Hochschule Wismar
University of Technology, Business and Design
Fakultät für Ingenieurswissenschaften

Bachelor Thesis

Entwicklung einer koordinierten Roboterapplikation mit Kawasaki-FSN300 Roboter

Eingereicht am: 23.04.2013

von: Christoph Rasch, geboren am

21.08.1989 in Parchim

Matrikelnummer: 114104

Gutachter: Prof. Dr.-Ing. T. Pawletta

Aufgabenstellung

Für die Bachelor Thesis Nr.: ..... lfd. Nr./

Bearbeiter: Rasch, Christoph Martrikel-NR.: 114104 Matrikel: BMA09

Thema:
Entwicklung einer koordinierten Roboterapplikation mit Kawasaki-FSN300 Roboter

In der FG CEA wurden verschiedene MATLAB Toolboxen zur Steuerungsentwicklung für KUKA und Kawasaki Roboter entwickelt.
In dieser Arbeit soll mit den Toolboxen eine koordinierte Roboterapplikation entwickelt werden, welche aus realen und visualisierten Objekten besteht.

Schwerpunkte:

Einarbeitung in die Grundlagen: Roboterkinematik, MatlabKK-Robotic Tbx., MatlabKK-VisualizationTbx., CAD-Schnittstelle
Systemanalyse der koordinierten Roboterapplikation bestehend aus Kawasaki-FSN300 Robotern und einem automatisierten Transportwagen
Geometrische Modellierung der Applikation mit Creo Parametric 2.0 und MatlabKK-VisualizationTbx.
Entwicklung einer Steuerungsschnittstelle für den visualisierten Transportwagen mit MATLAB
Entwicklung der gesamten Steuerung Roboter + Transportwagen als virtuelle Steuerung (mit MatlabKK-Robotic & Matlab-KK-Visualization Tbx.)
Entwicklung der gesamten Steuerung als kombinierte virtuelle und reale Steuerung
Test der hybriden Applikation mit realen und visualisierten Objekten

Die Arbeit ist nach den vorgegebenen Richtlinien anzufertigen. Darin eingeschlossen ist die Anfertigung einer WWW-Seite zu den wesentlichen Lösungswegen und Ergebnissen der Arbeit. Neben der geforderten Anzahl schriftlicher Exemplare ist die schriftliche Arbeit (Originaldateien & PDF-Dateien) sowie die entwickelte Software auf einem dokumentierten digitalen Datenträger abzugeben

1. Betreuer/Prüfer:	Prof. Dr. Thorsten Pawletta, HS Wismar, FIW/MVU, FG CEA
2. & 3. Betreuer/Prüfer:	MEng. Artur Schmidt, HS Wismar, FIW/MVU, FG CEA
	MEng. Tobias Schwatinski, HS Wismar, FIW/MVU, FG CEA

Ausgabedatum:
Abgabetermin:

Prof. Dr. rer. nat. Dr.-Ing. habil. Andreas Kossow
Vorsitzender des Prüfungsausschusses Maschinenbau

Zusammenfassung

In der Forschungsgruppe Computational Engineering and Automation (CEA) wurden die MatlabKK-Robotic- und MatlabKK-Visualization-Toolbox für MATLAB entwickelt. Diese ermöglichen die Visualisierung und Steuerung von KUKA- oder KAWASAKI-Roboter. Durch die Erweiterung der MatlabKK-Visualization-Toolbox um eine CAD-Schnittstelle ist es möglich, STL-Modelle in die Visualisierung zu integrieren. Die Visualisierung unterscheidet zwischen den drei Objekttypen: Roboter, Umweltobjekte und Parts. Umweltobjekte dienen der Dekoration, während Parts von Robotern bewegt werden können. Roboter besitzen eine eigene Kinematik, einen Greifer und einen begrenzten Arbeitsraum. In dieser Arbeit wird der Objekttyp der Roboter um einen Transportwagen ergänzt, um größere Distanzen zwischen den Robotern und die Grenzen der Arbeitsräume zu überwinden. Der Arbeitsraum stellt den durch den Greifer eines Roboters zugänglichen Bereich dar. Das Modell des Transportwagens wird mit Hilfe der CAD-Schnittstelle der MatlabKK-Visualization-Toolbox eingelesen. Der Anwender kann zudem selbst erstellte Stereolithographie-Objekte (STL) als Modell Transportwagens übernehmen. Dafür ist der Objekttyp `New_Robot` integriert. Der Transportwagen, auch `Cart` genannt, kann sich frei im Raum bewegen, Parts greifen und verschieben. Zusätzlich ist eine Methode zur Erzeugung eines Transportbands implementiert, auf dem sich ein `Cart` bewegen kann. Die Schwerpunkte dieser Arbeit bilden die Erweiterung der MatlabKK-Visualization-Toolbox um den Transportwagen, die Visualisierung eines Transportbandes und die Implementierung einer neuen Toolbox, welche die Bewegung des Transportbandes steuert.

Inhaltsverzeichnis

	Inhaltsverzeichnis	iv
	Abbildungsverzeichnis	v
	Tabellenverzeichnis	vi
	Verzeichnis der Abkürzungen	vii
1	Einleitung	1
2	Vorbetrachtungen zur Robotik	2
2.1	Kinematik von Robotern	2
2.2	Die MatlabKK-Robotic-Toolbox	5
2.3	Die MatlabKK-Visualization-Toolbox mit CAD-Schnittstelle	6
2.4	Erweiterungsmöglichkeiten der MatlabKK-Visualization-Toolbox	8
3	Erweiterung der MatlabKK-Visualization-Toolbox	10
3.1	Systemanalyse der MatlabKK-Visualization-Toolbox	10
3.2	Erweiterung der MatlabKK-Visualization-Toolbox um das Cart	12
3.3	Implementierung von `New_Robot`	20
3.4	Anwendungsbeispiele	22
3.4.1	Anwendungsbeispiel mit Cart	22
3.4.2	Anwendungsbeispiel mit New_Robot	23
4	Erweiterung der MatlabKK-Visualization-Toolbox um Transportbänder	26
4.1	Systemanalyse der theoretische Grundlagen	26
4.2	Implementierung der `create_line` Methode in die Visualization-Toolbox	31
4.3	Anwendungsbeispiel der `create_line` Methode	37
5	Implementierung der Conveyer-Toolbox	40
5.1	Zu betrachtende Funktionen der Conveyer-Toolbox	40
5.2	Implementierung der betrachteten Funktionen	46
6	Entwicklung einer koordinierten Roboterapplikation	48
6.1	Beispiel: Roboter mit Conveyer-Toolbox und `create_line` Methode	48
6.2	Beispiel: Roboter mit Conveyer-Toolbox und einem STL-Transportband Objekt	50
7	Zusammenfassung und Ausblick	53
	Literaturverzeichnis	54
	Anlagenverzeichnis und Anlagen	55

Abbildungsverzeichnis

Abbildung 1: Kawasaki FS003N 2
Abbildung 2: KUKA KR3 2
Abbildung 3: Zweigelenkroboter RV6-12 3
Abbildung 4: Koordinatensysteme des RV6-12 3
Abbildung 5: Kommunikation zwischen den Toolboxen 5
Abbildung 6: Visualisierung des Kawasaki FS003N 8
Abbildung 7: Arbeitsraum der Kawasaki-Roboter 9
Abbildung 8: Aufbau eines Sechs-Achs-Roboters 11
Abbildung 9: Darstellung des Carts in Creo Parametric 2.0 12
Abbildung 10: MATLAB-Code der `Cart` Fallunterscheidung im Konstruktor 13
Abbildung 11: Erweiterung der `create` Methode 13
Abbildung 12: Fallunterscheidung der `start` Methode 14
Abbildung 13: MATLAB-Code von `frame_cart` 15
Abbildung 14: Berechnung der inversen Kinematik in `rmove_pose_rel_ptp` 16
Abbildung 15: Fallunterscheidung in `rmove_axis_rel_lin` 17
Abbildung 16: Erweiterung der Fallunterscheidung in `correct_axis_abs` 18
Abbildung 17: Achswinkelbestimmung in `draw_axes` 18
Abbildung 18: Transformation des Greifers in `Draw` 19
Abbildung 19: Zeichnung des Carts in `Draw` 20
Abbildung 20: Implementierung der Eingangsvariablen 21
Abbildung 21: MATLAB-Code der `New_Robot` Fallunterscheidung 21
Abbildung 22: Visualisierung des Beispiels mit `Cart` 23
Abbildung 23: Visualisierung des Beispiels mit `New_Robot` 24
Abbildung 24: Problembeispiel für den STL-Export 25
Abbildung 25: Darstellung der Transportband-Objekte in MATLAB 26
Abbildung 26: Bemaßung des Transportbands in Relation zum Cart 27
Abbildung 27: Definition der Punkte und der Winkel `phi` 27
Abbildung 28: Visualisierung der Start- und Endpunkte 28
Abbildung 29: Extrudieren der Eckpunkte der Transportbänder 28
Abbildung 30: Definieren der Eckpunkte einer Rampe 29

Abbildung 31: Koordinaten der Transportbandabschnitte 30
Abbildung 32: Koordinatenmatrix der `create_line` Methode 30
Abbildung 33: Algorithmus der create_line Methode 31
Abbildung 34: Erstellen des Startpunkts in `create_line` 32
Abbildung 35: Berechnung der Mittelwerte und Einlesen des Modells 33
Abbildung 36: Positionieren und Skalieren der Transportbandabschnitte 34
Abbildung 37: Darstellung der Länge `r` 34
Abbildung 38: Skizze der Breite `s` 35
Abbildung 39: Extrusion des ersten Transportbandabschnittes 35
Abbildung 40: Definieren der Eckpunkte einer Rampe 37
Abbildung 41: Skizze der Geraden des Transportbands 38
Abbildung 42: Visualisierung des Transportbands mit `create_line` 39
Abbildung 43: MATLAB-Code der `cset` Funktion 46
Abbildung 44: Abbildung 41: MATLAB-Code der `cgrab` Funktion 46
Abbildung 45: MATLAB-Code der `conveyer` Funktion 47
Abbildung 46: Visualisierung der Roboterappikation mit `create_line` 50
Abbildung 47: CAD-Transportband mit STL-Export 51
Abbildung 48: Visualisierung der Roboterapplikation mit CAD-Transportband 52

Tabellenverzeichnis

Tabelle 1: DH-Parameter des RV6-12 3

Tabelle 2: Befehlsabfolge des Beispiels 7

Tabelle 3: Befehlsabfolge des Beispiels mit `Cart` 22

Tabelle 4: Befehlsabfolge des Beispiels mit `New_Robot` 23

Tabelle 5: Befehlsabfolge am Beispiel von `create_line` 38

Tabelle 6: Parameter der `conveyer` Funktion 41

Tabelle 7: Parameter der `cpoint` Funktion 41

Tabelle 8: Parameter der `cmove` Funktion 42

Tabelle 9: Properties der `cset` Funktion 42

Tabelle 10: Properties der `cget` Funktion 43

Tabelle 11: Parameter der `ccallback` Funktion 43

Tabelle 12: Parameter der `cis` Funktion 44

Tabelle 13: Parameter der `cwait` Funktion 44

Tabelle 14: Befehlsfolge der Applikation mit `create_line` 48

Tabelle 15: Befehlsfolge der Applikation mit STL-Transportband 51

Verzeichnis der Abkürzungen

CEA	Computational Engineering and Automation
STL	Stereolithographie
CAD	Computer-aided Design
TCP	Tool-Center-Point
DH-Parameter	Denavit-Hartenberg-Parameter
OOP	Objektorientierte Programmierung

1 Einleitung

Die in der Forschungsgruppe CEA entwickelten MatlabKK-Robotic- und MatlabKK-Visualization-Toolboxen können Kuka- und Kawasaki-Roboter steuern und visualisieren. Durch die Erweiterung der MatlabKK-Visualization-Toolbox um eine CAD-Schnittstelle können neue STL-Modelle von Robotern, Parts und Umweltobjekten problemlos in der Toolbox visualisiert werden. Die in den Toolboxen definierte Kinematik erlaubt es Robotern, Parts zu greifen und im Bereich ihres Arbeitsraums zu bewegen. Da der Arbeitsraum begrenz ist, müssen zur Überwindung größerer Distanzen mehrere Roboter nebeneinander aufgestellt sein und das Part an den nächstgelegenen Roboter weitergeben. Diese Vorgehensweise ist aufwändig und realitätsfern. In der Industrie werden Distanzen mit Hilfe von Transportbändern und Transportwagen überwunden. Diese Arbeit beschäftigt sich deshalb in Kapitel 3 mit der Implementierung von Transportwagen in die MatlabKK-Robotic-Toolbox, welche im Folgenden auch als Carts bezeichnet werden. Das Modell des Transportwagens wird als eine flache Scheibe vorgegeben. Auf Wunsch soll der Nutzer mit importierten STL-Dateien das Aussehen des Carts verändern und seinem Anwendungsbereich anpassen können. Außerdem soll in Kapitel eine Methode erstellt werden, welche es dem Nutzer ermöglicht, Transportbänder durch Vorgabe von Koordinaten zu visualisieren. Die Methode erleichtert dessen Visualisierung. Transportbänder können zudem durch STL-Import implementiert werden. Die Steuerung der Transportwagen erfolgt mit der neu implementierten Conveyer-Toolbox, welche auf die MatlabKK-Robotic-Toolbox aufbaut und deren Funktionen an die Anforderungen des Carts anpasst sind. Die Conveyer-Toolbox ist in Kapitel 5 vorgestellt. In Kapitel 6 sollen Beispiele für die Implementierungen angegeben werden.

2 Vorbetrachtungen zur Robotik

Dieses Kapitel befasst sich mit der grundlegenden Kinematik von 6-Achs-Robotern, wie diese definiert wird und welche Unterscheidungsmöglichkeiten es innerhalb der Kinematik gibt. Des Weiteren wird auf die MatlabKK-Robotic-Toolbox [1] und die MatlabKK-Visualization-Toolbox [2] mit CAD-Schnittstelle [3] eingegangen. Die Grundlage für Abschnitt 2.1 bildet [4].

2.1 Kinematik von Robotern

Sowohl der in Abbildung 2 dargestellte Kuka KR3 als auch der in Abbildung 1 gezeigte Kawasaki FS003N weisen je sechs Gelenke auf, welche durch ihre Gelenkwinkel definiert sind. Durch das Einstellen der Position und Orientierung kann man die Stellung des Roboters und somit die des Tool-Center-Point (TCP) verändern.

