Wissenschaftliche Projektarbeit

Implementierung einer Robotersteuerung eines Dosierprozesses mit dem Matlab® basierten Computer Aided Robotic System der FG CEA.

Betreuer:	Prof. Dr. Thorsten Pawletta, FG, CEA	
	M. Eng. Artur Schmidt, FG, CEA	
eingereicht von:	B. Eng Birger Freymann	Matrikel-Nr.: 111708
eingereicht am:	17.06.2013	

Inhaltsverzeichnis

Aufgabenstellung der Projektarbeit

Bearbeiter: Birger Freymann

Ausgabe: 23. Juli 2012

Thema: Implementierung einer vorhandenen Robotersteuerung des Dosierprozesses in das Computer Aided Robotics (CAR) System der Forschungsgruppe Computational Engineering and Automation.

Ein Forschungsschwerpunkt der FG CEA ist die Entwicklung von Robotersteuerungen nach dem Rapid Control Prototyping Ansatz in Matlab. Im Rahmen eines früheren Projektes ist eine Steuerungssoftware des Dosierprozesses entwickelt und simulativ getestet worden.
Ziel dieses Projektes ist die Implementierung der vorhandenen Steuerungssoftware in das CAE-System der FG CEA in Matlab. Dazu sind für die praktische Robotersteuerung die *KUKA-KAWASAKI Robotic Toolbox for Matlab* und für die 3D-Visualisierung die *KUKA-KAWASAKI Visualization Toolbox for Matlab* zu verwenden. Im Anschluss kann die Robotersteuerung praktisch im Labor in Betrieb genommen werden. Sämtliche Ergebnisse sind in schriftlicher Form beim Betreuer abzugeben.

Schwerpunkte:

- Einarbeitung in die 3D-Modellierung mittels *Surf-Objekten* in Matlab
- Vermessung des realen Dosierprozesses
- Einarbeitung in die *KUKA-KAWASAKI-Visualization Toolbox for Matlab*
- Modellierung des Dosierprozesses in der *KUKA-KAWASAKI-Visualization Toolbox for Matlab* mittels parametrierbaren *Surf-Objekten*
- Einarbeitung in die *KUKA-KAWASAKI-Robotic Toolbox for Matlab*
- Einarbeitung in die entwickelte Steuerungssoftware des Dosierprozesses
- Integration von *Robotic & Visualization Toolbox* und der Steuerungssoftware des Dosierprozesses
- vollständige Simulation der Steuerung im CAE-System
- **Optional:** praktische Inbetriebnahme der Robotersteuerung im Labor

Betreuer: M.Eng. Artur Schmidt, FG, CEA
Prof. Dr. Thorsten Pawletta, FG, CEA

Abkürzungsverzeichnis

CAD	Computer Aided Design
CAR	Computer Aided Robotics
CEA	Computational Engineering and Automation
FG	Forschungsgruppe
ID	Identifikationsnummer
KOS	Koordinatenursprung
RCP	Rapid Control Prototyping
TCP	Tool Center Point

Tabellenverzeichnis

Abbildungsverzeichnis

Anlagenverzeichnis

1. Einleitung

Im Rahmen dieser Arbeit soll basierend auf einer vorhandenen Robotersteuerung eines Dosierprozesses, die Nutzung des *Computer Aided Robotic* (CAR) Systems der Forschungsgruppe (FG) *Computational Engineering and Automation* (CEA) untersucht werden. Die Steuerungssoftware ist für einen realen Versuchsaufbau entwickelt worden und soll mit einem virtuellen Dosierprozess getestet werden. Dafür werden die für den Dosierprozess erforderlichen Stationen als Surf-Objekte modelliert.
Dazu wird zuvor die Modellierung mit Surf-Objekten an einem geometrisch einfachen Beispiel aufgezeigt. Der zu modellierende Dosierprozess besteht aus fünf Stationen. Diese werden geometrisch vermessen und mit Hilfe eines CAD-Programmes abgebildet. Um die Modellierung der Stationen zu vereinfachen wird ein Modellierungskonzept, basierend auf wiederverwendbaren und parametrierbaren Funktionen entwickelt. Anschließend wird prinzipiell das CAR-System der FG CEA vorgestellt und dessen Nutzung anhand eines Beispiels erklärt. Danach wird für den Dosierprozess ein virtueller Roboter vom Typ Kawasaki anhand des virtuellen Versuchsaufbaus im CAR System geteacht. Dabei auftretende Probleme werden analysiert und Lösungskonzepte entwickelt. Anschließend werden für die Steuerung des Dosierprozesses einschließlich Robotersteuerung virtuelle Kontrollelemente entwickelt und implementiert. Anschließend erfolgt eine Zusammenfassung und Bewertung der wesentlichen Untersuchungsergebnisse.

2. Systemanalyse des Dosierprozesses

Dieses Kapitel zeigt den Aufbau des Dosierprozesses, bestehend aus 5 Stationen und einem Roboter. Nachfolgend werden die Aufgaben und Funktionen der Stationen beschrieben und anschließend der Ablauf des Dosierprozesses an einem Beispiel aufgezeigt.

2.1. Physikalischer Aufbau

Abbildung 1 zeigt den Aufbau des Dosierprozesses im Versuchslabor. Zu sehen sind 5 Stationen und 1 Roboter, welche nachfolgend beschrieben werden.

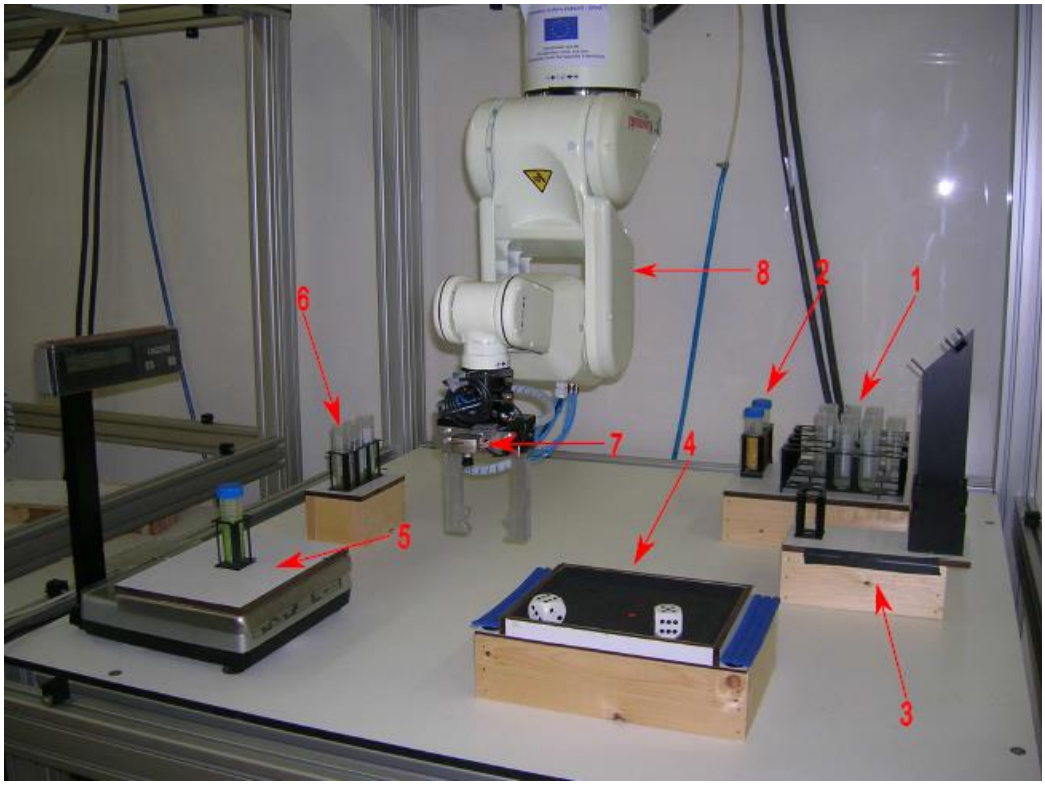


Abbildung 1: Aufbau des Gesamtsystems für den Dosierprozess

1 Anfangspuffer:

Im Anfangspuffer befinden sich 12 Reagenzgläser. Diese werden im Verlauf des Dosierprozesses mit gefärbtem Granulat befüllt.

2 Farbpuffer:

Im Farbpuffer befinden sich 3 Reagenzgläser welche mit unterschiedlich gefärbtem Granulat gefüllt sind. Sie dienen als Granulatquelle zur Befüllung der 12 Reagenzgläser des Anfangspuffers.

3 Umgreifstation:

Der Roboter kann den Inhalt eines gegriffenen Reagenzglases nicht ohne Umgreifen dosieren. Mit Hilfe der Umgreifstation kann der Greifer neu positioniert und das Granulat gezielt entleert werden.

4 Würfelbox mit Würfel:

Mit der Hilfe eines Würfels werden Zufallszahlen zwischen 1 und 6 generiert. Diese Zufallswerte, werden als Sollwerte in Gramm für die Befüllung der Reagenzgläser verwendet.

5 Waage:

Die Waage ist die Messeinheit des Dosierprozesses. Mit ihr wird der Inhalt eines Reagenzglases gewogen. Über eine RS232 Schnittstelle wird die Einwaage am Steuerungsrechner ausgewertet.

6 Endpuffer:

Im Endpuffer werden bis zu drei Reagenzgläser zwischengelagert. Ist der Endpuffer voll so wird er mit Hilfe des Roboters geleert.

7 Kamerasystem:

Das Kamerasystem besteht aus zwei Komponenten, dem Empfänger und der Funkkamera. Der Empfänger ist über einen USB-Anschluss an den PC angeschlossen. Die Bildverarbeitung erfolgt mit *Matlab*®.

8 Roboter:

Der Roboter stellt die wichtigste Prozesskomponente dar. Mit seinem Greifer bewegt er die Reagenzgläser und bedient die einzelnen Prozesskomponenten. Zusätzlich würfelt er, um die genannten Zufallszahlen zu generieren.

2.2. **Grundlegender Ablauf des Dosierprozesses**

Der Ablauf des Dosierprozesses ist in [1] detailliert beschrieben und lässt sich wie dort vorgeschlagen und in Tabelle 1 zusammengefasst in 16 Teilaufgaben zerlegen, die nachfolgend beschrieben werden.

Tabelle 1: Teilaufgaben des Dosierprozesses

	Teilaufgabe	Aufgabe
1	FromBeginningbufferToBalance	Befüllen
2	MoveToPicPose	
3	MoveToDiceMidPos	
4	MoveAndDice	
5	FromColourbufferToEncompass	
6	MoveToFilling	
7	Filling	
8	BackToEncompass	
9	FromBalanceToEndbufferAndPlaceIn	
10	FromEncompassToColourbuffer	
11	MoveToEndbufferAndTake	Entleeren
12	EmptyTubeAndMoveToBeginningbuffer	
13	MoveToBeginningbufferAndPlaceIn	
14	FromEncompassToBalance	
15	FromColourbufferToBalance	
16	FromBalanceToColourbuffer	

Die Teilaufgaben 1 bis 10 dienen zum Befüllen der Reagenzgläser mit Granulat. Teilaufgaben 11 bis 16 werden zum Entleeren der Reagenzgläser verwendet. Nachfolgend werden ausgehend vom in Abbildung 2 dargestellten Anfangszustand die Teilaufgaben kurz beschrieben.

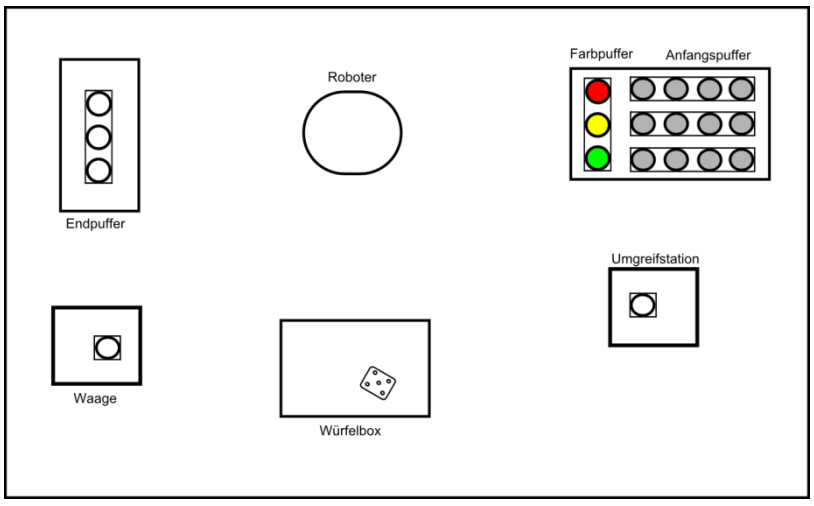


Abbildung 2: Anfangszustand des Dosierprozesses

1 Teilaufgabe (FromBeginningbufferToBalance)

Durch den Roboter wird ein Reagenzglas aus dem Anfangspuffer entnommen und zur Waage gebracht.

2 Teilaufgabe (MoveToPicPose)

Der Roboter positioniert sich über der Würfelbox, um mit einer am Greifer angebrachten Kamera ein Bild vom Würfel aufzunehmen.

3 Teilaufgabe (MoveToDiceMidPos)

Verfährt den Greifer in eine Position mittig über den Würfel.

4 Teilaufgabe (MoveAndDice)

Der Roboter fährt über die Würfelstation und nimmt ein Bild vom Würfel und der Box auf. Aus diesem Bild wird die aktuelle Augenzahl und die Orientierung des Würfels berechnet. Die Augenzahl dient als Sollwert in Gramm für die Befüllung des Reagenzglases. Anschließend würfelt der Roboter mit Hilfe seines Greifers den Würfel.

5 Teilaufgabe (FromColourbufferToEncompass)

Der Roboter entnimmt aus den Farbpuffer eines der drei Farbreagenzgläser und bringt es zur Umgreifstation. Dort wird das Reagenzglas wie in Abbildung 3a gezeigt abgelegt und der Roboter positioniert sich neu. Das Reagenzglas wird erneut gegriffen und am oberen Reagenzglashalter abgelegt. Anschließend erfolgt eine weitere Umpositionierung des Greifers. Nur aus einer Position der Greiferbacken wie in Abbildung 3b, kann der Inhalt des Reagenzglas ideal dosiert werden.

a) Farbreagenzglas quer zu den Greiferbacken b) Farbreagenzglas längs zu den Greiferbacken

Abbildung 3: Lage des Farbreagenzglases im Robotergreifer

6 Teilaufgabe (MoveToFilling)

Das in Teilaufgabe 5 umgegriffene Reagenzglas wird durch den Roboter zur Waage transportiert.

7 Teilaufgabe (Filling)

Der Roboter lässt das Reagenzglas um seine Längsachse rotieren. Dabei fällt Granulat in das auf der Waage befindliche Reagenzglas. Mit Hilfe der Waage wird die Istmenge an Granulat bestimmt. Übersteigt die Istmenge die Sollmenge so stoppt der Füllprozess.

8 Teilaufgabe (BackToEncompass)

Das Farbreagenzglas wird zur Umgreifstation zurückgebracht und dort wie in Abbildung 3b dargestellt abgelegt.

9 Teilaufgabe (FromBalanceToEndbufferAndPlaceIn)

Das befüllte Reagenzglas wird vom Roboter gegriffen, zum Endpuffer gebracht und an eine der drei freien Positionen abgelegt.

10 Teilaufgabe (FromEncompassToColourbuffer)

In dieser Teilaufgabe wird ein Farbreagenzglas, welches sich auf dem oberen Reagenzglashalter befindet, durch den Roboter gegriffen und zum Farbpuffer zurückgebracht.

Die Teilaufgaben 11 bis 16 dienen der Entleerung des vollen Endpuffers. Dabei ist zu beachten, dass die Reagenzgläser, deren Inhalt größer einen vordefinierten Grenzwert ist, nicht sofort in den Anfangspuffer zurückgestellt werden. Diese Reagenzgläser werden zuvor vom Roboter entleert. Nachfolgend wird dieser Sonderfall an einen Beispiel erklärt.

Abbildung 4 zeigt den Vorgang der Entleerung. Der Endpuffer ist mit drei Reagenzgläsern besetzt. Im ersten oberen Reagenzglas befinden sich 13 Gramm rotes Granulat. Das zweite Reagenzglas ist mit 9 Gramm rotem und das dritte Glas mit 6 Gramm gelbem Granulat befüllt (a). Somit muss das Reagenzglas auf Position eins des Endpuffers gelehrt werden, da sein Inhalt größer als 11 Gramm[1] ist. Auf der Umgreifstation befindet sich ein Farbreagenzglas mit gelbem und nicht rotem Granulat. Somit muss das Farbreagenzglas von der Umgreifstation zum Farbpuffer zurück gebracht werden, damit die Umgreifstation für den späteren Entleerungsschritt des Reagenzglases zur Verfügung steht (b). Im nächsten Schritt muss das rote Farbrea-

[1] 11 Gramm wurde als Standardwert für die Entleerung der Reagenzgläser gewählt

genzglas zur Waage transportiert und abgelegt werden (c). Nun erst kann das volle Reagenzglas aus dem Endpuffer entnommen werden und mit Hilfe der Umgreifstation entleert werden (d). Das leere Reagenzglas wird umgegriffen und wieder im Anfangspuffer abgelegt (e). Auch das Farbreagenzglas wird wieder zum Anfangspuffer zurücktransportiert (f). Die beiden im Endpuffer verbliebenen Reagenzgläser können jetzt direkt in den Anfangspuffer zurückgestellt werden, da ihr Inhalt jeweils kleiner als die vordefinierte Grenzwertmenge ist.

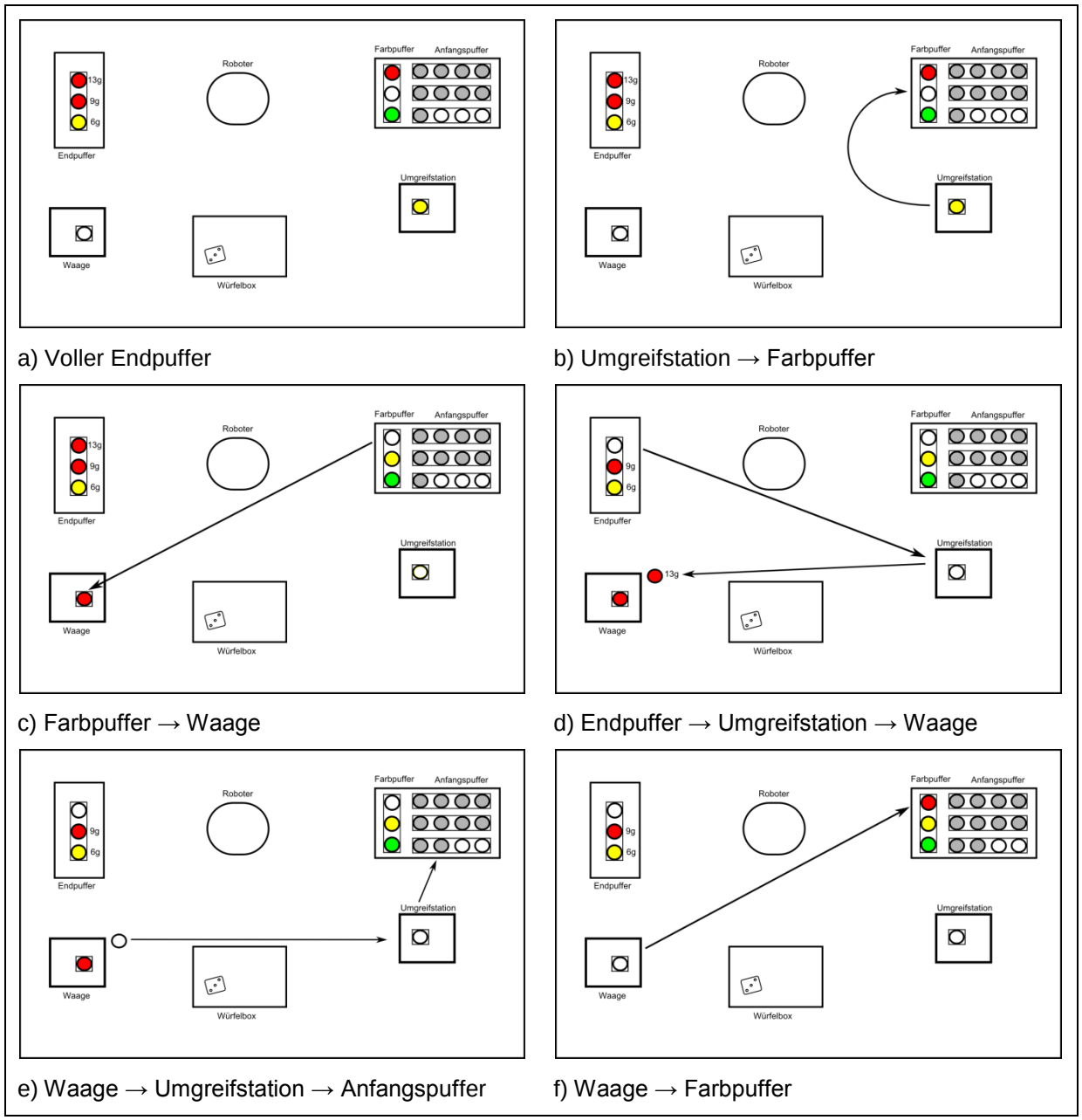


Abbildung 4: Dosierprozess mit vollem Endpuffer (Beispiel)

3. Geometrische Modellierung der Stationen mit *Surf-Objekten* in Matlab®

In diesem Abschnitt wird auf die Modellierung der Stationen als *Surf-Objekte* eingegangen. Dazu ist die messtechnische Erfassung der Abmessungen und Positionen der Stationen erforderlich. Diese sind in einem ersten Schritt in ein CAD-Programm[2] übertragen worden.

Anschließend erfolgt die Modellierung der Stationen als *Surf-Objekte*. Diese werden an einem Beispiel vorgestellt und erklärt. Des Weiteren wird ein Konzept gezeigt, um die Modellierung der Stationen zu vereinfachen.

3.1. Geometrische Grundlagen

Die technischen Zeichnungen der fünf für den Dosierprozess erforderlichen Funktionen sind im Anhang A1 dargestellt. Abbildung 5 zeigt das Layout der Stationen im Labor.

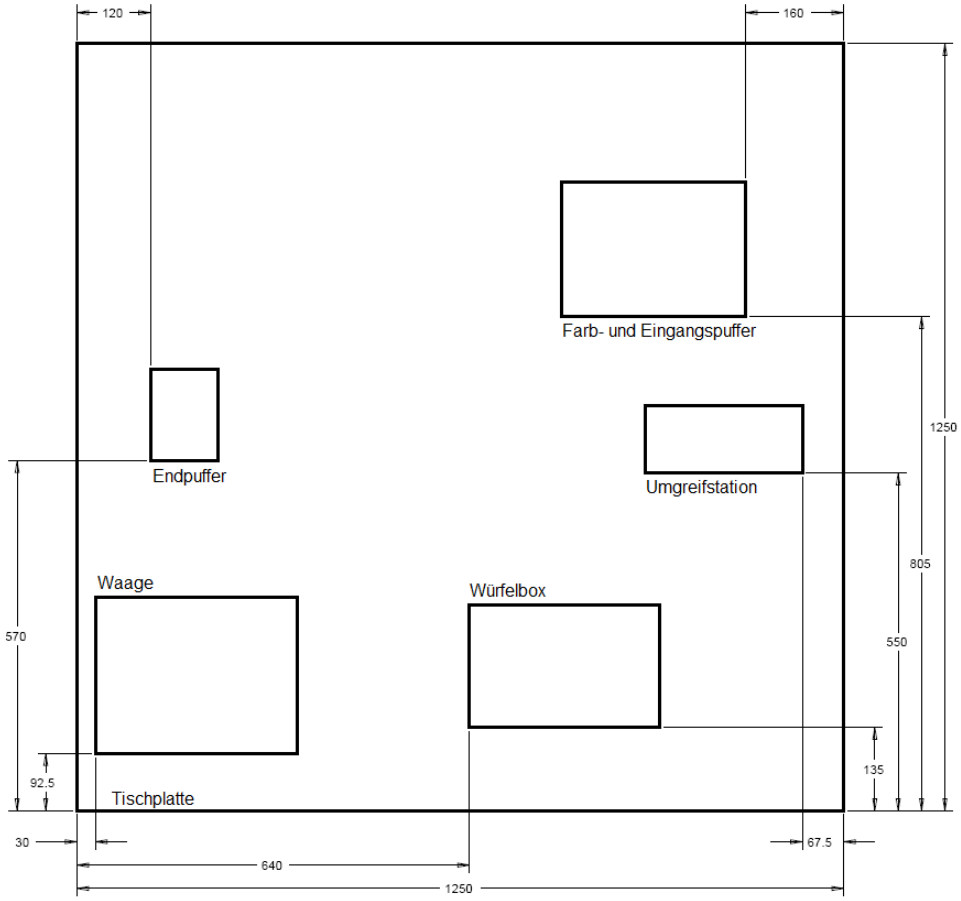


Abbildung 5: Layout der Stationen im Labor

[2] Hierfür wird *Creo® 2.0* verwendet (Bemaßung in mm).

