Technische Spezifikation

HTW Berlin
ZWL-Roboter

Autor: Gruppe ZWL
Letzte Änderung: 15. Dez 2022
Dateiname: ZWL-Roboter_Pflichtenheft.docx
Version: 1.0

Copyright

Version Historie

Version	Datum	Verantwortlich	Änderung
0.1	01.12.2022	Elian	Fachlicher Workflow
0.2	04.12.2022	Elian	technischer Workflow
0.3	05.12.2022	Elian	Komponentendiagramm
0.4	03.12.2022	Markus	Erkennung der Farben durch Kamera
0.5	05.12.2022	Maged	Abgrenzungen + Baugruppe Konstruieren + Technische Zeichnung für Einzelteile und Baugruppe
0.6	07.11.2022	Elian	Funktionalität
0.7	10.11.2022	Rayen	Erweiterungen Funktionalität
0.8	15.11.2022	Markus	Erweiterungen Überblick
0.9	15.12.2022	Alle	Allgemeine Erweiterungen
1.0	15.12.2022	Alle	Abgabe

Inhaltsverzeichnis

Abbildungsverzeichnis

Verzeichnis vorhandener Dokumente

Alle für die vorliegende Spezifikation ergänzenden Unterlagen müssen hier aufgeführt werden.

Dokument	Autor	Datum
3d Teile	Elian	07.12.2022
ZWL-Roboter_Workflow.graphml	Elian	11.12.2022
ZWL-Roboter_Technischer Workflow .graphml	Elian	12.12.2022