Abbildung 1: Kawasaki FS003N

Abbildung 2: KUKA KR3

Man unterscheidet zwischen der Vorwärts- und der Rückwärtskinematik, welche inverse Kinematik genannt wird. Um die Bemaßung und Grundstellung des Roboters eindeutig zu beschreiben, nutzt man die Denavit-Hartenberg-Parameter. Hierzu wird in jedes Gelenk und in die Basis ein eigenes Koordinatensystem gelegt.

Anhand der Verdrehung und Verschiebung der Koordinatensysteme auf das Vorherige, lassen sich die vier DH-Parameter bestimmen:

- Drehwinkel θ, welcher die x-Achsen zweier Koordinatensysteme durch Drehung um die z-Achse des vorherigen Systems zusammenfallen lässt
- Translation d, welche die Abstände der Ursprünge, gemessen in Richtung der z-Achse, minimal werden lässt
- Translation a in Richtung der x-Achse, um die Abstände der Ursprünge der Koordinatensysteme zu minimieren
- Drehwinkel α um die x-Koordinate, so dass die z-Achsen ineinander gedreht werden

Dies wird anhand eines Beispiels des Zweigelenkroboters RV6-12, nach [4], veranschaulicht:

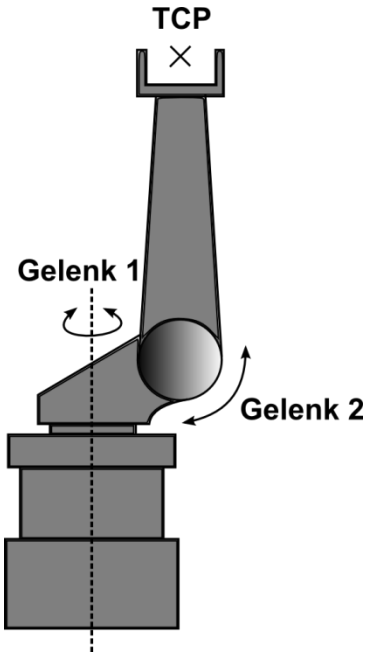

Abbildung 3: Zweigelenkroboter RV6-12

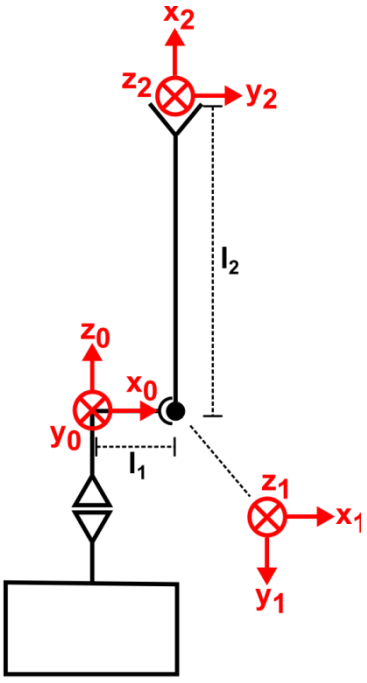

Abbildung 4: Koordinatensysteme des RV6-12

Tabelle 1: DH-Parameter des RV6-12

Gelenk	Θ in °	d in m	a in m	α in °
1	0	0	l_1	-90
2	-90	0	l_2	0

Mit Hilfe dieser Parameter lässt sich in der Vorwärtskinematik eine Frame-Matrix erstellen, welche die Rotation und Translation von dem vorrangegangenen Koordinatensystem zum Aktuellen wiedergibt. Um die neue Position und Verdrehung zu berechnen, wird die Frame-Matrix mit den aktuellen Koordinaten des Roboters multipliziert. So ist es möglich, durch Veränderung der Gelenkwinkel die Frames und damit die Stellung des TCP zu ändern. Die allgemeine Form einer Frame-Matrixist in Gleichung 1 aufgezeigt.

$$ {}^{i}_{i-1}\mathrm{T} = \begin{bmatrix} \cos(\theta_i) & -\sin(\theta_i)\cos(\alpha_i) & \sin(\theta_i)\sin(\alpha_i) & a_i\cos(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i)\cos(\alpha_i) & -\cos(\theta_i)\sin(\alpha_i) & a_i\sin(\theta_i) \\ 0 & \sin(\alpha_i) & \cos(\alpha_i) & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (1) $$

Die allgemeine Form stellt eine Frame-Matrix vom aktuellen Gelenk i in Bezug auf das vorrangegangene Gelenk i-1 dar. Für den in Abbildung 3 gezeigten Zweigelenkroboter ergibt sich für den TCP folgende Frame-Matrix:

$$ {}^{2}_{0}\mathrm{T} = {}^{1}_{0}\mathrm{T} * {}^{2}_{1}\mathrm{T} $$

Es werden somit die Rotation und Translation von der Roboterbasis zu Gelenk 2 (TCP) berechnet. Dabei ist die Multiplikation der vorrangegangenen Frame-Matrix von Gelenk 1 ebenfalls durchzuführen.

Anders als bei der Vorwärtskinematik wird bei der inversen Kinematik zuerst die Position des TCP vorgegeben. Der Tool Center Point muss sich im Arbeitsraum des Roboters befinden. Der Arbeitsraum stellt den Bereich dar, in dem sich der Roboter bewegen kann. Liegt die Position außerhalb des Arbeitsraums, so ist keine Bewegung dorthin durchführbar. Anschließend werden anhand geometrischer Berechnungen die einzelnen Gelenkwinkel bestimmt. Die Stellung des Roboterarms kann anschließend durch die Vorwärtskinematik beeinflusst werden.

2.2 Die MatlabKK-Robotic-Toolbox

Die MatlabKK-Robotic-Toolbox nach [1] bildet in dieser Arbeit die Grundlage für Steuerung der Roboter. Die MatlabKK-Visualization-Toolbox visualisiert die Kinematik von Kuka-Roboter und Kawasaki-Roboter. Die Steuerungssimulation basiert auf den Steuerungsbefehlen der Roboterhersteller. Per Simulation und Visualisierung entwickelte Steuerungsprogramme können dadurch direkt als reale Robotersteuerungen benutzt werden. Die Steuerungsbefehle der MatlabKK-Robotic-Toolbox bilden nach [1] 95% der Steuerungsbefehle der KUKA-KR2 sowie der Kawasaki AS Sprachen ab. Die Nutzung der Toolboxen erfolgt wie in Abbildung 5 dargestellt.

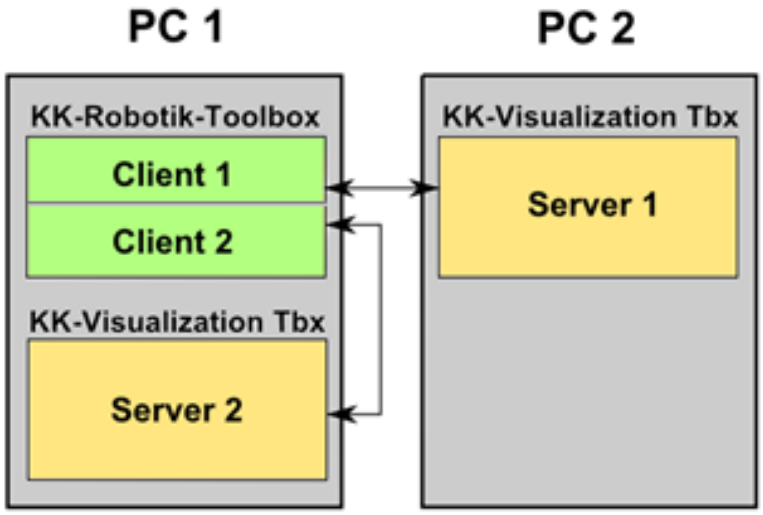

Abbildung 5: Kommunikation zwischen den Toolboxen

Es werden auf einem oder mehreren Rechnern mindestens zwei MATLAB-Instanzen erstellt. Dies bedeutet, dass MATLAB zweimal ausgeführt wird. Die erste Instanz symbolisiert den Visualisierungs-PC (Server), während die Zweite die Rolle des Control-PCs (Client) übernimmt. Der Nutzer kann über den Visualisierungs-PC die dazustellenden Objekte erzeugen und über den Control-PC die Steuerung der Roboterarme vornehmen.

2.3 Die MatlabKK-Visualization-Toolbox mit CAD-Schnittstelle

Die MatlabKK-Visualization-Toolbox nach [5] muss in Zusammenarbeit mit der MatlabKK-Robotic-Toolbox verwendet werden. Der Nutzer kann zwischen zwei Robotertypen wählen, welche er visualisieren und steuern kann. Für die grafische Darstellung benötigt das Programm die Datenpunkte der zu erstellenden Objekte. Diese werden gemäß [3] durch den Import von CAD-Objekten mit Hilfe der `STL_read` Funktion erstellt und automatisch bei der Wahl eines Robotertyps eingelesen. Bedingung dafür ist, dass das Modell als Stereolithographie-Datei (STL) abgespeichert ist.

Je nach Wahl des Robotertyps werden die vordefinierten Denavit-Hartenberg-Parameter verwendet. Somit sind für beide Robotertypen sowohl die DH-Parameter, als auch die Datenpunkte für die `patch` Funktion vorhanden. Die `patch` Funktion ist ein Befehl für die grafische Visualisierung von Datenpunkten in MATLAB, mit der Objekte in der MatlabKK-Visualization-Toolbox dargestellt werden. Der Nutzer kann der Visualisierung Umweltobjekte und Parts hinzufügen. Erstere sind starre, unbewegliche Körper. Parts sind Objekte, welche vom Roboter gegriffen und bewegt werden können.

Die Funktionsweise der Toolbox wird anhand eines Beispiels erläutert. Alternativ sind Weitere auf der CEA-Homepage [5] beschrieben.

Zuerst wird die Datenstruktur des entsprechenden Robotertyps in den Visualisierungs-Rechner mit Hilfe des Befehls `VirtualRobot.create(Type, Port, Pose, axis_property)` in MATLAB angelegt. Die Variable `Type` gibt an, ob es sich um einen Kawasaki- oder einen Kuka-Roboter handelt. Die nächste Variable definiert den Server-Port und `Pose` die Position der Basis des Roboters. `axis_property` ist ein optionaler Eingabewert. Mit ihm kann der Nutzer die Achseneinteilung festlegen. Wird dieser Wert nicht definiert, so legt das Programm diese automatisch fest.

Anschließend wird die Roboterdatenstruktur auf den Control-Rechner durch den Befehl `r1=robot(Key, Type, Connection, IP-Adress, Port)` erzeugt. Der `robot` Befehl liefert eine Handle-Variable auf der Datenstruktur, hier `r1`, zurück. Die `Key` Variable gibt an, ob die Verbindung zum Roboter geöffnet oder geschlossen ist, während die Wertzuweisungen von `Type` und `Port` von `VirtualRobot.create` übernommen werden müssen. `Connection` gibt an, ob es sich um eine serielle oder eine TCPIP-Verbindung handelt. Der Wert der Variable `IP-Adress` ist die IP-Adresse des Visualisierungsrechners. Für den Fall, dass die zwei MATLAB-Instanzen sich auf einem Rechner befinden, ist der Wert der Variable `'localhost'`.

Der Befehl `VirtualRobot.start_all` auf dem Visualisierungs-PC startet die grafische Darstellung. Der Nutzer kann vom Control-Rechner aus den Roboter steuern. Das Beispiel verwendet den Befehl `rmove(r1, 'home')`, welcher den Roboter mit der Kennzeichnung `r1` in die vordefinierte Position `home` steuert. Anschließend wird der Roboter durch Eingabe von `robot(r1, 'close')` geschlossen und die Visualisierung vom Visualisierungs-PC mit `VirtualRobot.delete_all` beendet.

Die Zusammenfassung der Befehle mit den definierten Variablen ist in Tabelle 2 veranschaulicht.

Tabelle 2: Befehlsabfolge des Beispiels

Befehlsabfolge	Visualisierungs-Rechner	Control-Rechner
1	VirtualRobot.create('Kawasaki', 40000, [0, 0, 0, 0, 0, 0])	
2		r1=robot('open', 'Kawasaki', 'tcpip', 'localhost', 40000)
3	VirtualRobot.start_all	
4		rmove(r1, 'home')
5		robot(r1, 'close')
6	VirtualRobot.delete_all	

Als Ergebnis der Eingabe visualisiert MATLAB, wie in Abbildung 6 gezeigt, den Kawasaki-Roboter in der Position `home`. Nach Ausführen des sechsten Befehls wird die MatlabKK-Visualization-Toolbox geschlossen.

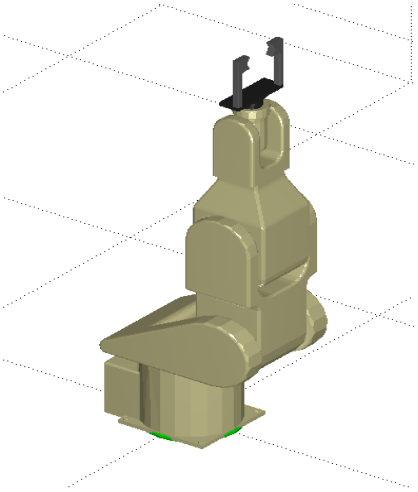

Abbildung 6: Visualisierung des Kawasaki FS003N

Der Nutzer könnte nach dem dritten Befehl zusätzlich Umweltobjekte und Parts einfügen, welche durch die `STL_read` Funktion importiert werden. Zu beachten ist, dass die Endung „.stl“ an die Namen der Dateien angefügt werden muss.

Die Toolbox verwendet die objektorientierte Programmierung (OOP) von MATLAB. Die Klasse `VirtualRobot` hat verschiedene Methoden, welche während der Ausführung der MatlabKK-Visualization-Toolbox aufgerufen werden und Eingangs- und Ausgabewerte besitzen.

2.4 Erweiterungsmöglichkeiten der MatlabKK-Visualization-Toolbox

Mit der MatlabKK-Visualization-Toolbox lassen sich Roboter und von ihnen gegriffene Parts visualisieren. Die Bewegung der Parts zwischen zwei Robotern kann jedoch zu Problemen führen.

Ein Problem stellt der begrenzte Arbeitsraum der Roboter dar. Sollen Parts über eine große Distanz transportiert werden, muss dies über mehrere aneinander gereihte Roboter realisiert werden. Diese Vorgehensweise ist aufwändig und in der Realität sehr kostenintensiv, weshalb hier eine Alternative gefunden werden muss.

Die Distanz könnte mit Hilfe eines Transportwagens oder eines Transportbands, wie in Abbildung 7 gezeigt, überwunden werden. Das Transportband sollte als Umweltobjekt einlesbar oder durch eine neu erstellte Methode automatisch generiert werden.