3.2. **Vorbetrachtungen zur Vereinheitlichung der Stationen**

Aus der Analyse des Dosierprozesses wird ersichtlich, dass viele der Stationen aus gleichen Elementarkörpern aufgebaut sind. Diese Eigenschaft kann zur Modellierung der Stationen genutzt werden. Aus einfachen Grundkörpern, wie z.B. einem Quader und einem Quader mit gerader Bohrung in der Mitte, lassen sich die Stationen Eingangspuffer, Waage und Farb- sowie Eingangspuffer zusammensetzen. Zur Modellierung der verbleibenden Stationen sind weitere Grundkörper notwendig.
Die Modellierung der Würfelbox kann mit Hilfe eines weiteren Grundkörpers erfolgen. Dieser bildet die Umrandung der Würfelbox.
Für die Umgreifstation werden abgeschnittene Quader erzeugt, um als Ausleger zu dienen. Der Grundkörper Quader mit Loch muss halbiert werden, um als Reagenzglashalter verwendet werden zu können. Des Weiteren ist es notwendig, diese Grundkörper um beliebe Achsen drehen zu können. In den nachfolgenden Abschnitten wird auf die Modellierung von *Surf-Objekten* und die Erzeugung der beschriebenen Grundkörper als parametrierbare Funktionen eingegangen.

3.3. **Modellierung mit *Surf-Objekten***

Dieser Abschnitt beschreibt die Modellierung von Umweltobjekten mit Hilfe von *Surf-Objekten*. Dabei wird der Umgang mit *Surf-Objekten* an einen einfachen und einen komplexeren Körper gezeigt. Das erste Beispiel zeigt die Modellierung eines Quaders und das zweite einen möglichen Ansatz zur Vereinfachung der Modellierung.

3.3.1. Einfaches Beispiel: Quader

Die Modellierung mit *Surf-Objekten* wird am Beispiel eines Quaders näher erklärt.

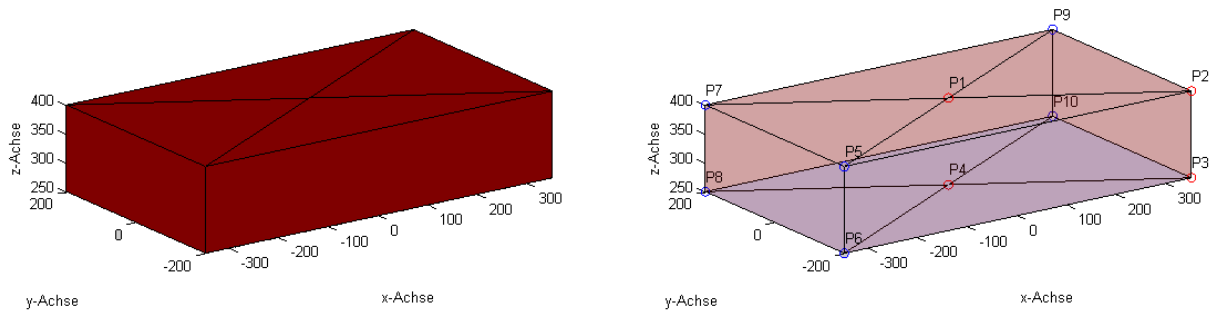


Abbildung 6: Modellierung mit Surf-Objekten am Beispiel eines Quaders.

Als erstes werden drei Matrizen angelegt, welche die X-, Y-, und Z-Koordinaten der Punkte des zu modellierenden Quaders aufnehmen (vgl. Abbildung 6 mit Tabelle 2). Jede Zeile der Matrizen beschreibt eine Konturlinie. Im Beispiel wird die erste Konturlinie durch die Punkte P1 bis P4 erzeugt. Der Körper kann mit 4 Konturlinien beschrieben werden. Dafür werden die X-, Y- und Z-Koordinaten der Punkte P1 bis P4 in die X-, Y- und Z-Matrizen eingeordnet. Beispielsweise hat der Punkt P1 die Koordinaten 0, 0, 400, die in der ersten Zeile das erste Element der jeweiligen Matrizen bilden. Punkt P2 hat die Koordinaten 350, -200, 400 und ist in Zeile 1 an der 2ten Stelle der jeweiligen Matrix eingeordnet.
Die Punkte P1 und P4 werden zu Beschreibung aller Konturlinien benötigt und sind jeweils der Start- bzw. Endpunkt. Dadurch ergibt sich, dass die erste Spalte der Matrizen überall die gleichen Zahlenwerte besitzt. Gleiches gilt jeweils für die letzte Spalte der Matrizen.
Die einzelnen so beschriebenen Konturlinien werden bei der Übergabe an die Surf-Funktion[3] automatisch flächenfüllend verbunden. Um den Quader zu schließen, ist die Wiederholung der ersten Konturlinie notwendig.

Tabelle 2: Surf-Objekte am Beispiel eines Quaders

x=[0, 350, 350, 0;	y=[0, -200, -200, 0;	z=[400, 400, 250, 250;
0, -350, -350, 0;	0, -200, -200, 0;	400, 400, 250, 250;
0, -350, -350, 0;	0, 200, 200, 0;	400, 400, 250, 250;
0, 350, 350, 0]	0, 200, 200, 0]	400, 400, 250, 250]
x=[x; x(1,:)]	y=[y; y(1,:)]	z=[z; z(1,:)]

[3] surf(x,y,z)

3.3.2. Komplexes Beispiel: Endpuffer

Dieser Unterabschnitt beschreibt die Modellierung einer Station unter Verwendung von einfachen Grundkörpern, mit deren Hilfe sich die Stationen nach dem Baukastenprinzip zusammenstellen und modellieren lassen. Ein Beispiel für dieses Vorgehen zeigt der in Abbildung 7 dargestellte Endpuffer.

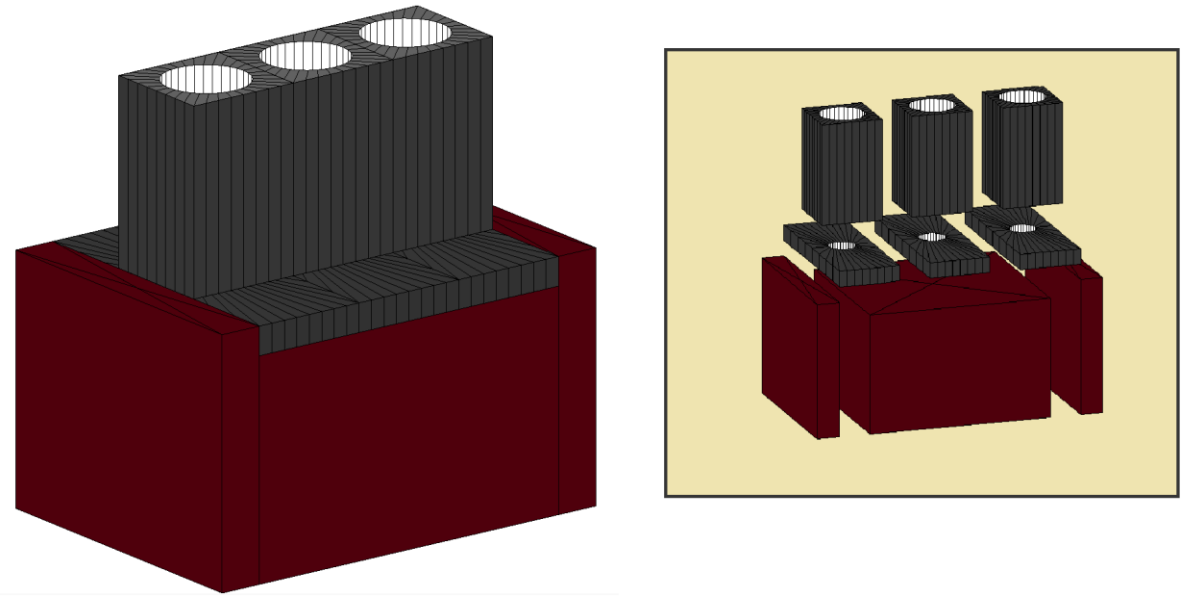


Abbildung 7: Endpuffer bestehend aus 2 Typen von Grundkörpern

Die verschiedenen Grundkörper des Endpuffers wurden unterschiedlich eingefärbt. Einfache Quader sind in dunkelrot und die Quader mit einer Bohrung in der Mitte sind in grau dargestellt. Das Beispiel zeigt, dass sich diese Station aus 3 roten und 6 grauen Grundkörpern aufbauen lässt. Mit Hilfe solcher Grundkörper lassen sich auch andere für den Prozess benötigte Stationen modellieren. Der folgende Abschnitt zeigt die hierfür notwendigen parametrierbaren Funktionen zur Modellierung der Grundkörper.

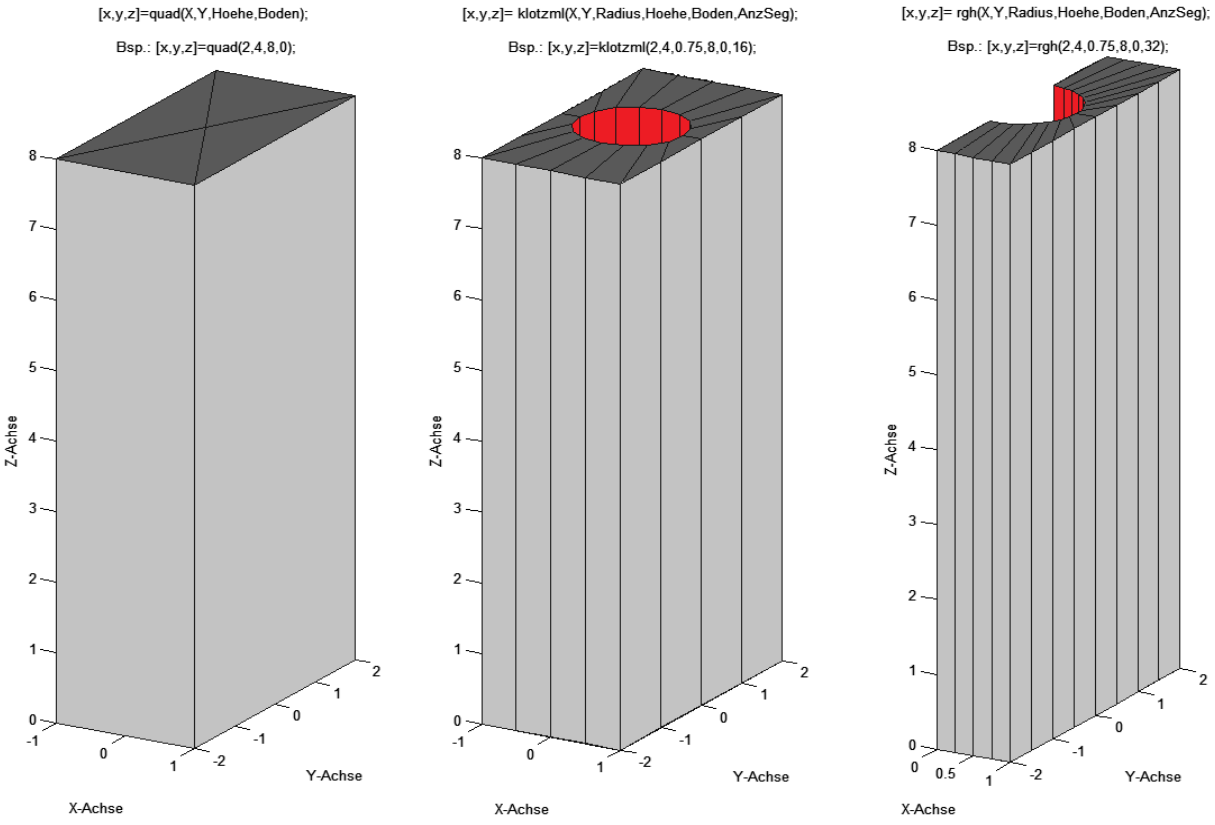
3.4. Modellierung der Stationen mit parametrierbaren Funktionen

Wie im vorherigen Abschnitt beschrieben, werden zur Erstellung der Stationen unterschiedliche Funktionen benötigt. Diese sind in Tabelle 3 zusammengefasst. Sie sind parametrierbar, um die Teilkörper der Stationen darstellen zu können.

Tabelle 3: Übersicht der Grundfunktionen zur Modellierung der Stationen

Funktion	Übergabeparameter	Erzeugung von …
[x,y,z]=quad	X, Y, Höhe, Boden	Quader(n)
[x,y,z]=klotzml	X, Y, Radius, Höhe; Boden, AnzSeg	Quader(n) mit zentrischer Bohrung
[x,y,z]=rgh	X, Y, Radius, Höhe; Boden, AnzSeg	Halben Reagenzglashalter(n)
[x,y,z]=quadschief	X, Y, HöheL; HöheL; Boden	Abgeschnittenen Quader(n)
[x,y,z]=umrandung	Xa, Ya, Xi, Yi, Höhe, Boden	Umrandung(en)
	Hilfsfunktionen	
[x,y,z]=cm2mm	X, Y, Z	Einheitenkonvertierung: cm zu mm
[x,y,z]=rot	X, Y, Z, rx, ry, rz	Rotation um Achse(n)

Die Abbildung 8 zeigt die Anwendung von drei dieser Funktionen an einem Beispiel. In Abbildung 8a ist die Funktion *quad* dargestellt. Diese dient zum Erstellen einfacher Quader, wie sie z.B. zur Modellierung des Tisches aus Unterabschnitt 3.3.1 verwendet werden können. Der Funktion müssen die X- und Y- Kantenlänge des Quaders übergeben werden. Der Koordinatenursprung des erzeugten Körper befindet sich in dessen X-Y-Zentrum. Ebenso benötigt die Funktion die Höhe und Bodenhöhe des Quaders (in Z-Richtung). Es sei darauf verwiesen, dass die Werte für Höhe und Boden vertauschbar sind, ohne dass sich die dabei erzeugten Körper optisch unterscheiden. Dies ist bei der späteren Modellierung der Stationen, die aus mehreren solcher Teilkörper bestehen, wichtig.



a) Quader b) Quader mit Bohrung c) Halber Quader mit Bohrung

Abbildung 8: parametrierbare Funktionen zur Modellierung

Abbildung 8b zeigt die Funktion *klotzml*. Diese Funktion dient der Erstellung von Quadern mit gerader Bohrung im Zentrum. Sie kann verwendet werden, um z.B. die Reagenzglashalter zu modellieren. Die Funktion baut auf der Funktion *quad* auf. Dabei werden weitere Übergabeparameter benötigt. Diese sind zum einen der Radius der Bohrung und die Anzahl der Segmente, aus denen sich der Körper zusammensetzen soll. Bei der Segmentanzahl ist zu beachten, dass nur Vielfache von 2 und Werte größer gleich 4 als Übergabewert erlaubt sind. Des Weiterem sind zur Darstellung der Bohrung Werte von mindestens 8 zu empfehlen.

Abbildung 8c zeigt die Funktion *rgh*. Diese wurde zur Modellierung des halben Reagenzglashalters bei der Umgreifstation erstellt. Sie baut auf der Funktion *klotzml* auf und dient zur Halbierung eines Reagenzglashalters. Die Übergabeparameter sind analog zu der Funktion *klotzml* zu wählen.

Abbildung 9 zeigt die Funktion *quadschief*. Dabei handelt es sich ebenfalls um eine Funktion die speziell für die Umgreifstation entwickelt worden ist. Als Übergabeparameter werden die X- und Y-Kantenlänge, die Höhe des Bodens, sowie die Körperhöhe der linken und rechten oberen Kante benötigt. Mit Hilfe dieser Funktion ist die Modellierung des Auslegers der Umgreifstation möglich.

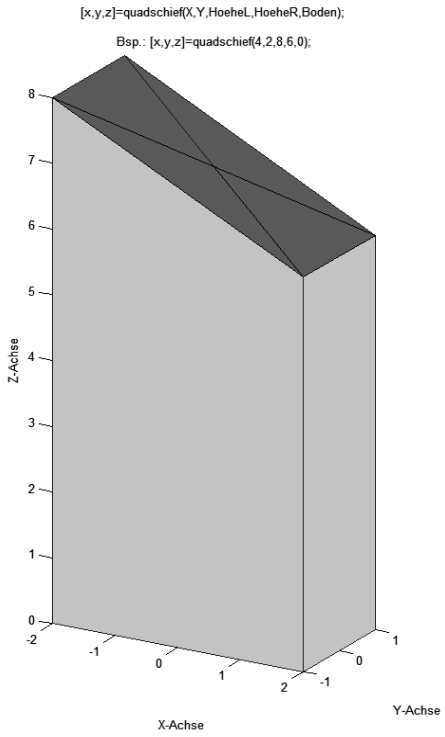


Abbildung 9: Beispiel: Abgeschnittener Quader

Abbildung 10 zeigt die Funktion *umrandung,* die für die Modellierung der Würfelbox verwendet wird. Bei der Verwendung von *Surf-Objekten* ist es nicht möglich, aus einem bereits erstellten Körper etwas zu entfernen. Aus diesem Grund wird eine eigene Funktion zur Modellierung der Umrandung verwendet. Alternativ ist es auch möglich, die Umrandung der Würfelbox mit 4 der oben beschriebenen Quader umzusetzten. Die Funktion *umrandung* benötigt neben der X- und Y-Kantenlänge der Außenkontur auch die X- und Y- Kantenlänge der Innenkontur. Weiterhin muss die Höhe des Bodens in Z-Richtung und die Gesamthöhe übergeben werden.

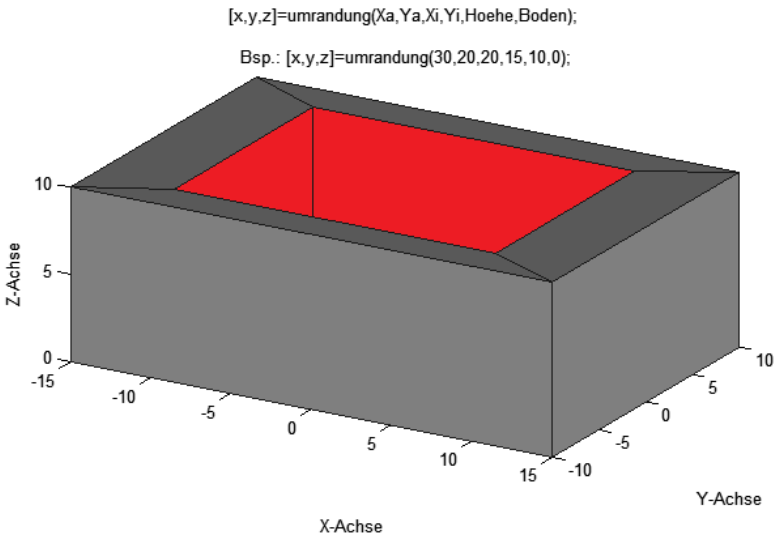


Abbildung 10: Beispiel: Funktion Umrandung

Um alle Stationen des Dosierprozesses zu modellieren sind noch zwei weitere Funktionen notwendig. Eine dieser Funktionen dient der Einheitenkonvertierung. Die Station wurde in cm ausgemessen und modelliert. Zur späteren Einbindung der Stationen ist eine Beschreibung in mm erforderlich. Deswegen wurde eine Funktion *cm2mm* erstellt.

Die letzte Funktion *rot* ist notwendig, um die erstellten Körper um beliebige Winkel drehen zu können. Der Funktion müssen die X, Y, Z -Matrizen des zu drehenden Körpers und die Winkel (α, β, γ) in Grad übergegen werden. Die Drehung erfolgt um den Punkt 0, 0, 0 des zu drehenden Körpers. Dies wird in Abbildung 11 am Beispiel eines Würfels gezeigt. Der Würfel wurde im Beispiel 10 Grad um die X-Achse, 20 Grad um y-Achse und 30 Grad um die Z-Achse gedreht. Die Drehung erfolgt im Beispiel jeweils im positiven Drehsinn um die jeweilige Achse. Negative Drehwinkel als Übergabeparameter sind aber ebenfalls möglich.

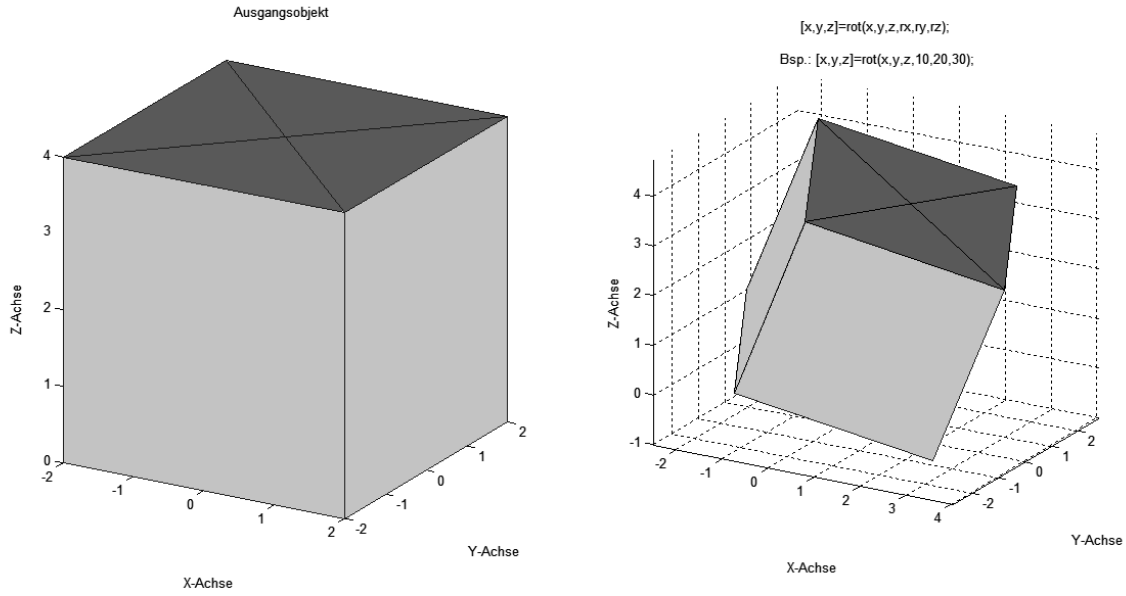
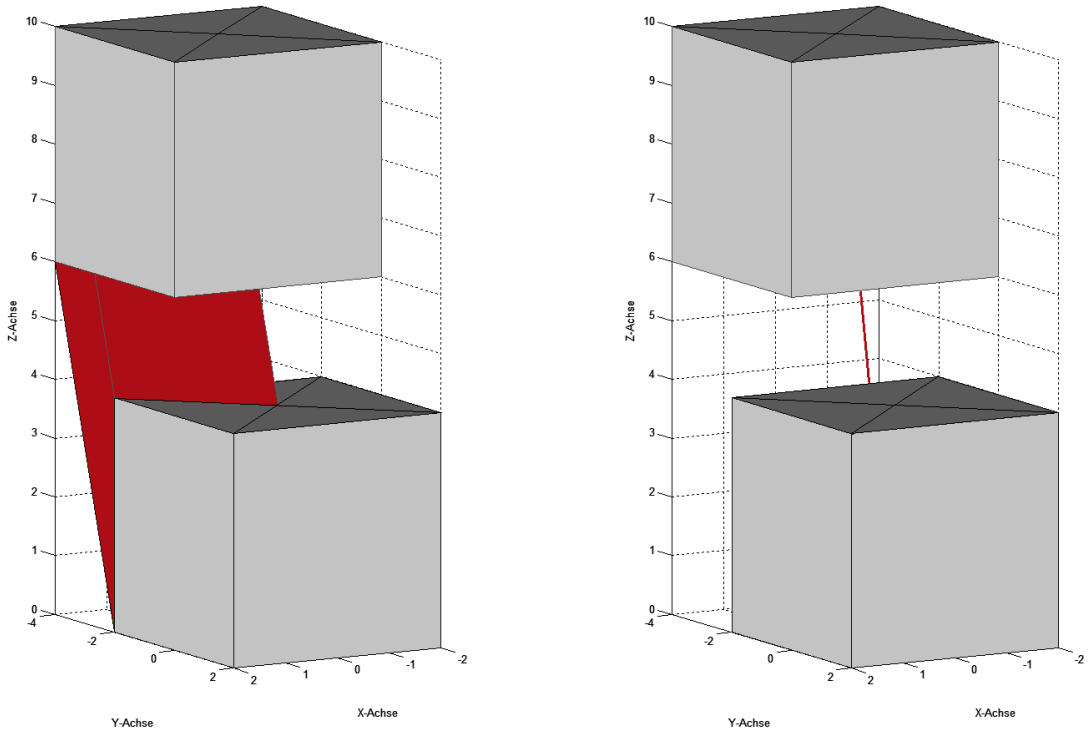


Abbildung 11: Beispiel: Rotation eines Würfels

Mit Hilfe der sieben Funktionen ist es möglich, alle Stationen des Dosierprozesses zu modellieren. Alle gezeigten Funktionen generieren Matrizen der Form 5 x n (n ϵ N) und sind miteinander kombinierbar. Damit aus diesen Teilkörpern die Stationen geformt werden können, müssen wie in Abbildung 12 zu sehen, Schnittstellen[4] integriert werden. Ohne geeignete Schnittstellen können Konturlinien nicht in gewünschter Form von einen zum anderen Grundkörper verlaufen. Abbildung 12a veranschaulicht rot eingefärbte Fehler, die beim Wechsel zwischen den Körpern entstehen können. Mit einer modellierten Schnittstelle, wie in Abbildung 12b, ist ein Übergang zwischen den Körpern zu sehen, der aber kaum störend ist.