1 Prozessüberblick

1.1 Fachlicher Workflow

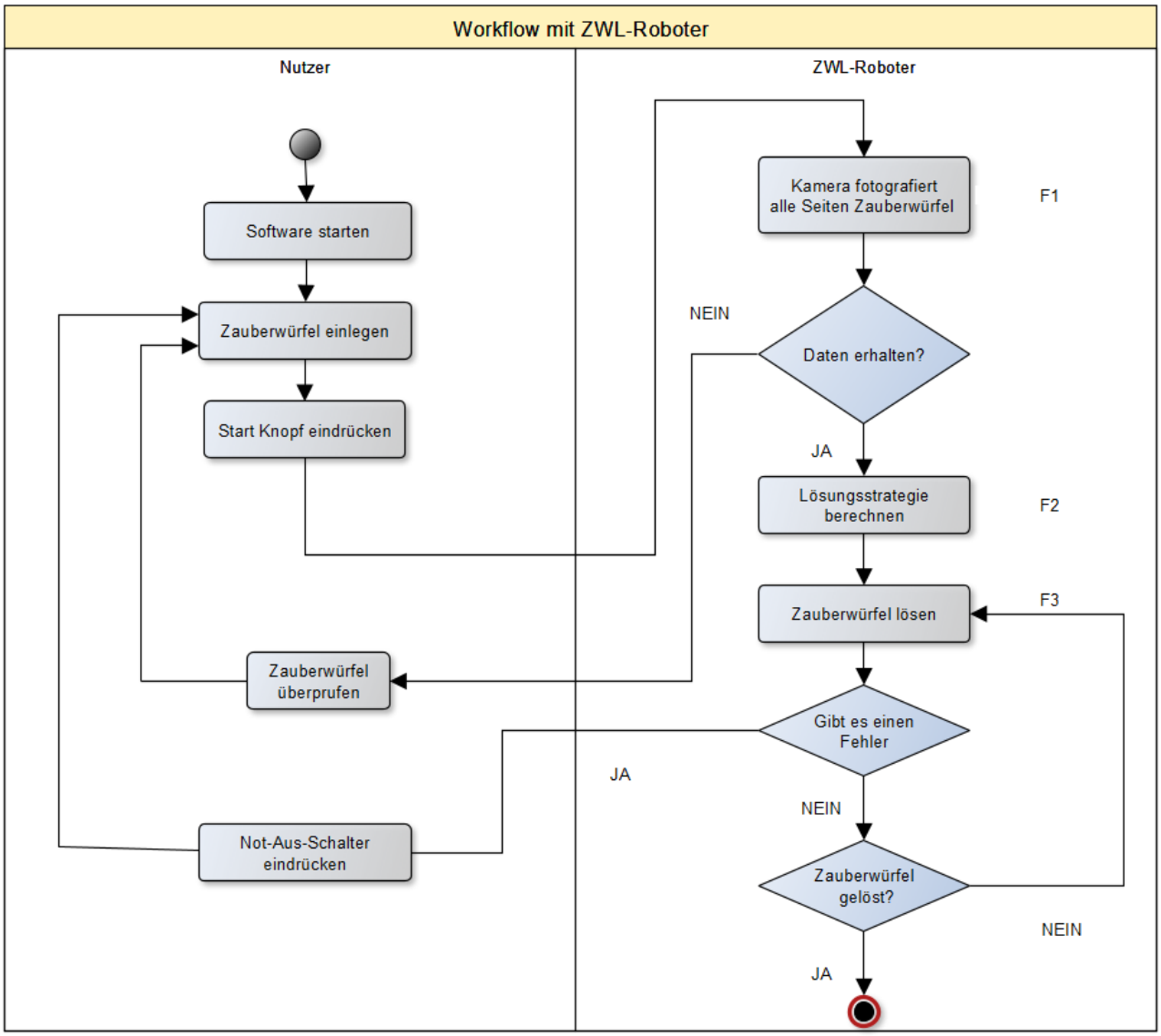


Abbildung 1: Fachlicher Workflow

1.2 Technischer Workflow

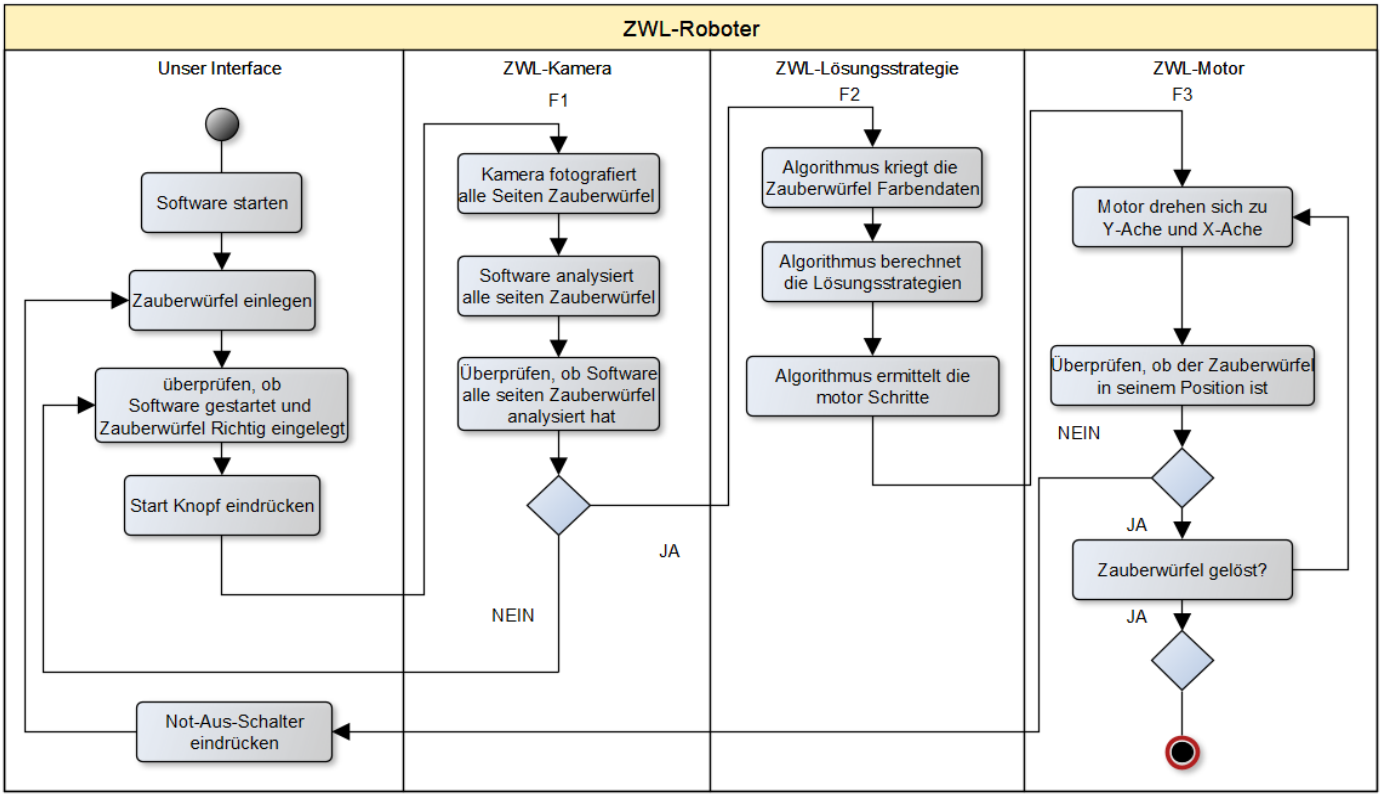


Abbildung 2: technischer Workflow

2 Technische Spezifikation SW

2.1 Überblick Komponenten

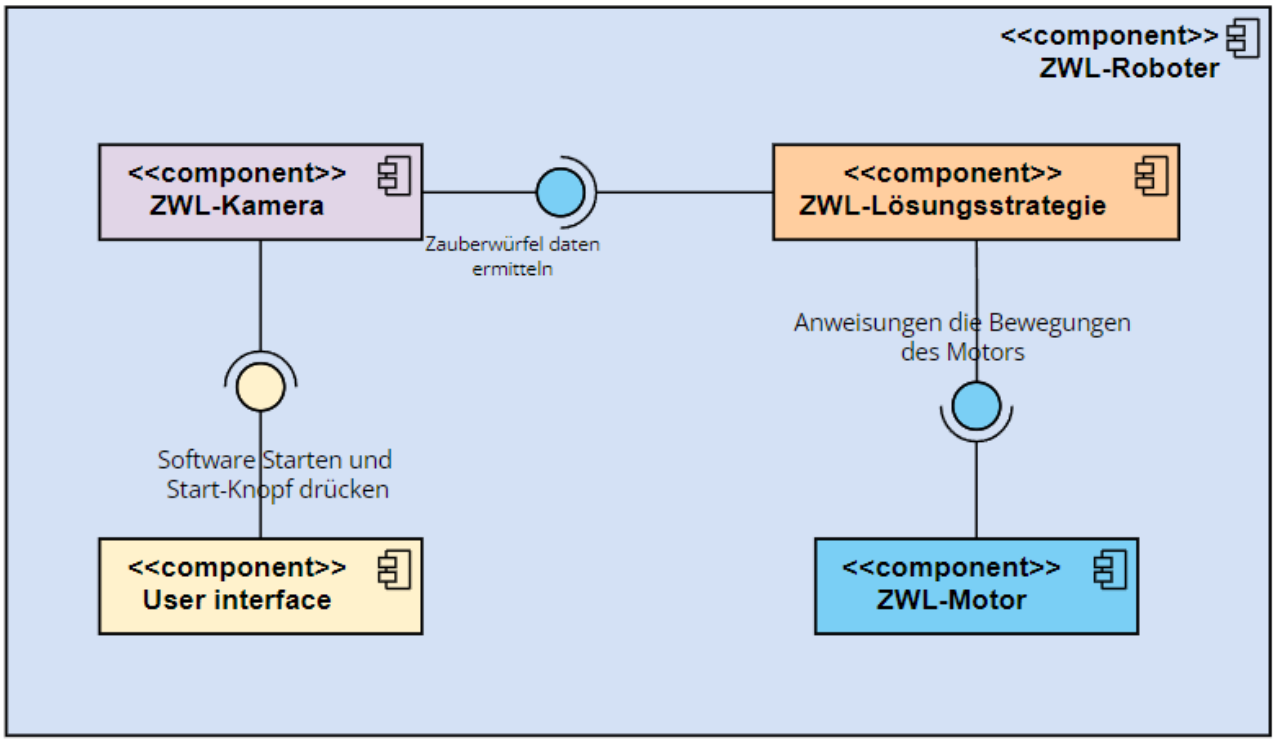


Abbildung **3**: Komponentendiagramm

2.2 Beschreibung der Implementierung

2.2.1 Erkennung der Farben durch Kamera

#	Komponentendetail	Erforderliche Arbeiten
T1	Farben Liste	Es ist eine Liste der möglichen Farben. Diese werden für erkennen der Farben des Würfels benutz, um Fehler zu reduzieren.
T2	GUI	• Hier werden die aktuellen Seiten des Würfe angezeigt und man kann sehen was erkannt wurde bzw. was nicht sowie den Status verfolgen. • Man erkennt das Raster wo der Würfel platziert werden soll für ein besseres Erkennen der Farben.
T3	Einlesen der Seiten	• Die Kamera erkennt die Farben auf einer Seite und speichert diese Werte ab. • Ein Signal zum Drehen des Würfels wird gesendet. • Eine weitere Seite wird erkannt und so weiter, bis alle Seiten in der GUI vorhanden sind und die Farben gespeichert sind.
T4	Übertragung der Farben	• Sobald alle Seiten im System gespeichert sind, wird es dem Algorithmus zur Verfügung gestellt für das weitere Berechnen. • Am Ende Erfolg nochmal die Kontrolle, um zu schauen ob alle Seiten eine gleiche Farbe haben.

