gesuchte Kante allerdings nicht in der dritten Ebene, kann sie auch nicht in die richtige Position gebracht werden (vgl. M30) (vgl. M31). In diesem Fall wird dieser Algorithmus auch verwendet. Er ist notwendig, um die Kante vorerst aus der zweiten Ebene in die dritte Ebene zu bewegen. Dann kann die Kante anschließend in die richtige Position gebracht werden und zur richtigen Stelle in die zweite Ebene bewegt werden. Das Einfügen übernimmt die Methode „SecondLayerPlatzieren()“ (vgl. M32).

Zuletzt wird die letzte Ebene durch „OLL“ und „PLL“ gelöst (vgl. M16). Dafür wird als erstes der „OLL“- Schritt von der Methode namens „OLL()“ durchgeführt (vgl. M34). In diesem Schritt werden alle gelben Felder nach oben gerichtet (vgl. M16).[38] Der Roboter tut dies wie folgt: Erst wird überprüft, ob zufälligerweise schon ein gelbes Kreuz auf der dritten Ebene vorhanden ist (vgl. M33). Ist dies aber nicht der Fall, so wird dieses erstellt. Dafür wird die Methode „GelbesKreuz“ aufgerufen (vgl. M35). Diese überprüft, wie die gelben Felder angeordnet sind. Je nach Muster wird dann ein bestimmter Algorithmus

[37] Vgl. Ruwix: „Steps of the advanced method“.
[38] Vgl. ebd.

ausgeführt.[39] Dabei entsteht das gelbe Kreuz, ohne etwas an der ersten oder zweiten Ebene zu verändern. Im darauffolgenden Schritt wird die Methode „OLLOrientation“ aufgerufen (vgl. M36). Diese orientiert die restlichen gelben Felder nach oben. Hierbei findet dasselbe Vorgehen statt: Es wird erneut geschaut, wie die gelben Felder angeordnet sind und ein entsprechender Algorithmus wird ausgeführt.[40] Als letzter Schritt wird der „PLL“-Schritt von der Methode „PLL()“ ausgeführt (vgl. M37). Dieser Schritt richtet nun noch den Rand der letzte Ebene.[41] Dieser Schritt ist notwendig, da sich die Ecken und Kanten der letzten Ebene noch an falschen Stellen befinden könnten (vgl. M16). Andernfalls wäre der Zauberwurzel schon gelöst. Ist es aber nicht der Fall, wird geschaut wie die Ecken der letzten Ebene angeordnet sind (vgl. M38). Müssen sie Diagonal getauscht werden, um richtig positioniert zu werden wird der „Y-Permutation“ Algorithmus ausgeführt.[42] Andernfalls der „T-Permutation“ Algorithmus.[43] Nun sind nur noch die Kanten der letzten Ebene vertauscht. Um diese schlussendlich richtig zu positionieren gibt es vier weitere Algorithmen (vgl. M39). Je nach Anordnung der Kanten wird dann der entsprechende ausgeführt und der Zauberwürfel ist gelöst.

5. Fazit

Zusammenfassend kann man sagen, dass mich die Facharbeit sehr viel Zeit gekostet hat. Vor allem der Bau des Lösungs-Roboters, welcher über drei neue Anläufe gebraucht hat und die Implementierung des Lösungsalgorithmus haben mich viele Stunden und auch viele Nerven gekostet. Dennoch kann ich guten Gewissens sagen, dass mir die Facharbeit sehr viel Freude bereitet hat. Dies liegt vermutlich auch daran, dass ich mich mit dem Thema „Zauberwürfel“ bereits auskannte. Und somit schon ein gutes Vorwissen hatte. Doch trotzdem habe ich noch sehr viel lernen können. Das vermutlich beste Beispiel dafür ist der Umgang mit dem Programm „Inventors“ von Autodesk. Dieses Programm ist dafür da, um 3D Objekte zu erstellen. Etwas womit ich mich bis dahin kaum auskannte, doch durch die Notwendigkeit von 3D Teilen für den Roboter, habe ich mir beigebracht damit umzugehen. Genau so habe ich durch diese Facharbeit gelernt mir Zeit gut einzuteilen, denn ansonsten wäre diese Facharbeit nicht rechtzeitig fertig geworden.

Schlussendlich war es definitiv viel Arbeit, welche sich bei dem letztendlichen Ergebnis für mich aber bezahlt macht.

---

[39] Vgl. JPerm: „2-Look OLL“.
[40] Vgl. ebd.
[41] Vgl. Ruwix: „Steps of the advanced method“.
[42] Vgl. JPerm: „2-Look PLL“.
[43] Vgl. ebd.

6. Literaturverzeichnis

Frank Klautke: „Lösungsbuch für Rubik´s Zauberwürfel“: http://www.klautke.de/files/download/Loesungsbuch_Zauberwuerfel.pdf (Google Suche: „was sind Algorithmen Zauberwürfel“, 28.03.2020)

Fu-berlin: „Die Klasse »java.lang.Integer« in Java“: http://userpage.fu-berlin.de/~ram/pub/pub_jf47ht81Ht/integer_java (31.03.2020)

Journaldev: „Two Dimensional Array in Java“: https://www.journaldev.com/747/two-dimensional-array-java (01.04.2020)

Lego: „EV3-Farbsensor Produktdetails“: https://www.lego.com/de-de/product/ev3-color-sensor-45506 (29.03.2020)

Lego: „EV3 Servomotor medium Produktdetails“: https://www.lego.com/de-de/product/ev3-medium-servo-motor-45503 (Google Suche: „Lego EV3 Motoren“, 29.03.2020)

Lego: „Großer EV3 Servomotor Produktdetails“: https://www.lego.com/de-de/product/ev3-large-servo-motor-45502 (Google Suche: „Lego EV3 Motoren“, 29.03.2020)

Lego: „Mindstorms Education“: https://www.lego.com/de-de/themes/mindstorms/learntoprogram (29.03.2020)