[4] Diese fokussiert die Punkte der jeweiligen Konturlinie in einen Punkt (Abbildung 12)



a) ohne Schnittstelle b) mit Schnittstelle

Abbildung 12: Beispiel: Zwei Würfel: Funktionsweise einer Schnittstelle

Der Quellcode der erstellten Funktionen ist im Anhang A8 beigefügt. Der Aufbau der Stationen, mit Hilfe der zuvor beschrieben Grundkörper, ist im Quelltext ausführlich dokumentiert.

4. Integration der geometrischen Modelle in das CAR-System der FG CEA

Das CAR-System (*Computer Aided Robotics*) der Forschungsgruppe CEA unterstützt die Entwicklung von Robotersteuerungen in der wissenschaftlich technischen Berechnungsumgebung Matlab®.

Im Unterabschnitt 4.1 werden erst die grundlegende Struktur und wesentliche Funktionen des CAR-Systems eingeführt. Anschließend wird die Integration der Visualisierung des Dosierprozesses im CAR-System vorgestellt.

4.1. Das CAR System der FG CEA

Zentraler Bestandteil der CAR-Systems ist die *KUKA-KAWASAKI Robotic Toolbox for Matlab*®. Diese ermöglicht die Anwendungsentwicklung von Robotersteuerung mit bis zu 99 Robotern und ist in Abbildung 13 schematisch dargestellt.

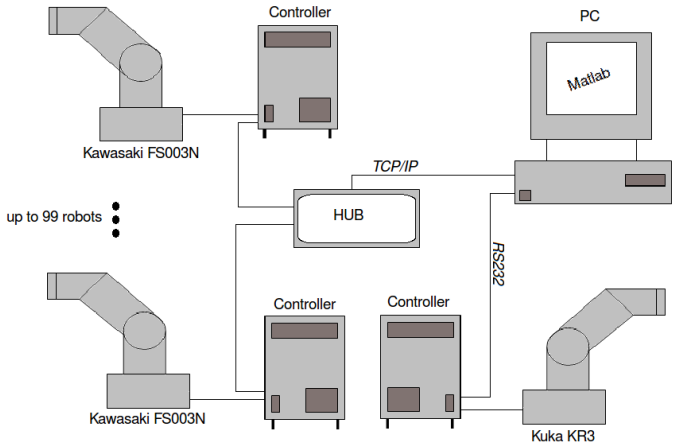


Abbildung 13: Anwendung der Robotic Toolbox for Matlab®nach [2]

Mit Hilfe eines einheitlichen Befehlssatzes lassen sich Kuka- und Kawasaki-Roboter steuern. Ein Auszug ausgewählter Befehle ist in Tabelle 4 dargestellt. Eine ausführliche Dokumentation befindet sich auf der Homepage der Forschungsgruppe [3].

Tabelle 4: Befehlsbeispiele *KUKA-KAWASAKI Robotic Toolbox for Matlab*®

Befehlsbeispiel	Wirkung
Robo_Handle=robot('open', 'Kawasaki', 'tcpip', … '192.168.0.3', 40000)	Verbindung mit Kawasaki-Roboter über TCP/IP
ID = rmove(Robo_Handle,‘home‘)	Roboter fährt auf Homeposition
rkill(Robo_Handle)	Stoppt Roboter und löscht Befehlspuffer
rstop(Robo_Handle)	Stoppt Bewegung des Roboters
rrun(Robo_Handle)	Reaktiviert Roboter-Befehlsinterpreter
Answer = ris(ID)	Prüft den Status des Roboters
P1 = rpoint(x,y,z,o,a,t)	Erzeugt Anfahrpunkt
robot(Robo_Handle, 'close')	Schließt Kommunikation zum Roboter

In Verbindung mit der *KUKA-KAWASAKI Visualization Toolbox for Matlab*® kann ein Projekt virtuell ohne Roboterunterstützung entwickelt werden. Die Verwendung dieser beiden Toolboxen ermöglicht eine Verkürzung der Entwicklungsphase und eine gefahrlose Überprüfung entwickelter Konzepte. Die *KUKA-KAWASAKI Visualization Toolbox for Matlab*® interpretiert dabei die mit Hilfe der *KUKA-KAWASAKI Robotic Toolbox for Matlab*® erzeugten Befehle und visualisiert die Steuerungsaktionen. Die Visualisierung kann bei Bedarf auf einen weiteren PC ausgelagert werden. Abbildung 14 zeigt die prinzipiellen Interaktionen der beiden Toolboxen.

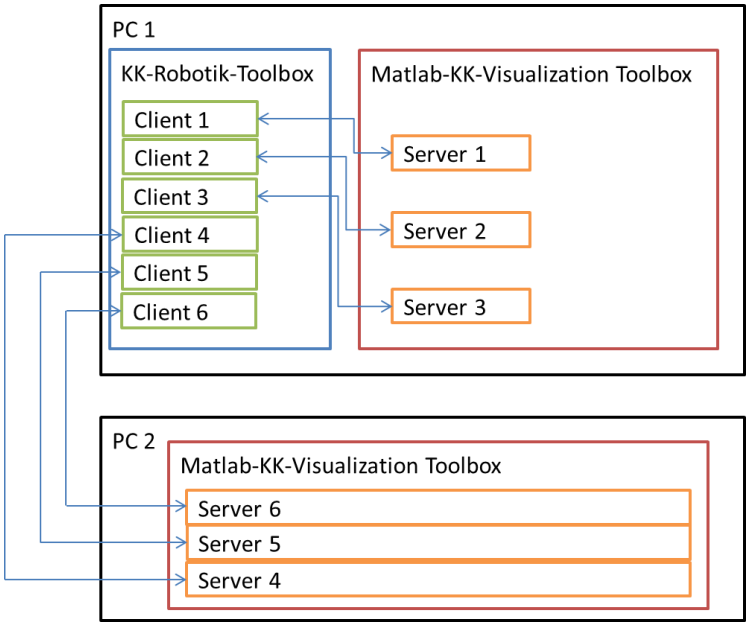


Abbildung 14: Interaktion der beiden Toolboxen der CEA Gruppe [4]

Hierfür wird das TCP/IP-Protokoll zur Datenkommunikation verwendet. Soll eine Steuerung am realen System getestet werden, so wird lediglich der Visualisierungs-PC durch den realen Roboter ersetzt.

Ein Anwendungsbeispiel der *Kuka-Kawasaki-Robotic Toolbox for Matlab*® und der *Kuka-Kawasaki Visualization Toolbox for Matlab*® ist in Tabelle 5 dargestellt und wird nachfolgend beschrieben.

Tabelle 5: Anwendungsbeispiel der Toolboxen

	Matlab®-Instanz **Visualisierungs-PC** mit *KK-Visualization Tbx for Mablab*	Matlab®-Instanz **Kontroll-PC** mit *KK-Robotic Tbx for Matlab*
1	VirtualRobot.create('Kawasaki', 40000, [0, 0, 0, 0, 0, 0])	
2		r1=robot('open', 'Kawasaki', 'tcpip', 'localhost', 40000)
3	VirtualRobot.start_all	
4	VirtualRobot.place_env('table', [0 -500 0 0 0 0]); VirtualRobot.place_env('base', [200 -450 400 0 0 0], [0,1,0]); VirtualRobot.place_env('base', [-200 -450 400 0 0 0], [0,1,0]); VirtualRobot.place_env('base', [0 -450 400 0 0 0], [0,1,0]); VirtualRobot.place_part('test_tube', [200 -450 425 0 0 0],[1,0,0]); VirtualRobot.place_part('test_tube', [-200 -450 425 0 0 0],[1,0,0]);	
5		load waypoints;
6		rmove(r1, waypoints);
7		robot(r1, 'close')
8	VirtualRobot.delete_all	

Abbildung 15 zeigt die Visualisierung des Versuchaufbaus nach Abarbeitung der Schritte 1 bis 4, der in Tabelle 5 gezeigten Befehle. Der Versuchsaufbau besteht aus 1 Roboter, 1 Tisch sowie 2 vereinfacht dargestellten Reagenzgläsern und 3 Grundplatten auf denen die Reagenzgläser stehen. Die verschiedenen Körper des Versuchsaufbaus sind als *Surf-Objekte* modelliert und als Mat-Files [5] gespeichert. Es ist möglich diese von der Homepage der Forschungsgruppe herunterzuladen [5].

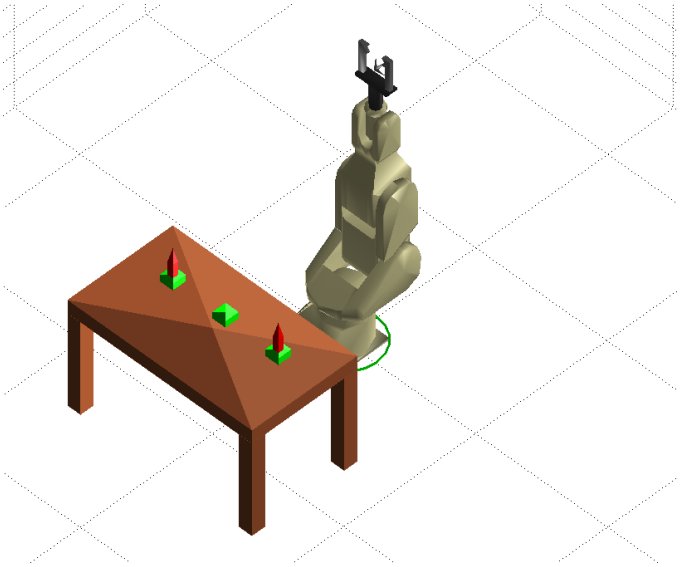


Abbildung 15: Beispiel *KUKA-KAWASAKI Visualization Toolbox for Matlab*®

[5] Mat -File (.mat) / Beispiel: „table.mat“

Das hier gezeigte Beispiel wird nachfolgend beschrieben und die verwendeten Befehle erläutert. Um dieses Beispiel zu testen ist es erforderlich, zwei Matlab®-Instanzen zu starten. Auch wenn beide Instanzen auf einem PC gestartet werden, werden Sie nachfolgend als Visualisierungs-PC und Kontroll-PC bezeichnet. Eine dient dabei als Visualisierungs-PC und die andere wird als Kontroll-PC verwendet. Die Befehlsabfolge wird in Tabelle 5 gezeigt.

Der 1. Befehl wird auf dem Visualisierungs-PC ausgeführt und legt den Typ des zu verwendenden Roboters fest. Ebenfalls wird durch den Befehl ein Port für die Kommunikation geöffnet und es wird die Ausrichtung des Roboters im Koordinatensystem festgelegt. Der 2. Befehl erfolgt auf dem Kontroll-PC, baut eine Kommunikation zum Roboter auf und vergibt ein Handle über, dass der Roboter angesprochen werden kann. Anschließend wird wieder auf den Visualisierungs-PC gewechselt. Befehl Nummer 3 startet die Visualisierung. Die Befehle unter Nummer 4 werden zum einlesen der modellierten *Surf-Objekte* verwendet. Dabei ist zu beachten, dass die *KUKA-KAWASAKI Visualization Toolbox for Matlab®* zwischen *Enviroment* und *Parts* unterscheidet. Diese beiden Grundtypen von Objekten unterscheiden sich in ihren Eigenschaften. Während sich Objekte vom Typ *Environment* nicht durch den Roboter bewegen lassen, ist dies mit den Elementen vom Typ *Part* möglich. Befehl Nummer 5 erfolgt auf dem Kontroll-PC und lädt Wegpunkte[6], welche durch den Roboter angefahren werden sollen. Befehl Nummer 6 nutzt diese Wegpunkte um den Roboter zu verfahren. Dabei greift der Roboter die Reagenzgläser und positioniert diese neu. Befehl Nummer 7 trennt die Kommunikation zum Visualisierungs-PC. Der letzte Befehl wird auf dem Visualisierungs-PC ausgeführt und beendet die Visualisierung.

[6] Diese stehen in [5] zum Download bereit.

4.2. **Integration der Stationen**

Die mit Hilfe der Funktionen aus Abschnitt 3.4 modellierten Stationen des Dosierprozesses wurden in *Mat-Files* gespeichert. Dieses Vorgehen erspart die Neuberechnung der Stationen beim Programmstart und ermöglicht eine einfache Integration in die Visualisierung. Dabei ist zu beachten, dass die 5 Stationen als *Environment* übernommen werden. Die 15 im Dosierprozess verwendeten Reagenzgläser werden als *Parts* in die Visualisierung integriert und können durch den Roboter bewegt werden. Abbildung 16 zeigt den Ausgangszustand des Dosierprozesses in der Visualisierung. Ein *Matlab®-Skript* zur Erstellung der Stationen ist im Anhang A8 hinterlegt. Eine Besonderheit stellt die Roboterzelle da. Die Roboterzelle ist nicht in das zur Erstellung der Stationen verwendete *Matlab®*-Skript integriert und wurde in einer weiteren Projektarbeit [6] erstellt.

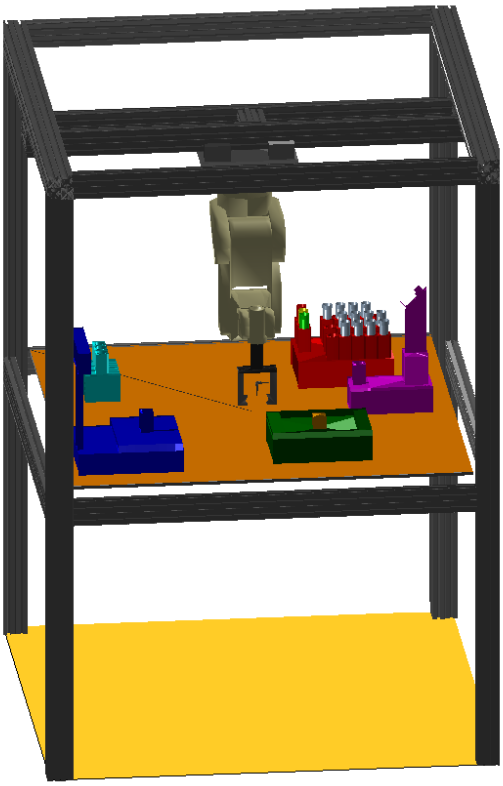


Abbildung 16: Dosierprozess in der Visualisierung

5. Die Steuerungssoftware des Dosierprozesses

In diesem Kapitel wird der Steuerungsaufbau beschrieben. Dabei wird auf das, in der Projektarbeit [1] entwickelte Steuerungskonzept des Dosierprozesses aufgebaut.

5.1. Grundlegender Aufbau

Der zugrunde liegende Aufbau der Steuerungssoftware besteht aus drei Komponenten, der *Steuerung*, dem *Prozess* und dem *Interface*. Abbildung 31 im Anhang A3 zeigt die *Steuerung* und den *Prozess*. Das *Interface* ist in den Prozess integriert. Die Steuerung erhält vom Prozess die aktuellen Prozesszustände. Diese werden durch die Steuerung ausgewertet und es werden Steuerwerte generiert und an den Prozess übermittelt. Abbildung 17 verdeutlicht dieses Vorgehen schematisch. Das Prozessinterface stellt dabei die Schnittstelle zwischen Prozess und Steuerung da und ist in den Prozess integriert.

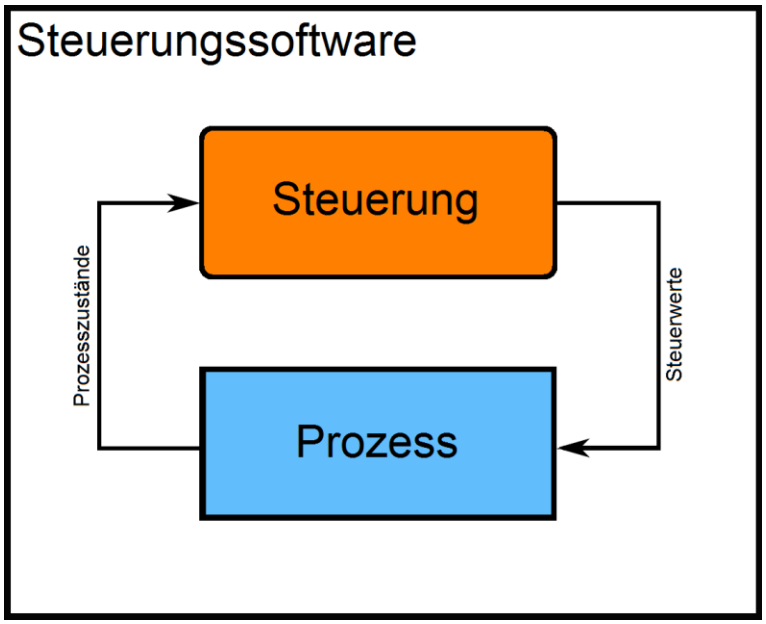


Abbildung 17: Schematischer Aufbau der Steuerungssoftware [1]

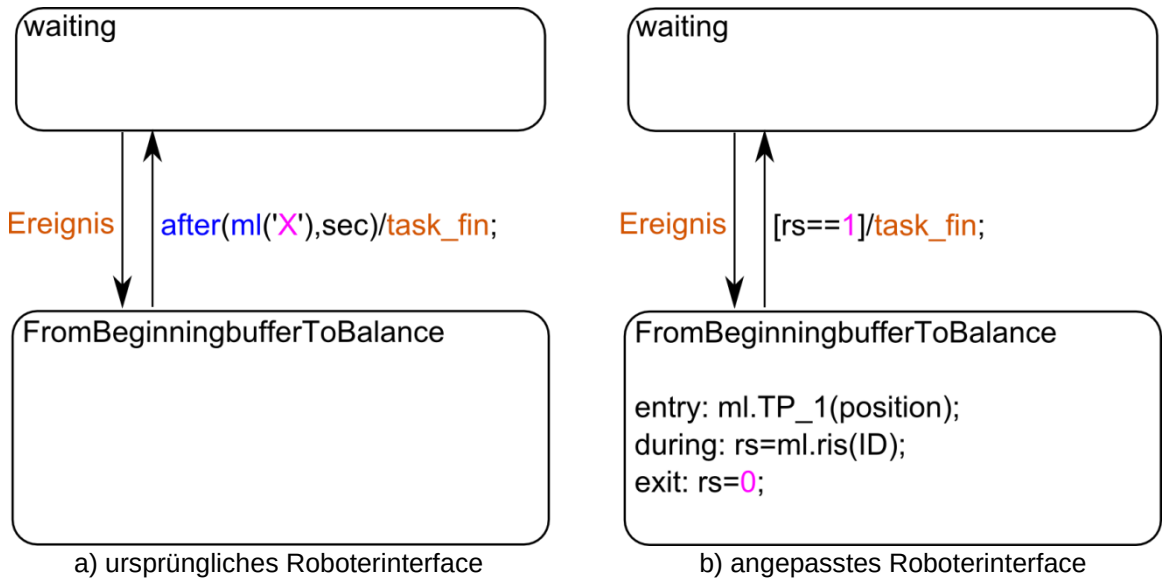
5.2. Anpassung der Steuerungssoftware an das CAR-System

Die in [1] entwickelte Steuerungssoftware wird um die Visualisierung des Dosierprozesses erweitert. Dafür ist es erforderlich das Roboterprozessinterface zu ändern. Im ursprünglichen Konzept wird die Bearbeitungsdauer eines Teilprozesses durch zufällig generierte Zeiten bestimmt (Abbildung 18a Position mit X markiert). Im Prozess mit dem virtuellen Roboter hingegen, hängt die Bearbeitungszeit nur von der Robotergeschwindigkeit und der Komplexität des Teilprozesses ab. Die ursprüngliche Steuerungssoftware unterteilt den Dosierprozess in 16 Teilaufgaben bzw. Teilpro-

zesse. Dieses Konzept wird für die neue Steuerungssoftware mit dem virtuellen Roboter übernommen.

Dafür ist es notwendig, die ermittelten Positionen, welche der Roboter anfahren kann, in eigene Funktionen zu integrieren. Diese entsprechen dann den zuvor genannten Teilprozessen. Tabelle 6 zeigt die Teilprozesse und die dazugehörende Funktion. Um dieses Vorgehen zu verdeutlichen folgt ein Beispiel.

Im Roboterprozessinterface gibt es einen Zustand *FromBeginningbufferToBalance.* Wird dieser betreten so wird auch die Funktion ID=TP_1(A_Pos) aufgerufen (Abbildung 18b). Die Steuerung übergibt an die Funktion die Nummer des Reagenzglases, welches vom Anfangspuffer zur Waage gebracht soll und der Roboter beginnt mit der Bearbeitung dieses Teilprozesses. Die Funktion generiert mit ihrem Aufruf eine Identifikationsnummer (ID). Diese wird genutzt, um den Status der Abarbeitung durch den Roboter zu überprüfen (during: rs[7]=ml.ris(ID)). Hierfür gibt es in der *KUKA-Kawasaki-Robotic Toolbox for Matlab®* eine eigene Funktion *ris*[8]. Ergibt die Überprüfung, dass die Teilaufgabe beendet ist (rs==1), so wird der Zustand wieder verlassen und ein Ereignis *task_fin* generiert. Abbildung 35 im Anhang A3 zeigt das komplette und durch die Funktionen aus Tabelle 6 erweiterte Roboterprozessinterface[9].



a) ursprüngliches Roboterinterface b) angepasstes Roboterinterface

Abbildung 18 ursprüngliches und angepasstes Roboterinterface (schematisch)

[7] Die Variable rs steht für RoboterStatus und wird beim Verlassen eines Zustandes auf 0 gesetzt.
[8] Die Funktion überprüft den Roboterstatus. Sie wird genutzt um einen Zustand erst dann zu verlassen, wenn der Roboter eine entsprechende Aktion beendet hat und dient somit der Synchronisierung der Steuerung und mit der Visualisierung.
[9] Mit Hilfe dieser 14 Funktionen lässt sich der gesamte Dosierprozess modellieren.

Tabelle 6: Funktionen der Teilprozesse zur Integration ins Prozessinterface

Aufgabe	Funktion
FromBeginningbufferToBalance	ID=TP_1(A_Pos) {A_Pos ϵ N | 1 ≤ A_Pos ≤ 12}
MoveToPicPose	ID=TP_2()
MoveAndDice	ID=TP_2_1()
FromColourbufferToEncompass	ID=TP_3(C_Pos) {C_Pos ϵ N | 1 ≤ C_Pos ≤ 3}
MoveToFilling	ID=TP_4()
Filling	ID=TP_5()
BackToEncompass	ID=TP_6()
FromBalanceToEndbufferAndPlaceIn	ID=TP_7(E_Pos) {E_Pos ϵ N | 1 ≤ E_Pos ≤ 3}
FromEncompassToColourbuffer	ID=TP_8(C_Pos) {C_Pos ϵ N | 1 ≤ C_Pos ≤ 3}
MoveToEndbufferAndTake	ID=TP_9(E_Pos) {E_Pos ϵ N | 1 ≤ E_Pos ≤ 3}
EmptyTubeAndMoveToBeginningbuffer	ID=TP_10(A_Pos) {A_Pos ϵ N | 1 ≤ A_Pos ≤ 12}
MoveToBeginningbufferAndPlaceIn	ID=TP_11(A_Pos) {A_Pos ϵ N | 1 ≤ A_Pos ≤ 12}
FromEncompassToBalance	ID=TP_12()
FromColourbufferToBalance	ID=TP_13(C_Pos) {C_Pos ϵ N | 1 ≤ C_Pos ≤ 3}
FromBalanceToColourbuffer	ID=TP_14(C_Pos) {C_Pos ϵ N | 1 ≤ C_Pos ≤ 3}
A_Pos.: Position im Anfangspuffer C_Pos.: Position im Farbpuffer E_Pos.: Position im Endpuffer	

6. Erweiterung der Steuerungssoftware um eine Kontrolloberfläche

Dieses Kapitel beschreibt die Erweiterung der Steuerungssoftware durch Kontrollelemente. Diese ermöglichen es, die Hintergrundprozesse wie z.B. das Würfeln zu veranschaulichen oder den aktuellen Füllstand darzustellen.