T1: Farben Liste

- Eine Liste ist dazu da um die möglichen Farben direkt abrufbar sind. Somit kann die Erkennung schneller geschehen, da es mit den aktuellen Farben vergleicht und somit die mögliche Farbverfälschungen wegen den schlechten Lichtverhältnissen minimieren kann.
- Es ist eine Liste, wo die 6 Grundfarben des Würfels gespeichert sind, zudem kann es auch erweitert werden, damit auch andere Farben unterstützt werden.

T2: GUI

- Es wird eine GUI für die Visualisierung des Scannens benötigt, damit der Benutzer nachverfolgen kann was aktuell gemacht wird.
- Es wird mit OpenCV zusammen mit Python Umgebung verwendet. Dadurch wird es sichergestellt, dass es auf jedem Windows Rechner laufen wird.
- Um den Vorgang zu starten, muss das Software gestartet werden. Eine vorherige Installation ist nicht notwendig. Er reicht, wenn in Betriebssystem die Standardmäßigen Direktiven und Software installiert ist.
- Sobald gestartet wird es direkt die Seiten Fotografieren (Scannen), somit erfolgt das Starten automatisch.

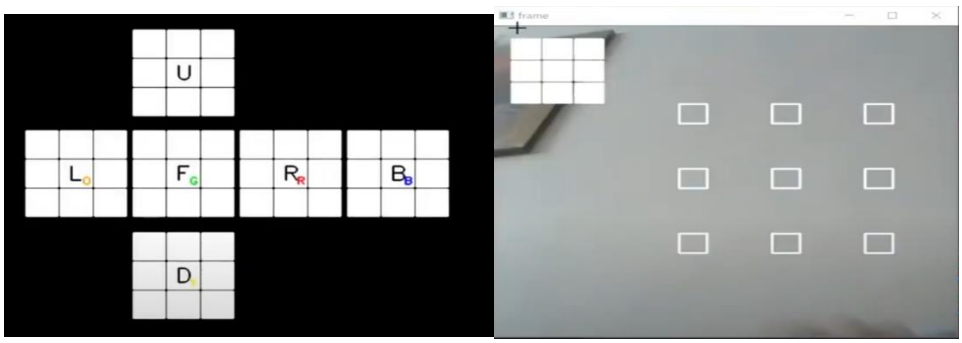


Abbildung 4: Kamera Grafische Anzeiger

T3: Einlesen der Seiten

- Die Kamera scannt eine Seite des Würfels und gibt den Befehl damit dieser sich weiterdreht. Danach werden die anderen Seiten genauso gescannt und die Werte abgespeichert
- Nach den zuvor Ausgesuchter Methode beginnt man mit der Grünen Seite (also die mittlere Farbe ist Grün). Danach wird es erst vorwärts gedreht und zum Schluss werden die beiden Seitlichen Ansichten auch gescannt.
- Die GUI ist dieselbe, lediglich die Werte werden dann dort angezeigt.

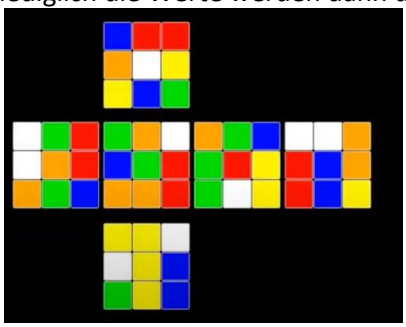


Abbildung 5: Fotografische Ansichten

T4: Unterfunktion B

- Da der Code in den Algorithmus zu Lösungsberechnung integriert ist sendet er die Farben direkt als Matrix für die weitere Berechnung. Diese erfolgt dann sobald alle Seiten erkannt wurden.
- Es gibt keine Exceptiones, denn es werden nur die Farben erkannt, sofern nichts eingelegt ist, wird dort auch nichts erkannt. Auch wenn man den Würfel nicht wie vorgesehen mit Grün nach oben zuerst einlegt, wird kein Fehler auftreten. Den Fehler sieht man dann möglicherweise während des Lösens, dass da etwas nicht stimmt.

2.2.2 Lösungsweg berechnen

2.2.2.1 Grundlagen

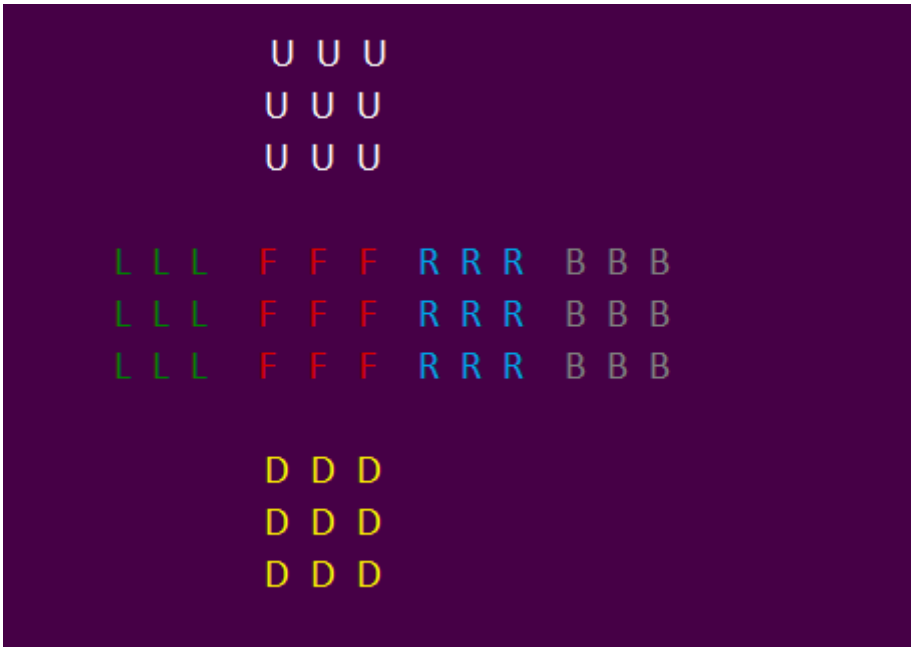
- Zauberwürfel haben 6 Seiten, sie sind mit *U L F R B D* (Up , Left , Front , Right , Back , Down) beschriftet.