Philohome: „LEGO® 9V Technic Motors compared characteristics“: https://www.philohome.com/motors/motorcomp.htm (29.03.2020)

SourceForge: „Installieren des leJOS Eclipse Plugins“: https://sourceforge.net/p/lejos/wiki/Installing%20the%20Eclipse%20plugin/ (29.03.2020)

SourceForge: „Installiere leJOS“: https://sourceforge.net/p/lejos/wiki/Installing%20leJOS/ (29.03.2020)

leJOS: „Java“: http://www.lejos.org/rcx-faq.php (29.03.2020)

leJOS: „Packages“: http://www.lejos.org/ev3/docs/ (29.03.2020)

Coding-unicorn: „Verbindung mit dem EV3“: https://coding-unicorn.de/lego-roboter/verbindung-mit-dem-ev3/ (3103.2020)

Geolino: „Fünfmal staunen über den Zauberwürfel“: https://www.geo.de/geolino/kreativ/1134-rtkl-gib-mir-fuenf-fuenfmal-staunen-ueber-den-zauberwuerfel (20.03.2020)

HiTechnic Sensors: „NXT Color Sensor V2“: https://modernroboticsinc.com/product/hitechnic-nxt-color-sensor-v2/ (29.03.2020)

JPerm: „2-Look OLL“: https://jperm.net/algs/2look/oll (01.04.2020)

JPerm: „2-Look PLL“: https://jperm.net/algs/2look/pll (01.04.2020)

Panjutorials: „Boolsche Variable“: https://panjutorials.de/tutorials/java-tutorial-programmieren-lernen-fuer-anfaenger/lektionen/boolsche-variable/

Speedcubing Schweiz: „Speedcubing“: https://www.speedcubing.ch/ (Google suche: „Speedcubing“, 28.03.2020)

Roberta: „10 Klassen -und Methodenübersicht“: https://www.roberta-home.de/fileadmin/user_upload/WebBooks/JavaBand/RobertaBuchch10.html (29.02.2020)

Ruwix: „Advanced CFOP / Fridrich Method“: https://ruwix.com/the-rubiks-cube/different-rubiks-cube-solving-methods/ (Google Suche: „most popular speedcube methods“, 28.03.2020)

Ruwix: „Steps of the advanced method“: https://ruwix.com/the-rubiks-cube/advanced-cfop-fridrich/ (Google Suche: „CFOP“, 27.03.2020)

Wikibooks: „Zauberwürfel/ 3x3x3/ Notation“: https://de.wikibooks.org/wiki/Zauberw%C3%BCrfel/_3x3x3/_Notation (31.03.2020)

World Cube Association: „Rangliste“: https://www.worldcubeassociation.org/results/rankings/333/single (Google Suche: „cubing world records“, 28.03.2020)

Bildequellen

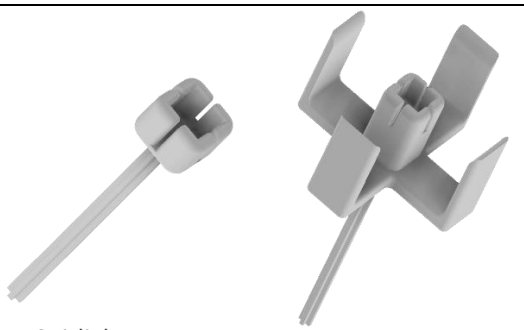
Bild des Deckblatts: E-Technology News: „The NYPD is using machine learning to spot crime patterns – StateScoop“: https://e-technologynews.com/the-nypd-is-using-machine-learning-to-spot-crime-patterns-statescoop/ (01.04.2020)

M14: Wikibooks: „Zauberwürfel/ 3x3x3/ Notation“: https://de.wikibooks.org/wiki/Zauberw%C3%BCrfel/_3x3x3/_Notation (31.03.2020)

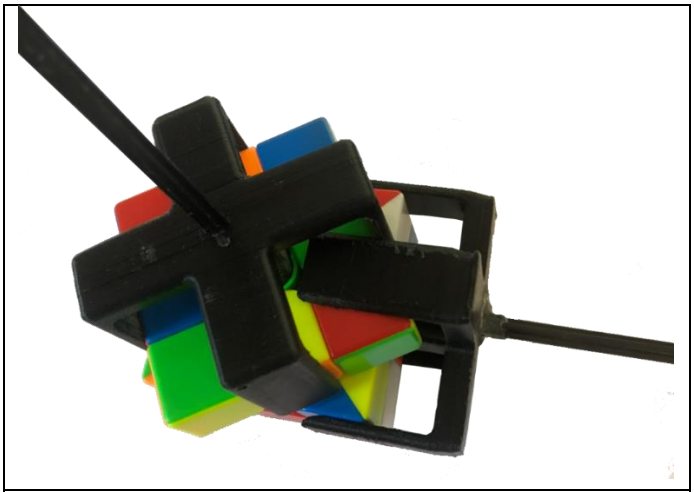
M16: Ruwix: „Steps of the advanced method“: https://ruwix.com/the-rubiks-cube/advanced-cfop-fridrich/ (Google Suche: „CFOP“, 27.03.2020)

M18/M19/M30: (Pfeil) Freepik: „Pfeile Icons“: https://de.freepik.com/freie-ikonen/herunterladen_970801.htm#page=1&query=pfeile&position=36 (01.04.2020)

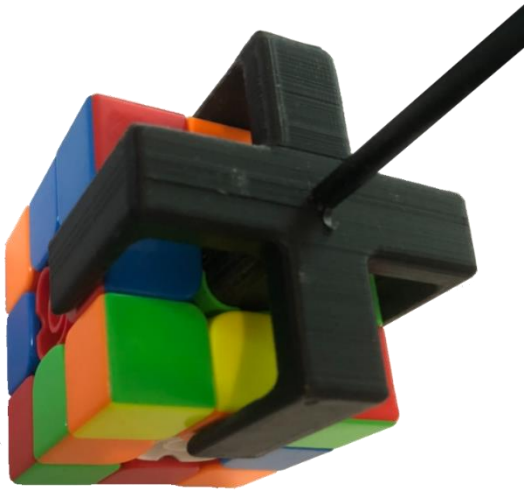
7. Anhang



M1: Darstellung der Zauberwürfel-Halterungen.