6.1. Visualisierung des Würfels

Der Würfel kann in der Visualisierung nicht gewürfelt werden, da keine Kinematik zur Generierung einer künstlichen Schwerkraft vorgesehen ist. Stattdessen wird in der Simulation ein Zufallsgenerator zur Erzeugung der benötigten Zufallszahlen verwendet. Um die Augenzahl darzustellen, wurde ein künstlicher Würfel integriert. Dieser stellt den aktuellen Würfelwert optisch dar (Abbildung 19). Damit lässt sich die spätere Befüllung der Reagenzgläser durch den Betrachter nachvollziehen und überwachen. Die Aktualisierung der gewürfelten Augenzahl erfolgt bei dem Zugriff auf die Funktion *MakePicture* durch die Prozesssteuerung.

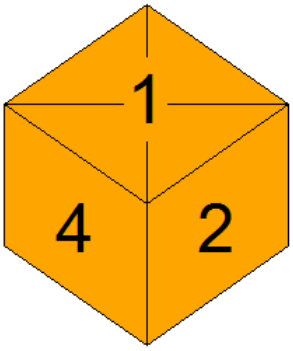


Abbildung 19: Visualisierung des Würfels

6.2. **Visualisierung des Displays der Waage**

Abbildung 20 zeigt das Display der Waage in der Visualisierung. Das virtuelle Display verhält sich analog zu einer realen Waage und wird nach dem Einsetzen eines Reagenzglases in die Waage auf 0 Gramm tariert. Somit ist für einen kurzen Moment die Füllmenge des aktuellen Reagenzglases auf der Waage zu sehen, bevor diese tariert wird (in der Funktion *tara*). Zusammen mit dem zuvor erwähnten Würfel lässt sich der Prozess durch den Nutzer überwachen. Bei der Entnahme des Reagenzglases von der Waage wird diese erneut tariert, damit sie keine negativen Werte anzeigt.

Abbildung 20: Visualisierung des Displays der Waage

6.3. **Visualisierung der Füllzustände der Reagenzgläser**

Abbildung 21 zeigt das 3 Kontrollelement. Hier wird der jeweilige Füllstand aller Reagenzgläser gezeigt. Weiterhin ist eine eventuell vorhandene Vorbefüllung und der jeweilige *JobTyp*[10] zu erkennen. Die Anzeige ist mit dem Prozess synchronisiert und wird erst beim Einfüllen von Granulat durch die Funktion *Weighing* oder Ausleeren eines Reagenzglases durch die Funktion *EmptyTubeAndMoveToBeginningbuffer*[11] aktualisiert.

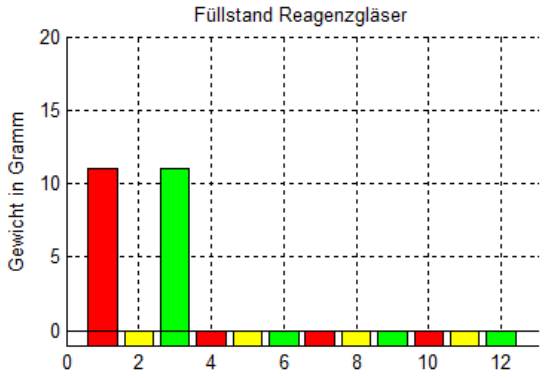


Abbildung 21: Visualisierung der Füllstände der Reagenzgläser

[10] Rot oder Gelb oder Grün
[11] Siehe Tabelle 6

7. Funktionsnachweis der Steuerungssoftware mit dem CAR-System

In diesem Kapitel wird der Funktionsnachweis der Steuerungssoftware in Kombination mit dem CAR-System der FG untersucht. Dabei auftretende Probleme werden auf ihre Ursache untersucht und es werden Lösungen erarbeitet.

7.1. Probleme bei der Simulation mit dem CAR-System

Bei ersten Tests mit dem virtuellen Roboter fiel auf, dass dieser sich von seinem realen Vorbild leicht unterscheidet. So befinden sich die Greiferbacken nicht im *ToolCenterPoint* (TCP) und sind um 13mm versetzt (Abbildung 22). Des Weiteren ist der Greifer des realen Roboters länger als beim virtuellen Roboter.

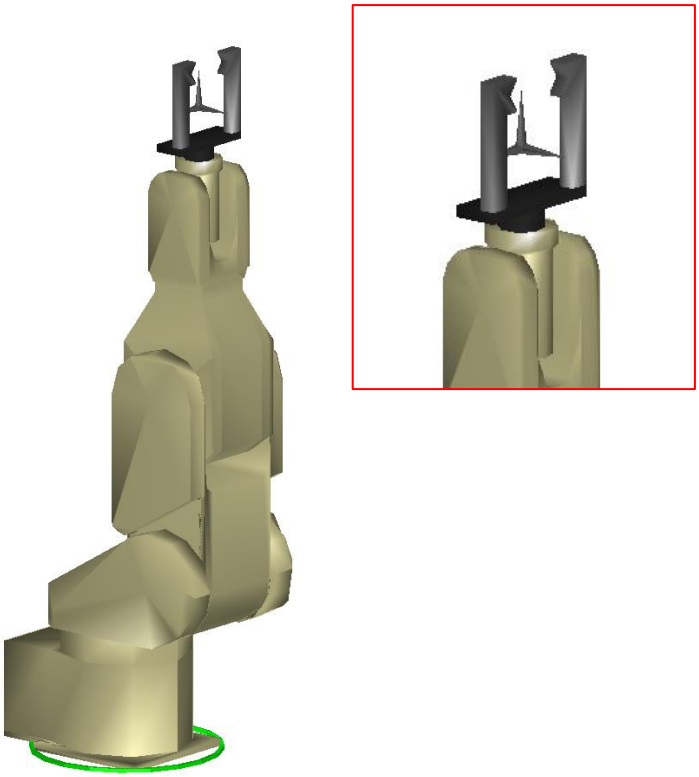


Abbildung 22: ursprünglicher Roboter

Ein weiteres Problem zeigt sich beim Anfahren des Endpuffers. Der reale Roboter kann eine Position in der *below* oder in die *above* Stellung[12] anfahren. Dies ist beim virtuellen Roboter bislang nicht möglich. Die benötigten Positionen der Reagenzgläser im Endpuffer lassen sich ohne die Umschaltung der Stellung nicht anfahren.

[12] Dies ist eine Funktionalität die in den realen Roboter zur Verfügung steht und bislang nicht in die *KUKA-KAWASAKI Robotic Toolbox for Matlab*® integriert ist.

Erste Tests die Stationen mit dem Roboter anzufahren, erfolgen mit den in [1] beschrieben Roboterkoordinaten. Dabei zeigt sich eine schlechte Positionierungsgenauigkeit bei einigen Stationen. Dies ist z.B. bei der Waage von Bedeutung. Diese ist zwischenzeitlich für andere Versuche auf dem Tisch umgestellt worden, sodass sich ihre ursprüngliche Position nicht genau rekonstruieren lässt. Somit ist eine individuelle Anpassung der in [1] ermittelten Koordinaten notwendig.
Ebenso muss die Steuerungssoftware an die Integration CAR-Systems der FG-CEA angepasst werden. Eine Dokumentation der Änderungen an der Steuerungssoftware ist im Anhang A3 - A7 zu finden.

7.2. **Problembehebung**

Um den realen Roboter besser zu entsprechen wird der virtuelle Roboter durch eine veränderte Variante ersetzt. Diese ist in Abbildung 23 dargestellt. Dabei wird der Anschlussflansch des Greifers verlängert und die Greiferbacken werden um 13mm versetz.

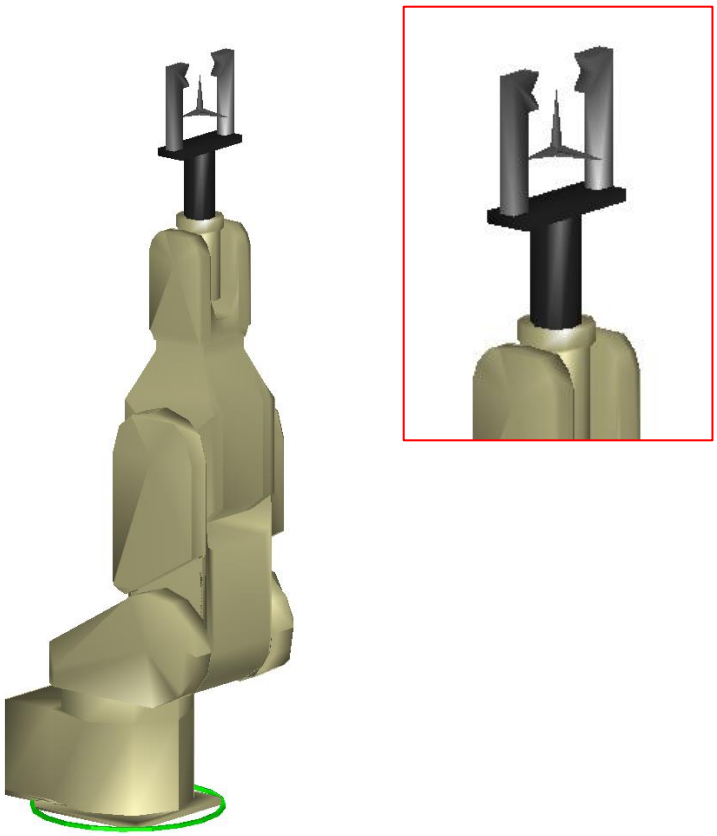


Abbildung 23: aktualisierter Roboter

Da sich der Endpuffer im virtuellen Versuchsaufbau nicht durch den Roboter anfahren lässt, muss dieser neu platziert werden. Abbildung 24 zeigt die neue Position des Endpuffers. Die Stationen wurden bezogen auf ihren Flächenschwerpunkt bemaßt. Der Koordinatenursprung (KOS) befindet sich im Flächenschwerpunkt des Tisches[13].

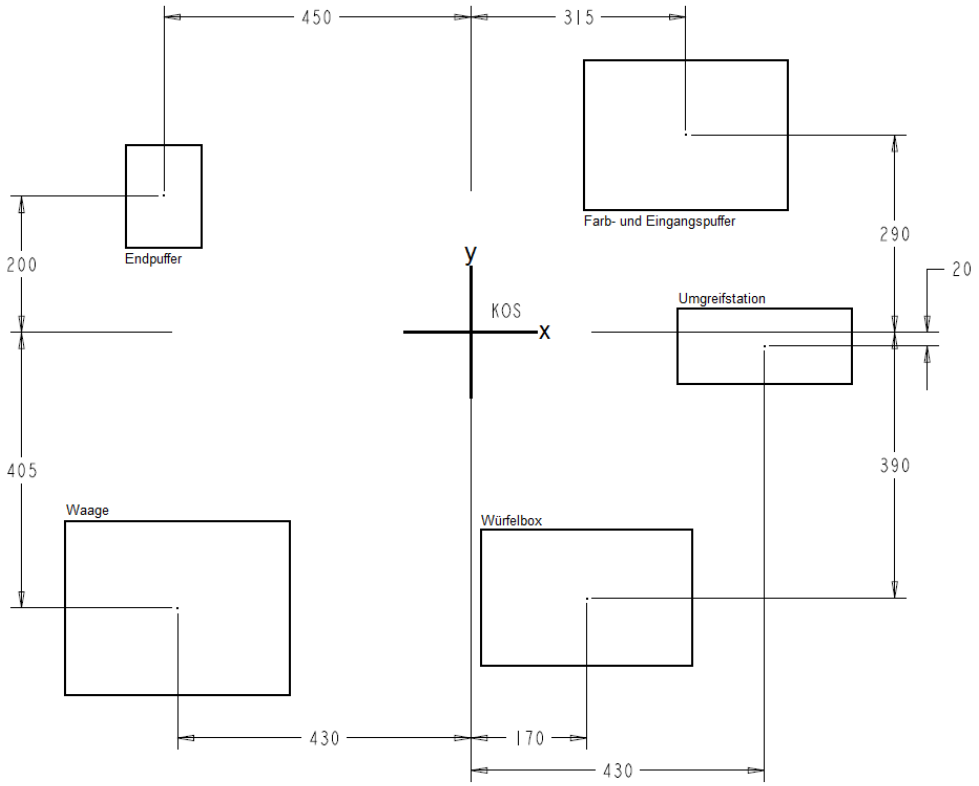


Abbildung 24: Layout der Stationen in der Visualisierung

[13] Durch diese Art der Bemaßung lassen sich die Positionen der Stationen in der Visualisierung leichter identifizieren. Die in Abbildung 24 dargestellten X- und Y-Werte sind im Quellcode des Matlab®-Skriptes A3_V-Pc (Anhang 0) zu sehen.

8. Zusammenfassung

Im Rahmen der Arbeit sollte die virtuelle Umsetzung einer vorhandenen Robotersteuerung eines Dosierprozesses mit dem *Computer Aided Robotics* (CAR) System der Forschungsgruppe (FG) *Computational Engineering and Automation* (CEA) untersucht werden. Dabei war die von der Forschungsgruppe entwickelte *KUKA-KAWASAKI Robotic Toolbox for Matlab*® und die *KUKA-KAWASAKI Visualization Toolbox for Matlab*® zu verwenden. Die Stationen des Dosierprozess wurden geometrisch erfasst und mit Hilfe eines CAD Programmes nachgebildet. Dies diente der anschließenden Modellierung der Stationen mit *Surf-Objekten* und der Integration in das *CAR-System*. Danach erfolgte ein Einteachen eines virtuellen Roboters für den virtuellen Dosierprozess. Die ermittelten Koordinaten wurden in die Steuerung integriert, wobei auftretende Probleme analysiert und behoben wurden. Abschließend erfolgte die Integration virtueller Kontrollelemente für die Waage, den Würfel und die 12 Reagenzgläser des Eingangspuffers des Dosierprozesses. Diese gestatten dem Benutzer die Überwachung des Dosierprozesses während einer Simulation der Steuerungsvorgänge. Die Funktionsfähigkeit der Robotersteuerung konnte mit dem CAR System in der Simulation nachgewiesen werden. Dabei zeigten sich geringe Unterschiede im Bewegungsverhalten des virtuellen Roboters zum realen Roboter. Bei der Anwendung der virtuellen Steuerung auf den realen Dosierprozess, sind die eingeteachten Roboterkoordinaten zu überprüfen.

Literaturverzeichnis

[1] A. Schmidt, „Entwicklung einer kooperierenden Roboterapplikation auf Basis des RCP-Ansatzes in der Matlab/Simulink-Umgebung,“ Projektarbeit, Wismar, 2011.

[2] M. Christern, A. Schmidt, T. Schwatinski und T. Pawletta, „KUKA-KAWASAKI-Robotic Toolbox for Matlab®,“ April 2013. [Online]. Available: http://www.mb.hs-wismar.de/cea/KK_Robotic_Tbx/KK_Robotic_Tbx.html.

[3] M. Christern und A. Schmidt, „Welcome to the KK-Robotic-Toolbox online documentation,“ April 2013. [Online]. Available: http://www.mb.hs-wismar.de/cea/KK_Robotic_Tbx/docu_html/rtb.html.

[4] T. Schwatinsky, T. Pawletta und J. Otto, „KUKA-KAWASAKI-Visualization Toolbox for Matlab®,“ April 2013. [Online]. Available: http://www.mb.hs-wismar.de/cea/MatlabKK_Robotic-and-Visualization_Tbx/MatlabKK_Robotic_Visualization_Tbx.html.

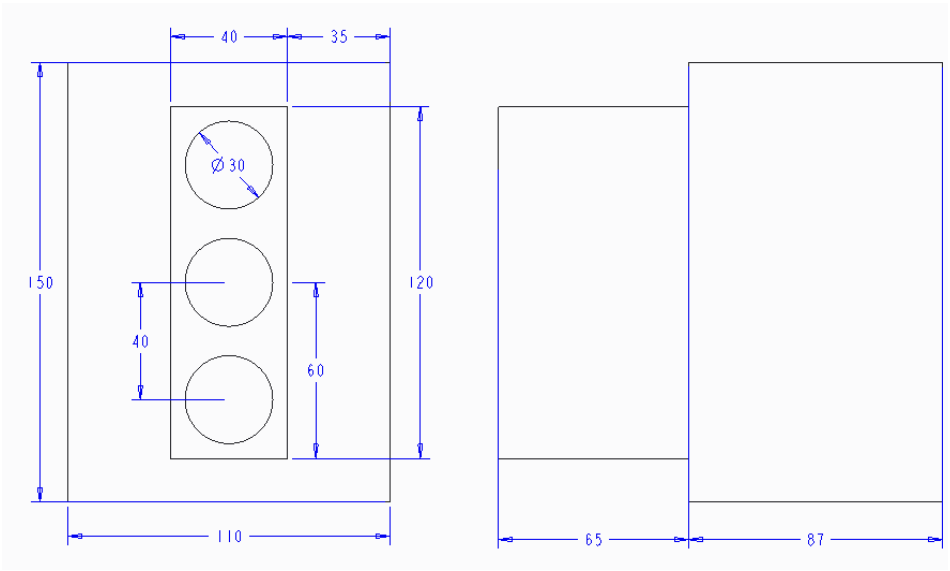
[5] T. Schwartinski, „KK-Robotik-Tollbox for Matlab,“ April 2013. [Online]. Available: http://www.mb.hs-wismar.de/cea/MatlabKK_Robotic-and-Visualization_Tbx/docu/fulldocu.html.

[6] T. Ebert, „Durchgängige Entwicklung von einfachen Robotersteuerungen im Computer Aided Robotics (CAR) System der Foschungsgruppe Computational Engineering and Automation (FG CEA) der Hochschule Wismar,“ 2013.

Anlagen

A1: Technische Zeichnungen der 5 Stationen

Endpuffer



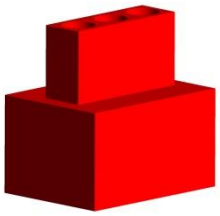
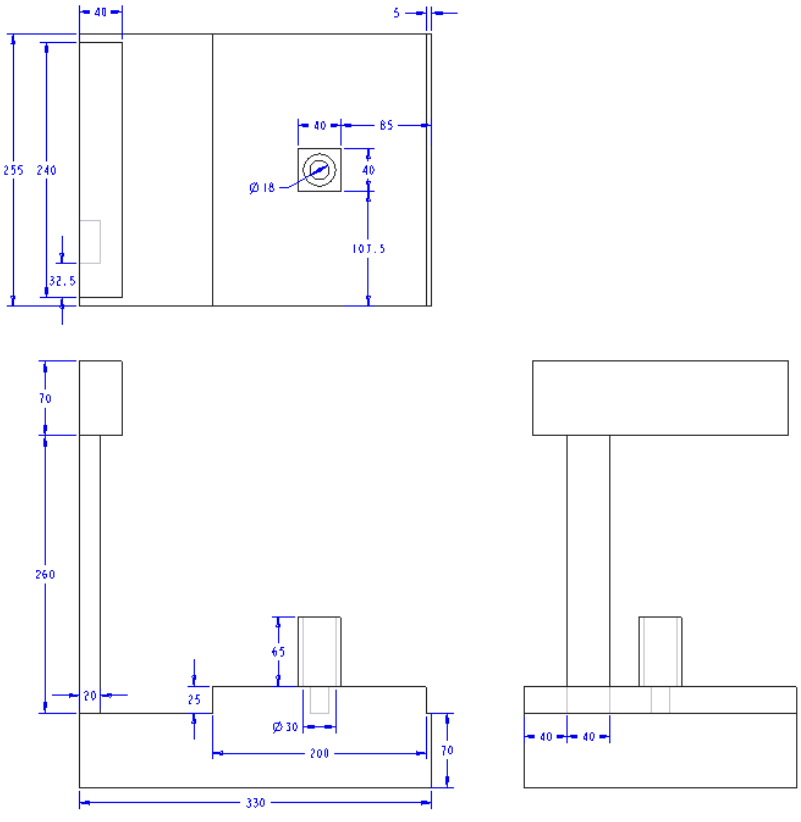


Abbildung 25: Endpuffer mit Bemaßung

Waage



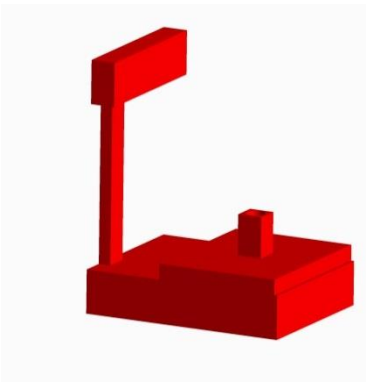
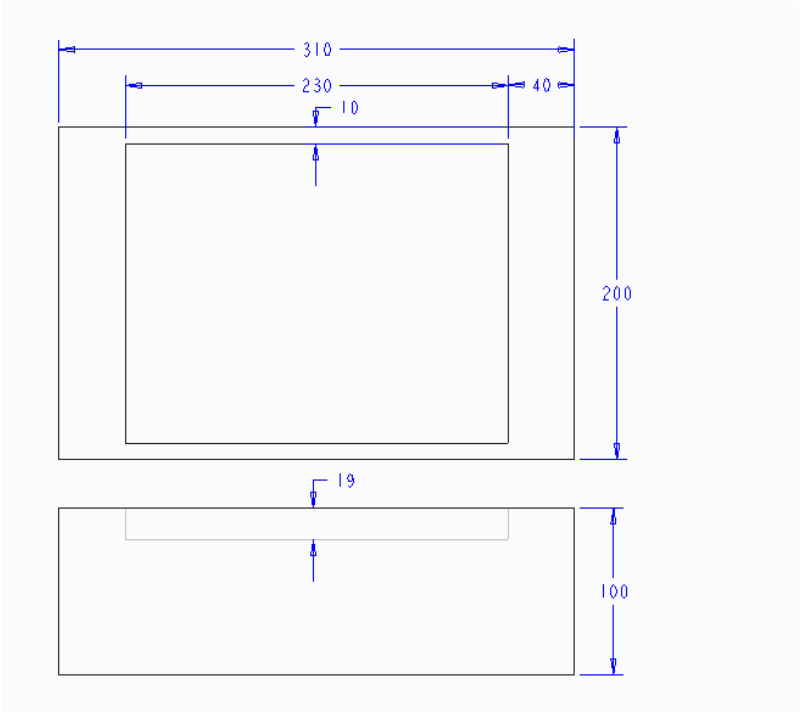


Abbildung 26: Waage mit Bemaßung

Würfelbox



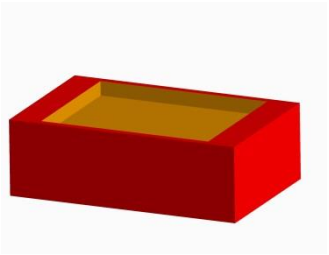
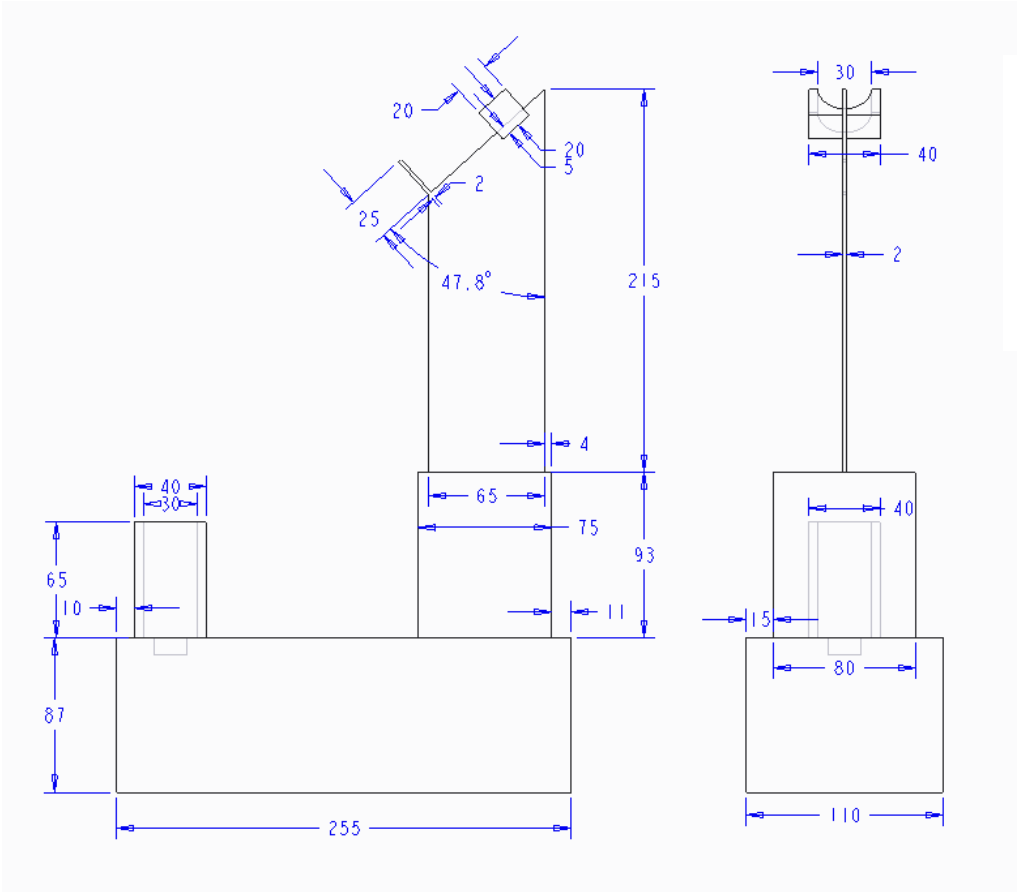