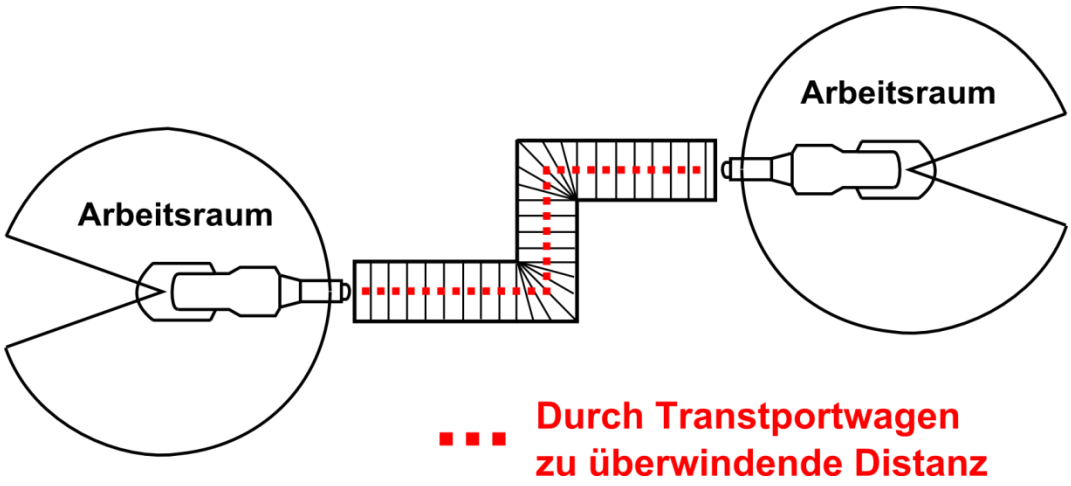

Abbildung 7: Arbeitsraum der Kawasaki-Roboter

Zudem sollte der Nutzer durch selbst gewählte STL-Dateien das Modell des Transportwagens festlegen können. Diese Option ermöglicht es, die Visualisierung an die entsprechenden Anforderungen oder an real existierende Transportwagen besser anpassen zu können.

Die nachfolgenden Kapitel behandeln die Integration von Transportwagen und und Transportbändern in die MatlabKK-Visualization Toolbox und deren virtuelle Steuerung mit der MatlabKK-Robotic Toolbox.

3 Erweiterung der MatlabKK-Visualization-Toolbox

In diesem Kapitel wird die Erweiterung der MatlabKK-Visualization-Toolbox gemäß der Problemanalyse in Kapitel 2.4 erläutert. Schwerpunkte bilden die Systemanalyse der Toolbox und die Erweiterung der vorhandenen Robotertypen `Kuka` und `Kawasaki` um einen Transportwagen. Anschließend wird das Einlesen eigener STL-Modelle vorgestellt, um die Visualisierung von Transportwagen flexibler zu gestalten.

3.1 Systemanalyse der MatlabKK-Visualization-Toolbox

Die Systemanalyse behandelt die Implementierung eines neuen Robotertyps in die Toolbox. Es wird die Kompatibilität der vorhandenen Kinematik für die Erweiterung um einen Transportwagen untersucht und die notwendigen Änderungen beschrieben.

Die Kinematik wird mit Hilfe von Frame-Matrizen realisiert, welche die DH-Parameter des Roboters voraussetzen. Der Transportwagen, auch `Cart` genannt, wird als Robotertyp implementiert. Es ist demnach notwendig die Kinematik der Roboter um das `Cart` zu erweitern. Um eine hohe Kompatibilität des Transportwagens mit den Funktionen der MatlabKK-Robotic-Toolbox zu gewährleisten, wird die grundlegende Kinematik beibehalten und die Frames werden mittels Fallunterscheidung angepasst.

Die Kuka- und Kawasaki-Roboter besitzen jeweils sechs Gelenke und sieben Armteile, wie es in Abbildung 8 zu sehen ist. Um die Verschiebung und Verdrehung von einem Gelenk zum Nächsten zu bestimmen, verwendet die Toolbox insgesamt acht Frames. Frame 1 berechnet die Verschiebung und Verdrehung vom Basiskoordinatensystem zur Basis des Roboters. Es folgt jeweils ein Frame für die

fünf Armteile und den Greifer. Das achte Frame berechnet die Orientierung des gegriffenen Parts, welches vom Roboter bewegt wird.

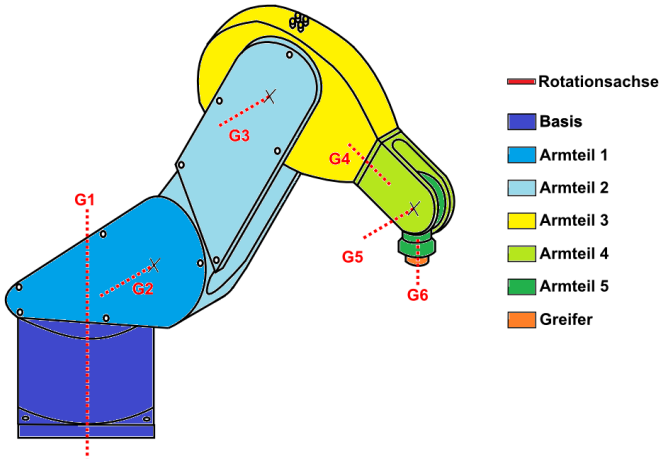

Abbildung 8: Aufbau eines Sechs-Achs-Roboters

Wie in Kapitel 2.1 beschrieben, verwenden die Frames die definierten DH-Parameter der Roboter. Da der Transportwagen nur aus einem Armteil besteht, wird auch nur ein Frame für die Berechnung der Verschiebungen und Verdrehungen benötigt. Das `Cart` hat keine eigenen DH-Parameter, weshalb das Frame aus der Rotationsmatrix mit den Verdrehungswinkeln a, b und c und den Verschiebungen x, y und z besteht.

$$R = \begin{bmatrix} \cos(a)\cos(b) & \cos(c)\sin(b) + \sin(c)\sin(a)\cos(b) & \sin(c)\sin(b) - \cos(c)\sin(a)\cos(b) \\ -\cos(a)\sin(b) & \cos(c)\cos(b) - \sin(c)\sin(a)\sin(b) & \sin(c)\cos(b) + \cos(c)\sin(a)\sin(b) \\ \sin(a) & -\sin(c)\cos(a) & \cos(c)\cos(a) \end{bmatrix}$$

Die Rotationsmatrix R wird in die Frame-Matrix eingesetzt:

$$T = \begin{bmatrix} & [R\ 3x3] & & \begin{matrix} x \\ y \\ z \end{matrix} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Da das `Cart` als Greifer betrachtet werden kann, wird die gezeigte Frame-Matrix als Frame 7 festgelegt und die Restlichen als Einheitsmatrizen definiert. Auf diese Weise können die Frames des Transportwagen an die Kinematik der Roboter angepasst werden.

3.2 Erweiterung der MatlabKK-Visualization-Toolbox um das Cart

Das `Cart` soll mit der `STL_read` Funktion nach MATLAB importiert werden. Da diese STL-Dateien voraussetzt, muss der Transportwagen mit Hilfe eines CAD-Programms modelliert und als STL-Datei exportiert werden. Das CAD-Objekt ist in Abbildung 9 mit der Bedienoberfläche des Exports in Creo Parametric 2.0 dargestellt.

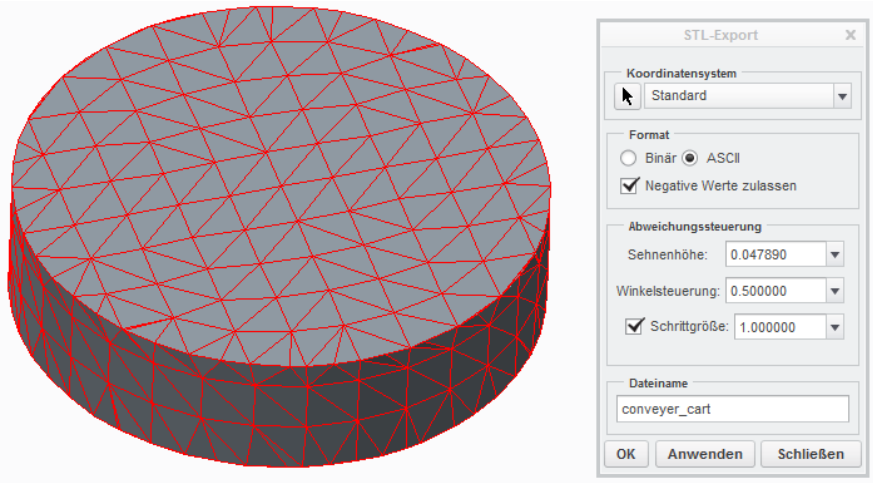

Abbildung 9: Darstellung des Carts in Creo Parametric 2.0

Die Bemaßung des Carts kann je nach Anwendung unterschiedlich definiert werden, indem das vordefinierte Maß mit einem Faktor `b_fak` multipliziert wird. Als vordefinierte Maße gelten eine Höhe von 2mm und ein Durchmesser von 10mm. Die Breite des Carts ist maximal als die 10-fache Breite der zu greifenden Parts festzulegen, da es sonst zu Fehlern bei deren Erkennung kommen kann.

Der Import des Modells erfolgt im Konstruktor `VirtualRobot`. Dieser wird beim Starten der MatlabKK-Visualization-Toolbox ausgeführt. Die Methode ist im Folgenden gezeigt:

```
obj = VirtualRobot(Type, port, pose, s_axis)
```

Beim Aufrufen des Konstruktors wird als Eingabeparameter der Typ des Roboters mit der Variablen `Type` übergeben. Gibt der Nutzer vor, dass der Robotertyp `Cart` ist, wird die in Abbildung 10 dargestellte Fallunterscheidung vom Programm ausgeführt. Die importierten Parameter werden in der Struktur `obj.Modell6` als Objekt des Greifers gespeichert.

```
case 'Cart'
    %Laedt den Wagen als Modell 6 (Greifer)
    [Typ ,Koord_cart, Faces_cart,Color_cart] = STL_read('cart.stl');
    obj.Modell6.Faces = Faces_cart;
    obj.Modell6.Vertices = Koord_cart;
    obj.Modell6.Color = Color_cart;

    %Vergrössern des Wagens
    b = property/10;
    obj.Modell6.Vertices = obj.Modell6.Vertices*b;
```

Abbildung 10: MATLAB-Code der `Cart` Fallunterscheidung im Konstruktor

Der Konstruktur wird um einen fünften Eingangswert `property` erweitert, welcher die gewünschte Breite des Carts darstellt. Es werden die Koordinaten des Transportwagens mit dem Faktor `b` multipliziert, um die Höhe und den Durchmesser zu vergrößern. Die Breite des Carts entspricht dem fünften Eingangswert des Konstruktors.

Neben dem Konstruktor sind 15 weitere Methoden an den Robotertyp anzupassen, darunter sechs für die Bewegung der Roboter und insgesamt sieben Start-, Positions- und Zeichenmethoden. Zudem ist die statische Methode `frame_cart` zur Berechnung der Frame-Matrix des Carts neu implementiert.

Die im Konstruktor ergänzte Eingangsvariable `property` muss auch in der `create` Methode, mit der die Roboter implementiert werden, definiert sein. Da nur der Typ `Cart` auf diese zugreift, wird die Breite des Carts standardisiert auf 100mm gesetzt, sollten weniger als fünf Eingangsvariablen definiert sein. Die Methode ist in Abbildung 11 dargestellt.

```
function create(Type,port,s_axis,property)
    %%% create
    ...

    if nargin < 5
        property = 100;
    end
```

Abbildung 11: Erweiterung der `create` Methode

In der Methode `start` werden die Frame-Matrizen der Roboter implementiert. Diese wird um eine Abfrage erweitert, ob die Eingangsvariable `obj.Type` den Wert `'Cart'` besitzt. Bei einer Verneinung werden die vorhandenen Frames der Roboter benutzt. Andernfalls werden die Verschiebungen `x`, `y` und `z` und die Verdrehungen `a`, `b` und `c` durch die Variablenstruktur `obj.pose` definiert. Da Frame-Matrizen `F(1)` bis `F(6)` keinen Einfluss auf die Kinematik des Transportwagens haben, werden diese als Einheitsmatrizen mit der `eye` Funktion implementiert. Die für die Roboter verwendete Methode `dh2Tr` benötigt als Eingabewerte die Denavit-Hartenberg-Parameter. Da der Transportwagen keine DH-Parameter besitzt, kann `F(7)` mit Hilfe der neuen, statischen Methode `frame_cart` bestimmt werden. `frame_cart` berechnet, wie in Kapitel 3.1 gezeigt, die Rotationsmatrix des Carts anhand der vorgegebenen Verdrehungen `a`, `b` und `c`. Anschließend wird die Transformationsmatrix mit den Verschiebungen `x`, `y` und `z` gebildet.