M3: Untere Zauberwürfel-Halterung verkantet.



M4: Untere Zauberwürfel-Halterung greift um den Zauberwürfel herum.

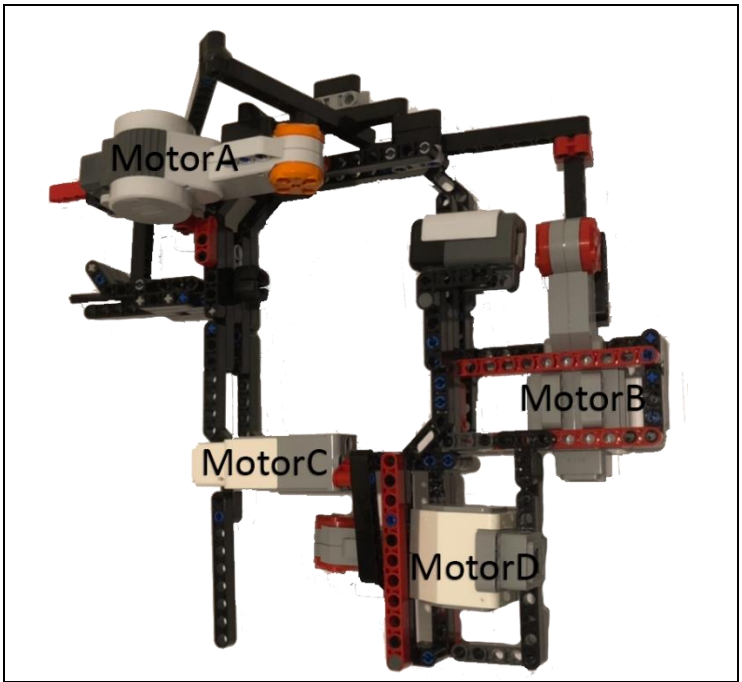
```
01 import lejos.hardware.Button;       //Button-Package wird importiert.
02 import lejos.hardware.lcd.LCD;      //LCD-Package wird importiert.
03
04 public class HelloWorldTest{
05 //Eine neue Klasse namens „HelloWorldTest“ wird erzeugt.
06
07     public static void main(String[] args){
08     //Diese Methode wird immer beim Programmstart ausgeführt.
09           LCD.drawString("Hello World", 0, 0);
10           //Auf dem LCD des Lego Mindstorms erscheint oben links „Hello
11           World“.
12           Button.waitForAnyPress();
13           //Das Programm läuft so lange, bis ein Knopf gedrückt wird.
14     }
15 }
```

Legende:	Die Klassen sind blau Die Methoden sind hellgrün und *kursiv*

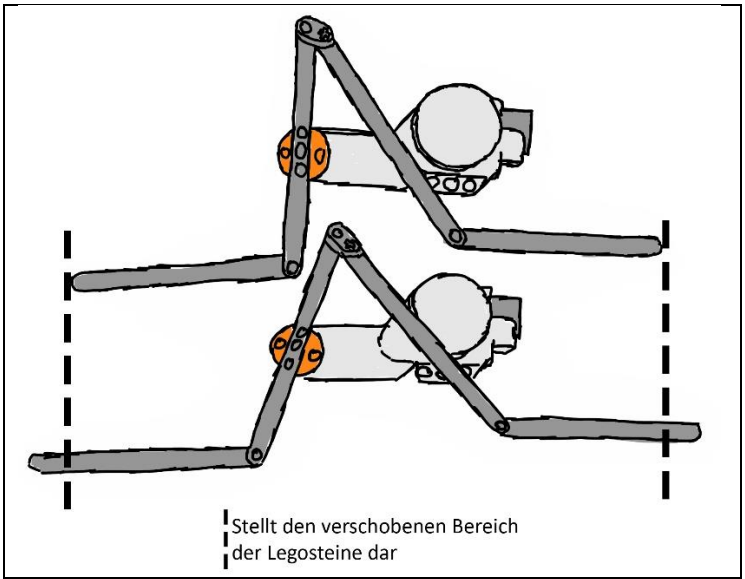
M5: Beispielhafte Implementation für „Hello World“.

```
01     static boolean Oben;
02     // Boolean namens „Oben“ wird zum Start des leJOS Programms erzeugt.
03 public static boolean UntenBewegen(){
04 //Eine neue Methode namens „UntenBewegen“ wird erstellt. Diese gibt
05 immer einen boolean nach ihrer Ausführung zurück.
06     if(Oben == false){
07           //Wenn der Zustand von „Oben“ „falsch“ ist, werden die
08           Aktionen innerhalb der geschweiften Klammer ausgeführt.
09           MotorC.rotate(135);
10           //Der „MotorC“ wird um 135 Grad im Uhrzeigersinn gedreht
11           return Oben=true;
12           //Der Wert von „Oben“ wird auf „wahr“ gesetzt.
13     }
14     else{
15           //Ist der Zustand von „Oben“ nicht „falsch“, so werden
16           die Aktionen innerhalb der geschweiften Klammer ausgeführt.
17           MotorC.rotate(-135);
18           //Der MotorC wird um 135 Grad gegen den Uhrzeigersinn
19           gedreht.
20           return Oben=false;
21           //Der Wert von „Oben“ wird auf „falsch“ gesetzt.
22     }
23 }
```

M7: Implementation der Methode „UntenBewegen()“.



M8: Roboter Servomotoren Benennung (Von hinten).