Abbildung 27: Würfelbox mit Bemaßung

Umgreifstation



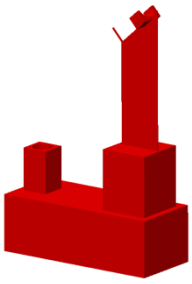
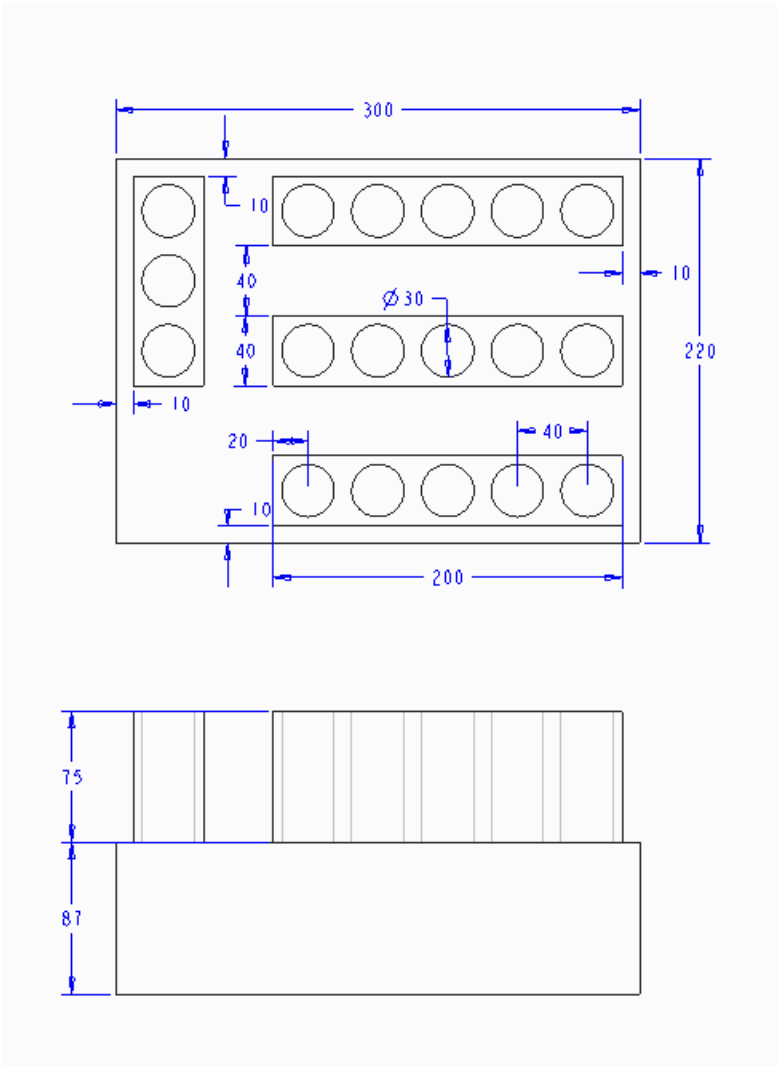


Abbildung 28: Umgreifstation mit Bemaßung

Farb- und Eingangspuffer



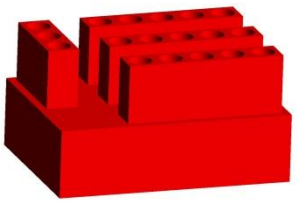


Abbildung 29: Farb- und Eingangspuffer mit Bemaßung

A2: Anleitung: Start der Laborautomatisierung

Auf der beigefügten CD befindet sich ein Ordner *Laborautomatisierung* (Abbildung 30). In diesem Ordner gibt es zwei Unterordner. Der Ordner *Control_PC* beinhaltet die Dateien für den Steuerungsrechner. Im Ordner *Visualisierung_PC* sind alle nötigen Dateien für den Visualisierungsrechner hinterlegt. Des Weiterem befindet sich in diesem Ordner die Datei *Modell_Kawasaki.mat*. Diese dient zur Aktualisierung des Roboters in der installierten *KUKA-KAWASAKI-Visualization Toolbox for Matlab®* und muss die dort bereits vorhandene Datei ersetzen.

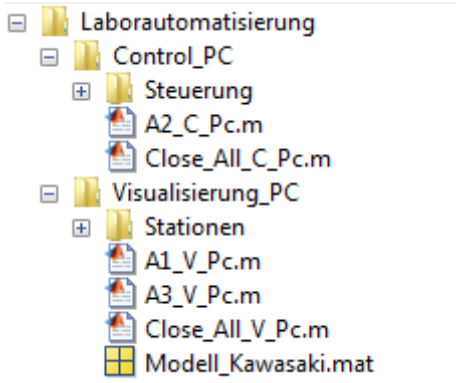


Abbildung 30: Ordnerstruktur Laborautomatisierung

Tabelle 7 zeigt die nötigen Schritte zum Start der Visualisierung und Steuerung auf dem Visualisierungs- und Kontrollrechner.

Tabelle 7: Startabfolge der Laborautomatisierung

Schritt	Visualisierung-PC	Kontroll-PC
1	A1_V_Pc.m	---
2	---	A2_C_Pc.m
3	A3_V_Pc.m	---
4	---	Starten der Prozess-Steuerung
	Beenden der Simulation:	
5	---	Stoppen der Prozess-Steuerung
6	Close_All_C_Pc.m	Close_All_V_Pc.m

A3: Prozess und Steuerung in Matlab®

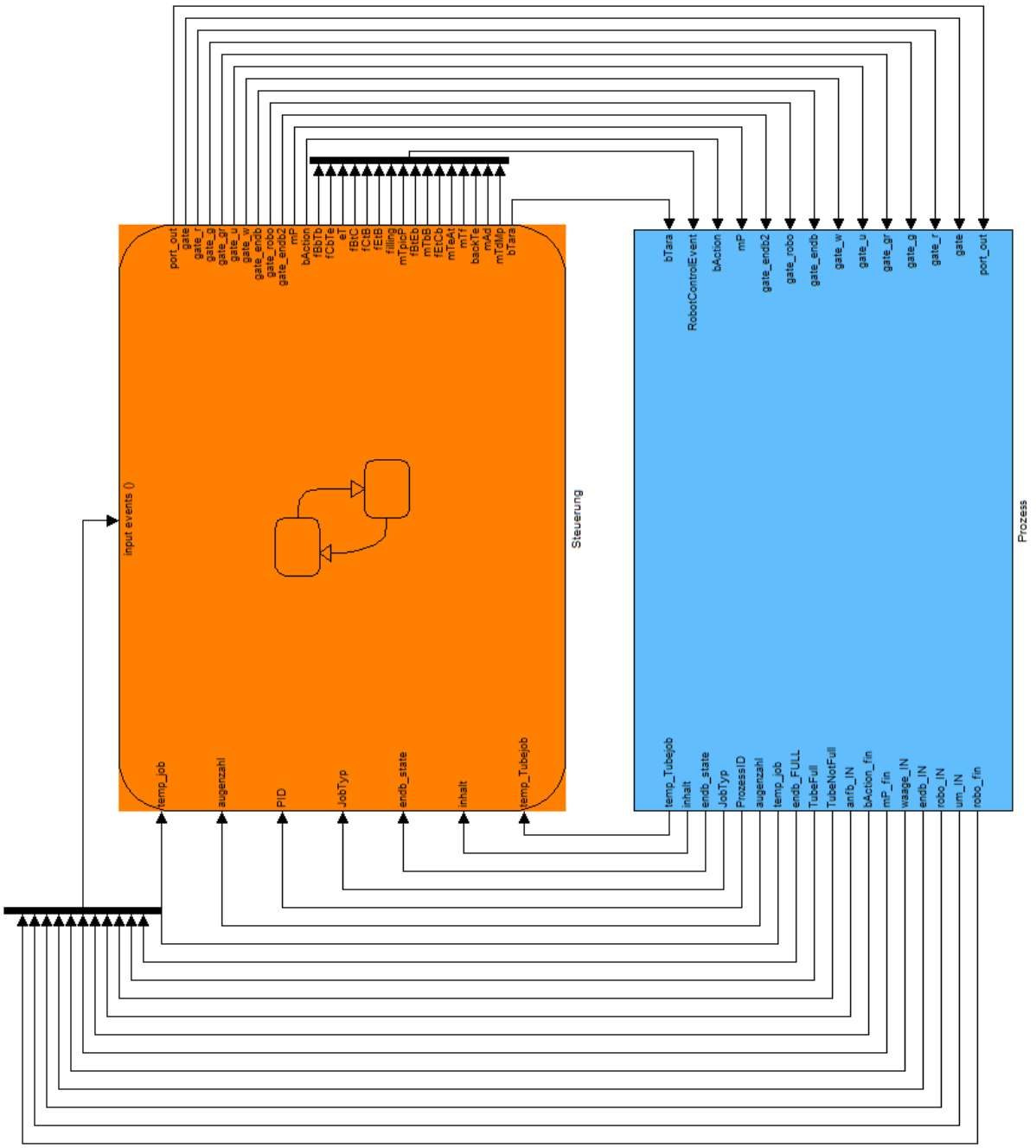


Abbildung 31: Prozess und Steuerung des Dosierprozesses

Das Prozessinterface ist in den Prozess integriert. Der Prozess wird in Abbildung 32 in geöffneter Form dargestellt. Im unteren Bildbereich ist das *ProzessInterface* zu sehen. Abbildung 33 zeigt die erste Ebene des geöffneten Prozessinterfaces. Abbildung 34 veranschaulicht die 2. Ebene des Prozessinterfaces. Dabei ist das *Roboter-Interface*, das *BalanceInterface* und *WCamInterface* zu sehen. Abbildung 35 zeigt die in das *RoboterInface* integrierten Teilprozesse des Dosierprozesses.

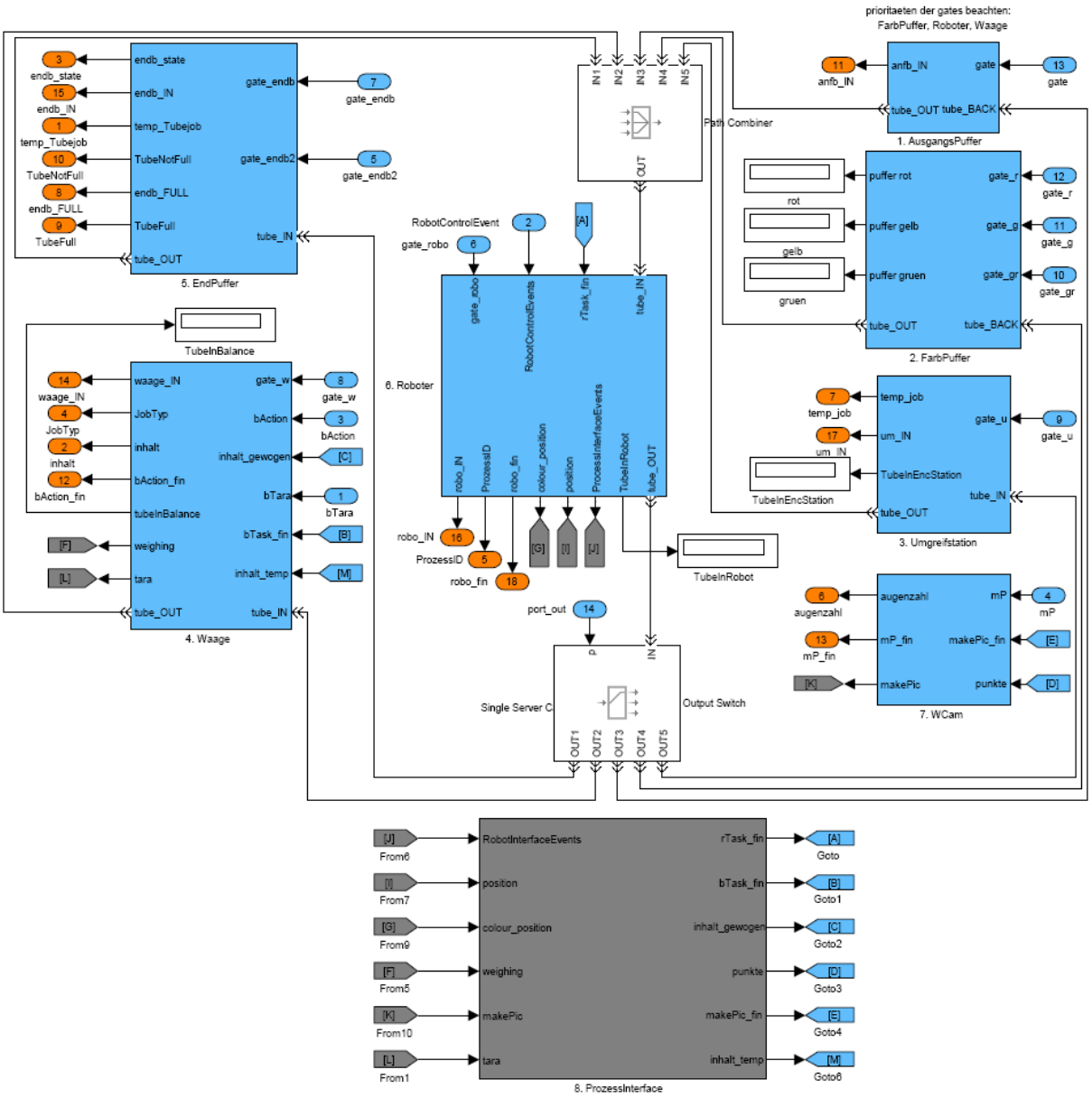


Abbildung 32: Prozess mit Prozessinterface

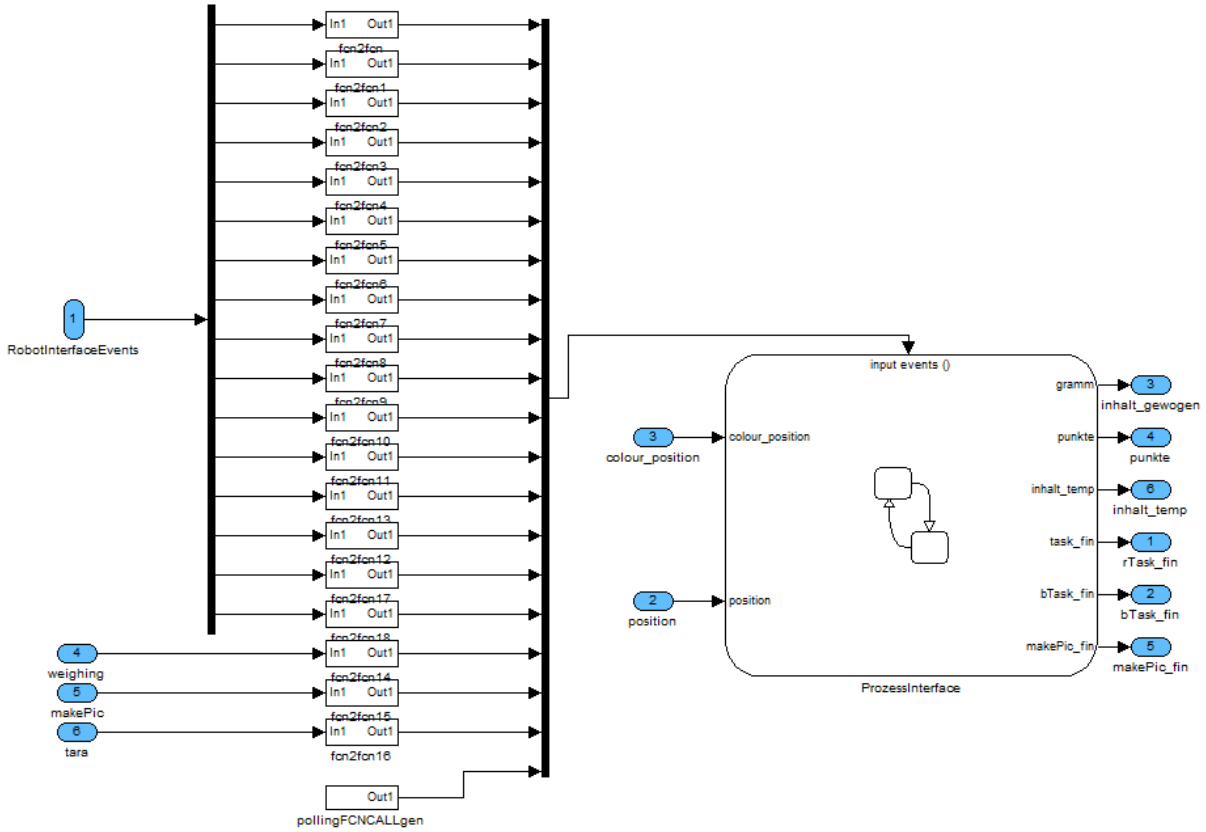


Abbildung 33: Prozessinterface 1. Ebene

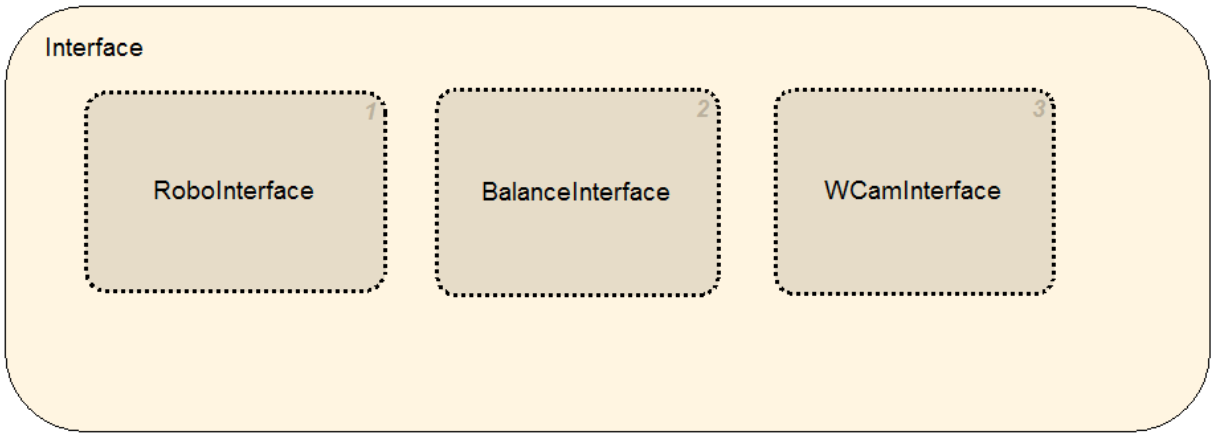


Abbildung 34: Prozessinterface 2. Ebene

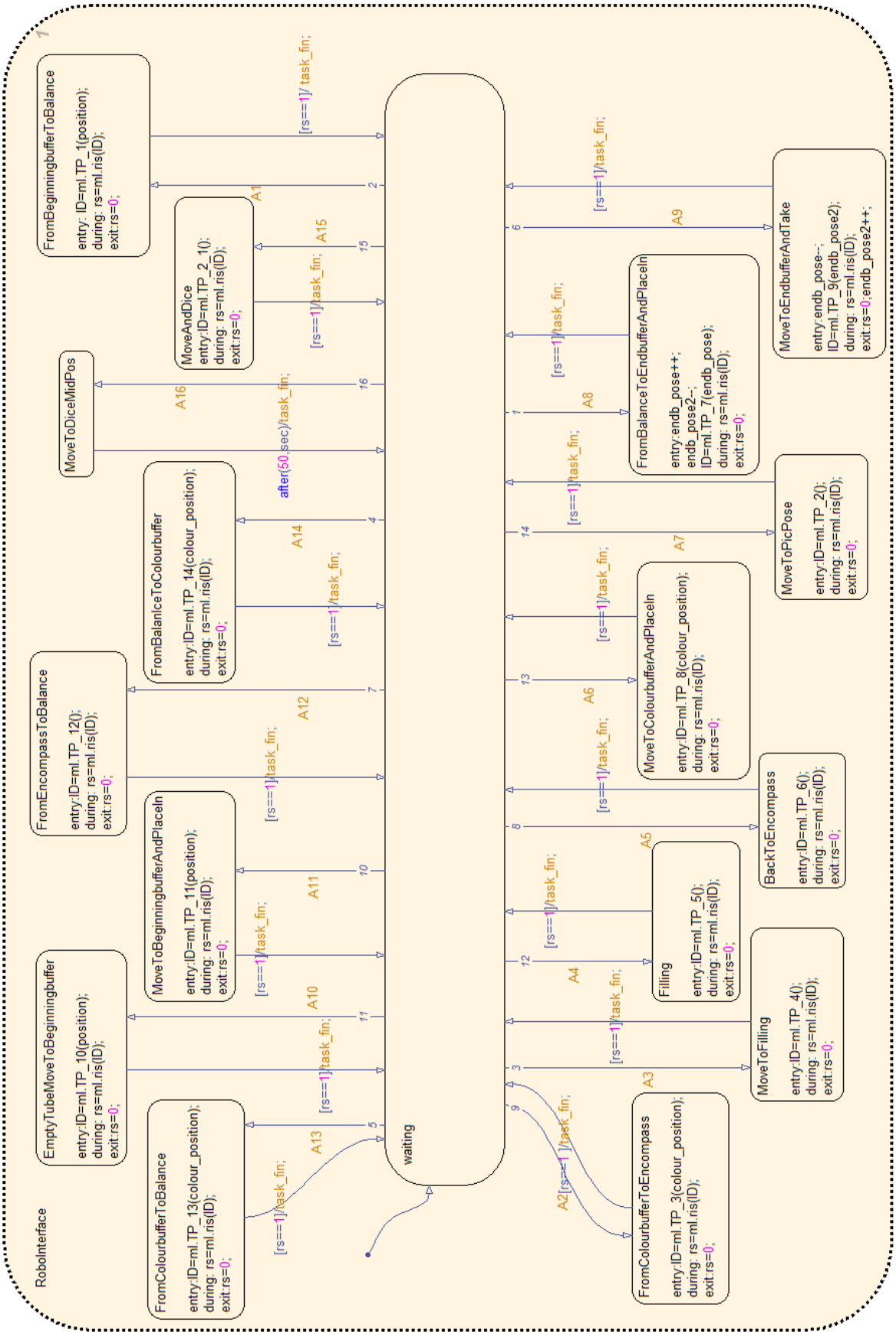


Abbildung 35: Roboterprozessinterface mit integrierten Teilprozessen

A4: Steuerung des Endpuffers

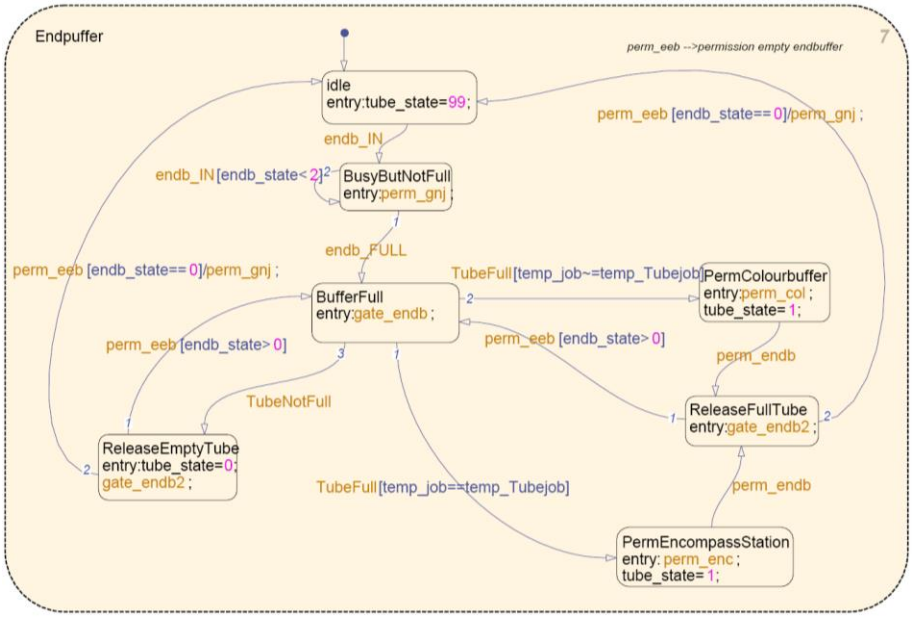


Abbildung 36: Steuerung des Endpuffers in der ursprünglichen Variante

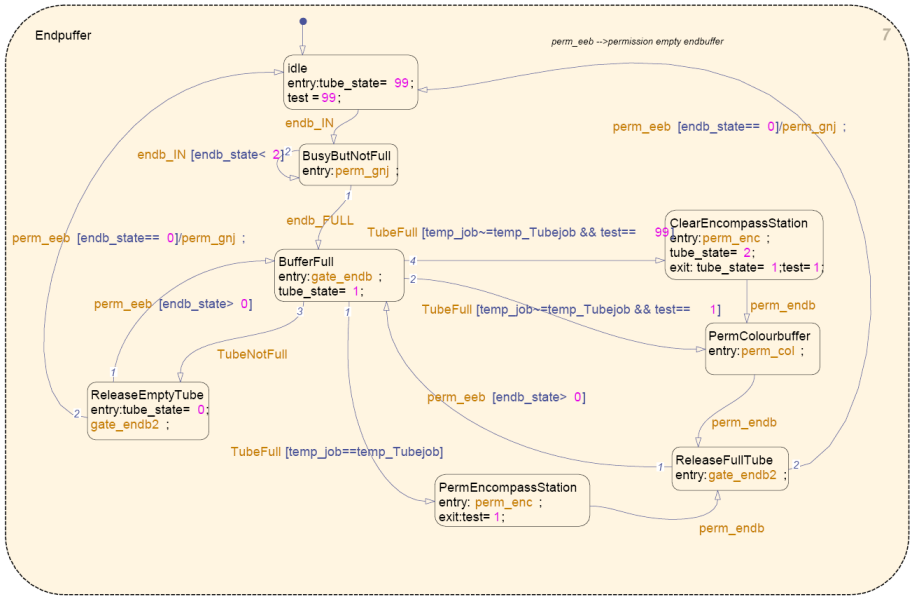


Abbildung 37: Steuerung des Endpuffers in der angepassten Variante

Liegt auf der Umgreifstation ein Farbreagenzglas mit falschem *JobTyp* (Rot, Gelb, Grün), so wird dieses zuerst in den Farbpuffer zurückgestellt (*ClearEncompassStation*). *Test=1* wird gesetzt, sodass dieser Zustand nicht wieder betreten wird. Danach wird aus den Farbenpuffer das passende Farbreagenzglas entnommen und zur Waage gebracht. Das Reagenzglas wird mit Hilfe der Umgreifstation geleert und anschließend zum Eingangspuffer zurückgebracht. Ohne die vorherige Kontrolle der Umgreifstation wurde es zur Kollision zwischen verbliebenen Farbreagenzglas und zu leerenden Reagenzglas kommen.