- Bewegungen in einer Lösung beschreiben, welche Seite des Würfels gedreht werden soll und in welche Richtung. *U* bedeutet Abbiegeseite *U* ein Viertel drehen im Uhrzeigersinn. *U'* bedeutet Abbiegeseite *U* eine Vierteldrehung gegen den Uhrzeigersinn. *U2* bedeutet Abbiegeseite *U* zwei Viertel Kurven (Richtung spielt keine Rolle weil das gleiche Ergebnisse erzeugen).

- Wir nehmen ein Standardfarbschema von Weiß auf *U*, Grün auf *L*, Rot auf F, Blau auf *R*, Orange auf B und Gelb auf *D* an

- Die Quadrate jedes Würfels können über ihre Farbe (rot, grün, blau usw.) oder über die Seite, zu der sie gehören, beschrieben werden. Wir setzen immer Weiß oben (Seite U), so dass Sie einfach jedes weiße Quadrat als U, jedes gelbe Quadrat als D usw. beschriften können. Es braucht viel weniger Zeichen, um den Zustand des Würfels mit ULFRBD zu beschreiben, als mit Farbnamen, also verwenden wir von nun an ersteres.

- Ein Würfel kann über eine Zeichenfolge für jedes Quadrat beschrieben werden. Die Zeichenfolge für ein gelöstes 3x3x3 wäre :
  "UUUUUUUUURRRRRRRRRFFFFFFFFFDDDDDDDDDLLLLLLLLLBBBBBBBBB"



Um einen optimalen Algorithmus zu finden, um 3x3x3 zu lösen, haben wir zuerst den 2x2x2 aufgrund seiner Einfachheit beobachtet und von dort aus können wir vielleicht definieren, mit welchem Algorithmus wir gehen werden :

Jetzt tun wir so, als hätten wir viel Zeit zur Verfügung und etwa 3 Millionen gelöste 2x2x2 Würfel. Wir nehmen den ersten Würfel und machen eine U-Drehung, die den Würfel in den Zustand UUUUFFLLRRFFBBRRLLBBDDDDDD versetzt :

Wenn wir also einen Würfel finden, der wie UUUUFFLLRRFFBBRRLLBBDDDD aussieht, wissen wir, dass wir ein U' (gegen Uhr U-Drehung) tun können, um es wieder zu lösen
=> damit haben wir den ersten Eintrag in unserer lookup Tabelle.

Wiederholen Sie nun diesen Vorgang für Züge L, F, R usw. (beginnend mit einem gelösten Würfel jedes Mal) und wann immer Sie einen Würfelstatus finden, der nicht in Ihrer Lookup Tabelle enthalten ist, fügen Sie ihn zusammen mit der Umkehrung der Schritte hinzu, die Sie unternommen haben, um den Würfel in diesen Zustand zu bringen ... Wir kehren die Schritte um, damit Sie vom verschlüsselten Zustand zurück in den gelösten Zustand gelangen können. Glücklicherweise gibt es viele Beispiele für Lookup Tabellen im Internet, aus denen wir wählen können.

Eine typische Lookup tabelle enthält 3.674.160 mögliche Zustände für einen 2x2x2-Cube. In der Informatik haben wir eine breite erste Suche durchgeführt, um alle möglichen Würfelzustände zu finden.

=> Ein 2x2x2 Löser kann einfach den Zauberwurfel Status in der Lookup Tabelle nachschlagen, um die Lösung abzurufen.

- **Binäre Suche :**

Wie gesagt, es gibt 3,6 Millionen Einträge in unserer 2x2x2 Lookup Tabelle, wenn wir den aktuellen Zauberwurfel Status mit allen 3,6 Millionen Einträgen vergleichen würden, bis wir eine Übereinstimmung gefunden hätten. Das wird eine Weile dauern würde. Im schlimmsten Fall wäre der gesuchte Eintrag der allerletzte. Dies wäre O (n).
Wenn wir die Einträge in der Nachschlagetabelle sortieren, können wir in die Mitte der Liste springen und diesen Eintrag mit dem gesuchten Eintrag vergleichen. Wenn diejenige, nach der wir suchen, "weniger" ist als die in der Mitte der Datei, dann wissen wir, dass wir in der ersten Hälfte der Datei suchen müssen ... Wir haben gerade die Hälfte der Datei in einer einzigen Suche eliminiert. An diesem Punkt können wir die erste Hälfte der Datei halbieren und herausfinden, ob wir im ersten Viertel der Datei oder im zweiten Quartal suchen sollten.

Wir wiederholen diesen Vorgang, bis wir den gesuchten Element gefunden haben. Dies wird als binäre Suche bezeichnet und ist sehr effizient, es ist O (log n).

- **Pseudocode:**

```
binarySearch(arr, x, low, high)
        repeat until low = high
               mid = (low + high)/2
                   if (x == arr[mid])
                   return mid

                   else if (x > arr[mid]) // x is on the right side
                       low = mid + 1

                   else                   // x is on the left side
                       high = mid – 1
```

Arr ist die Liste wo wir die Suche durchführen werden
X ist der gescuhte Element
Low : Stelle der erste/kleinste Element in der Liste
High : Selle der letzte/grössteElement

Ist es möglich, alles, was oben steht, auf 3x3x3 Zauberwurfel anzuwenden?

Die Antwort ist leider Nein :

Es gibt 43.252.003.274.489.856.000 (43 Trillionen) mögliche Kombinationen, in denen ein 3x3x3 sein kann, so dass dies viel zu groß ist, als dass wir es über eine einzige Lookup Tabelle tun könnten. Google hat 2010 35 Jahre CPU-Zeit investiert, um alle 43 Trillionen Staaten zu lösen.Sie haben nur bewiesen, dass alle 43 Trillionen 3x3x3-Würfelzustände in 20 Zügen oder weniger gelöst werden können. => God´s Number

Ein normaler Mensch kann sich nicht einfach alle möglichen Algorithmen merken, um einen Rubik's Cube mit weniger als 20 schritten zu lösen. Dies war nur mathematisch bewiesen, aber praktisch schwer zu implementieren.