Es werden zudem die Gelenkverschiebungswerte für das Cart für die erste Darstellung definiert. Die Gelenkverschiebungswerte sind Korrekturwerte für die Position der Roboter. Die Verschiebungen der Koordinaten des Carts in y-, x- und z-Richtung und die Verdrehung sind jeweils 0. Die Fallunterscheidung in der Methode `start` ist in Abbildung 12 dargestellt.

```
if strcmp(obj.Type,'Cart') == 0
    %Wenn es sich um einen Roboter handelt:
    ...
else
    %Wenn es sich um ein Cart handelt:
    %Frame 1 bis 6 sind Einheitsframes, Frame 7 vom Cart
    %(Greifer) mit der Methode frame_cart definiert
    x = obj.pose(1);
    y = obj.pose(2);
    z = obj.pose(3);
    a = obj.pose(4);
    b = obj.pose(5);
    c = obj.pose(6);

    F(1).Frames = eye(4);
    F(2).Frames = eye(4);
    F(3).Frames = eye(4);
    F(4).Frames = eye(4);
    F(5).Frames = eye(4);
    F(6).Frames = eye(4);
    F(7).Frames = frame_cart(x,y,z,a,b,c);
end
```

Abbildung 12: Fallunterscheidung der `start` Methode

Die für die Bestimmung der Frame-Matrizen notwendige Methode `frame_cart` ist in Abbildung 13 gezeigt.

```
function [Frame]=frame_cart(x,y,z,a,b,c)
    %Neue Methode, welche das Frame für das Cart definiert, als
    %Eingabewerte dient die aktuelle Position

    a = a*pi/180;       sa=sin(a); ca=cos(a);
    b = b*pi/180;       sb=sin(b); cb=cos(b);
    c = c*pi/180;       sc=sin(c); cc=cos(c);

    R=[ -sa*sc+ca*cb*cc -sa*cc-ca*cb*sc ca*sb;...
         ca*sc+sa*cb*cc  ca*cc-sa*cb*sc sa*sb;...
        -sb*cc           sb*sc          cb];

    Frame =  [R(1,1) R(1,2) R(1,3) x;
              R(2,1) R(2,2) R(2,3) y;
              R(3,1) R(3,2) R(3,3) z;
              0      0      0      1];
end
```

Abbildung 13: MATLAB-Code von `frame_cart`

Das Cart muss zusätzlich in die Bewegungsmethoden der Roboter implementiert werden. Bei den sechs Methoden handelt es sich um:

- `rmove_pose_rel_ptp`
- `rmove_pose_ptp`
- `rmove_pose_rel_lin`
- `rmove_pose_lin`
- `rmove_axis_rel_lin`
- `romve_axis_lin`

Grundsätzlich darf die Kinematik der Roboter nicht verändert werden. Demnach müssen die Methoden durch Fallunterscheidungen an den Transportwagen angepasst werden. Da die Kinematik des Cart mit nur einem Frame berechnet wird, ist die Berechnung der inversen Kinematik mit der Methode `invkin` überflüssig. Die Methode bestimmt die einzelnen Verdrehungen der Gelenke der Roboter, wenn der TCP eine vorgegebene Orientierung erreichen soll. Als Gelenkverdrehungen ist die Position des Carts, welche die Verschiebungen und Verdrehungen der Koordinaten beinhaltet, anzunehmen. Demzufolge kann durch eine Abfrage die inverse Kinematik beim Typ `Cart` entfallen. Die Abfrage ist in Abbildung 14 am Beispiel von `rmove_pose_rel_ptp` aufgezeigt.

```
if strcmp(obj.Type,'Cart') == 0
    %Berechnung der Achswinkel in der Zielposition mittels inverser
    %Kinematik
    [theta_target] = obj.invkin(target_pos(1),target_pos(2),...
        target_pos(3),target_pos(4),target_pos(5),target_pos(6));
else
    %Bei dem Cart ist keine inverse Kinematik notwendig
    theta_target = target_pos;
end
```

Abbildung 14: Berechnung der inversen Kinematik in `rmove_pose_rel_ptp`

Diese Vorgehensweise kann für die Bewegungsmethoden `rmove_pose_ptp`, `rmove_pose_rel_lin` und `rmove_pose_lin` übernommen werden. Bei den Methoden `rmove_axis_rel_lin` und `rmove_axis_lin` sind die Frame-Matrizen auf das Cart anzupassen.

`frame_cart` benötigt die Winkel im Gradmaß, weshalb diese vor der Übergabe umgewandelt werden. Anschließend definieren die Methoden die Frames `F(1)` bis `F(8)` als Einheitsmatrizen, wobei `F(7)` durch `frame_cart` berechnet wird. Die Erweiterung ist in Abbildung 15 am Beispiel von `rmove_axis_rel_lin` zu sehen.

```
if strcmp(obj.Type,'Cart') == 0
    %Wenn es sich um einen Roboter handelt:
...
else
    %Wenn es sich um ein Cart handelt:
    x = theta_target(1);
    y = theta_target(2);
    z = theta_target(3);
    a = theta_target(4)*180/pi;
    b = theta_target(5)*180/pi;
    c = theta_target(6)*180/pi;

    F(1).Frames=eye(4);
    F(2).Frames=eye(4);
    F(3).Frames=eye(4);
    F(4).Frames=eye(4);
    F(5).Frames=eye(4);
    F(6).Frames=eye(4);
    F(7).Frames=frame_cart(x,y,z,a,b,c);
    F(8).Frames=eye(4);
end
```

Abbildung 15: Fallunterscheidung in `rmove_axis_rel_lin`

Die Methoden `rget_pose`, `correct_axis_abs`, `correct_axis_rel` und `correct_pose_abs` benutzen definierte Korrekturwerte für die Verschiebungen und Verdrehungen der Gelenke. Diese Werte werden durch Erweiterung einer implementierten Fallunterscheidung auf das Cart angepasst. Da keine vorangegangenen Verdrehungen oder Verschiebungen beim Transportwagen vorliegen, sind eventuelle Korrekturwerte zu entfernen. Dies wird in Abbildung 16 verdeutlicht und gilt für alle vier Methoden.

```
%Achsenkorrektur entsprechend der Abweichung zwischen Drehsinn
%und Nullposition durch die Dh-Parameter und der Realitaet.
switch obj.Type
    case 'Kawasaki'
        theta=[-a1 a2 a3 a4 a5 a6+67.5];
    case 'Cart'
        theta= [a1,a2,a3,a4,a5,a6];
    otherwise
        theta=-[a1 a2+90 a3-90 a4 a5 a6];
end
```

Abbildung 16: Erweiterung der Fallunterscheidung in `correct_axis_abs`

Die Berechnung der Achswinkelsätze erfolgt mit Hilfe der Methoden `draw_k` und `draw_axes`. Die Anpassung an das Cart erfolgt in `draw_k` erneut durch das Überspringen der Bestimmung der inversen Kinematik durch eine Abfrage, wie es bei den Bewegungsfunktionen der Fall ist. Die Methode `draw_axes` definiert die aktuellen Werte, indem die Methode `rget_pose` verwendet wird. Somit kann die Verwendung der Struktur `obj.data.act_theta`, welche für das Cart nicht definiert ist, vermieden werden. Die Änderung in `draw_axes` ist in Abbildung 17 dargestellt.

```
%aktuelle Achswinkel
if strcmp(obj.Type,'Cart') == 0
    theta_act = obj.data.act_theta;
else
    [k1,k2,k3,k4,k5,k6] = rget_pose(obj);
    theta_act = [k1,k2,k3,k4,k5,k6];
end
```

Abbildung 17: Achswinkelbestimmung in `draw_axes`

Die Zeichenmethode `Draw` muss an drei Stellen geändert werden. In dieser Methode werden mit Hilfe der importierten Koordinaten die Objekte visualisiert und das Frame des gegriffenen Objekts definiert. Da sich die Frame-Matrizen des Transportwages von denen der Roboter unterscheiden, sind diese durch eine Abfrage an das Cart anzupassen. Es werden die Verschiebungen `x`, `y` und `z` und die Verdrehungen `a`, `b` und `c` mit `axes_set` definiert.

Anschließend werden die Frames des Carts definiert. Das mit `frame_cart` definierte `F(7)` wird mit den importierten Koordinaten des Carts für die Methode `transf_achse` als Eingabewerte definiert. Als Ergebnis liefert die Methode die neuen Koordinatenwerte, die nach der Bewegung vorliegen. Diese Koordinaten werden zudem für das gegriffene Objekt übernommen. Die Implementierung des Frames des gegriffenen Parts ist in Abbildung 18 dargestellt.

```
%Cart (Greifer)
[xc7, yc7, zc7]=obj.transf_achse(F(7).Frames,...
    obj.Modell6.Vertices(:,1), obj.Modell6.Vertices(:,2),...
    obj.Modell6.Vertices(:,3));
obj.current_TCP_Frame=F(7).Frames;

%Keine Darstellungen des geschlossenen Greifers
%notwendig
if obj.clamped==1
    if obj.picked~=0
        %Darstellung des gegriffenen Objektes
        parts(obj.picked).F_Rob_in_0=F(7).Frames;
        parts(obj.picked).Draw;
    end
end
```

Abbildung 18: Transformation des Greifers in `Draw`

Die Visualisierung der Objekte erfolgt mittels der `patch` Funktion. Es werden alle Armteile der Roboter dargestellt. Da das Cart aus nur einem Armteil besteht, muss eine Fallunterscheidung getroffen werden.

Das erste Ausführen der `Draw` Methode ruft einmalig die `patch` Funktion auf. Bei jeder neuen Berechnung der Koordinaten der darzustellenden Objekte ist die Visualisierung durch die `set` Funktion zu aktualisieren. Die Position des TCP wird als Position des Carts, welches gleichzeitig der Greifer ist, definiert. Die Fallunterscheidung ist in Abbildung 19 veranschaulicht.

```
if strcmp(obj.Type,'Cart') == 0
    %Zeichnung der Roboter
    ...
else
    %Zeichnung des Carts
    Cart = [xc7,yc7,zc7];

    obj.current_TCP.x=xc7;
    obj.current_TCP.y=yc7;
    obj.current_TCP.z=zc7;

    if isempty(obj.mdl_cart)
        fc = [120 120 120]/255;
        obj.mdl_cart = patch('vertices' ,Cart,'faces', obj.Modell6.Faces);
        set(obj.mdl_cart,'FaceColor',fc,'EdgeColor','none',...
            'FaceLighting','gouraud');
    else
        %Verändert die Koordinaten
        set(obj.mdl_cart,'Vertices',Cart);
    end
end
```

Abbildung 19: Zeichnung des Carts in `Draw`

3.3 Implementierung von New_Robot

Um das Modell des Transportwagens auf die speziellen Anwendungsbereiche des Nutzers oder real existierende Transportwagen anzupassen, ist der Robotertyp `New_Robot` implementiert. Mit diesem ist es möglich, dass der Nutzer eigene STL-Objekte als Modell des Transportwagens festlegt. Die Fallunterscheidung von `New_Robot` ist im Konstruktor implementiert. Um die zu visualisierende STL-Datei einzulesen, kann der Anwender den Namen der Datei als Eingabewert `property` beim Konstruktor hinzufügen. Die Definition der Variable `s_axis` ist nicht erforderlich, da diese standardisiert implementiert wird. Die Variable gibt die oberen und unteren Grenzen der Achsen in der Visualisierung an. Zudem kann der Name der STL-Datei auch als vierte Eingangsgröße angegeben werden. In diesem Fall wird `s_axis` auch standardisiert definiert. Die Implementierung der Eingangsvariablen ist in Abbildung 20 gezeigt.

```
function obj=VirtualRobot(Type,port,pose,s_axis,property)
            %%% Konstruktor
            ...

            if strcmp(Type,'New_Robot') == 1
                if nargin<5
                    property = s_axis;
                    s_axis=[-1250 1250 -1250 1250 0 1000];
                end
            end
```

Abbildung 20: Implementierung der Eingangsvariablen

Nach der Fallerkennung von `New_Robot` wird der Name der STL-Datei, in dem das Objekt exportiert wurde, eingelesen. Es ist darauf zu achten, dass der Name in Anführungszeichen steht und die Endung „.stl“ angehängt wird. Die Parameter werden, wie beim Transportwagen, in der Variablen `obj.Modell6` gespeichert und der Robotertyp zu `Cart` verändert. Somit wird die Kinematik von `Cart` übernommen und das eingelesene Objekt kann als Transportwagen verwendet werden. Die Fallunterscheidung ist in Abbildung 21 aufgezeigt.

```
case 'New_Robot'
    % Erzeugt ein Cart aus einem vom Nutzer vorgegebenen
    % Modell
    % Laedt neues STL-Modell ein, Namen mit Anfuehrungs-
    % zeichen eingeben, Modell wird in der im CAD-Programm
    % festgelegten Groesse visualisiert

    %Laedt den Roboter als Cart-Modell 6 (Greifer)
    [Typ ,Koordinaten_cart, Faces_cart,Color_cart] = STL_read(property);
    obj.Modell6.Faces = Faces_cart;
    obj.Modell6.Vertices = Koordinaten_cart;
    obj.Modell6.Color = Color_cart;

    % Übernimmt die Eigenschaften/Kinematik vom Cart
    obj.Type = 'Cart';
```

Abbildung 21: MATLAB-Code der `New_Robot` Fallunterscheidung

3.4 Anwendungsbeispiele

In den beiden nachfolgenden Unterabschnitten werden jeweils ein Anwendungsbeispiel für den Typ `Cart` sowie den Typ `New_Robot` aufgezeigt.

3.4.1 Anwendungsbeispiel mit Cart

Dieses Beispiel verdeutlicht die Verwendung eines Carts mit Hilfe der MatlabKK-Robotic-Toolbox und der MatlabKK-Visualization-Toolbox. Die Befehlsabfolge ist in Tabelle 3 dokumentiert.

Tabelle 3: Befehlsabfolge des Beispiels mit `Cart`

Befehlsabfolge	Visualisierungs-Rechner	Control-Rechner
1	VirtualRobot.create('Cart', 40000, [0, 0, 0, 0, 0, 0],100)	
2		c1=robot('open', 'Kawasaki', 'tcpip', 'localhost', 40000)
3	VirtualRobot.start_all	
4		P=rpoint(400, 0, 0, 0, 0, 0, 'axis', 'rel', 'ptp');
5		rmove(c1,P)
6		robot(c1, 'close')
7	VirtualRobot.delete_all	

Der erste Schritt beinhaltet das Starten der Visualisierungs-Toolbox. Die Variable `Type` in der `create` Methode ist als `'Cart'` definiert, damit ein Transportwagen visualisiert wird. Der Control-Rechner lädt die Robotic-Toolbox mit der Funktion `robot` und weist dem Cart das Handle `c1` zu. Durch `VirtualRobot.start_all` wird die Visualisierung gestartet. Da die Kinematik der Roboter auf das Cart angepasst ist, sind die Befehle der Robotic-Toolbox ausreichend. `rpoint` erstellt eine Variablenstruktur, in der das Cart sich um 400mm von der aktuellen Position in x-Richtung bewegt. Die Definition der Koordinaten mit `'axis'` hat auf die Position keinen Einfluss.

Sowohl `'axis'`, als auch `'pose'` berechnen denselben Punkt, da das Cart aus nur einem Modell besteht und keine Achsen besitzt. Die Eingabewerte werden bei beiden Definitionen als die Verschiebungen x, y und z und Verdrehungen a, b und c der Koordinaten definiert.

Der Wert `'ptp'` bedeutet, dass eine Bewegung von der aktuellen zur Zielposition vorgenommen wird. Die Funktion `rmove` führt die Bewegung des Carts zu der Position aus. Anschließend wird die Robotic-Toolbox geschlossen und die Visualisierung mit `VirtualRobot.delete_all` beendet. Ein Ausschnitt der Visualisierung ist in Abbildung 22 zu sehen.

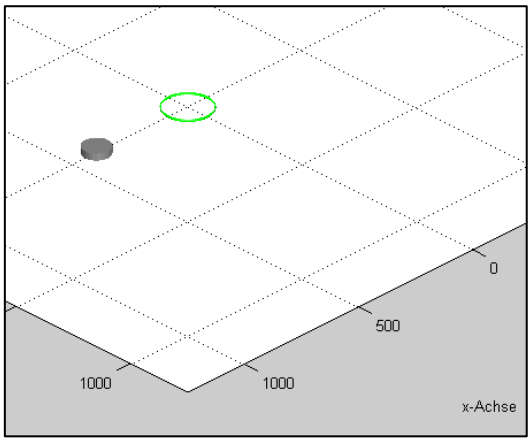

Abbildung 22: Visualisierung des Beispiels mit `Cart`

3.4.2 Anwendungsbeispiel mit New_Robot

In diesem Beispiel wird verdeutlicht, wie selbst erstellte STL-Modelle als Transportwagen eingelesen werden können. Dies ermöglicht es dem Nutzer, das Modell des Carts auf seine Anwendungen anzupassen. Die Befehlsabfolge ist in Tabelle 4 gezeigt.