A5: Prozess des Roboters

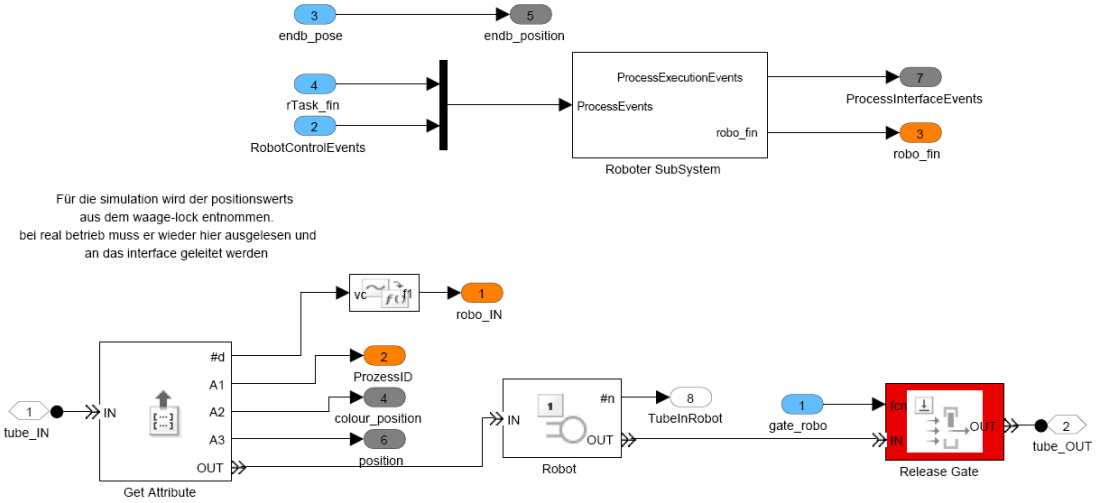


Abbildung 38: Prozess des Roboters in der ursprünglichen Variante

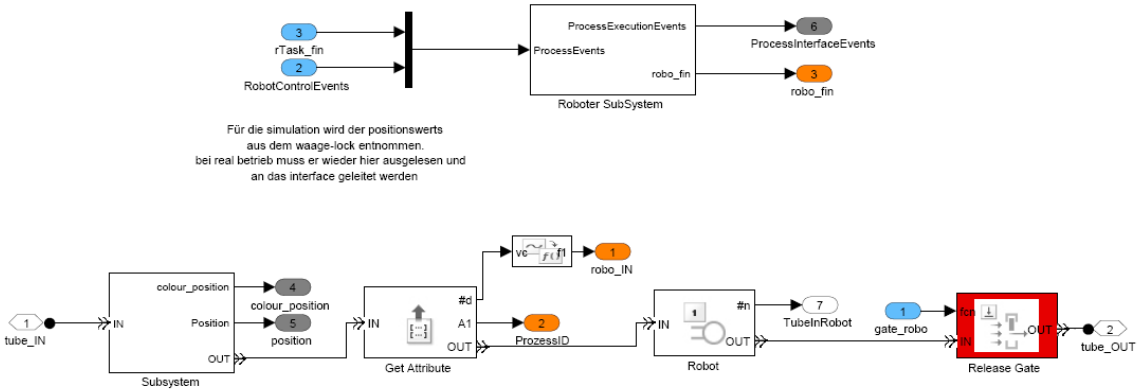


Abbildung 39: Steuerung des Roboters in der angepassten Variante

A6: Prozess der Umgreifstation

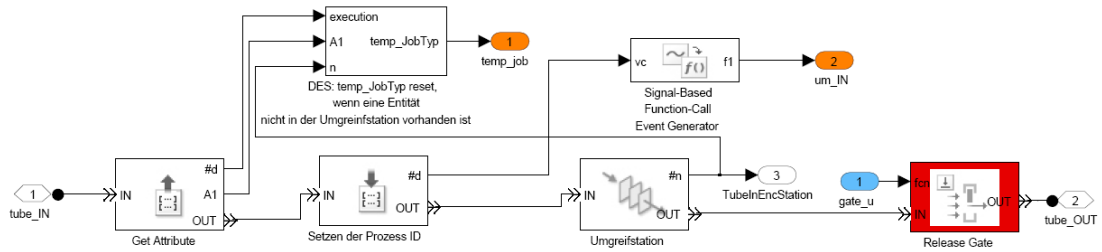


Abbildung 40: Prozess der Umgreifstation in der ursprünglichen Variante

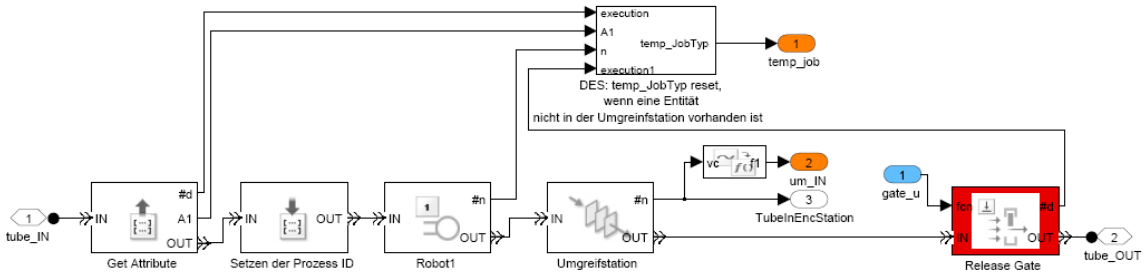


Abbildung 41: Prozess der Umgreifstation in der angepassten Variante

Dem Block der Umgreifstation ist ein Nullserver vorangestellt. Dieser verzögert die Entität um 0 Sekunden damit an die Steuerung das Signal *JobTemp* übermittelt wird.

A7: Steuerung des Roboters

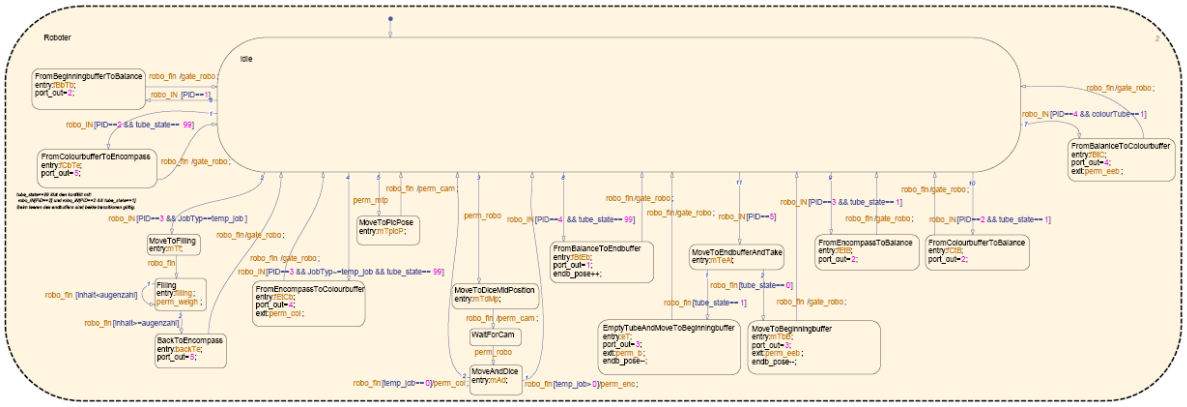


Abbildung 42: Steuerung des Roboters in der ursprünglichen Variante

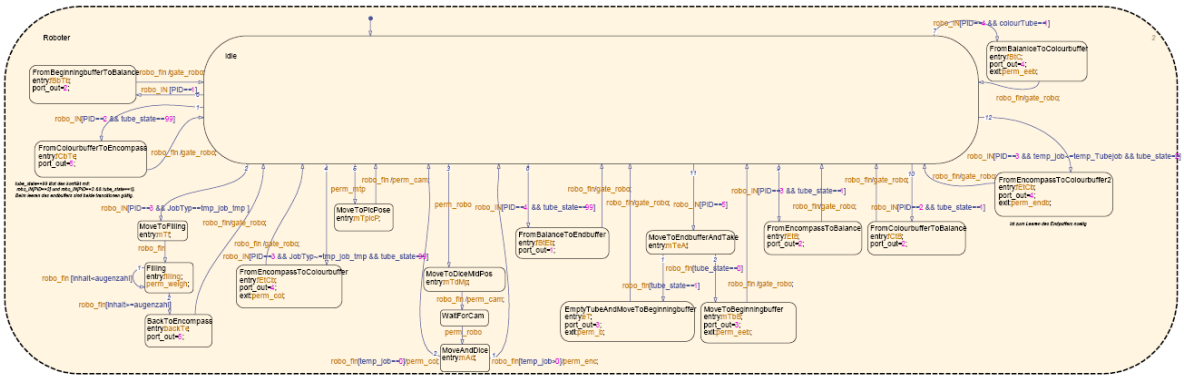


Abbildung 43: Steuerung des Roboters in der angepassten Variante

Vergleicht man die Abbildung 42 mit der Abbildung 43, wird ersichtlich, dass die Steuerung des Roboters um einen weiteren Zustand ergänzt ist. Diese Zustand (*FromEncompassToColourpuffer*) ist erforderlich um ein volles Reagenzglas zu entleeren, wenn sich auf der Umgreifstation ein Farbreagenzglas mit einem anderen JobTyp befindet. Andernfalls wurde es bei der Entleerung zu einer Kollision eines Reagenzglases mit einem Farbreagenzglas kommen.

A8: Quellcode der Funktion zum Erstellen der Stationen

```
%% Station aus Surf-Objekten modellieren

function zmain()
A_gen = input('Baugruppen speichern ? (j/n) ','s');
scrsz=get(0,'ScreenSize');
f1=figure;
set(f1,'Position',[.1*scrsz(3),.1*scrsz(4),.8*scrsz(3),.8*scrsz(4)]);
%set(f1, 'Position',scrsz);
colormap('gray')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%table
[x,y,z]=table;
[x,y,z]=cm2mm(x,y,z);
if A_gen =='j'
save('table.mat','x','y','z');
end
surf(x,y,z)
axis equal
hold on

%input_puffer
[x,y,z]=i_puffer;
[x,y,z]=cm2mm(x,y,z);
if A_gen =='j'
save('i_puffer.mat','x','y','z');
end
surf(x+940,y+915,z)

%libra
[x,y,z]=libra;
[x,y,z]=cm2mm(x,y,z);
if A_gen =='j'
save('libra.mat','x','y','z');
end
surf(x+195,y+220,z)

%output_puffer
[x,y,z]=o_puffer;
[x,y,z]=cm2mm(x,y,z);
if A_gen =='j'
save('o_puffer.mat','x','y','z');
end
surf(x+190,y+645,z)

%cube_station
[x,y,z]=cube_station;
[x,y,z]=cm2mm(x,y,z);
if A_gen =='j'
save('cube_station','x','y','z');
end
surf(x+795,y+235,z)

%cross_station
[x,y,z]=cross_station;
[x,y,z]=cm2mm(x,y,z);
if A_gen =='j'
save('cross_station.mat','x','y','z');
end
surf(x+1055,y+605,z)
```

```
%test_tubes
[x,y,z]=test_tube;
[x,y,z]=cm2mm(x,y,z);
if A_gen =='j'
save('test_tube.mat','x','y','z');
end
surf(x+820,y+995,z+91)
surf(x+820,y+955,z+91)
surf(x+820,y+915,z+91)

surf(x+900,y+995,z+91);surf(x+940,y+995,z+91);surf(x+980,y+995,z+91);
surf(x+1020,y+995,z+91);surf(x+1060,y+995,z+91);surf(x+900,y+915,z+91);
surf(x+940,y+915,z+91);surf(x+980,y+915,z+91);surf(x+1020,y+915,z+91);
surf(x+1060,y+915,z+91);surf(x+900,y+835,z+91);surf(x+940,y+835,z+91);
surf(x+980,y+835,z+91);surf(x+1020,y+835,z+91);surf(x+1060,y+835,z+91)

%Würfel
[x,y,z]=quad(40,40,81,121);
if A_gen =='j'
save('Wuerfel.mat','x','y','z');
end
[x,y,z]=rot(x,y,z,0,0,30);
surf(x+795,y+235,z)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
grid
view(16,16)
%Hintergrund rot einfaerben
set(gca,'Color','red')
rotate3d on
xlabel('X-Achse')
ylabel('Y-Achse')
zlabel('Z-Achse')
end

%% Grundfunktion: Konvertieren cm:mm
function [x,y,z]=cm2mm(x,y,z)
x=x*10;
y=y*10;
z=z*10;
end
```

```
%% Grundfunktion: Quader mit zentrischer Bohrung

function [x,y,z]= klotzml(a,b,ri,Hoehe,Boden,n)
%INFO:
%Koordinatenursprung im Zentrum des Körpers
%Bohrung mit Radius "ri" im X,Y-Zentrum des Quaders
%Anzahl der Segmente n bestimmt Genauigkeit (Achtung: Datenmenge steigt!)
%Bedingung: n=2*N; n>=8

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%           _____________                                              %%%
%%%          /             /|        (X)Makiert Bohrung                  %%%
%%%         /     (X)     / |                                            %%%
%%%        /____________ /  |         _ _ Hoehe                          %%%
%%%       |             |   |          |               KOS:              %%%
%%%       |             |   |   _ _    |               z   y             %%%
%%%       |             |  /    /      |               |  /              %%%
%%%       |             | /0   /b      |               | /               %%%
%%%      0|_____________|/   _/_      _|_ Boden        |/____x           %%%
%%%              0                                                       %%%
%%%       |------------|                                                 %%%
%%%             a                                                        %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

i=360/n;
a=a/2;
b=b/2;

%Hilfskontur um Koordinaten für äußeres Quadrat zu berechnen
r=[a*ones(1,n/8),a:-a/(n/8):-a,-a*ones(1,n/4-1),-
a:a/(n/8):a,a*ones(1,n/8)]';
k=[0:b/(n/8):b,b*ones(1,n/4-1),b:-b/(n/8):-b,-b*ones(1,n/4-1),-
b:b/(n/8):0]';

x=[ri*cosd(0:i:360)',r,r,ri*cosd(0:i:360)',ri*cosd(0:i:360)'];
x=[x;[ri,ri,ri,ri,ri]];

y=[ri*sind(0:i:360)',k,k,ri*sind(0:i:360)',ri*sind(0:i:360)'];
y=[y;zeros(1,5)];

z=ones(n+1,1)*[Boden,Boden,Hoehe,Hoehe,Boden];
z=[z;ones(1,5)*Boden];

%Schnittstelle um Konturlinien in einen Punkt zusammenzufassen
x=[[ri,ri,ri,ri,ri];x;[ri,ri,ri,ri,ri]];
y=[zeros(1,5);y;zeros(1,5)];
z=[ones(1,5)*Boden;z;ones(1,5)*Boden];
end
```

```
%% Grundfunktion: Quader

function [x,y,z]=quad(x,y,Hoehe,Boden)
%INFO:
%Koordinatenursprung im Zentrum des Körpers

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%          _____________                                              %%%
%%%         /            /|                                             %%%
%%%        /            / |                                             %%%
%%%       /___________ /  |          _ _ Hoehe                          %%%
%%%      |            |   |           |            KOS:                 %%%
%%%      |            |   |    _ _    |            z   y                %%%
%%%      |            |  /    /       |            |  /                 %%%
%%%      |            | / 0  /b       |            | /                  %%%
%%%      |____________|/   _/_       _|_ Boden     |/____x              %%%
%%%             0                                                       %%%
%%%      |------------|                                                 %%%
%%%            a                                                        %%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

x=x/2;
y=y/2;

x=[0, x,  x, 0, 0;
   0,-x, -x, 0, 0;
   0,-x, -x, 0, 0;
   0, x,  x, 0, 0];
x=[x;x(1,:)];

y=[0,-y, -y, 0, 0;
   0,-y, -y, 0, 0;
   0, y,  y, 0, 0;
   0, y,  y, 0, 0];
y=[y;y(1,:)];

z=ones(5,1)*[Boden,Boden,Hoehe,Hoehe,Boden];

%Erzeugung einer Schnittstelle zu anderen Körpern
x=[zeros(1,5);x;zeros(1,5)];
y=[zeros(1,5);y;zeros(1,5)];
z=[ones(1,5)*Hoehe;z;ones(1,5)*Boden];
end
```

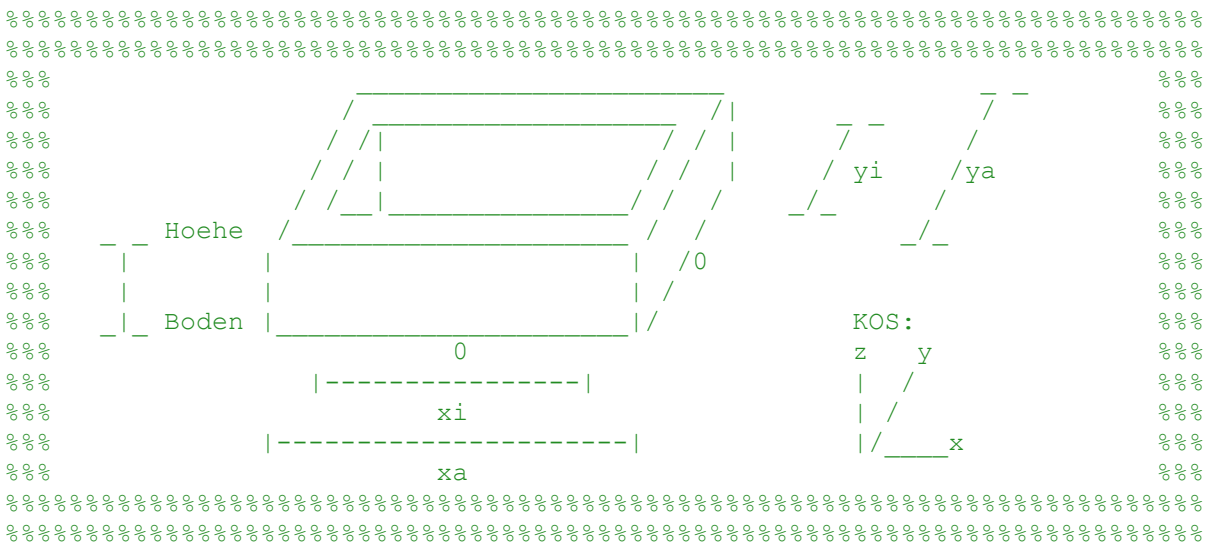
```
%% Grundfunktion: Schiefer Quader

function [x,y,z]=quadschief(x,y,HoeheL,HoeheR,Boden)
%mittlere Hoehe zwischen Hoehe_links und Hoehe_rechts
mi=mean([HoeheL,HoeheR]);
x=x/2;
y=y/2;

x=[0, x, x, 0, 0;
   0,-x,-x, 0, 0;
   0,-x,-x, 0, 0;
   0, x, x, 0, 0];
x=[x;x(1,:)];

y=[0,-y,-y, 0, 0;
   0,-y,-y, 0, 0;
   0, y, y, 0, 0;
   0, y, y, 0, 0];
y=[y;y(1,:)];

z=[Boden, Boden, HoeheR, mi, Boden;
   Boden, Boden, HoeheL, mi, Boden;
   Boden, Boden, HoeheL, mi, Boden;
   Boden, Boden, HoeheR, mi, Boden;
   Boden, Boden, HoeheR, mi, Boden];

%Schnittstelle zu anderen Körpern durch die Zusammenfassung der Konturlinie
%in einen Punkt ( Nutzung von ones() und zeros() )
x=[zeros(1,5);x;zeros(1,5)];
y=[zeros(1,5);y;zeros(1,5)];
z=[ones(1,5)*mi;z;ones(1,5)*Boden];
end
```

```
%% Grundfunktion: Halber test_tubehalter

function [x,y,z]= rgh(a,b,ri,Hoehe,Boden,n)
i=360/n;
a=a/2;
b=b/2;

%Komplexe Hilfsmatrizen (aus Beispielen abgeleitet)
r=[a*ones(1,n/8),a:-a/(n/8):-a,-a*ones(1,n/4-1),-
a:a/(n/8):a,a*ones(1,n/8)]';
k=[0:b/(n/8):b,b*ones(1,n/4-1),b:-b/(n/8):-b,-b*ones(1,n/4-1),-
b:b/(n/8):0]';

x=[ri*cosd(0:i:360)',r,r,ri*cosd(0:i:360)',ri*cosd(0:i:360)'];
x=[x;[ri,ri,ri,ri,ri]];

y=[ri*sind(0:i:360)',k,k,ri*sind(0:i:360)',ri*sind(0:i:360)'];
y=[y;zeros(1,5)];

z=[ones(n+1,2)*Boden,ones(n+1,2)*Hoehe,ones(n+1,1)*Boden];
z=[z;ones(1,5)*Boden];

%Körper halbieren
k=n/4+1;
%Hilfswert für geschlossene Kontur
h=ri+(b-ri)/2;

%Schnittstellen erzeugen und Datenmenge halbieren
x=[[h,h,h,h,h];[0,0,0,0,0];x(end-k:end,:);...
    x(1:k,:);[0,0,0,0,0];[h,h,h,h,h]];

y=[[-h,-h,-h,-h,-h];[-h,-h,-h,-h,-h];y(end-k:end,:);y(1:k,:);...
   [h,h,h,h,h];[h,h,h,h,h]];

z=[[Hoehe,Hoehe,Hoehe,Hoehe,Hoehe];[Boden,Boden,Hoehe,Hoehe,Boden];...
    z(end-k:end,:);z(1:k,:);[Boden,Boden,Hoehe,Hoehe,Boden];...
    [Boden,Boden,Boden,Boden,Boden]];
end
```

```
%% Grundfunktion: Drehung um x,y,z-Achse

function [x,y,z]=rot(x,y,z,a,b,c)
%Uebergabeparameter:
%x,y,z sind Matrizen (beschreiben Körper)
%Drehung um die X-Achse: mit a [Grad]
%Drehung um die Y-Achse: mit b [Grad]
%Drehung um die Z-Achse: mit c [Grad]

[o,p]=size(x);
%Spaltenvektoren aus x,y,z erzeugen und in coords zusammenführen
coords=[x(:) y(:) z(:)];
%Drehung um x-Achse
Ra=[1 0 0; 0 cosd(a) sind(a); 0 -sind(a) cosd(a)];
coords=coords*Ra;
%Drehung um y-Achse
Rb=[cosd(b) 0 -sind(b); 0 1 0; sind(b) 0 cosd(b)];
coords=coords*Rb;
%Drehung um y-Achse
Rc=[cosd(c) sind(c) 0; -sind(c) cosd(c) 0; 0 0 1];
coords=coords*Rc;
%Ausgangsmatrix wieder herstellen
x=reshape(coords(:,1),o,p);
y=reshape(coords(:,2),o,p);
z=reshape(coords(:,3),o,p);
end
```