Die gute Nachricht ist, dass der 3x3x3 1974 erfunden wurde und viele Kluge Menschen die letzten 40 Jahre damit verbracht haben, herauszufinden, wie man Zauberwurfel sehr effizient löst. Dies bedeutet, dass viele Algorithmen speziell für die Verwendung durch Computer bereitgestellt wurden, um ihre Prozessleistungen zu nutzen.

2.2.2.2 Mögliche Algorithmen zum lösen eines 3x3x3 :

- **CFOP**:

Diese Methode wird häufig beim Speedcubing verwendet und es ist auf die folgenden Schritte basiert:

1. ein weißes Kreuz zu bilden, bei dem das Kantenstück mit den Mittelstücken auf der angrenzenden Seite übereinstimmt.
2. F2L (first two layers)
3. OLL (orient last layer)
4. PLL (permute last layer)

- **Roux Method:**

Die Roux-Methode hat eine Reihe von Vorteilen. Es verwendet weniger Bewegungen und Algorithmen und ist intuitiver. Der Blockaufbau kann sich jedoch als schwierig erweisen

1. den 1x2x3-Block auf der linken Seite des Würfels erstellen
2. Zweite Block erstellen
3. CMLL Schritt: Es besteht aus acht Anpassungen von A bis H sowie weiteren sechs Anordnungen innerhalb jeder Anpassung. Jede Anordnung beinhaltet den Austausch einer Reihe von Ecken, bis auf eine.
4. Die letzten 6 kanten lösen

- **Metha Method:**

1. FB (First Block) : Der erste Blockschritt ist wie der erste Block der Roux-Methode, außer dass der Block auf der D-Seite und nicht auf der L- oder R-Fläche wie in Roux erstellt wird.
2. 3QB Schritt : Richten Sie zuerst die Zentren aus, indem Sie ein u oder u2 oder u' machen. Dann machen Sie noch einen, damit die Zentren falsch ausgerichtet sind und lösen Sie die hinteren links, hinten rechten und vorderen linken Randschlitze.
3. EOLE Schritt : Dies wird in der Regel erreicht, indem sichergestellt wird, dass die untere rechte Kante ausgerichtet ist, und dann R U R', R U' R', F' U F oder F' U' F verwendet, um zu wechseln, welche Kanten sich im vorderen rechten Schlitz befinden, und dann, sobald die Kanten ausgerichtet sind, die Kante einfügen, die vorne rechts sein sollte.
4. 6CO Schritt
5. 6CP Schritt
6. L5EP step

- **Layer-By-Layer:**

Layer-by-Layer oder normalerweise nur LBL ist eine Gruppe von Methoden, die den Würfel in Schichten löst. Bei der grundlegenden LBL-Methode für Anfänger beendet der Solver die Schichten nacheinander: die ersten Schichtkanten, dann die Ecken, dann die Kanten der zweiten Schicht und schließlich die letzte Ebene. Dies ist eine gängige Methode für neue Cuber, um sie selbst zu entdecken.

1. Cross
2. F2L
3. OLL
4. PLL

Algorithmen für Maschinen geeigenet:

Alle oben vorgestellten Methoden sind Methoden, die entwickelt wurden, um dem menschlichen Geist und der menschlichen Denkweise zu entsprechen. Aber was wichtig zu wissen ist, dass all diese Methoden nicht immer eine optimale Lösung bieten (Gottes Zahl).
Deshalb haben Mathematiker neue Methoden entwickelt, die auf Computern implementiert werden können, um die enorme Geschwindigkeit , die sie bieten können zu nutzen.

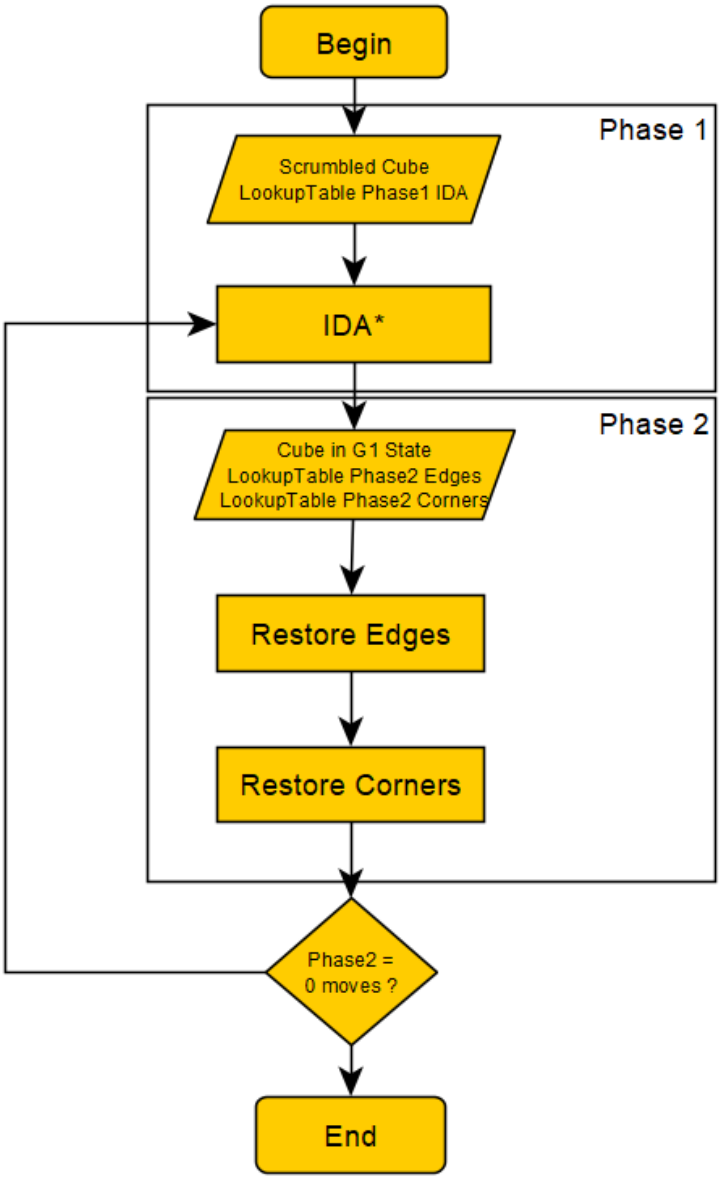
Optimale Lösung Algorithmen:

Die optimalen Online-Rubik's Cube-Solver-Programme verwenden Herbert Kociembas Zwei-Phasen-Algorithmus, der typischerweise eine Lösung von 20 Zügen oder weniger bestimmen kann. Der Benutzer muss die Farbkonfiguration des verschlüsselten Würfels einstellen, und das Programm gibt die Schritte zurück, die erforderlich sind, um es zu lösen. Wir werden auch Kociemba Algorithmus für unsere Program verwenden.