M9: Skizze des Auseinanderschiebens und Zusammenziehens der Legostangen.

```
01 public static void LinksDrehen(int i){
02 //Eine neue Methode namens „LinksDrehen“ wird erstellt. Diese
03 benötigt bei einem Aufruf immer einen Wert in Form eines Integer.
04     MotorB.rotate(i);
05     //Der „MotorB“ wird um die Gradzahl gedreht, welche zum Zeitpunkt
06     des Methodenaufrufs als Integer angegeben wurde.
07 }
```

M10: Implementation der Methode „LinksDrehen()“.

```
public static double runden(double l){
//Angegebener Zahlenwert wird der Variable „l“ zugewiesen.
            l=l*10;
            //Der Werte der Variable „l“ wird mit zehn multipliziert.
            l=Math.round(l);
            // Der Wert der Variable „l“ wird auf null Nachkommastellen
            gerundet.
            return l/10;
            //Der Wert der Variable „l“ wird durch zehn subtrahiert und
            als „Ergebnis“ zurückgegeben.
      }
```

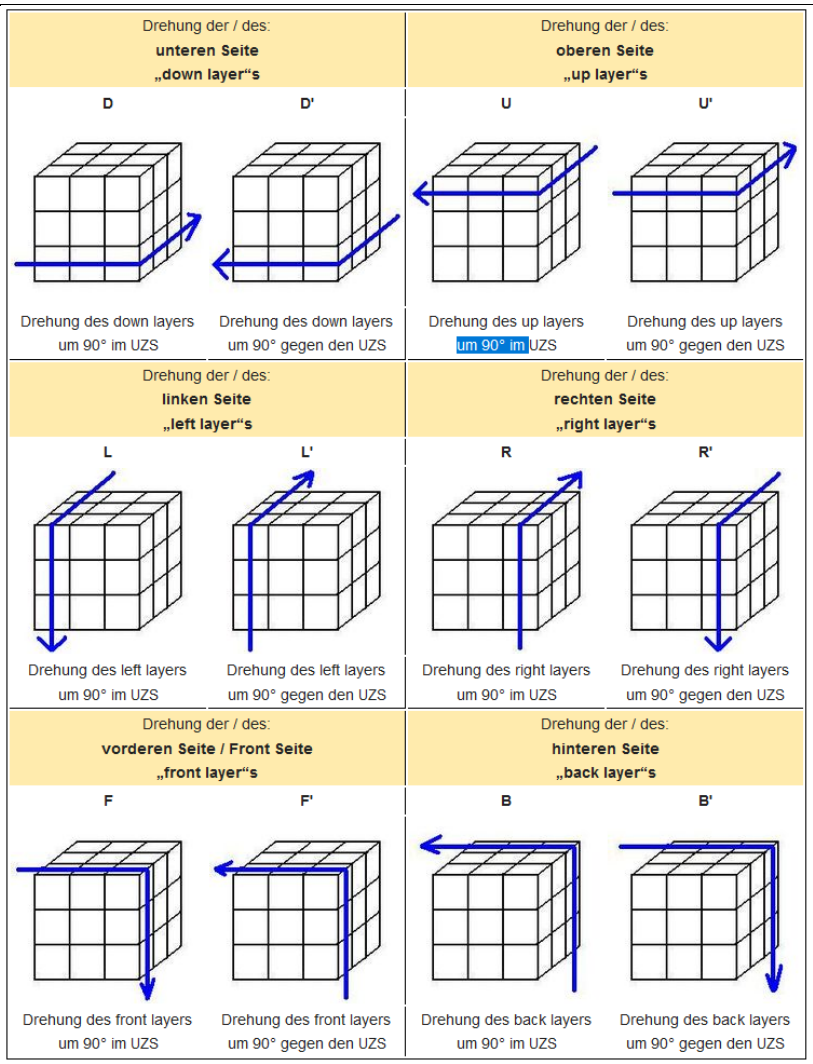
M11: Implementation der Methode „runden()“.

```
            static double R = 0; //Variable „R“ wird erstellt und ihr 0
            wird zugewiesen.
            static double G = 0; //Variable „G“ wird erstellt und ihr 0
            wird zugewiesen.
            static double B = 0; //Variable „B“ wird erstellt und ihr 0
            wird zugewiesen.
      public static void RGBWerte() {
            SensorMode Farbe = Farbsensor.getRGBMode();
            //Farbsensor wird auf den RGB-Modus eingestellt.
            float[] FarbArray = new float[Farbe.sampleSize()];
            //Ein Array wird erstellt. Die Größe des Arrays variiert ist
            abhängig davon, wie viele Werte „color.sampleSize()“ besitzt.
            In diesem Fall sind es drei Werte (RGB).

            Farbe.fetchSample(FarbArray, 0);
            //Die gemessenen RGB-Werte des Farbsensors werden nun in dem
            Array „FarbArray“ gespeichert.
            R = runden(FarbArray[0]);
            //Die Variable „R“ wird der, auf eine Nachkommastelle
            gerundete, Zahl des Rot-Werts zugeschrieben.
            G = runden(FarbArray[1]);
            //Die Variable „G“ wird der, auf eine Nachkommastelle
            gerundete, Zahl des Gelb-Werts zugeschrieben.
            B = runden(FarbArray[2]);
            //Die Variable „B“ wird der, auf eine Nachkommastelle
            gerundete, Zahl des Blau-Werts zugeschrieben.
      }
```