Grundfunktion: Umrandung

```
function [x,y,z]=umrandung(xa,ya,xi,yi,Hoehe,Boden)
```



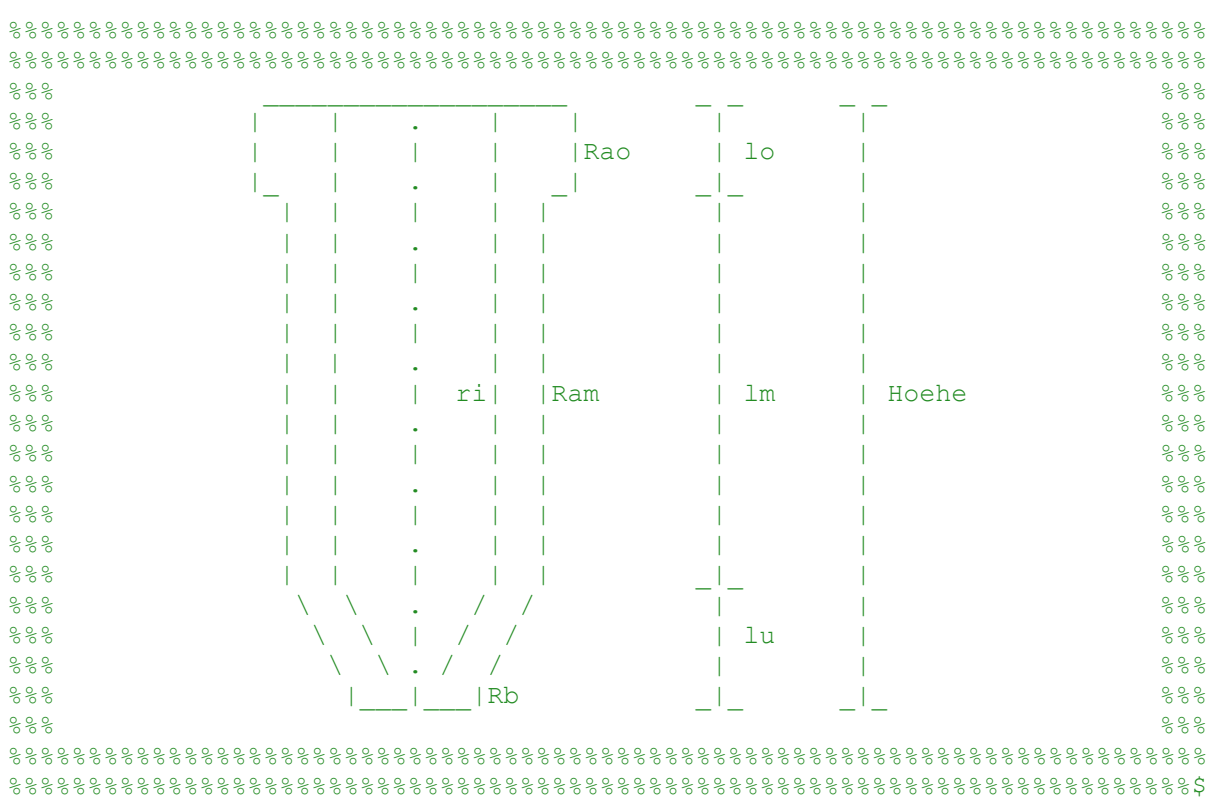

```
xa=xa/2;
ya=ya/2;
xi=xi/2;
yi=yi/2;

x=[ xa, xa, xa, xa, xa;
    xi, xa, xa, xi, xi;
   -xi,-xa,-xa,-xi,-xi;
   -xi,-xa,-xa,-xi,-xi;
    xi, xa, xa, xi, xi;
    xi, xa, xa, xi, xi];

y=[ ya, ya, ya, ya, ya;
    yi, ya, ya, yi, yi;
    yi, ya, ya, yi, yi;
   -yi,-ya,-ya,-yi,-yi;
   -yi,-ya,-ya,-yi,-yi
    yi, ya, ya, yi, yi];

z=[ones(1,5)*Boden;
   ones(5,1)*[Boden,Boden,Hoehe,Hoehe,Boden]];
end
```

```
%% Körper: table
function [x,y,z]=table
x=[0,140;
    0,140];
y=[0,0;
    140,140];
z=zeros(2);
end

%% Körper: i_puffer
function [x,y,z]=i_puffer
%Grundkörper
[xgk,ygk,zgk]=quad(30,22,0,8.7);

%Reagenzglashalter (rea)
[xrea,yrea,zrea]= klotzml(4,4,1.5,16.2,8.7,32);

%Dreierkombination oben links
xrea1=xrea-12;
yrea1=yrea+8;
xrea2=xrea-12;
yrea2=yrea+4;
xrea3=xrea-12;

%Fünfwerkombination (fu)
xfu=[xrea-8;xrea-4;xrea;xrea+8;xrea+4];
yfu=[yrea;yrea;yrea;yrea;yrea];
zfu=[zrea;zrea;zrea;zrea;zrea];

x=[xrea1;xrea2;xrea3;xgk;zeros(1,5);xfu+4;zeros(1,5);...
xfu+4;zeros(1,5);xfu+4];
y=[yrea1;yrea2;yrea;ygk;zeros(1,5);yfu+8;zeros(1,5);yfu;zeros(1,5);yfu-8];
z=[zrea;zrea;zrea;zgk;zeros(1,5);zfu;zeros(1,5);zfu;zeros(1,5);zfu];
end

%% Körper: libra
function [x,y,z]=libra
%test_tubehalter (rea)
[xrea,yrea,zrea]= klotzml(4,4,1.5,16,9.5,32);
xrea=xrea+6;

%Messplatte mit Loch (mp)
[xmp,ymp,zmp]=klotzml(20,25.5,0.9,9.5,7,32);
xmp=xmp+6;

%Grundkörper (gk)
[xgk,ygk,zgk]=quad(33,25.5,0,7);

%Anzeigehalter (anzh)
[xanzh,yanzh,zanzh]=quad(2,4,7,33);
xanzh=xanzh-15.5;
yanzh=yanzh-6.75;

%Anzeige (anz)
[xanz,yanz,zanz]=quad(4,24,40,33);
xanz=xanz-14.5;

x=[xrea;xmp;xgk;xanzh;xanz];
y=[yrea;ymp;ygk;yanzh;yanz];
z=[zrea;zmp;zgk;zanzh;zanz];
end
```

```
%% Körper: o_puffer
function [x,y,z]=o_puffer
%Grundkörper Seiten(gks)
[xgks,ygks,zgks]=quad(11,1.5,0,8.7);

ygks1=ygks+6.75;
ygks2=ygks-6.75;

%Grundkörper Mitte(gkm)
[xgkm,ygkm,zgkm]=quad(11,12,0,7.7);

%Grundkörper Basis 1-3(gkb)
[xgkb,ygkb,zgkb]= klotzml(11,4,0.9,8.7,7.7,32);

ygkb1=ygkb+4;
ygkb3=ygkb-4;

%test_tubehalter 1-3(rea)
[xrea,yrea,zrea]= klotzml(4,4,1.5,15.2,8.7,32);
yrea1=yrea-4;
yrea3=yrea+4;

x=[xgks;xgkb;xrea;xrea;xrea;xgkb;xgkb;xrea;xgks;xgkm];
y=[ygks1;ygkb1;yrea3;yrea3;yrea;ygkb;ygkb3;yrea1;ygks2;ygkm];
z=[zgks;zgkb;zrea;zrea;zrea;zgkb;zgkb;zrea;zgks;zgkm];
end

%% Körper: Würfelstation
function [x,y,z]=cube_station
%Grundkörper (gk)
[xgk,ygk,zgk]=quad(31,20,0,8.1);

%Umrandung (um)
[xum,yum,zum]=umrandung(31,20,23,18,10.1,8.1);
zum=zum+0.001;

x=[xgk;xum];
y=[ygk;yum];
z=[zgk;zum];
end

%% Körper: cross_station
function [x,y,z]=cross_station
%test_tubehalter (gp)
[xrea,yrea,zrea]= klotzml(4,4,1.5,15.2,8.7,32);
xrea=xrea-9.75;
%Grundplatte mit Loch (gp)
[xgp,ygp,zgp]= klotzml(6,11,0.9,8.7,7.7,32);
xgp=xgp-9.75;

%Grundkörper (gk)
[xgk,ygk,zgk]=quad(25.5,11,0,7.7);

%Abdeckung (ab)
[xab,yab,zab]=quad(19.5,11,7.7,8.7);
xab=xab+3;

%Halter (ha1)
[xha1,yha1,zha1]=quad(7.5,8,8.7,18);
xha1=xha1+7.9;
```

```
a=atand(6.3/6.5);        %Winkel
b=90-a;                  %Winkel Spitze oben rechts
h=0.5/sind(b);           %Höhe der Kante rechts außen
g=0.5/tand(b);

%Halter (ha2)
[xha2,yha2,zha2]=quadschief(6.5,0.2,33.2-h,39.5-h,18);
xha2=xha2+8;

%Halter rechts (hr)
[xhr,yhr,zhr]=quadschief(0.5,0.2,1.7,1.7-g,0);
[xhr,yhr,zhr]=rot(xhr,yhr,zhr,0,b,0);

gs=1.7-g+sqrt((h/2)^2-0.25^2);
xs=sind(b)*gs;

xhr=xhr+8+6.5/2-xs;
zhr=zhr+39.5-h/2-gs*cosd(b);

%Reganzglashalter (rgh)
[xrgh,yrgh,zrgh]= rgh(4,4,1.5,2,0,32);
[xrgh,yrgh,zrgh]=rot(xrgh,yrgh,zrgh,0,b,0);
xrgh=xrgh+8+6.5/2-sind(b)*3.7-sind(a)*1.5;
zrgh=zrgh+39.5-3.7*cosd(b)+1.5*cosd(a);

hhau=6.5/sind(b)-1.7-2-0.2;

%Halter unten (hau)
[xhau,yhau,zhau]=quad(0.5,0.2,hhau,0);
[xhau,yhau,zhau]=rot(xhau,yhau,zhau,0,b,0);
xhau=xhau+8+6.5/2-(3.7+hhau)*sind(b)+0.5/2*sind(a);
zhau=zhau+39.5-(3.7+hhau)*cosd(b)-0.5/2*cosd(a);

%Halter ende (he)
[xhe,yhe,zhe]=quad(0.2,0.2,0,3);
[xhe,yhe,zhe]=rot(xhe,yhe,zhe,0,-a,0);

xhe=xhe+8+6.5/2-(3.8+hhau)*sind(b)+0.5*sind(a);
zhe=zhe+39.5-(3.8+hhau)*cosd(b)-0.5*cosd(a);

x=[xrea;xgp;xgk;xab;xha1;xha2;xhr;xrgh;xhau;xhe];
y=[yrea;ygp;ygk;yab;yha1;yha2;yhr;yrgh;yhau;yhe];
z=[zrea;zgp;zgk;zab;zha1;zha2;zhr;zrgh;zhau;zhe];
end
```

```
%% Körper: test_tube

function [x,y,z]=test_tube
Rao=1.55;            %Radius aussen oben
lo=1.2;              %Länge des Gewindes
Ram=1.45;            %Radius aussen Mitte
lm=9.3;              %Länge des Mittelteils
Hoehe=11.5;          %Höhe des gesamten Reagenzglases
ri=1.35;             %Radius Innen
lu=Hoehe-lo-lm;      %Länge des Unterteils (Kegel)
Rb=0.3;              %Radius der Basis unten

n=32;                %Anzahl Segmente
```




```
%Spaltenvektor erzeugen
x=cosd(0:(360/n):360)'*[0,Rb,Ram,Ram,Rao,Rao,ri,ri,0,0];
y=sind(0:(360/n):360)'*[0,Rb,Ram,Ram,Rao,Rao,ri,ri,0,0];
z=ones(n+1,1)*[0,0,lu,(Hoehe-lo),(Hoehe-lo),Hoehe,Hoehe,lu,(Ram-ri),0];
end
```