2.2.2.3 Kociemba Algorithm

Seit dem Debüt des Rubik's Cube vor vielen Jahrzehnten hat es viele Fragen aufgeworfen, die weit über die Lösung hinausgehen. Eines dieser Rätsel war die optimale Lösung für einen gegebenen verschlüsselten Zustand. Nachdem wir das Rätsel gelöst hatten, wollten wir es weiter vorantreiben. Wie können wir einen bestimmten Würfel in der geringstmöglichen Anzahl von Zügen lösen? Mathematik und jahrzehntelange Forschung haben bewiesen, dass jedes Gerangel in weniger als 20 Zügen gelöst werden kann (God´s Number) :

- **Workflow :**



- **Two Phase Algorithm:**

Pseudocode:

```
path              current search path (acts like a stack)
node              current node (last node in current path)
g                 the cost to reach current node
f                 estimated cost of the cheapest path (root..node..goal)
h(node)           estimated cost of the cheapest path (node..goal)
cost(node, succ)  step cost function
is_goal(node)     goal test
successors(node)  node expanding function, expand nodes ordered by g + h(node)
ida_star(root)    return either NOT_FOUND or a pair with the best path and its
cost

procedure ida_start(root)
    bound := h(root)
    path := [root]
    loop
        t := search(path, 0, bound)
        if t = FOUND then return (path, bound)
        if t = ∞ then return NOT_FOUND
        bound := t
    end loop
end procedure

function search(path, g, bound)
    node := path.last
    f := g + h(node)
    if f > bound then return f
    if is_goal(node) then return FOUND
    min := ∞
    for succ in successors(node) do
        if succ not in path then
            path.push(succ)
            t := search(path, g + cost(node, succ), bound)
            if t = FOUND then return FOUND
            if t < min then min := t
            path.pop()
        end if
    end for
    return min
end function
```