M12: Implementation der Methode „RGBWerte()“.

```
static int Farbe =0;
public static void FarbWert() {
	RGBWerte();
	//Die Methode „RGBWerte()“ wird ausgeführt.
	if(R ==1 && G==0.5 && B==0.0 || R==1 && G==0.4 && B==0.0 || […]) {
		Farbe = 4;
		//Entsprechen die RGB-Werte einer dieser Werte, so wird die
	Variable „Farbe“ auf 4 gesetzt (Orange). Andernfalls wird
	das gleiche Vorgehen bei den anderen Möglichkeiten
		durchgeführt.

	}

	else if(R==0.6 && G==0.7 && B==0.2 || […]) { //Gelb
		Farbe = 6;
		//Die Variable „Farbe“ wird auf 6 gesetzt (Gelb).

	}
	else if(R==0.3 && G==0.6 && B==0.3 || […]) { //Grün
		Farbe = 5;
		//Die Variable „Farbe“ wird auf 5 gesetzt (Grün).

	}

	else if(R==0.7 && G==0.2 && B==0.1 || […]) { // Rot
		Farbe = 2;
		//Die Variable „Farbe“ wird auf 2 gesetzt (Rot).

	}

	else if(R==0.2 && G==0.3 && B==0.4 || […]){ // Blau
		Farbe = 3;
		//Die Variable „Farbe“ wird auf 3 gesetzt (Blau).

	}

	else if(R==0.9 && G==0.9 && B==0.7 || […]) { // Weiss
		Farbe = 1;
		//Die Variable „Farbe“ wird auf 1 gesetzt (Weiß).

	}

	else {
		FarbWert();
		//Trifft keiner der Fälle zu, so wird die Methode
		„FarbWerte()“ erneut ausgeführt.
	}
}
```

M13: Implementation der methode „FarbWert()“.



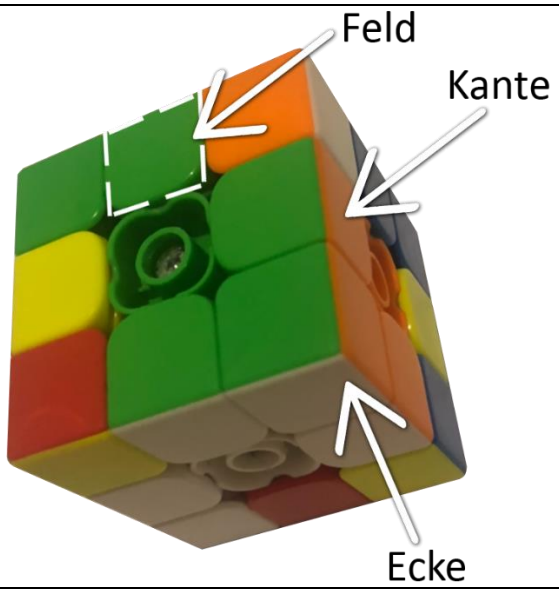
M14: Grundbewegungen des Zauberwürfels.

```
public static void Rs() {
        SeitenBewegen();
        UntenDrehen(180);
        SeitenBewegen();
        LinksDrehen(90);
        SeitenBewegen();
        UntenDrehen(-180);
        SeitenBewegen();
    }
```

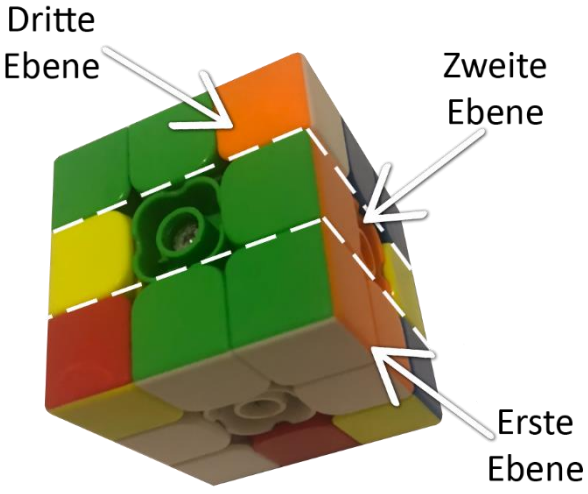
M15: Implementation der Methode „Rs()“.

```
static int[][] Wurfel =
      {
            {1,1,1},
            {1,1,1},
            {1,1,1},
            {2,2,2,3,3,3,4,4,4,5,5,5},
            {2,2,2,3,3,3,4,4,4,5,5,5},
            {2,2,2,3,3,3,4,4,4,5,5,5},
            {6,6,6},
            {6,6,6},
            {6,6,6}
      };
```

M17: Implementierung des zweidimensionalen „Zauberwürfel-Arrays“ .



M18: Beschriftung der Kanten und Ecken.



M19: Ebenen des Zauberwüfels.

```
public static void NachObenOrientieren(int f) {
		if(Wurfel[1][1]!=f) {

			if(Wurfel[4][1]==f) {
				X();
			}
			else if(Wurfel[4][4]==f) {
				Y();
				X();
			}
			else if(Wurfel[4][7]==f) {
				Xs();
			}
			else if(Wurfel[4][10]==f) {
				Ys();
				X();
			}
			else if(Wurfel[7][1]==f) {
				X();
				X();
			}

		}
	}
```

M20: Implementation der Methode „NachObenOrientieren(int f)“.

```
public static void SeiteNachLinksOrientieren(int F) {
			if(Wurfel[4][10]!=F) {

				if(Wurfel[4][4]==F) {
					Y();
					Y();
				}
				else if(Wurfel[4][7]==F) {
					Ys();
				}
				else if(Wurfel[4][1]==F) {
					Y();
				}

			}
		}
```

M21: Implementation der Methode „SeiteNachLinksOrientieren(int F)“.

```
	static int[][] GefundeneFarben = new int[9][2];
public static void FindeFarbe(int farbe) {
	int a = 0;
	int b = 0;
	for(int z = 0; z < Wurfel.length; z++) {
		for(int s = 0; s < Wurfel[z].length; s++) {
			if(Wurfel[z][s] == farbe) {
				if(z==1 && s==1) {

				}
				else {
					GefundeneFarben[a][b]=z;
					GefundeneFarben[a][b+1]=s;
					a++;
				}
				}
			}
		}
		 }
```

M22: Implementation der Methode „FindeFarben(int farbe)“.

```
	static int[] RichtigeKante = new int[2];
public static void FindePassendeKante(int farbe1, int farbe2) {
	FindeFarbe(farbe1);
	for(int i=0; i<8; i++) {
		KantenPaare(GefundeneFarben[i][0],GefundeneFarben[i][1]);
		if(Paar[0]==1 && Paar[1]==1) {
		}
		else {
			if(Wurfel[Paar[0]][Paar[1]]==farbe2) {

				RichtigeKante[0]=GefundeneFarben[i][0];
				RichtigeKante[1]=GefundeneFarben[i][1];
			}
		}
	}
}
```