A9: Quellcode des Skriptes robo_init

```
%% Globale Variablen um Funktionen TP_1 bis TP_14 anzulegen
global Robo

%% ROBO HOMEPOSITION UND ENTWIRRUNGSPUNKT FÜR DIE KAMERA
home_pose =rpoint(0,359,275,0,0,0,'speed',50;
Robo.home_pose=home_pose;

% %Hilfspunkt zur Konfigurationsentwirrung
p_entwirrung = rpoint(0,540,196,90,90,-90,'speed',50);

%% GREIFER zu/auf - PUNKTE
Clamp_on  = rpoint(0,0,0,0,0,0,'rel','signal',[-9 10],'wait',0.5);
Clamp_off = rpoint(0,0,0,0,0,0,'rel','signal',[9 -10],'wait',0.5);

%% ANFANGSPUFFER UND FARBPUFFER
%obere rechte Ecke eingeteacht
Ta_ANFANGSPUFFERinBASE = [rotz(pi) [485, -178 180]';0 0 0 1];

% Greifposition
Ta_GREIFinANFANGSPUFFER = [rotz(pi/2) [150 110 -37]'; 0 0 0 1];

%geteachte Zwischenposition (Sicherheitspunkt)
panfang_greif(1) = rpoint(350 , 20  , 180 , 0 ,0 , -90,'speed',50);

%Greifposition
pos = frame2kartpos2(Ta_ANFANGSPUFFERinBASE * Ta_GREIFinANFANGSPUFFER);
panfang_greif(2) = rpoint(pos(1),pos(2),pos(3),pos(4),pos(5),pos(6),...
'speed',50);

%     |--------------------------|
%     |     ---     ----------   |
%     |  R  |O|     |O|O|O|O|X|  |
%     |  Gr |O|     ----------   |
%     |  Ge |O|     |X|O|O|O|O|  |
%     |     ---     ----------   |
%     |             |X|O|O|O|O|  |
%     |             ----------   |
%     |--------------------------|

P = [  30-0.93      30+0      -37;
      30+40-.1      30+0      -37;
         30+80      30+0      -37;
        30+120      30+0      -37;
        30-0.1     30+80      -37;
       30+40.2     30+80      -37;
       30+80.1     30+80      -37;
        30+120     30+80      -37;
      30+40.15    30+160      -37;
         30+80    30+160      -37;
        30+120    30+160      -37;
      30+160.1    30+160      -37];

for k=1:12;
    pos = frame2kartpos2(Ta_ANFANGSPUFFERinBASE *[rotz(pi/2) P(k,:)'; ...
      0 0 0 1]);
    panfang_r(k) = rpoint(pos(1),pos(2),pos(3),pos(4),pos(5),pos(6),...
      'speed',50,'wait',0.5);
end
```

```
%% Farbpuffer Reagenzglaspositionen und Rotationsmatrix

% Farben-Reagenzgläser Robo-points
%Position_Rot
pos = frame2kartpos2(Ta_ANFANGSPUFFERinBASE*...
      [rotz(0) [270.2 190.1 -37]'; 0 0 0 1]);
pf(1) = rpoint(pos(1),pos(2),pos(3),pos(4),pos(5),pos(6),'speed',50);

%Position_Gelb
pos = frame2kartpos2(Ta_ANFANGSPUFFERinBASE *[rotz(0) [270.2 150 -37]';...
      0 0 0 1]);
pf(2) = rpoint(pos(1),pos(2),pos(3),pos(4),pos(5),pos(6),'speed',50);

%Position_Gruen
pos = frame2kartpos2(Ta_ANFANGSPUFFERinBASE *[rotz(0) [270 110 -37]';...
      0 0 0 1]);
pf(3) = rpoint(pos(1),pos(2),pos(3),pos(4),pos(5),pos(6),'speed',50);

%runterfahren um 120mm (Reagenzglasaufnahmepunkt)
panfangspuffer_h_aufnahme = rpoint(0,0,120,0,0,0,'rel','lin','speed',50);
panfangspuffer_h_ablage   = rpoint(0,0,120,0,0,0,'rel','lin','speed',50);

%hochfahren um 120mm nach Reagenzglasaufnahme
panfangspuffer_h_anheben  = rpoint(0,0,-120,0,0,0,'rel','lin','speed',50);

%Zwischenpunkt liegt zwischen der Umgreifstation und der Waage
p_zwischen = rpoint(-22,354,143,-64,0,62.8,'speed',50);

%% UMGREIFSTATION
% Beschreibung des Teilprozesses mit Frames
% Umgreifstation in Base-KOS Berechnung

%untere rechte Ecke eingeteacht
Tu_UMGREIFSTATIONinBASE = [rotz(-0.5*pi) [322.5 77 160]';0 0 0 1];

% Reagenzglasuebergabe
Tu_UEBERGABEinUMGREIFSTATION=[rotz(0) [55 30 -17]'; 0 0 0 1];

% 1. Reagenzglas (unten)
Tu_R1inUMGREIFSTATION=[rotz(0) [55 30 0]'; 0 0 0 1];

% Greifer in R1
% z=200 TCP zum Greifmittelpunkt verschoben
Tu_GREIFERinR1 =[rotz(0) [0 0 200]';0 0 0 1];

% Drehung in Greifer
Tu_DREHUNGinGREIFER = [rotz(pi/2)*roty(pi/2)*rotz(-pi/2) [0 0 0]';...
      0 0 0 1];

Tu_POSKORR = [rotz(0) [0 0 -167]'; 0 0 0 1];

% Reagenzglasaufnahme in Drehung1
Tu_AUFNAHMEinDREHUNG = [rotz(0) [0 -80 0]'; 0 0 0 1];

% Reagenzglasaufnahme2 in Reagenzglasaufnahme
Tu_AUFNAHME2inAUFNAHME = [rotz(0) [0 3 0]'; 0 0 0 1];

% Greifer in Drehung2
Tu_GREIFERinDREHUNG2 =Tu_GREIFERinR1;
```

```
% Drehung2 in Greifer 45° Drehung
Tu_DREHUNG2inGREIFER = [rotz(pi/2)*roty(-pi/4)*rotz(-pi/2) [0 0 0]'; ...
      0 0 0 1];

% Positionskorrektur zum TCP
Tu_POSKORR2 = [rotz(0) [0 0 -200]'; 0 0 0 1];

% Positionierung zum Umgreifen 45° Lage
Tu_POSUMinDREHUNG2 = [rotz(0) [0 228 0]'; 0 0 0 1];

% Ablägen in Posum ( Positionierung zum Umgreifen )
Tu_ABLAGEinPOSUM = [rotz(0) [0 0 38.5]'; 0 0 0 1];

% Greifer in Ablage
Tu_GREIFERinABLAGE =Tu_GREIFERinR1;

% Drehung3 in Greifer
Tu_DREHUNG3inGREIFER = [rotz(pi/2)*roty(pi/4)*rotz(-pi/2) [0 0 0]';...
      0 0 0 1];

% Positionierung2 in Drehung3
Tu_POSNUM2inDREHUNG3 = [rotz(0) [0 0 -145]'; 0 0 0 1];

% Greifer in Positionierung2
Tu_GREIFERinPOSNUM2 =Tu_GREIFERinR1;

% Drehung4 in  Greifer
Tu_DREHUNG4inGREIFER = [rotz(pi/2)*roty(45*pi/180)*rotz(-pi/2) [0 0 0]';...
      0 0 0 1];

% Positionierung3 in Drehung4
Tu_POSNUM3inDREHUNG4 = [rotz(0) [0 -69.5 80]'; 0 0 0 1];

% Positionierung4 in Positionierung3
Tu_POSNUM4inPOSNUM3 = [rotz(0) [0 58 0]'; 0 0 0 1];

% Berechnung der gesuchten Frames
Tu_UEBERGABEinBASE = Tu_UMGREIFSTATIONinBASE*Tu_UEBERGABEinUMGREIFSTATION;
Tu_R1inBASE        = Tu_UMGREIFSTATIONinBASE *Tu_R1inUMGREIFSTATION;
Tu_DREHUNGinBASE   = Tu_R1inBASE*Tu_GREIFERinR1*Tu_DREHUNGinGREIFER*...
      Tu_POSKORR;
Tu_AUFNAHMEinBASE  = Tu_DREHUNGinBASE*Tu_AUFNAHMEinDREHUNG;
Tu_AUFNAHME2inBASE = Tu_AUFNAHMEinBASE*Tu_AUFNAHME2inAUFNAHME;
Tu_DREHUNG2inBASE  = Tu_DREHUNGinBASE*Tu_GREIFERinDREHUNG2*...
      Tu_DREHUNG2inGREIFER*   Tu_POSKORR2;
Tu_POSUMinBASE     = Tu_DREHUNG2inBASE*Tu_POSUMinDREHUNG2;
Tu_ABLAGEinBASE    = Tu_POSUMinBASE*Tu_ABLAGEinPOSUM;
Tu_DREHUNG3inBASE  = Tu_ABLAGEinBASE*Tu_GREIFERinABLAGE*...
      Tu_DREHUNG3inGREIFER*Tu_POSKORR2;
Tu_POSNUM2inBASE   = Tu_DREHUNG3inBASE*Tu_POSNUM2inDREHUNG3;
Tu_DREHUNG4inBASE  = Tu_POSNUM2inBASE*Tu_GREIFERinPOSNUM2*...
      Tu_DREHUNG4inGREIFER*Tu_POSKORR2;
Tu_POSNUM3inBASE   = Tu_DREHUNG4inBASE*Tu_POSNUM3inDREHUNG4;
Tu_POSNUM4inBASE   = Tu_POSNUM3inBASE*Tu_POSNUM4inPOSNUM3;
```

```
%Roboterpunkte (Robo-Struktur-Points)
pos = frame2kartpos2(Tu_UEBERGABEinBASE);
pu_uebergabe = rpoint(pos(1),pos(2),pos(3),pos(4),pos(5),pos(6),...
      'speed',50);
pos = frame2kartpos2(Tu_R1inBASE );
pu_r1 = rpoint(pos(1),pos(2),pos(3),pos(4),pos(5),pos(6),'speed',50);

%wird für die Teilaufgabe 8 benötigt
%-------------------------------------------------------------------------
pu_r12 = [rpoint(0,0,-100,0,0,0,'rel','lin'),pu_r1];
%-------------------------------------------------------------------------

%geteachte Punkte zur Aufnahme und Ablage des Reagenzglases
%-------------------------------------------------------------------------
pu_ablaegen = rpoint(0,0,113,0,0,0,'rel','lin','speed',50,'wait',1);
%-------------------------------------------------------------------------

pos = frame2kartpos2(Tu_DREHUNGinBASE);
pu_drehung1 = rpoint(pos(1),pos(2),pos(3),pos(4),pos(5),pos(6),'speed',50);

pu_drehung1_lin =
rpoint(pos(1),pos(2),pos(3),pos(4),pos(5),pos(6),'lin',...
      'speed',50);

pos = frame2kartpos2(Tu_AUFNAHMEinBASE);
pu_aufnahme = rpoint(pos(1),pos(2),pos(3),pos(4),pos(5),pos(6), ...
      'lin','speed',50,'wait',0.5);
pos = frame2kartpos2(Tu_AUFNAHME2inBASE);

pu_aufnahme2 = rpoint(pos(1),pos(2),pos(3),pos(4),pos(5),pos(6),...
      'lin','speed',50,'wait',0.5);
pos = frame2kartpos2(Tu_POSUMinBASE);

pu_posum = rpoint(pos(1),pos(2),pos(3),pos(4),pos(5),pos(6),...
      'lin','speed',50);

pos = frame2kartpos2(Tu_DREHUNG2inBASE);
pu_drehung2 = rpoint(pos(1),pos(2),pos(3),pos(4),pos(5),pos(6),...
      'lin','speed',50);

pos= frame2kartpos2(Tu_ABLAGEinBASE);
pu_posum_ablage = rpoint(pos(1),pos(2),pos(3),pos(4),pos(5),pos(6),...
      'lin','speed',50,'wait',0.5);
pos = frame2kartpos2(Tu_DREHUNG3inBASE);

pu_drehung3 = rpoint(pos(1),pos(2),pos(3),pos(4),pos(5),pos(6),...
      'lin','speed',50);
pos = frame2kartpos2(Tu_POSNUM2inBASE);

pu_posnum2 = rpoint(pos(1),pos(2),pos(3),pos(4),pos(5),pos(6),...
      'lin','speed',50);
pos = frame2kartpos2(Tu_DREHUNG4inBASE);

pu_drehung4 = rpoint(pos(1),pos(2),pos(3),pos(4),pos(5),pos(6),...
      'lin','speed',50);
pos = frame2kartpos2(Tu_POSNUM3inBASE);

pu_posnum3 = rpoint(pos(1),pos(2),pos(3),pos(4),pos(5),pos(6),...
      'lin','speed',50,'wait',0.5);
pos = frame2kartpos2(Tu_POSNUM4inBASE );
```

```
pu_posnum4 = rpoint(pos(1),pos(2),pos(3),pos(4),pos(5),pos(6),...
      'lin','speed',50);

pu_linX2=rpoint(0,0,-100,0,0,0,'rel','lin');

%pu_uebergabe --> für Farbe auf die Umgreifstation lägen
TP_umgreifen =
[pu_r1,pu_ablaegen,Clamp_off,pu_r1,pu_drehung1,pu_aufnahme,Clamp_on,...
      pu_drehung1_lin,pu_drehung2,pu_posum,pu_posum_ablage,Clamp_off,...
      pu_posnum2,pu_drehung4,pu_posnum3,Clamp_on,pu_posnum4];

TP_umgreifen2= [pu_posnum3,Clamp_off,pu_posum_ablage,Clamp_on, ...
      pu_posum,pu_drehung2,pu_drehung1,pu_aufnahme,Clamp_off,pu_r1, ...
      pu_ablaegen, Clamp_on,pu_linX2,pu_r1];

%% FRAMES FÜR DIE KAMERAPOSITIONIERUNG UND POSITION ZUM WUERFELN
% WuerfelBox-NULL-Frame (linke untere ecke)
T_WBOXinBASE = [rotz(pi) [73 497.1 321]'; 0 0 0 1];

% WuerfelBox-MID-Frame
T_BOXMIDinWBOX = [rotz(0) [-125 95 0]'; 0 0 0 1];

%CamPosition-BoxMID-Frame (Kameraposition über der BoxMitte)
T_BOXCAMinBOXMID = [rotz(0) [0 -52 -322]'; 0 0 0 1];

%CamPosition-PicMID-Frame (Kameraposition über der BildMitte)
T_PICCAMinBOXCAM = [rotz(0) [-37 9 0]'; 0 0 0 1];

%PicPose in Base (Kamera steht genau über der Mitte der Wuerfelbox
pos = frame2kartpos2(T_WBOXinBASE * T_BOXMIDinWBOX *T_BOXCAMinBOXMID);
pic_pose = rpoint(pos(1),pos(2),pos(3),pos(4),pos(5),pos(6), ...
      'speed',50,'wait',1);

DWB=rpoint(10,50,345,0,0,0,'rel','ptp','wait',0.5);
DWB2=rpoint(0,0,-50,0,0,0,'rel','ptp','wait',1);

%% WAAGE
%Reagenzglass der Waage in Base-KOS Berechnung
% Waage in Base Beschreibung zuer Sicherheit ist z=180 gesetzt
% rechte untere Ecke eingeteacht und das Frame aufgebaut
Tw_WAAGEinBASE = [rotz(0.5*pi), [-245 275.5 180]';[0 0 0 1]];

% Reagenzglass in Waage Beschreibung
Tw_RinWAAGE = [rotz(0),[131.5 105 0]'; 0 0 0 1];

%Positionierung zum Befüllen
Tw_BEFUELLENinWAAGE = [rotz(40*pi/180)*roty(120*pi/180)*rotz(-pi/2), ...
      [-70 -40 135]';0 0 0 1];

%Drehung in Befüllen
Tw_DREHUNGinBEFUELLEN = [rotz(pi/2)*roty(-50*pi/180)*rotz(-pi/2),...
      [20 5 25]';0 0 0 1];

%Drehung2 in Drehung
Tw_DREHUNG2inDREHUNG = [rotz(pi/2)*roty(-20*pi/180)*rotz(-pi/2),...
      [0 40 10]';0 0 0 1];
%Drehpunkt
Tw_DREHPUNKTinDREHUNG2 = [rotz(0), [0 0 100]';0 0 0 1];

%Drehpunktkorrektur
Tw_DREHPUNKT_KORR = [rotz(0), [0 0 -100]';0 0 0 1];
```

```
%Frames und RoboPoints
pos=frame2kartpos2(Tw_WAAGEinBASE*Tw_RinWAAGE );
pw_r1 = rpoint(pos(1),pos(2),pos(3),pos(4),pos(5),pos(6),'speed',50);

pos=frame2kartpos2(Tw_WAAGEinBASE*Tw_BEFUELLENinWAAGE);
pw_befuellen = rpoint(pos(1),pos(2),pos(3),pos(4),pos(5),pos(6), ...
      'speed',50);%4

pw_drehung =rpoint(5.6,-7,-18,0,-57,0,'rel','axis');
pos=frame2kartpos2(Tw_WAAGEinBASE*Tw_BEFUELLENinWAAGE*...
      Tw_DREHUNGinBEFUELLEN);

pw_drehung1 = rpoint(pos(1),pos(2),pos(3),pos(4),pos(5),pos(6), ...
      'speed',50);

pos = frame2kartpos2(Tw_WAAGEinBASE*Tw_BEFUELLENinWAAGE*...
      Tw_DREHUNGinBEFUELLEN*Tw_DREHPUNKTinDREHUNG2*...
      Tw_DREHUNG2inDREHUNG*Tw_DREHPUNKT_KORR);

pw_drehung2 = rpoint(pos(1),pos(2),pos(3),pos(4),pos(5),pos(6), ...
      'lin','speed',50);%

pw_drehung3=rpoint(0,0,0,0,-80,0,'rel','axis');

%Hoehenkorrektur zum aufnehmen
pw_h  = rpoint(0,0, 85,0,0,0,'lin','rel','speed',50,'wait',0.2)
pw_hz = rpoint(0,0,-85,0,0,0,'lin','rel','speed',50,'wait',0.2);

%Zwischen bzw. Hilfspunkt
pw_zwischen = rpoint(-78,323,335,97,130,-90,'speed',50);

pw_D1=rpoint(0,0,0,-30,0,0,'rel','lin');
pw_D2=rpoint(0,0,0,60,0,0,'rel','lin');

%% ENDPUFFER

% Endbuffer in Base Beschreibung
% rechte untere Ecke eingeteacht und das Frame aufgebaut
Tend_ENDPUFFERinBASE = [rotz(0.5*pi), [-395+20 -275+2 180]';[0 0 0 1]];

%Rotationsmatrix ist für alle Reagenzgläser gleich
%Roboterpunkte (Robo-Structure-Points)
pos = frame2kartpos2(Tend_ENDPUFFERinBASE*[rotz(pi/2) [75 55 -50]'; ...
      0 0 0 1]);

pend_greif = rpoint(pos(1),pos(2),pos(3),pos(4),pos(5),pos(6), ...
      'speed',50);%4

pos = frame2kartpos2(Tend_ENDPUFFERinBASE*[rotz(pi/2) , [35 55 -33]'; ...
      [0 0 0 1]]);

pend_r(1) = rpoint(pos(1),pos(2),pos(3),pos(4),pos(5),pos(6), ...
      'lin','speed',50);
pos= frame2kartpos2(Tend_ENDPUFFERinBASE*[rotz(pi/2) , [75 55 -33]'; ...
      [0 0 0 1]]);

pend_r(2) = rpoint(pos(1),pos(2),pos(3),pos(4),pos(5),pos(6), ...
      'lin','speed',50

pos = frame2kartpos2(Tend_ENDPUFFERinBASE*[rotz(pi/2) , [115 55 -33]'; ...
      [0 0 0 1]]);
```

```
pend_r(3) = rpoint(pos(1),pos(2),pos(3),pos(4),pos(5),pos(6), ...
      'lin','speed',50);

%Hoehenkorrekturfür alle reagenzgläser zum Ablegen
pend_h       = rpoint(0,0,126,0,0,0,'rel','lin','speed',50);

%aufnahme
pend_haufn   = rpoint(0,0,126,0,0,0,'rel','lin','speed',50);
pend_anheben = rpoint(0,0,-71,0,0,0,'rel','lin','speed',50);

%Konfigurationskorrekturpunkt
pend_pos = rpoint(0,0,0,0,0,0,'speed',50);

%% ROBOTERAUFGABEN
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Aufgabe1 (FromBeginningbufferToBalance)
TP1=[panfang_greif,panfang_r,panfangspuffer_h_aufnahme,Clamp_on, ...
      panfangspuffer_h_anheben,pu_uebergabe,p_zwischen,pw_r1,pw_h, ...
      Clamp_off,pw_r1];

%% Aufgabe2 (MoveAndDice)
TP2 = [home_pose,pic_pose];
TP2_1=[DWB,DWB2,pic_pose];

%% Aufgabe3 (FromColourbufferToEncompass) Umgreifstation
TP3 = [panfang_greif(2),pf,panfangspuffer_h_aufnahme,Clamp_on, ...
      panfangspuffer_h_anheben,pu_uebergabe,pu_r1,pu_ablaegen, ...
      Clamp_off,pu_r1,pu_drehung1,pu_aufnahme,Clamp_on,pu_drehung1_lin, ...
      pu_drehung2,pu_posum,pu_posum_ablage,Clamp_off,pu_posnum2];

%% Aufgabe4 (MoveToFilling)
TP4 =[pu_drehung4,pu_posnum3,Clamp_on,pu_posnum4,pw_zwischen,...
      pw_befuellen,pw_drehung];

%% Aufgabe5 (Filling)
TP5 = [pw_D1,pw_D2,pw_D1];

%% Aufgabe6 (BackToEncompass)
TP6 = [pw_befuellen,pw_zwischen,pu_posnum4,pu_posnum3,Clamp_off, ...
      pu_posnum2];

%% Aufgabe7 (FromBalanceToEndbufferAndPlaceIn)
TP7=[pw_zwischen,pw_r1,pw_h,Clamp_on,pw_hz,pu_r1,pend_greif,pend_r,...
      pend_h,Clamp_off,pend_greif];

%% Aufgabe8 (FromEncompassToColourbuffer)
TP8=[pu_uebergabe,pu_posum,pu_posum_ablage,Clamp_on,pu_posum, ...
      pu_drehung2,pu_drehung1,pu_aufnahme,Clamp_off,pu_r1,pu_ablaegen, ...
      Clamp_on,pu_linX2,pu_r1,pf,panfangspuffer_h_ablage,Clamp_off, ...
      panfangspuffer_h_anheben];

%% Aufgabe9 (MoveToEndbufferAndTake)
TP9=[pu_r1,pend_greif,pend_r,pend_haufn,Clamp_on,pend_anheben,...
      pend_greif,pu_r1];

%% Aufgabe10 (EmptyTubeAndMoveToBeginningbuffer)
TP10=[TP_umgreifen,pw_zwischen,pw_befuellen,pw_drehung,pw_drehung2, ...
      pw_drehung3,pw_befuellen,pw_zwischen,pu_posnum4,TP_umgreifen2, ...
      pu_uebergabe,panfang_r,panfangspuffer_h_ablage,Clamp_off, ...
      panfangspuffer_h_anheben,panfang_greif(2)];
```

```
%% Aufgabe11 (MoveToBeginningbuffer)
TP11=[pu_uebergabe,panfang_greif(2),panfang_r,panfangspuffer_h_ablage, ...
      Clamp_off,panfangspuffer_h_anheben,panfang_greif(2)];

%% Aufgabe12 (FromEncompassToBalance)
TP12 =[pu_uebergabe,pu_posum,pu_posum_ablage,Clamp_on,pu_posum, ...
      pu_drehung2,pu_drehung1,pu_aufnahme,Clamp_off,pu_drehung1,pu_r12, ...
      pu_ablaegen,Clamp_on,pu_r12,pu_uebergabe,p_zwischen,pw_r1,pw_h, ...
      Clamp_off,pw_r1];

%% Aufgabe13 (FromColourbufferToBalance)
TP13=[panfang_greif(2),pf,panfangspuffer_h_aufnahme,Clamp_on,...
      panfangspuffer_h_anheben,pu_uebergabe,p_zwischen,pw_r1,pw_h,...
      Clamp_off,pw_r1];

%% Aufgabe14 (FromBalanceToColourbuffer)
TP14=[p_zwischen,pw_r1,pw_h,Clamp_on,pw_hz,pu_uebergabe, ...
      panfang_greif(2),pf,panfangspuffer_h_ablage,Clamp_off, ...
      panfangspuffer_h_anheben];

%% Globale Variablen um Funktionen TP_1 bis TP_14 anzulegen
Robo.TP1=TP1;
Robo.TP2=TP2;
Robo.TP2_1=TP2_1;
Robo.TP3=TP3;
Robo.TP4=TP4;
Robo.TP5=TP5;
Robo.TP6=TP6;
Robo.TP7=TP7;
Robo.TP8=TP8;
Robo.TP9=TP9;
Robo.TP10=TP10;
Robo.TP11=TP11;
Robo.TP12=TP12;
Robo.TP13=TP13;
Robo.TP14=TP14;
save('WayPoints','Robo')
```

A10: Quellcode der Funktion A1_V_Pc

```
%Erzeugt einen Roboter vom Typ Kawasaki un öffnet Port 40000
VirtualRobot.create('Kawasaki', 40000, [-20, 2, 858, 180, 180, 0],[-700 700
-700 700 -942 1058]);
```

A11: Quellcode der Funktion A2_C_Pc

```
%Laborautomatisierung
global en_Mat tube_capa en_MatTemp last_pic_call Robo

%--------------------------------------------------------------------------
load('WayPoints.mat')

Robo.hnd=robot('open', 'Kawasaki', 'tcpip', 'localhost', 40000);
rmove(Robo.hnd,Robo.home_pose);  %Startet Robot in Homeposition
%--------------------------------------------------------------------------
%Erzeugen leerer Reagenzgläser
%Hier können die Anfangszustände (Anfangsinhalt in Gramm) der
%Reagenzgläser festgelegt werden.
en_Mat = [11 0 11 0 0 0 0 0 0 0 0 0];
```

```
%Erzeugen einer Zwischenspeichermatrix
en_MatTemp =zeros(1,12);

%Festlegung des JobTypes                rot = 1; gelb = 2; gruen = 3;
JobTypVec = [1 2 3 1 2 3 1 2 3 1 2 3];

%Zulässiger Reagenzglasinhalt in Gramm
tube_capa = 11;

%Wichtig für die MakePicture Funktion
last_pic_call = 1;

%Interface polling time
t_polling = 0.1;

%--------------------------------------------------------------------------
%Öffnet die Prozess-Steuerung
open_system('Laborautomatisierung_SIM_VISU')
%==========================================================================
%% Visualisierung der Kontrollelemente

load('Wuerfel.mat');                        %lädt Pkte des Würfels
WC=[255/255 165/255 0/255];                 %Würfel-Farbe (RGB)

figure('name','Übersicht','NumberTitle','off');
subplot(3,1,1)
surf(x,y,z)                                 %Visualisiert den Würfel
axis equal
view(-45,45)
colormap(WC)
set(gca,'xTick',[],'yTick',[],'zTick',[]);  %Achsenbeschriftung ausblenden


%Erzeugt Beschriftung des Würfels und generiert Händels, welche global
%gespeichert werden und zur Aktualisierung der Grafik dienen
Robo.o=text(0,0,122,'','FontUnits','normalized','FontSize',0.2,...
    'HorizontalAlignment','center','BackgroundColor',WC);
Robo.l=text(-20,0,100,'','FontUnits','normalized','FontSize',0.2,...
    'HorizontalAlignment','center');
Robo.r=text(0,-20,100,'','FontUnits','normalized','FontSize',0.2,...
    'HorizontalAlignment','center');

%nachfolgende Matrix enthält die Zahlenwerte zur Visualisierung der Seiten
%des Würfels (Seiten werden zufällig gewählt)
Robo.w=[2 3; 3 5; 5 4; 4 2;     %Kombination für 1 gewürfelt
        1 3; 3 6; 6 4; 1 4;     %Kombination für 2
        1 2; 2 6; 6 5; 5 1;     %Kombination für 3
        1 5; 5 6; 6 2; 2 1;     %Kombination für 4
        1 3; 3 6; 6 4; 4 1;     %Kombination für 5
        2 4; 4 5; 5 3; 3 2];    %Kombination für 6
%==========================================================================
%% Zweites Subplot (Waage)
subplot(3,1,2);
imshow('Display.png'); %Zeigt Display der Waage an

%Händel wird global gespeichert -->zur späteren Aktualisierung des Displays
Ro-
bo.D=text(200,100,0,'0.00g','FontUnits','normalized','FontSize',0.2,'Horizo
ntalAlignment','center');
%==========================================================================
```

```
%% Drittes Subplot (Balkendiagramm der Füllstande)
subplot(3,1,3);

%Füllstände (en_Mat) und Farbbelegung (JobTyp) werden zur Erstellung des
%farbigen Balkendiagramms miteinander verknüpft. Dafür ist die aufteilung
%in drei seperate balkendiagramme notwendig, welche mit hold on
%übereinandergelegt werden (seperate Balkenfarbe in Matlab nicht
%vorgesehen) --> siehe OnlineDoku Matlab (hier Trick verwendet: hold on)

%Hilfsvariable welche zur Speicherung der Füllstände dient und bei Tara
%nicht auf Null resetet wird (zur Visualisierung notwendig)
Robo.en_Mat2=en_Mat;

%muss zur Aktualisierung der Füllstände global gespeichert werden
Robo.JobTypVec=JobTypVec;
one_neg=-ones(1,12); %Erzeugt Hilfsvektor um Farbbelegung dazustellen

hold on
%Händels werden global zur späteren Aktualisierung gespeichert
Robo.b2_r = bar(((JobTypVec==1).*en_Mat),'r');
Robo.b2_y = bar(((JobTypVec==2).*en_Mat),'y');
Robo.b2_g = bar(((JobTypVec==3).*en_Mat),'g');

%Farbbelegung der Reagenzgläser im Balkendiagramm darstellen
bar(((JobTypVec==1).*one_neg),'r')
bar(((JobTypVec==2).*one_neg),'y')
bar(((JobTypVec==3).*one_neg),'g')
hold off

grid on
title('Füllstand Reagenzgläser')
ylabel('Gewicht in Gramm')

%Achseneinstellung vornehmen (Y-Achse mit -1 für Farbbelegung notwendig)
set(gca,'XLim',[0 13], 'YLim',[-1 20]);grid on;
```

A12: Quellcode der Funktion A3_V_Pc

```
%Robot-Visualisation
VirtualRobot.start_all

z=0; %Höhenkoordinate um Stationen neu zu positionieren
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Envirement/Umgebung
VirtualRobot.place_env('Zelle', [0 540 z 0 0 0],[122/255 122/255 122/255]);
VirtualRobot.place_env('libra', [-430 -405 z 0 0 0],[0 0 1]);
VirtualRobot.place_env('cube_station', [170 -390 z 0 0 0],[0 100/256 0]);
VirtualRobot.place_env('i_puffer', [315 290 z 0 0 0],[1 0 0]);
VirtualRobot.place_env('o_puffer', [-450 200 z 0 0 0],[0 1 1]);
VirtualRobot.place_env('cross_station', [430 -20 z 0 0 0],[1 0 1]);
VirtualRobot.place_env('Wuerfel',[170 -390 z 0 0 0],[1 0.64 0]);
VirtualRobot.place_env('Tischplatte', [-625 -625 z 0 0 0],[1 140/255 0]);
VirtualRobot.place_env('Boden', [-700 -700 -942+z 0 0 0],[1 0.75 37/255]);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%moving parts/Bewegliche Teile
%      |--------------------------|
%      |     ---      ----------  |
%      |  R  |O| 1                |
%      |  Ge |O| 2    ----------  |
%      |  Gr |O| 3                |
%      |     ---      ----------  |
%      |                          |
%      |              ----------  |
%      |--------------------------|
VirtualRobot.place_part('test_tube', [195 370 91+z 0 0 0],[1 0 0]); %rot
VirtualRobot.place_part('test_tube', [195 330 91+z 0 0 0],[1 1 0]); %gelb
VirtualRobot.place_part('test_tube', [195 290 91+z 0 0 0],[0 1 0]); %grün

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Rechte Reihe --> angefangen mit linken Reagenzglas
%      |--------------------------|
%      |              ----------  |
%      |           1 |O|O|O|O|O| 5|
%      |              ----------  |
%      |                          |
%      |              ----------  |
%      |                          |
%      |              ----------  |
%      |--------------------------|
VirtualRobot.place_part('test_tube', [275 370 91+z 0 0
0],[198/255,226/255,255/255]);
VirtualRobot.place_part('test_tube', [315 370 91+z 0 0
0],[198/255,226/255,255/255]);
VirtualRobot.place_part('test_tube', [355 370 91+z 0 0
0],[198/255,226/255,255/255]);
VirtualRobot.place_part('test_tube', [395 370 91+z 0 0
0],[198/255,226/255,255/255]);
%VirtualRobot.place_part('test_tube', [435 370 91+z 0 0
0],[198/255,226/255,255/255]);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%mittlere Reihe --> angefangen mit linken Reagenzglas
%      |--------------------------|
%      |              ----------  |
%      |                          |
%      |              ----------  |
%      |           1 |O|O|O|O|O| 5|
%      |              ----------  |
%      |                          |
%      |              ----------  |
%      |--------------------------|
%VirtualRobot.place_part('test_tube', [275 290 91+z 0 0
0],[198/255,226/255,255/255]);
VirtualRobot.place_part('test_tube', [315 290 91+z 0 0 0], ...
[198/255,226/255,255/255]);
VirtualRobot.place_part('test_tube', [355 290 91+z 0 0 0], ...
[198/255,226/255,255/255]);
VirtualRobot.place_part('test_tube', [395 290 91+z 0 0 0], ...
[198/255,226/255,255/255]);
VirtualRobot.place_part('test_tube', [435 290 91+z 0 0 0], ...
[198/255,226/255,255/255]);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%Linke Reihe --> angefangen mit linken Reagenzglas
%      |--------------------------|
%      |              -----------  |
%      |                           |
%      |              -----------  |
%      |                           |
%      |              -----------  |
%      |            1 |O|O|O|O|O| 5|
%      |              -----------  |
%      |--------------------------|
%VirtualRobot.place_part('test_tube', [275 210 91+z 0 0
0],[198/255,226/255,255/255]);
VirtualRobot.place_part('test_tube', [315 210 91+z 0 0 0], ...
      [198/255,226/255,255/255]);
VirtualRobot.place_part('test_tube', [355 210 91+z 0 0 0], ...
      [198/255,226/255,255/255]);
VirtualRobot.place_part('test_tube', [395 210 91+z 0 0 0], ...
      [198/255,226/255,255/255]);
VirtualRobot.place_part('test_tube', [435 210 91+z 0 0 0], ...
      [198/255,226/255,255/255]);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%% Einstellung zur Modellbetrachtung %%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Vorzugswinkel zur Zellenbetrachtung
view(-6,20)

%rotate3d on
set(gca,'Color',[204/255,204/255,204/255])

%Achsen ausblenden
set(gca,'xTick', [],'yTick',[],'zTick',[]);

%Achsenbeschriftung ausschalten
xlabel('');ylabel('');zlabel('')
zoom on
grid off
clc
```

Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die hier vorliegende Arbeit selbstständig, ohne fremde Hilfe und nur unter Verwendung der in dieser Arbeit aufgeführten Hilfsmittel angefertigt habe.

Ort, Datum Birger Freymann