Tabelle 4: Befehlsabfolge des Beispiels mit `New_Robot`

Befehlsabfolge	Visualisierungs-Rechner	Control-Rechner
1	VirtualRobot.create('New_Robot', 40000, [0, 0, 0, 0, 0, 0], 'conveyer_car.stl')	

2		c2=robot('open', 'Kawasaki', 'tcpip', 'localhost', 40000)
3	VirtualRobot.start_all	
4		P=rpoint(400, 0, 0, 0, 0, 0, 'pose', 'abs', 'ptp');
5		rmove(c2,P)
6		robot(c2, 'close')
7	VirtualRobot.delete_all	

Als Typ wird im ersten Schritt `'New_Robot'` definiert. Im Beispiel wird das Modell `'conveyer_car.stl'` aufgerufen, welches eine Ablagefläche mit befestigten Rollen darstellt. Das Objekt wurde in Creo Parametric 2.0 erstellt und nach MATLAB importiert. Die Endung „.stl“ und die Übergabe in Form eines Skripts müssen vorhanden sein. Die Funktion `robot` öffnet die Robotic-Toolbox und `VirtualRobot.start_all` staret die Visualisierung. `rpoint` erstellt eine Position, welche sich 400mm in positiver Richtung auf der X-Achse befindet. Die Variable `'abs'` gibt an, dass der Punkt ohne Relation zur aktuellen Position erstellt wird. Die Koordinaten beziehen sich auf das Welt-Koordinatensystem. Die Funktion `rmove` führt die lineare Bewegung zu dem definierten Punkt aus. Die letzten beiden Befehle schließen die Visualisierung und Abbildung 23 zeigt einen Ausschnitt der Visualisierung.

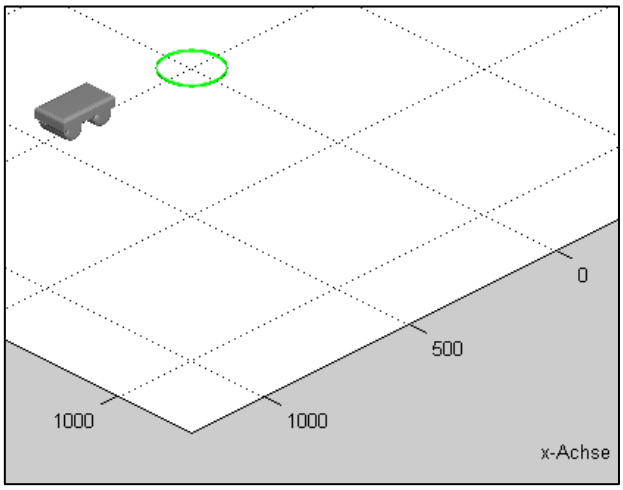

Abbildung 23: Visualisierung des Beispiels mit `New_Robot`

Bei dem STL-Export der dazustellenden Objekte ist auf die Triangulation der Modelle zu achten. Bei dem Greifen von in der Nähe befindlichen Parts wird überprüft, ob sich Koordinatenpunkte des Transportwagens mit den Grenzen des Parts überschneiden. In einigen Fällen kann das Part vom Transportwagen berührt, aber nicht gegriffen werden. Ein Vergleich für eine gute und eine ungünstige Triangulation ist in Abbildung 24 zu sehen. Wenn ein Objekt in der Mitte des Carts positioniert ist, kann es passieren, dass dieses zwar die Flächen des Carts schneidet, jedoch nicht die Eckpunkte der Dreiecke. Es ist somit darauf zu achten, dass gleichmäßig über den Transportwagen Koordinatenpunkte verteilt sind. Dies kann mit dem Einstellen einer Schrittweite beim STL-Export definiert werden.

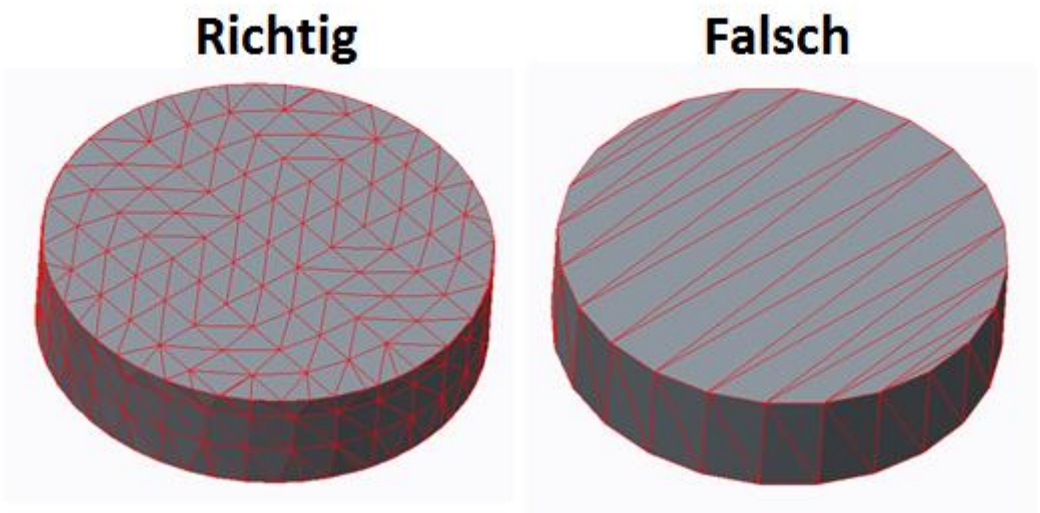

Abbildung 24: Problembeispiel für den STL-Export

In der Abbildung ist zu erkennen, dass das linke Objekt eine gute Vernetzung auf der Oberfläche besitzt. Die Chance, dass ein Part erkannt wird, ist erheblich größer als bei dem rechten Objekt. Sobald ein Part klein genug ist, um die Randpunkte des rechten Objekts nicht zu schneiden, wird es nicht erkannt.

4 Erweiterung der MatlabKK-Visualization-Toolbox um Transportbänder

Dieses Kapitel befasst sich mit der Implementierung einer `create_line` Methode in die MatlabKK-Visualization-Toolbox, welche es den Nutzer ermöglicht, automatisch Transportbänder zu visualisieren und virtuell zu steuern. Zudem wird ein Beispiel zur Anwendung dieser Methode vorgestellt.

4.1 Systemanalyse der theoretischen Grundlagen

Das Transportband soll durch Vorgabe von Punkten definiert werden. Diese sind vom Nutzer festgelegt und geben an, wo sich die Eckpunkte des Transportbandes befinden. Das Transportband besteht aus zwei Bestandteilen, dem Start- und Endpunkt und den Transportbandabschnitten. Die Objekte der Bauteile sind in dem CAD-Programm Creo Parametric 2.0 modelliert und lassen sich durch die Funktion `STL_read` nach MATLAB importieren. In Abbildung 25 sind die MATLAB-Modelle für `'conveyer_line.stl'` links und `'conveyer_start.stl'` rechts dargestellt. Die Breite ist mit 100mm definiert. Die Farben der Ecklinien sind zur Veranschaulichung auf `'black'` gesetzt.

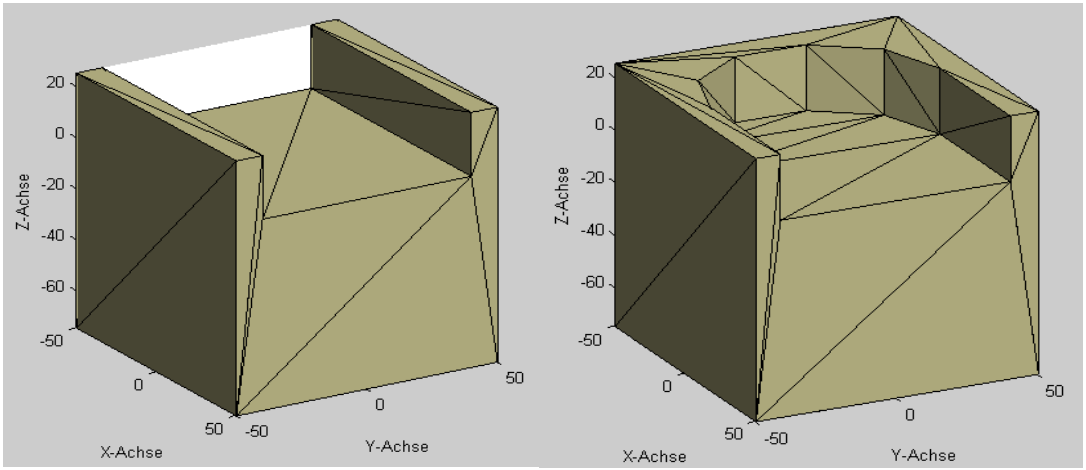

Abbildung 25: Darstellung der Transportband-Objekte in MATLAB

Die Bemaßungen sind an den Durchmesser `b` des Transportwagens angepasst. Standardisiert ist eine Höhe von 75mm mit 25mm hohen Rändern vorgegeben. Die Dicke der Ränder betragen ein Zehntel des Durchmessers, während die Breite der Modelle mit 1.4*`b` bestimmt sind. Die Größenrelationen des Transportbands im Bezug zum Cart sind in Abbildung 26 skizziert.

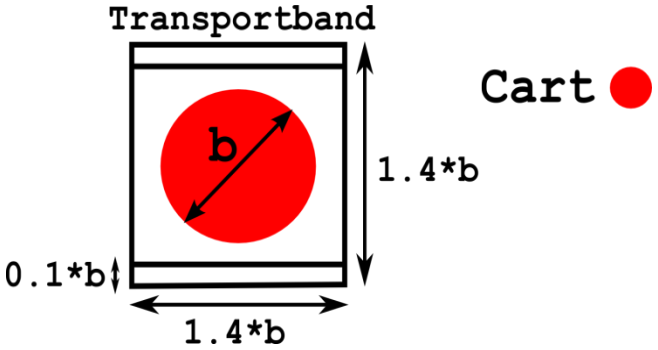

Abbildung 26: Bemaßung des Transportbands in Relation zum Cart

Im ersten Schritt werden mit der Methode `create_line` die Punkte, durch die das Transportband verbunden ist, definiert. Es müssen mindestens drei Positionen definiert werden. Eine maximale Anzahl der Punkte gibt es nicht. Zwischen jeweils zwei Positionen kann eine Gerade gelegt werden, welche einen Winkel `phi` zum Welt-Koordinatensystem besitzt. Als Beispiel wird die Visualisierung eines Transportbands, welches durch drei Punkte definiert ist, in Abbildung 27 vorgestellt. Die blau gekennzeichneten Linien stellen die Geraden g1 und g2 dar, über die das Transportband gelegt werden soll.

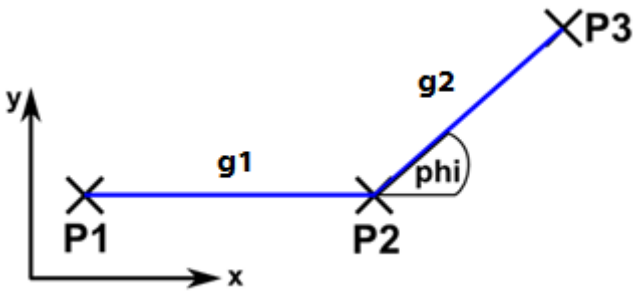

Abbildung 27: Definition der Punkte und der Winkel `phi`

Nach der Definition der drei Punkte werden jeweils an der ersten und an der letzten Position der Start- und Endpunkt mit dem Modell `'conveyer_start.stl'` erzeugt. Durch die Drehung um den Winkel `phi` können die Modelle in Richtung der Geraden ausgerichtet werden. Das Ergebnis ist in Abbildung 28 skizziert.

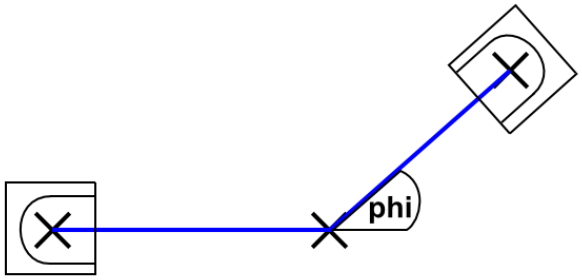

Abbildung 28: Visualisierung der Start- und Endpunkte

Das eigentliche Transportband wird erstellt, indem das Modell des Transportbands, `'conveyer_line.stl'`, in die Mittelpunkte der Geraden gesetzt wird und die Eckpunkte zu den Enden der Geraden extrudiert werden. Die `create_line` Methode muss unterscheiden, ob sich an den Enden Start- und Endpunkte oder der Schnittpunkt mit einer anderen Geraden befindet. Sollte das Transportband eine geschlossene Linie bilden und der erste und letzte vorgegebene Punkt sind identisch, sind keine Start- und Endpunkte zu visualisieren. Abbildung 29 skizziert die Vorgehensweise.

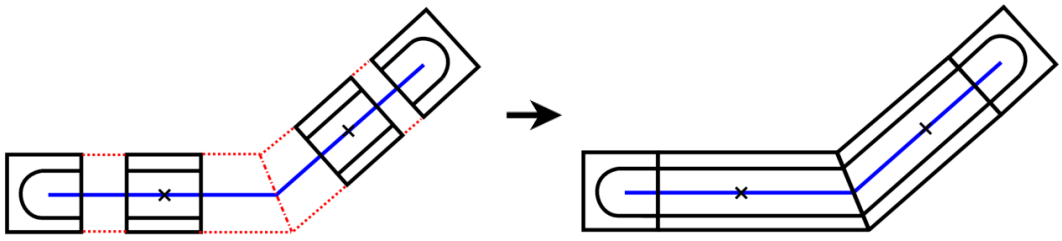

Abbildung 29: Extrudieren der Eckpunkte der Transportbänder

Das Extrudieren der Eckpunkte erfolgt mit der allgemeinen Gleichung:

Koordinaten_neu = Koordinaten_alt + (Extrudierlänge * Richtungsvorzeichen)

Eine ähnliche Vorgehensweise wird bei der Erstellung von Rampen verwendet, welche Höhenunterschiede in Richtung der z-Achse überwinden können. Die Transportbandmodelle werden in den Mittelpunkt M der Geraden gelegt, aber werden die Eckpunkte nicht extrudiert. Die neuen Eckpunkte sind als die vom Nutzer vorgegebenen Positionen definiert. Die Eckpunktkoordinaten der angrenzenden Transportbandabschnitte bleiben unverändert. Als Bedingung ist definiert, dass sich lediglich die x- oder die y-Koordinaten ändern. Es ist keine Implementierung von Rampen, die eine Änderung in x-, y- und z-Richtung haben, vorgesehen.