M23: Implementation der Methode „FindePassendeKante(int farbe1, int farbe2)“.

```
      static int[] Paar = new int[2];
public static void KantenPaare(int z, int s) {
            if(z == 0 && s == 1) {
                  Paar[0]= 3;
                  Paar[1]= 7;
            }
            else if(z==1 && s==0) {
                  Paar[0]= 3;
                  Paar[1]= 10;
            }
            else if(z==4 && s==2) {
                  Paar[0]= 4;
                  Paar[1]= 3;
            }
             […]
            else {
                  Paar[0]=1;
                  Paar[1]=1;
            }
             }
```

M25: Implementation der Methode „KantenPaare(int z, int s)“.

```
public static void KreuzTeilPlatzieren(int z, int s) {
		if(z==1 && s==0) {
		}
		else {
			if(z==0 && s==1){
				B2();
				D();
				L2();
			}
			else if(z==1 && s==2){
				R2();
				D2();
				L2();
			}
			 […]
			else {

			}
		}
	}
```

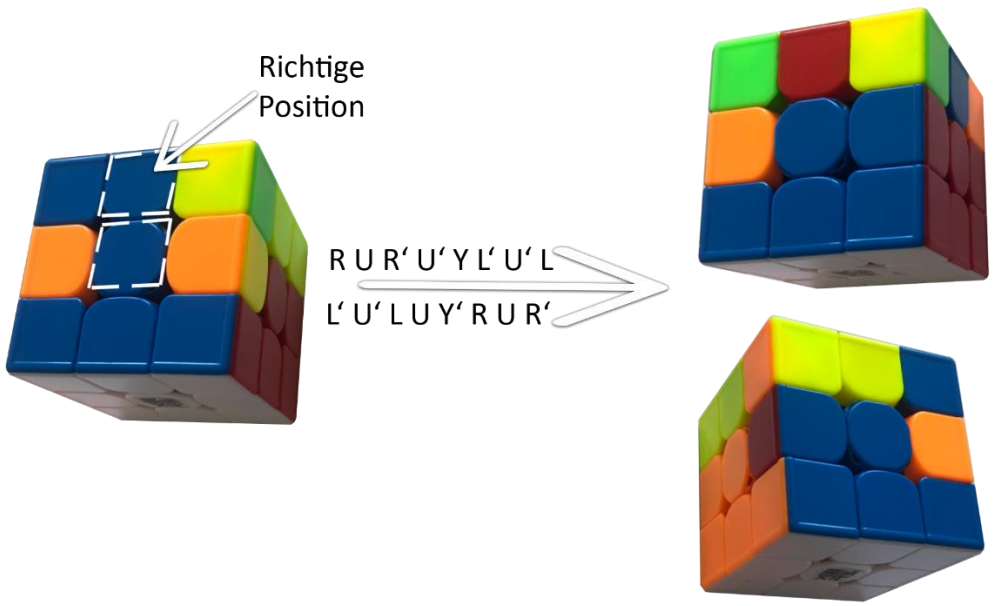
M26: Implementation der Methode „KreuzTeilePlatzieren(int z, int s)“.

```
	 static int[]RichtigeEcke=new int[2];
	public static void FindePassendeEcke(int farbe1, int farbe2, int
farbe3) {
		FindeFarbe(farbe1);
		for(int i=0; i<8; i++) {
		int E1Z=EckenPaare(GefundeneFarben[i][0],
		GefundeneFarben[i][1])[0][0];

		int E1S=EckenPaare(GefundeneFarben[i][0],
		GefundeneFarben[i][1])[0][1];

		int E2Z=EckenPaare(GefundeneFarben[i][0],
		GefundeneFarben[i][1])[1][0];

		int E2S=EckenPaare(GefundeneFarben[i][0],
		GefundeneFarben[i][1])[1][1];
		if(E1Z==1 && E1S==1 || E2Z== 1 && E2S ==1) {

		}
		else {
			if(Wurfel[E1Z][E1S]==farbe2 && Wurfel[E2Z][E2S]==farbe3
			|| Wurfel[E1Z][E1S]==farbe3 &&
			Wurfel[E2Z][E2S]==farbe2) {
				RichtigeEcke[0]=GefundeneFarben[i][0];
				RichtigeEcke[1]=GefundeneFarben[i][1];
			}
		}
		}
	}
```

M27: Implementation der Methode „FindePassendeEcke(int farbe1, int farbe2, int farbe3)“.

```
public static int[][] EckenPaare(int z, int s) {
		if(z==0 && s==0) {
			int[][] EckPaar= {{3,9},{3,8}};
			return EckPaar;
		}
		else if(z==0 && s==2) {
			int[][] EckPaar= {{3,5},{3,6}};
			return EckPaar;
		}
		else if(z==2 && s==0) {
			int[][] EckPaar= {{3,0},{3,11}};
			return EckPaar;
		}
		 […]
		else {
			int[][] EckPaar= {{1,1},{1,1}};
			return EckPaar;
		}
	}
```

M29: Implementation der Methode „EckenPaare(int z, int s)“.