3 Technische Spezifikation Konstruktion

3.1 Baugruppen

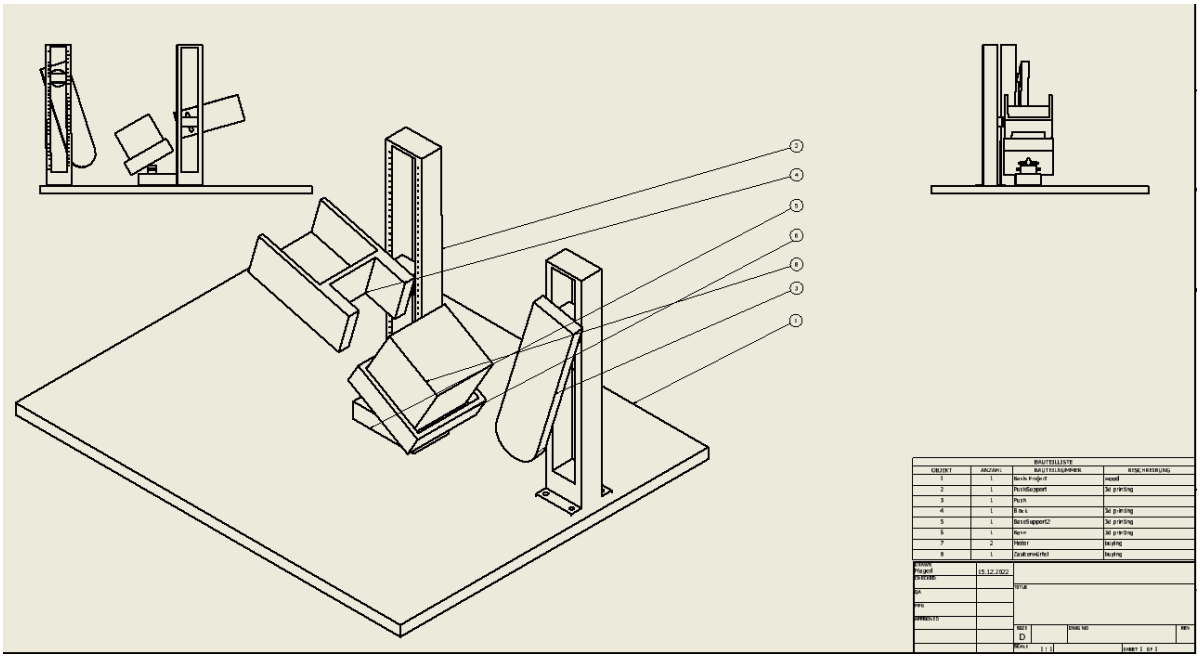


Abbildung 6: Technische Zeichnung für Baugruppe

3.2 Einzelteile

3.2.1 Base:

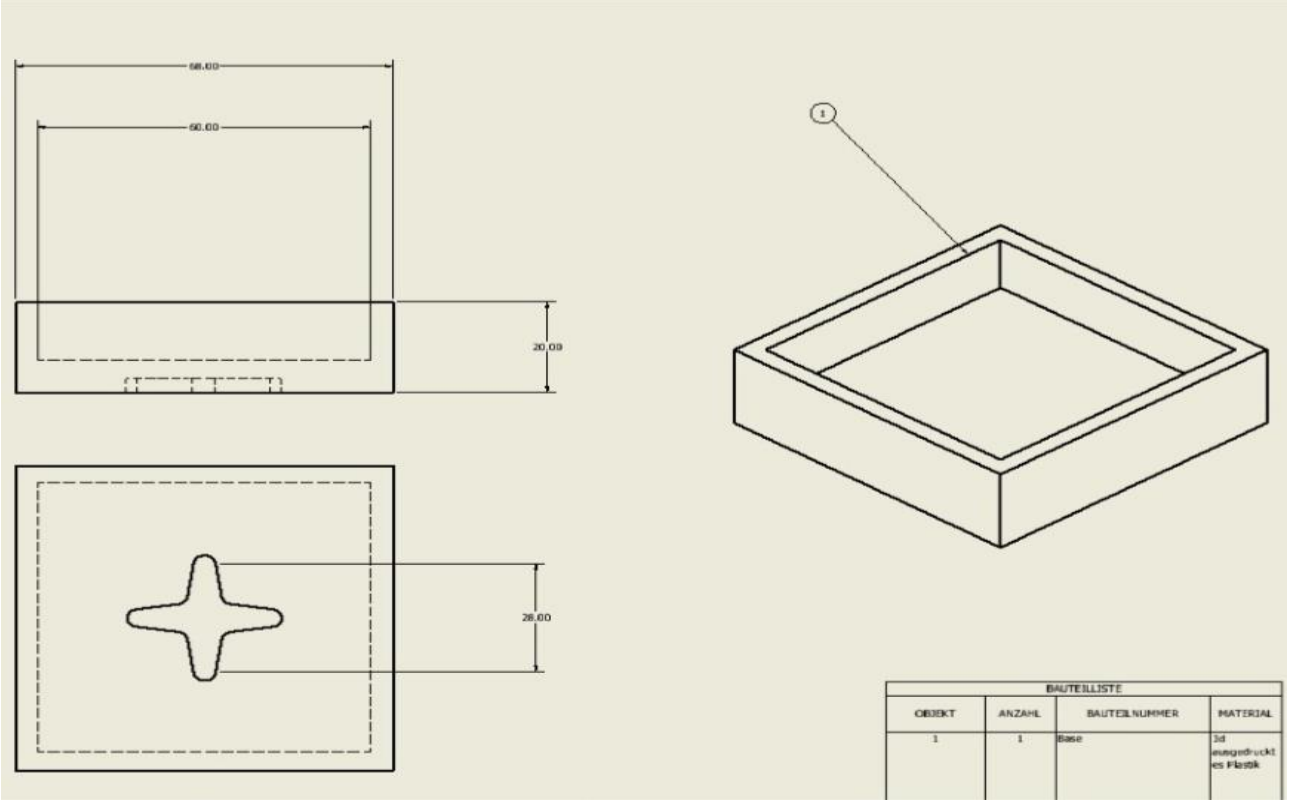


Abbildung 7: Technische Zeichnung für Einzelteil Base

3.2.2 Block:

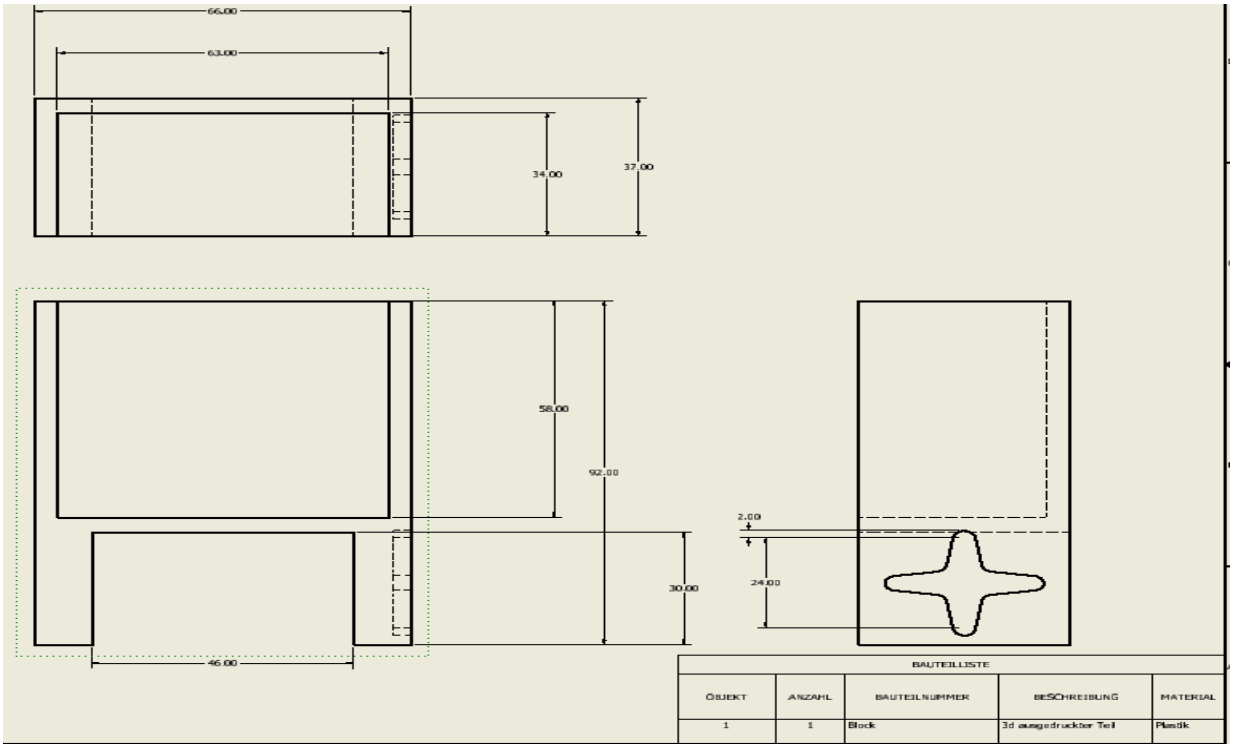


Abbildung 8: Technische Zeichnung für Einzelteil Block

3.2.3 Push:

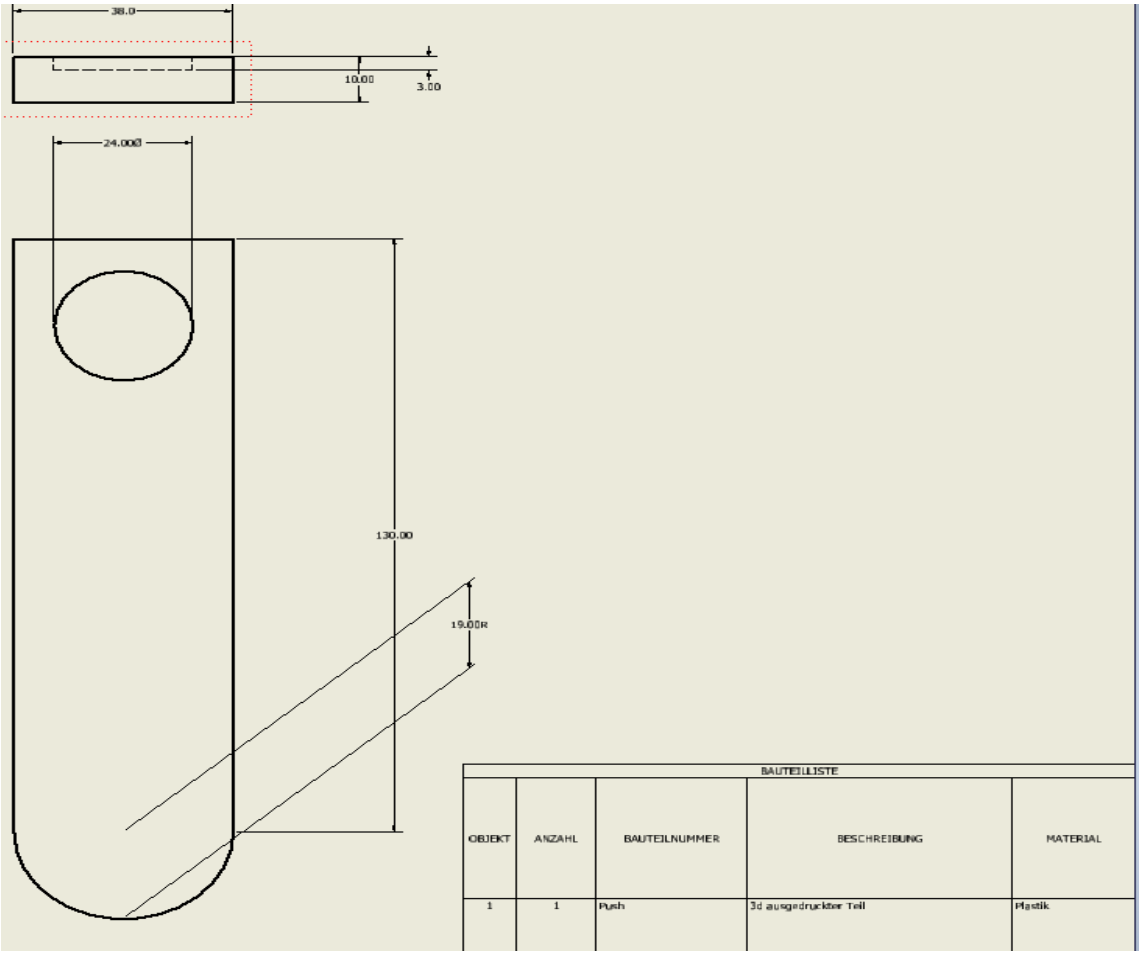


Abbildung 9: Technische Zeichnung für Einzelteil Push

3.2.4 Support:

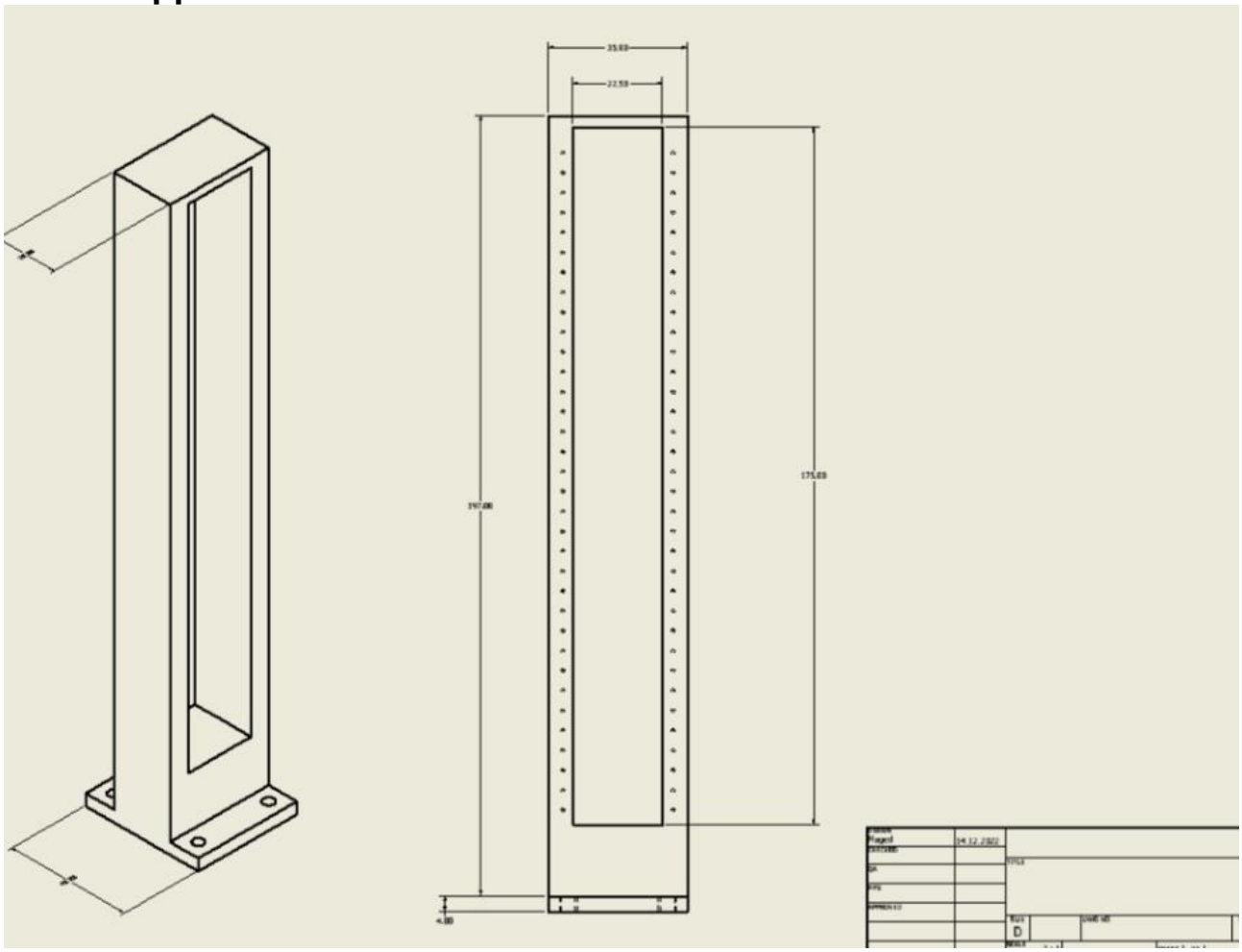


Abbildung 10: Technische Zeichnung für Einzelteil Support

3.3 Berechnungen

Berechnungen sind optional zu erstellen und wo sie sinnvoll sind; z.B. bei Bewegungen, Lasten etc.
Bei den Berechnungen sind die Berechnungsverfahren und die Ergebnisse zu dokumentieren.

4 Offene Fragen

#	Issue	Status	Owner	Deadline
1	Offene Fragen an Auftraggeber	• O (offen) • B (in Bearbeitung) • A (abgeschlossen)	Müller	Datum

Modul Abhängigkeiten

- Welche Abhängigkeiten gibt es zu anderen Modulen?
- Einfache Aufzählungsform