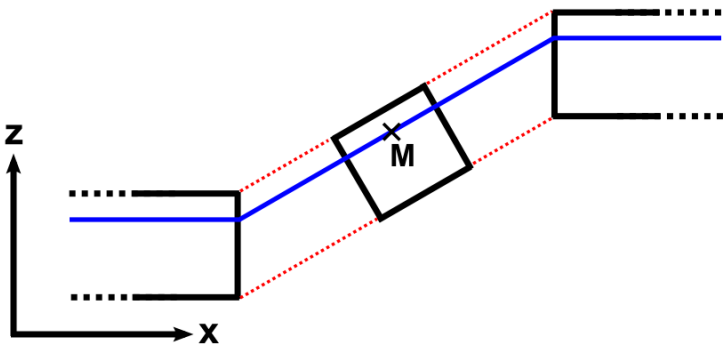

Abbildung 30: Definieren der Eckpunkte einer Rampe

Da die Koordinaten der Transportbandabschnitte unterschiedlich lang extrudiert werden, muss eine Unterscheidung der Eckpunkte getroffen werden. Das importiere STL-Objekt `'conveyer_line.stl'` besteht aus insgesamt 16 Eckpunkten, welche insgesamt 84 Koordinaten besitzen. Somit sind jedem der 16 Eckpunkte vier bis sieben Koordinaten zugeordnet. Der Grund ist die Zusammensetzung der dreidimensionalen STL-Objekte aus einer Vielzahl von Dreiecken. An den 16 Ecken können sich die Koordinaten mehrerer Dreiecke schneiden. Die Anzahl der Koordinaten entspricht den Eckpunkten der sich schneidenden Dreiecke. Um den Transportbandabschnitt zu extrudieren, müssen die Eckpunkte mit den dazugehörigen Koordinaten neu definiert werden. Zudem muss untersucht werden, in welcher Ausrichtung sich das Objekt befindet, um die

Punkte in die korrekte Richtung zu extrudieren. Die 16 Eckpunkte des Modells und die dazugehörigen Koordinaten sind in Abbildung 31 skizziert.

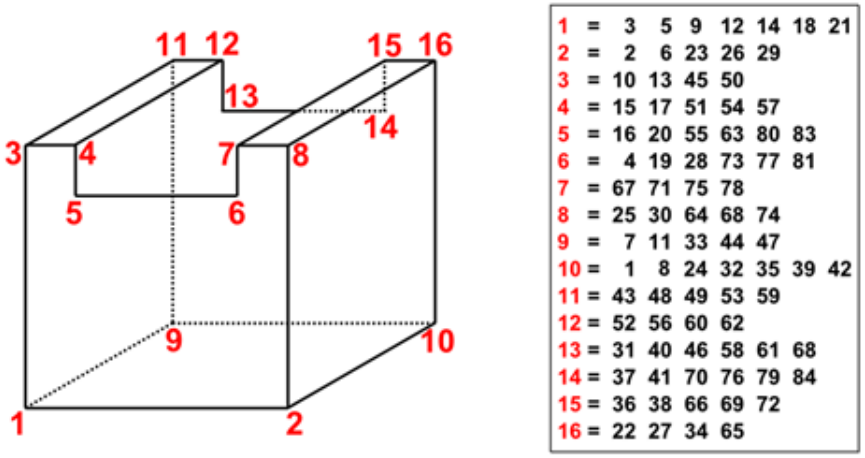

Abbildung 31: Koordinaten der Transportbandabschnitte

Die Abbildung zeigt, welcher Eckpunkt durch welche Koordinaten der Dreiecke definiert ist. Um Beispielsweise den Eckpunkt 3 zu extrudieren, müssen die Koordinaten 10, 13, 45 und 50 des eingelesen STL-Modells verändert werden. Die Funktionsweise der `create_line` Methode ist in Abbildung 32 dargestellt.

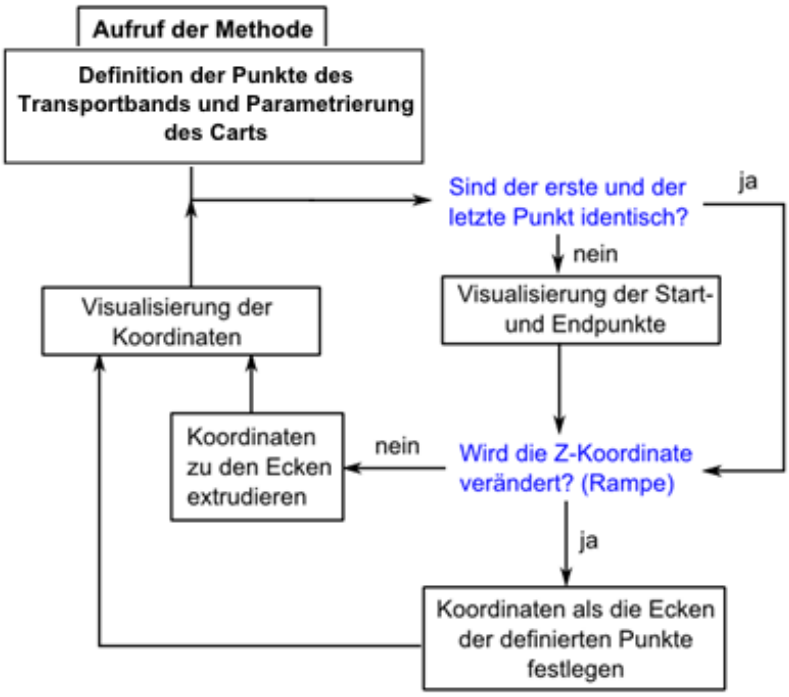

Abbildung 32: Algorithmus der `create_line` Methode

4.2 Implementierung der create_line Methode in die Visualization-Toolbox

Die theoretische Vorgehensweise aus Kapitel 4.1 ist in der statischen Methode `create_line(b,P)` implementiert. Als Eingabewerte werden der gewünschte Durchmesser `b` des Transportwagens und die Positionen der gewünschten Punkte des Transportbands als Matrix `P` übergeben.

Das erstellte Transportband soll ein Objekt sein, auf dem sich das Cart in der Visualisierung bewegen kann. Deshalb wird zu Beginn der Methode der Durchmesser des Carts festgelegt. Die Bemaßung des Transportbands ergibt sich durch die in Abschnitt 4.1 erklärte Parametrierung. Da das importierte Cart Modell eine Breite von 10mm besitzt, muss das Transportband um einen neu definierten Faktor `b_fak`, welcher einem Zehntel der definierten Breite `b` entspricht, vergrößert werden. Für die eindeutige Zuordnung der Koordinaten zu einem Eckpunkt wird eine Matrix erzeugt, welche die Eckpunkte des Transportbandmodells als Zeilen und die dazugehörigen Koordinaten als Spalten beinhaltet. Die Matrix ist in Abbildung 33 dargestellt.

```
% Jede Zeile entspricht einem Punkt
% Jede Spalte einer Koordinate
VP = [3,  5,  9,  12, 14, 18, 21;
      2,  6,  23, 26, 29,  0,  0;
      10, 13, 45, 50,  0,  0,  0;
      15, 17, 51, 54, 57,  0,  0;
      16, 20, 55, 63, 80, 83,  0;
      4,  19, 28, 73, 77, 81,  0;
      67, 71, 75, 78,  0,  0,  0;
      25, 30, 64, 68, 74,  0,  0;
      7,  11, 33, 44, 47,  0,  0;
      1,   8, 24, 32, 35, 39, 42;
      43, 48, 49, 53, 59,  0,  0;
      52, 56, 60, 62,  0,  0,  0;
      31, 40, 46, 58, 61, 82,  0;
      37, 41, 70, 76, 79, 84,  0;
      36, 38, 66, 69, 72,  0,  0;
      22, 27, 34, 65,  0,  0,  0];
```

Abbildung 33: Koordinatenmatrix der `create_line` Methode

Nicht mit Koordinaten zugewiesene Elemente sind mit dem Wert null definiert, da eine Matrix in jeder Zeile die gleiche Anzahl an Spalten besitzen muss.

Sollten die erste und letzte Zeile der vom Nutzer definierten Matrix P nicht identisch sein, werden die Start- und Endpunkte visualisiert. Die Überprüfung erfolgt mit einer `if` Abfrage, welche die drei Spalten der Zeilen einzeln überprüft. Mit der Funktion `STL_read` wird die STL-Datei `'conveyer_line.stl'` eingelesen und die Koordinaten und Faces des Objekts definiert. Als Beispiel ist das Erstellen des Startpunkts in Abbildung 34 veranschaulicht.

```
if P(1,1) ~= P(end,1) || P(1,2) ~= P(end,2) || P(1,3) ~= P(end,3)
    ...
    SEP = 0;
    % ==> Erstellen des Startpunkts
    % Einlesen des Objekts
    [Typ, K_start, F_start] = STL_read('conveyer_start.stl');

    % Startpunkt muss eventuell um phi_s gedreht werden
    x = P(2,1)-P(1,1);
    y = P(2,2)-P(1,2);
    phi_s = atan2(x,y);
    phi_s = phi_s - (pi/2);

    % Positionieren und Skalieren des Startpunkts
    K_start = K_start * rotz(phi_s);
    K_start(:,1) = K_start(:,1)*b_fak+P(1,1);
    K_start(:,2) = K_start(:,2)*b_fak+P(1,2);
    K_start(:,3) = K_start(:,3)+P(1,3);

    % Visualisierung des Startpunkts
    obj_start = patch('Vertices',K_start,'Faces',F_start);
    set(obj_start,'Edgecolor','none','Facecolor',Color,...
        'FaceLighting','gouraud');
```

Abbildung 34: Erstellen des Startpunkts in `create_line`

Die Abfrage erfolgt durch den Vergleich des ersten Punkts `P(1)` mit dem letzten Punkt `P(end)` der vom Nutzer festgelegten Positionen. Die Variable `SEP` gibt an, ob der Start- mit dem Endpunkt zusammenfällt. Sie ist für die spätere Fallunterscheidung beim Extrudieren erforderlich.

Der Winkel, um den die Modelle gedreht werden, wird mit Hilfe von `atan2` bestimmt. Dies hat den Vorteil gegenüber anderen Winkelfunktionen, dass sich das Ergebnis in allen vier Quadranten des Koordinatensystems berechnen kann. Somit ist das Vorzeichen des Winkels angegeben und eine zusätzliche Fallunterscheidung entfällt. Die Koordinaten der Start- und Endpunkte werden um den berechneten Winkel um die Z-Achse gedreht ,mit dem Faktor `b_fak` skaliert durch Addition der definierten Punkte positioniert. Um die Enden des Transportbandes zu visualisieren, werden die Koordinaten und die Faces der `patch` Funktion übergeben. Die Funktion `set` verändert die Farbe und die Beleuchtung der Objekte.

Die Winkel zwischen allen Positionen werden in dem Vektor `phi` mit der Funktion `atan2` analog zum Beispiel auf Seite 32 bestimmt. Die Funktion `STL_read` definiert die Koordinaten `K_line` und Faces der Transportbandmodelle. Um die Positionierung im Mittelpunkt der Geraden zwischen zwei Punkten vorzunehmen, werden die mittleren Strecken `xm`, `ym` und `zm` berechnet. Diese sind, wie in Abbildung 35 gezeigt, nach dem Einlesen des Transportbandmodells bestimmt.

```
% Einlesen der Line
[Typ, K_line, F_line] = STL_read('conveyer_line.stl');

% Mittelpunkt berechnen, von dem die Lines extrudiert werden
xm = (P(i+1,1)-P(i,1))/2;
ym = (P(i+1,2)-P(i,2))/2;
zm = (P(i+1,3)-P(i,3))/2;
```

Abbildung 35: Berechnung der Mittelwerte und Einlesen des Modells

Anschließend werden die Modelle um den ihnen zugeordneten Winkel `-phi` gedreht und mit dem Faktor `b_fak` skaliert. Der Winkel ist negativ, da die anfängliche Ausrichtung des Modells verdreht ist. Die Positionierung erfolgt mittels der Addition von den Positionen mit den Mittelwerten und ist in Abbildung 36 gezeigt.

```
% Positionieren und Skalieren der Lines
% Es erfolgt nur eine Drehung um die z-Achse
% Drehung um x und y später in case 0
K_line = K_line * rotz(-phi(i));
K_line(:,1) = K_line(:,1)*b_fak+(P(i,1)+xm);
K_line(:,2) = K_line(:,2)*b_fak+(P(i,2)+ym);
K_line(:,3) = K_line(:,3)+P(i,3);
```

Abbildung 36: Positionieren und Skalieren der Transportbandabschnitte

Für das Extrudieren auf der Ebene sind die zu extrudierenden Längen `r` und `r2` implementiert. Die Variable r ist die Länge, um die der Transportbandabschnitt gedehnt werden muss, um an die vom Nutzer vorgegebenen Punkte anzuschließen. Die Hypotenuse r berechnet sich mit Hilfe des Satzes des Pythagoras mit den Mittelwerten `xm` und `ym` und ist in Abbildung 37 skizziert.

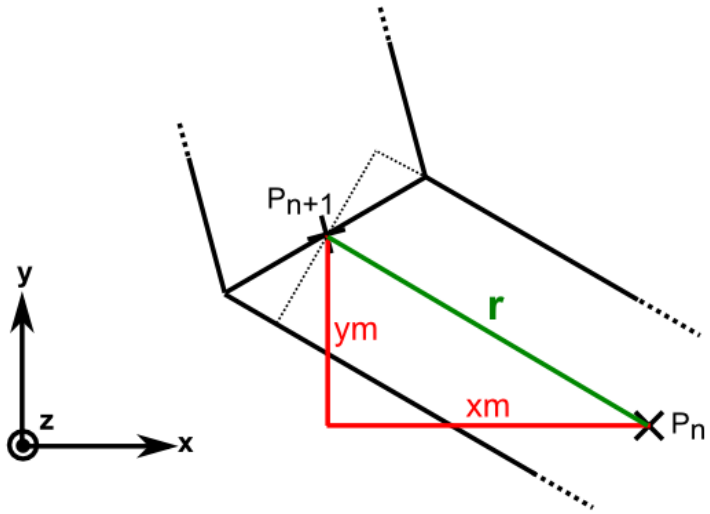

Abbildung 37: Darstellung der Länge `r`

Die Länge `r2` ist von dem Winkel `alpha` und der Breite `s` abhängig. Die Variable `s` gibt an, um wie viele Millimeter die Extrusion korrigiert werden muss, um die schräge Kante der Ecke zu erzeugen. Jedem Eckpunkt des Transportbandabschnitts ist eine definierte Breite `s` zugeordnet. Je nach Punkt ist die Breite von -0.7*`b` bis +0.7*`b`. Die Bestimmung von `r2` erfolgt mit Hilfe des Tangenssatzes. In Abbildung 38 ist eine Ecke und deren Parameter `s`, `alpha` und die daraus resultierende Länge `r2` skizziert.