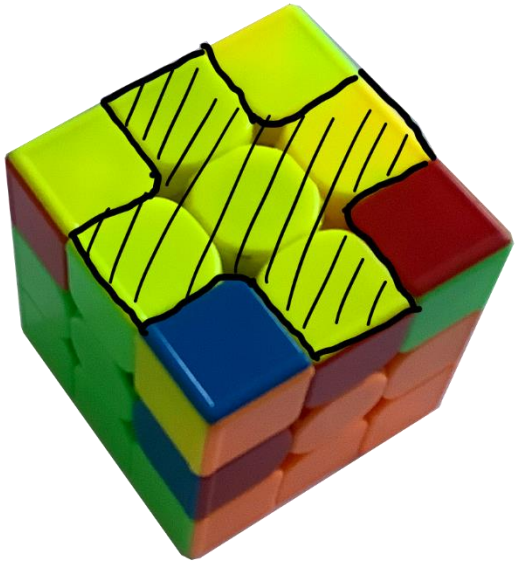
M30: „SecondLayer“ Algorithmus.

```
public static void InPositionBringen(int[] Kante) {
            int z=Kante[0];
            int s=Kante[1];
            if(z==5 && s==10 || z==7 && s==0) {

            }else{
                  if(z==5 && s== 1 || z==6 && s==1) {
                        Ds();
                  }
                  else if(z==5 && s== 4 || z==7 && s==2) {
                        D2();
                  }
                  else if(z==5 && s== 7 || z==8 && s==1) {
                        D();
                  }
                  else if(z==4 && s==0 || z==4 && s==11){
                        Y();
                        SeconLayerAlgorithm();
                        Ys();
                        D2();

                  }
                  else if(z==4 && s==2 || z==4 && s==3){
                        Y2();
                        SeconLayerAlgorithm();
                        Ys();
                        D();
                  }
                  else if(z==4 && s==5 || z==4 && s==6){
                        Ys();
                        SeconLayerAlgorithm();
                        Ys();
                  }
                  else if(z==4 && s==8 || z==4 && s==9){
                        SeconLayerAlgorithm();
                        Ds();
                  }
            }
      }
```

M31: Implementation der Methode „InPositionBringen(int[] Kante)“.

```
public static void SecondLayerPlatzieren(int farbe) {
		if(Wurfel[5][10]==farbe) {
			Ys();
			D();
			L();
			D();
			Ls();
			Ds();
			Fs();
			Ds();
			F();
		}
		else {
			D2();
			Ls();
			Ds();
			L();
			D();
			B();
			D();
			Bs();
		}
	}
```

M32: Implementation der Methode „SecondLayerPlatzieren(int Farbe)“.



M33: Darstellung von „Gelbes Kreuz“.

```
	public static void OLL() {
		NachObenOrientieren(6);
		if(Wurfel[0][1]==6 && Wurfel[1][0]==6 && Wurfel[1][2]==6 &&
		Wurfel[2][1]==6) {
		}
		else {
			GelbesKreuz();
		}
		LCD.drawString("GelbesKreuzFertig", 0, 0);
		Button.waitForAnyPress();
		NachObenOrientieren(6);
		if(Wurfel[0][0]==6 && Wurfel[0][2]==6 && Wurfel[2][0]==6 &&
		Wurfel[2][2]==6) {
			LCD.drawString("Absturz 1", 0, 7);
		}
		else {
			OLLOrientation();
		}
	}
```

M34: Implementation der Methode „OLL()“.

```
	public static void GelbesKreuz() {
		NachObenOrientieren(1);
		if(Wurfel[6][1]!=6 && Wurfel[7][0]!=6 && Wurfel[7][2]!=6 &&
		Wurfel[8][1]!=6) {//Dot Shape
			f(); r(); u(); rs(); us(); fs(); D2(); f(); u(); r();
		us(); rs(); fs();
		}
		else if(Wurfel[6][1]!=6 && Wurfel[7][0]==6 && Wurfel[7][2]==6
		&& Wurfel[8][1]!=6) { //I Shape
			f(); r(); u(); rs(); us(); fs();
		}
		else if(Wurfel[6][1]==6 && Wurfel[7][0]!=6 && Wurfel[7][2]!=6
		&& Wurfel[8][1]==6 ) { //I Shape
			D(); f(); r(); u(); rs(); us(); fs();
		}
		else if(Wurfel[6][1]!=6 && Wurfel[7][0]!=6 && Wurfel[7][2]==6
		&& Wurfel[8][1]==6 ) { //L Shape
			Ds(); f(); u(); r(); us(); rs(); fs();
		}
		else if(Wurfel[6][1]!=6 && Wurfel[7][0]==6 && Wurfel[7][2]!=6
		&& Wurfel[8][1]==6 ) { /L Shape
			D2(); f(); u(); r(); us(); rs(); fs();
		}
		else if(Wurfel[6][1]==6 && Wurfel[7][0]==6 && Wurfel[7][2]!=6
		&& Wurfel[8][1]!=6 ) { /L Shape
			D(); f(); u(); r(); us(); rs(); fs();
		}
		else if(Wurfel[6][1]==6 && Wurfel[7][0]!=6 && Wurfel[7][2]==6
		&& Wurfel[8][1]!=6 ) {/L Shape
			f(); u(); r(); us(); rs(); fs();
		}
		NachObenOrientieren(6);
	}
```

M35: Implementation der Methode „GelbesKreuz()“.

```
public static void OLLOrientation() {
		NachObenOrientieren(6);
		if(Wurfel[3][0]==6 && Wurfel[2][2]==6 && Wurfel[3][6]==6 &&
		Wurfel[3][9]==6) { //Antisune
			Ls(); Us(); L(); Us(); Ls(); U2(); L();
		}
		else if(Wurfel[3][3]==6 && Wurfel[3][5]==6 && Wurfel[3][9]==6
		&& Wurfel[3][11]==6){ //H
			R(); U(); Rs(); U(); R(); Us(); Rs(); U(); R(); U2();
			Rs();
		}
		else if(Wurfel[0][0]==6 && Wurfel[3][11]==6 && Wurfel[3][6]==6
		&& Wurfel[2][2]==6){ //L
			Rs(); Fs(); Ls(); F(); R(); Fs(); L(); F();
		}
		else if(Wurfel[3][0]==6 && Wurfel[3][3]==6 && Wurfel[3][5]==6
		&& Wurfel[3][8]==6){ //Pi
			L(); Us(); Rs(); U(); Ls(); U(); R(); U(); Rs(); U();
			R();
		}
		else if(Wurfel[2][0]==6 && Wurfel[3][2]==6 && Wurfel[3][5]==6
		&& Wurfel[3][8]==6){ //Sune
			R(); U(); Rs(); U(); R(); U2(); Rs();
		}
		else if(Wurfel[0][0]==6 && Wurfel[2][0]==6 && Wurfel[3][2]==6
		&& Wurfel[3][6]==6){ //T
			Rs(); Fs(); L(); F(); R(); Fs(); Ls(); F();
		}
		else if(Wurfel[0][0]==6 && Wurfel[0][2]==6 && Wurfel[3][0]==6
		&& Wurfel[3][2]==6){ //U
			R2(); D(); Rs(); U2(); R(); Ds(); R(); U2(); Rs();
		}
		else {
			Y();
			OLLOrientation();
		}
	}
```

M36: Implementation der Methode „OLLOrientation()“.

```
public static void PLL() {
		NachObenOrientieren(6);
		PLLOrientierung();
		LCD.drawString("OrientierungPPL", 0, 3);
		PLLAusrichtung();
	}
```

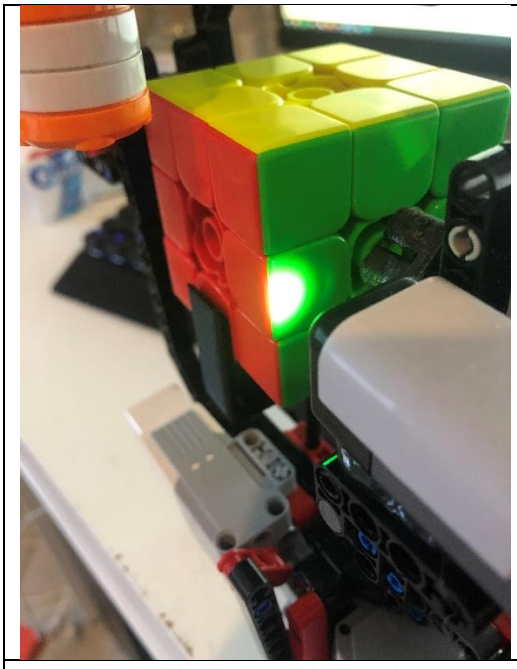
M37: Implementation von der Methode „PLL()“.

```
public static void PLLOrientierung() {
		if(Wurfel[3][5]== Wurfel[3][3]) {
			Ls(); Us(); L(); U(); L(); Fs(); L2(); U(); L(); U();
		Ls(); Us(); L(); F(); //T-Permutation
		}
		else if(Wurfel[3][0]!=Wurfel[3][2] &&
		Wurfel[3][3]!=Wurfel[3][5] && Wurfel[3][6]!=Wurfel[3][8] &&
		Wurfel[3][9]!=Wurfel[3][11]){
			F2(); D(); R2(); U(); R2(); Ds(); Rs(); Us(); R();
			F2(); Rs(); U(); R(); //Y-Permutation
		}
		else if(Wurfel[3][11]== Wurfel[3][9] &&
			Wurfel[3][5]==Wurfel[3][3]) {

		}
		else {
			Y();
			PLLOrientierung();
		}
	}
```

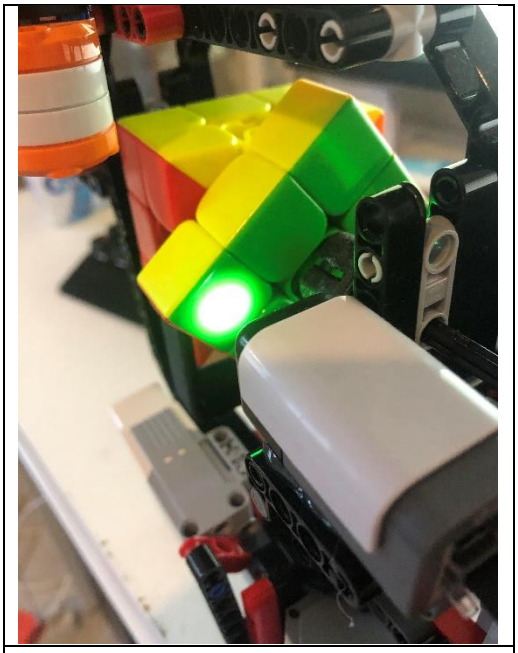
M38: Implementation der Methode „PLLOrientierung()“.

```
public static void PLLAusrichtung() {
		NachObenOrientieren(6);
		if(Wurfel[3][6]==Wurfel[3][7]){//U-Permutation
			if(Wurfel[3][9]==Wurfel[3][4]) { //Ua-Permutation
				R(); Us(); R(); U(); R(); U(); R(); Us(); Rs();
				Us(); R2();
			}
			else if(Wurfel[3][3]==Wurfel[3][10]){//Ub-Permutation
				Ls(); U(); Ls(); Us(); Ls(); Us(); Ls(); U();
			L(); U(); L2();
			}
		}
		else if(Wurfel[3][6]!=Wurfel[3][7]){
			if(Wurfel[3][0]==Wurfel[3][4]) {//Z-Permutation
				Rs(); Us(); Rs(); F(); R(); Fs(); U(); R(); Fs();
				Us(); Ls(); U(); L(); F();
			}
			else if(Wurfel[3][0]==Wurfel[3][7]) {//H-Permutation
				L(); R(); U2(); Ls(); Rs(); Fs(); Bs(); U2();
			F(); B();
			}
		}
		else if(Wurfel[3][6]==Wurfel[3][7] &&
			Wurfel[3][0]==Wurfel[3][1]) {

		}
		else {
			Y();
			PLLAusrichtung();
		}
	}
```

M39: Implementation der Methode „PLLAusrichtung()“.



M40: Abbildung, wie der Farbsensor eine Kante des Zauberwürfels scannt.



M41: Abbildung, wie der Farbsensor eine Ecke des Zauberwürfels scannt.

8. Schlusserklärung

Ich erkläre, dass ich die Facharbeit ohne fremde Hilfe angefertigt und nur die im Literaturverzeichnis angeführten Quellen und Hilfsmittel benutzt habe.