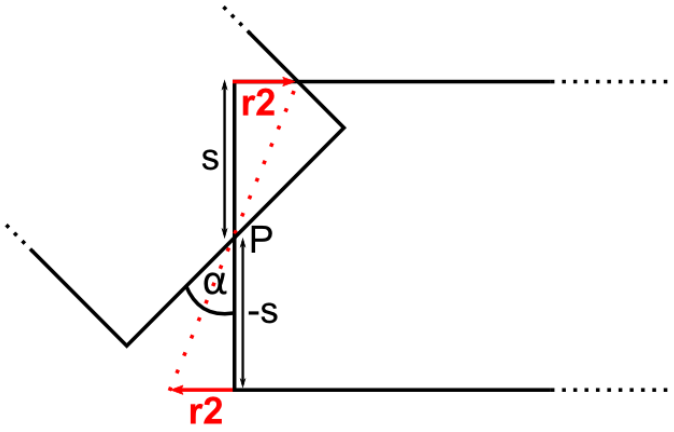

Abbildung 38: Skizze der Breite s

Die Extrusionen müssen für alle Eckpunkte des Transportbandabschnittes erfolgen. Da diese in ihrer Ausrichtung verschieden sein können, werden die Punkte 1 bis 8 und 9 bis 16 durch Fallunterscheidung mit unterschiedlichen Längen extrudiert. Die Extrusion des ersten Transportbandabschnittes ist in Abbildung 39 dargestellt.

```
for k = 1:8
    for l = 1:length(VP(k,:))
        Zeile = VP(k,l);
        if Zeile ~= 0
            % Aenderung der Koordinaten
            switch k
                case 1
                    s = 0.7*b;
                ...
                case 8
                    s = -0.7*b;
            end
            % Wenn Start- und Endpunkt zusammenfallen
            % wird Fallunterscheidung betrieben
            if SEP == 0
                r2 = tan(alpha/2)*s;
                K_line(Zeile,1) = K_line(Zeile,1)-(r+S18+f18*r2)*cos(phi(i));
                K_line(Zeile,2) = K_line(Zeile,2)-(r+S18+f18*r2)*sin(phi(i));
            else
                r2 = tan(alpha/2)*-s;
                K_line(Zeile,1) = K_line(Zeile,1)-(r+S916+f916*r2)*cos(phi(i));
                K_line(Zeile,2) = K_line(Zeile,2)-(r+S916+f916*r2)*sin(phi(i));
            end
        end
    end
end
```

Abbildung 39: Extrusion des ersten Transportbandabschnittes

Zudem wird bei dem Erstellen des ersten und letzten Abschnitts eine zusätzliche Abfrage getroffen, ob ein Start- oder Endpunktmodell angrenzt. Besitzt die Variable `SEP` den Wert 0, grenzt ein Start- oder Endpunkt an das Objekt und die Länge `r2` entfällt. Dies ist durch die Variablen `S18`, `f18`, `S916` und `f916` geregelt. Die Variablen werden am Anfang der Implementierung durch Fallunterscheidung definiert. `f18` und `f916` geben an, ob die Extrusion um die Länge `r2` erweitert wird. `S18` und `S916` sind definierte Korrekturwerte für die Länge `r`. Diese Methodik wird für die restlichen Abschnitte des Transportbandes übernommen.

Bei einem Höhenunterschied zwischen zwei Punkten werden Rampen erstellt. Die Vorgehensweise ist mit der Extrusion in der Ebene vergleichbar. Da eine Rampe achsparallel verlaufen muss, kann eine Unterscheidung getroffen werden, ob die Rampe in x- oder y-Richtung verläuft. Es können somit keine Rampen erstellt werden, welche schräg verlaufen und gleichzeitig einen Höhenunterschied überwinden.

Die für die Bestimmung der Ausrichtung der Rampe erstellte Fallunterscheidung implementiert die Variablen `xd` und `yd`, welche die Ausrichtung definieren. Rampen überwinden Höhenunterschiede, weshalb die Transportbandabschnitte um die X- oder Y-Achse mit Hilfe der Methoden `rotx` und `roty` gedreht werden müssen. Die Positionierung verwendet zusätzlich den Mittelwert `zm` zur Anpassung der Höhe.

Die einzelnen Eckpunkte der Transportbandabschnitte besitzen bei einer Rampe jeweils eine unterschiedliche Breite `s` und eine Höhe `sz`. Da die Eckpunkte als die vom Nutzer vorgegebenen Positionen definiert werden, ist keine Extrusion notwendig.

Das Definieren der neuen Koordinaten erfolgt durch Addieren der Breiten und Höhen mit den Koordinaten der Positionen. Die Variablen `xd` und `yd` bestimmen die Ausrichtung der Rampe durch Multiplikation mit den Werten von `s` und `sz`. Abbildung 40 stellt die Methodik am Beispiel der Eckpunkte 1 bis 8 dar.

```
for k = 1:8
    for l = 1:length(VP(k,:))
        Zeile = VP(k,l);
            if Zeile ~= 0
                % Aenderung der Koordinaten
                switch k
                    case 1
                        s = -0.7*b;
                        sz = -75;
                    ...
                    case 8
                        s = 0.7*b;
                        sz = 25;
                end
                % Punkte werden zu Punkten von i gesetzt
                K_line(Zeile,1) = P(i,1)+xd*(s);
                K_line(Zeile,2) = P(i,2)+yd*(s);
                K_line(Zeile,3) = P(i,3)+sz;
            end
    end
end
```

Abbildung 40: Definieren der Eckpunkte einer Rampe

Nach dem Variieren der Koordinaten wird das Modell durch die `patch` Funktion visualisiert und mit Hilfe von `set` die Farben und Lichtverhältnisse angepasst.

4.3 Anwendungsbeispiel der create_line Methode

Um die Vorgehensweise beim Benutzen der `create_line` Methode zu verdeutlichen, wird an dieser Stelle ein Cart mit dem dazugehörigen Transportband visualisiert. Der Durchmesser des Transportwagens soll 100mm betragen. Das Transportband besteht aus 11 Punkten und besitzt insgesamt zwei Rampen. Da der Durchmesser des Carts gegeben ist, wird die Parametrierung des Transportbands auf das Cart angepasst. Der erste Eingangsparameter der `create_line` Methode muss demnach auch 100mm betragen.

Zudem sind der Start- und der Endpunkt nicht identisch. Die einzugebenden Befehle sind in Tabelle 5 dokumentiert.

Tabelle 5: Befehlsabfolge am Beispiel von `create_line`

Befehlsabfolge	Visualisierungs-Rechner	Control-Rechner
1	VirtualRobot.create('Cart', 40000, [0, 0, 0, 0, 0, 0],100)	
2		c=robot('open', 'Kawasaki', 'tcpip', 'localhost', 40000)
3	VirtualRobot.start_all	
4	VirtualRobot.create_line(100, [0 0 0; 200 0 0; 400 0 200; 600 0 200; 900 -400 200; 1100 -500 200; 1100 -900 200; 800 -1200 200; 600 -1200 200; 0 -1200 0; -400 -1200 0])	
5		robot(c, 'close')
6	VirtualRobot.delete_all	

Die Positionen der Eckpunkte des Transportbandes sind als Matrix zu definieren. Abbildung 41 visualisiert die zu erstellenden Geraden mit der `plot3` Funktion.

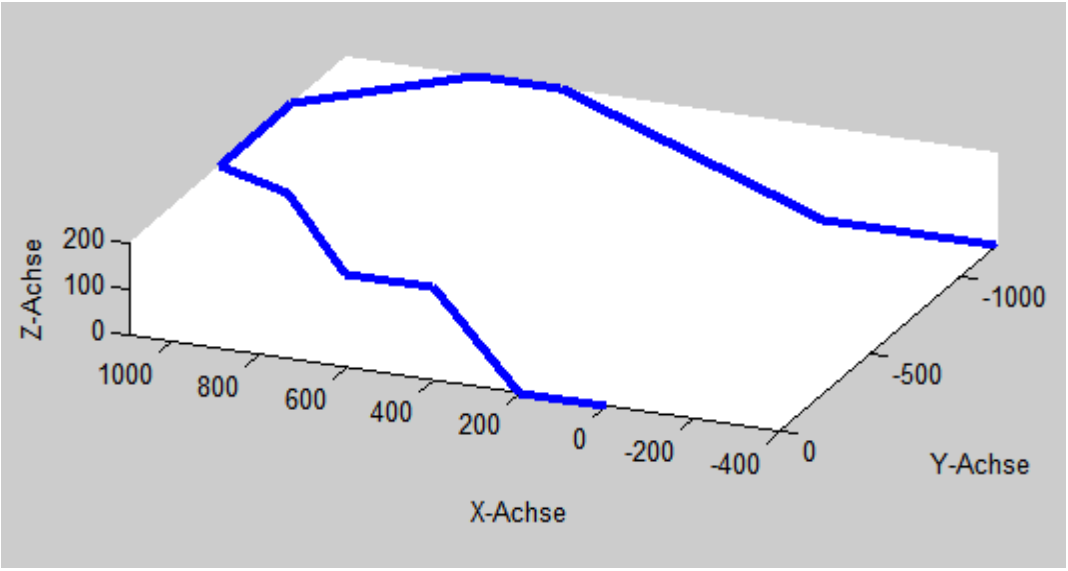

Abbildung 41: Skizze der Geraden des Transportbands

Das Ergebnis der Visualisierung ist in Abbildung 42 dargestellt.

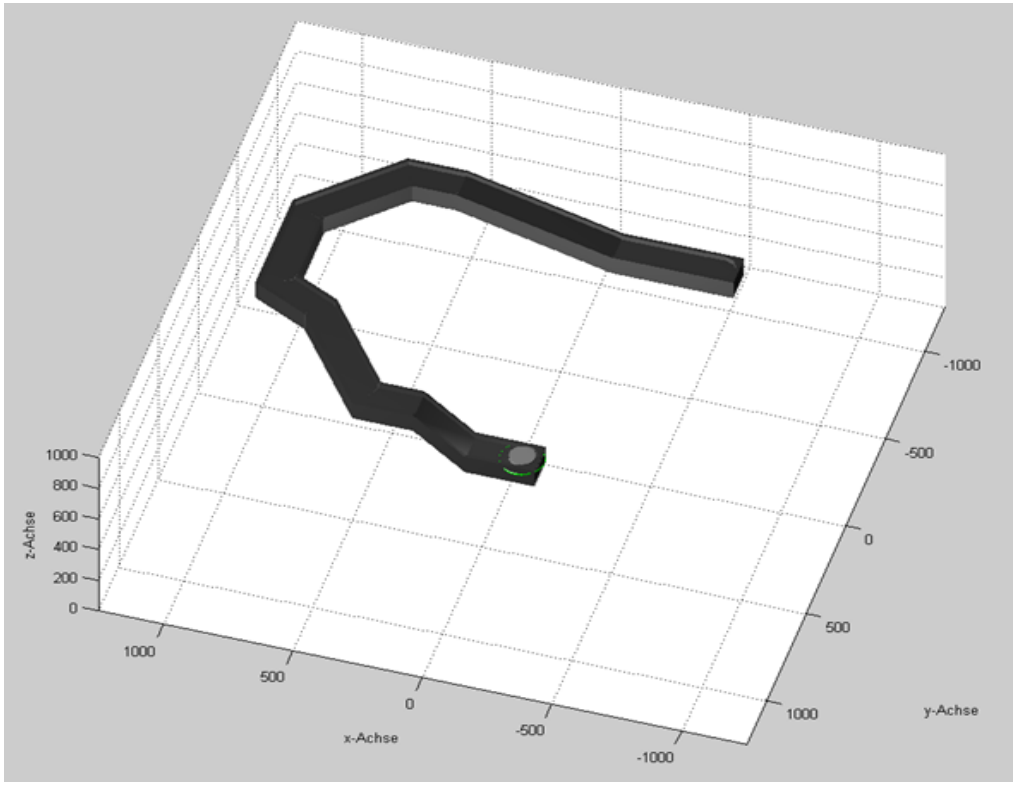

Abbildung 42: Visualisierung des Transportbands mit `create_line`

Das Transportband ist im Verhältnis zum Cart frei parametriert. Der Transportwagen hat ein kleines Spiel zu den Rändern des Transportbands und hebt sich durch eine hellere Farbe von diesem ab. Die Geraden der Transportbandabschnitte und die Rampen sind lückenlos miteinander verbunden und besitzen keine überstehenden Kanten oder Ränder. Im Vergleich zu Abbildung 41 ist zu erkennen, dass das Transportband korrekt visualisiert ist. Die `create_line` Methode ermöglicht somit einen unkomplizierten Weg zum Erstellen eigener Transportbänder.

5 Implementierung der Conveyer-Toolbox

Da eine Veränderung der MatlabKK-Robotic-Toolbox nicht gewünscht ist, soll in diesem Kapitel die Implementierung einer neuen Toolbox erläutert werden. Als Grundlage für die zu erstellende Conveyer-Toolbox dienen einige Funktionen der Robotic-Toolbox. Zudem werden zwei neue Funktionen für das Greifen und Loslassen von Parts implementiert.

5.1 Zu betrachtende Funktionen der Conveyer-Toolbox

Die Conveyer-Toolbox besteht aus insgesamt 16 Funktionen, welche auf die MatlabKK-Robotic-Toolbox aufbauen. Es können einige Funktionen aus der Robotic-Toolbox übernommen werden, da die Kinematik des implementierten Carts in der MatlabKK-Visualization-Toolbox mit der Kinematik der vordefinierten Roboter übereinstimmt. Somit ist eine Änderung der Funktionen kaum notwendig.

Zur Erleichterung der Bedienung werden zudem die Funktionen `cgrab` und `crelease` erstellt. Diese ermöglichen es dem Nutzer Parts zu greifen und loszulassen, ohne `cset` benutzen zu müssen. Die Funktionen haben `rset` als Grundlage und sind neu implementiert.

Um Verwechslungen mit der Robotic-Toolbox zu vermeiden, sind alle Namen mit dem Anfangsbuchstaben „c“ versehen. Die Funktionen der Conveyer-Toolbox sind im Folgenden erläutert.

`h = conveyer (connection, type, IP, port)`

Die Funktion öffnet oder schließt eine Verbindung zu einem Cart durch TCP/IP oder einem seriellen Port und speichert die Daten als Handle `h`. Die Grundlage bildet `robot` aus der Robotic-Toolbox.

Tabelle 6: Parameter der `conveyer` Funktion

Parameter	Beschreibung	Beispielwerte
connection	Öffnen oder schließen der Verbindung	`'open'` `'closed'`
type	Art der Verbindung, wahlweise seriell oder TCP/IP	`'tcpip'` `'serial'`
IP	IP-Adresse des Carts, alternativ auch `localhost` verwendbar	`'192.168.0.1'` `'localhost'`
port	Portnummer des Carts mit der IP-Adresse	`40000` `40001`
COM-port	Alternative zu `IP` und `Port`, gibt die serielle Port-Adresse des Carts an	`'COM1'` `'COM2'`

`P = cpoint (x, y, z, o, a, t, relation)`

`Cpoint` erstellt einen durch das Cart anfahrbaren Punkt auf der Basis von `rpoint` und speichert diesen als Variable `P`. Optional kann auch `'speed'` mit dem dazugehörigen Wert ergänzt werden.

Tabelle 7: Parameter der `cpoint` Funktion

Parameter	Beschreibung	Beispielwerte
x, y, z	Verschiebungen der Koordinaten in x-, y- und z-Richtung	`180, 0, 0` `0, 0, -200`
o, a, t	Verdrehungen der Koordinaten um die x-, y- und z-Achse	`0, 0, 1.57` `pi/2, 0, 0`
relation	Art der Koordinaten als absolute Koordinaten oder relativ zum Cart	`'abs'` `'rel'`

`ID = cmove (handle, position)`

Diese Funktion bewegt das Cart zu einer von `cpoint` vorgegebenen Position. Als Ausgangspunkt dient `rmove`. `Position` besteht aus den Verschiebungen der Koordinaten x, y, z und den Verdrehungen o, a, t.

Tabelle 8: Parameter der `cmove` Funktion

Parameter	Beschreibung	Beispielwerte
position(1:3)	Verschiebungen der Koordinaten in x-, y- und z-Richtung	`180, 0, 0` `0, 0, -200`
position(4:6)	Verdrehungen der Koordinaten um die x-, y- und z-Achse	`0, 0, 1.57` `pi/2, 0, 0`

`ID = cset (handle, property, value)`

Mit diesem Kommando kann der Nutzer Eigenschaften des Interpreter des Carts und der Geschwindigkeit verändern. Das Cart ist durch `handle` definiert. Die Basis der Funktion ist `rset`.

Tabelle 9: Properties der `cset` Funktion

Property	Beschreibung	Value

`'interpreter'`	Schaltet den Interpreter des Cart-Controllers an, aus, oder löscht den Speicher des Interpreters	`'on'`
		`'off'`
		`'erase'`

`'speed'`	Verändert die prozentuale Bewegungsgeschwindigkeit des Carts	`10` `30`
`'signal'`	Gibt die Öffnung des Greifers an, [-9 10] öffnet, [9 -10] schließt diesen	`[-9 10]` `[9 -10]`

Ein sofortiges Ausführen der Aktion kann durch die Erweiterung `'now'` bewirkt werden.

`Ausgabe = cget (handle, property, value)`

Cget gibt die Werte einer Position oder der Geschwindigkeit des Carts aus. Das Cart ist durch `handle` definiert und Ausgangspunkt dieser Funktion ist `rget`.

Tabelle 10: Properties der `cget` Funktion

Property	Beschreibung	Ausgabe
`'pose'`	Gibt die Position des Carts zurück	`Positions-struktur`
`'speed'`	Gibt die Geschwindigkeit des Carts zurück	`Wert in Prozent`
`'signal'`	Gibt die Öffnung des Greifers an, [-9 10] öffnet, [9 -10] schließt diesen	`[-9 10]` `[9 -10]`

`ccallback (ID,{@function-handle,argumentlist}, beginning)`

Ccallback führt definierte Funktionen nach Ablauf oder Start einer Aktion automatisch aus. Die Grundlage bildet `rcallback`.

Tabelle 11: Parameter der `ccallback` Funktion

Parameter	Beschreibung	Beispielwerte
ID	Identifikations-Vektor des an das Cart gesendeten Kommandos, wird nach `cset` oder `cmove` zurückgegeben	`ID`
@function-handle	Handle der auszuführenden Funktion	`@Funktions-handle`
argumentlist	Optional. Eine durch Komma abgetrennte Liste der Argumente der Funktion	`Liste der Argumente`
beginning	Gibt an, ob die Funktion nach dem Beenden oder nach dem Starten des ID-Kommandos ausgeführt wird	`'finished'` `'started'`

`Ausgabe = cis (ID, question)`

Diese Funktion gibt an, ob eine Bewegung gestartet oder beendet ist. Bei Prozessen kann zudem die Bewegung auch pending, d.h. gerade in der Ausführung, sein. Als Basis dient `ris`.

Tabelle 12: Parameter der `cis` Funktion

Parameter	Beschreibung	Ausgabe
ID	Identifikations-Vektor des an das Cart gesendeten Kommandos, wird nach `cset`, `rprocess` oder `cmove` zurückgegeben	–
question	Definiert die Frage wahlweise als `'started'`, `'finished'` oder `'pending'` bei Prozessen	`true/false`

`cwait (ID, condition)`

Mit dieser Funktion kann die Eingabe im Command Window unterdrückt werden, bis eine Bedingung (`condition`) erfüllt ist. Ausgangspunkt bildet `rwait`.

Tabelle 13: Parameter der `cwait` Funktion

Parameter	Beschreibung
ID	Identifikations-Vektor des an das Cart gesendeten Kommandos, wird nach `cset`, `rprocess` oder `cmove` zurückgegeben
condition	Definiert die Bedingung wahlweise als `'started'`, `'finished'` bei Funktionen oder zusätzlich `'pending'` bei Prozessen

`cbrake (handle)`

Diese Funktion beendet alle Bewegungen des Carts mit der Bezeichnung `handle` oder aller Carts, wenn kein `handle` definiert ist. Es können weiterhin Kommandos empfangen werden. Die Grundlage ist `rbrake`. Nach der Eingabe von `crun` kann die Bewegung wieder aufgenommen werden.

cstop (handle)

Das Kommando beendet und löscht alle Bewegungen des Carts mit der Bezeichnung `handle` oder aller Carts, wenn `handle` undefiniert ist. Kommandos werden weiterhin empfangen. Die Basis bildet `rstop`. Nach der Eingabe von `crun` kann die Bewegung wieder aufgenommen werden.

ckill (handle)

Ckill beendet und löscht alle Bewegungen des Carts mit der Bezeichnung `handle` oder aller Carts, wenn kein `handle` definiert ist. Zudem kehrt das Cart in den Ausgangsstatus zurück. Grundlage ist `rkill`. Nach der Eingabe von `crun` kann die Bewegung wieder aufgenommen werden.

crun (handle)

Diese Funktion lässt das Cart `handle` die Bewegung wieder aufnehmen. Wenn kein `handle` definiert ist, wirkt diese Aktion für alle Carts. Ausgangspunkt von `crun` ist `rrun`.

creset (handle)

Dieses Kommando reaktiviert das Cart `handle` nach dem Ausschalten des Interpreters. Sollte `handle` undefiniert sein, werden alle Carts reaktiviert. Basis dieser Funktion ist `rreset`.

cgrab (handle)

`Cgrab` ist eine neu erstellte Funktion, welche durch `cset` die `signal` Eigenschaft eines Carts mit der Bezeichnung `handle` verändert. `Signal` wird auf den Wert `[-9 10]` gesetzt und das Cart wird das in der Nähe befindliche Part greifen.

crelease (handle)

Das neue Kommando `crelease` setzt mit Hilfe der `cset` Funktion den Wert der `signal` Eigenschaft auf `[9 -10]`. Das Cart wird das gegriffene Part loslassen.

5.2 Implementierung der betrachteten Funktionen

Die Implementierung der Conveyer-Toolbox erfolgt durch den Aufruf der Funktionen der MatlabKK-Robotic-Toolbox. Zudem wird die Funktion `conveyer` erweitert und `cgrab` und `crelease` neu erstellt. Der MATLAB-Code einer Funktion Robotic-Toolbox ist in Abbildung 43 dargestellt.

cset.m

```
1   function R = cset(Handle,varargin)
2
3 -     R = rset(Handle,varargin{:});
4
5 -   end
```

Abbildung 43: MATLAB-Code der `cset` Funktion

Als Beispiel ist `cset` gewählt worden, welche auf die Funktion `rset` der MatlabKK-Robotic-Toolbox aufbaut. Durch den Aufruf kann eine Veränderung der Robotic-Toolbox vermieden werden. Die Eingangsparameter von `cset` werden an `rset` übergeben. Falls ein Eingangsparameter `varargin` ist, muss eine geschweifte Klammer mit einem Colon-Operator folgen, da MATLAB die Variable sonst nicht als Cell-Array erkennt und einen Fehler ausgibt.

cgrab.m

```
1   function cgrab(Handle)
2
3 -   cset(Handle,'signal',[-9 10]);
4
5 -   end
```

Abbildung 44: MATLAB-Code der `cgrab` Funktion

In Abbildung 44 ist `cgrab` dargestellt. Zusammen mit `crelease` bilden sie die zwei neu implementierten, in der Robotik-Toolbox nicht vorhandenen, Funktionen.

Nach dem Aufruf der Funktionen wird von dem Transportwagen ein in der Nähe befindliches Part gegriffen oder losgelassen. Ohne `cgrab` und `crelease` kann dies mit der Funktion `cset(Handle,'signal',Parameter)` realisiert werden. `Parameter` besteht aus einem Spaltenvektor mit zwei Werten. `[-9 10]` bewirkt das Greifen und `[9 -10]` das Loslassen eines Parts.

```
function Handle = conveyer(connection, type, IP, port)

    Handle = robot(connection, 'Kawasaki', type, IP, port);

    fprintf(1,[
    '\n\n'...
    'o-------------------------------------------------------------o\n'...
    '| C O N V E Y E R - T O O L B O X      (Ver. 2013/03/11)     |\n'...
    '| (w) by Christoph Rasch                                     |\n'...
    '| Research Group : Computational Engineering and Automation |\n'...
    '| University of Applied Sciences Wismar, Germany             |\n'...
    'o-------------------------------------------------------------o\n\n']);

end
```

Abbildung 45: MATLAB-Code der `conveyer` Funktion

Die in Abbildung 45 gezeigte Funktion `conveyer` basiert auf `robot` und erweitert diese um die Ausgabe der Toolbox-Beschreibung. In der Beschreibung sind der Name der Toolbox, das Datum der Version, der Autor und das Institut, welches die Toolbox veröffentlicht, angegeben.

6 Entwicklung einer koordinierten Roboterapplikation

Dieses Kapitel befasst sich mit der Realisierung einer koordinierten Roboterapplikation. Es werden Beispiele für die Anwendung der Conveyer-Toolbox vorgestellt. Die Transportbänder sind durch die Methode `create_line` oder durch Hinzufügen von einer STL-Datei als Umweltmodell visualisiert. Die entstehenden Ergebnisse werden vorgestellt und dokumentiert.

6.1 Beispiel: Roboter mit Conveyer-Toolbox und create_line Methode

Die Verwendung der `create_line` Methode ermöglicht das Erstellen von Transportbändern durch Vorgabe von Punkten. In diesem Beispiel wird ein halbkreisförmiges Transportband visualisiert, auf dem ein Transportwagen ein Objekt zwischen zwei Kawasaki-Robotern bewegt. Die dazu notwendige Bewegungsfolge ist in Tabelle 14 dargestellt.

Tabelle 14: Befehlsfolge der Applikation mit `create_line`

Befehlsabfolge	Visualisierungs-Rechner	Control-Rechner
1	VirtualRobot.create('Cart', 40000, [-2000, -1500, 0, 0, 0, 0],[-2500, 2500, -1750, 750, -100, 1250],200)	
2		c=conveyer('open', 'tcpip', 'localhost', 40000)
3	VirtualRobot.create('Kawasaki', 40001, [-1500, -1500, -100, 0, 0, 0], [-2500, 2500, -1750, 750, -100, 1250])	
4		r1=robot('open', 'Kawasaki', 'tcpip', 'localhost', 40001)
5	VirtualRobot.create('Kawasaki', 40002, [1500, -1500, -100, 0, 0, 0], [-2500, 2500, -1750, 750, -100, 1250])	

6		r2=robot('open', 'Kawasaki', 'tcpip', 'localhost', 40002)
7	VirtualRobot.start_all	
8	VirtualRobot.create_line(200, [-2000 -1500 0; -2000 -1200 0; -2000 -800 300; -2000 -500 300; -1500 0 300; -1000 200 300; 0 350 300; 1000 200 300; 1500 0 300; 2000 -500 300; 2000 -800 300; 2000 -1200 0; 2000 -1500 0])	
9	VirtualRobot.place_env('table.stl', [-900 -1500 -100 0 0 0]);	
10	VirtualRobot.place_part('test_tube.stl', [-1200 -1450 300 0 0 0], [1,0,0]);	
11	VirtualRobot.place_env('table.stl', [900 -1500 -100 0 0 0]);	
12		c_example1;
19	VirtualRobot.delete_all	

Das Ausführen der Befehle visualisiert zwei Kawasaki Roboter, welche durch ein Transportband mit einem Cart verbunden sind. Zudem stehen neben den Robotern zwei Tische. Auf einem Tisch befindet sich ein Reagenzglas, welches von einem Kawasaki gegriffen und auf den Transportwagen platziert wird. Dieser transportiert das Reagenzglas zum anderen Roboter, welcher es auf den in seiner Nähe befindlichen Tisch stellt. Es ist darauf zu achten, dass der Control-Rechner auf die MatlabKK-Robotic-Toolbox und die Conveyer-Toolbox zugreifen kann. Die Bewegungsabläufe sind im Anhang in dem MATLAB Skript `c_example1.m` abgespeichert.

Das Ergebnis der Befehlsabfolge ist in Abbildung 46 dargestellt. Wie vorgegeben wird das Reagenzglas vom ersten zum zweiten Kawasaki Roboter transportiert. Während der Visualisierung traten keine Fehler auf. Das Ausführen der Befehle war ohne Einschränkung möglich. Die Funktionsfähigkeit ist damit nachgewiesen.

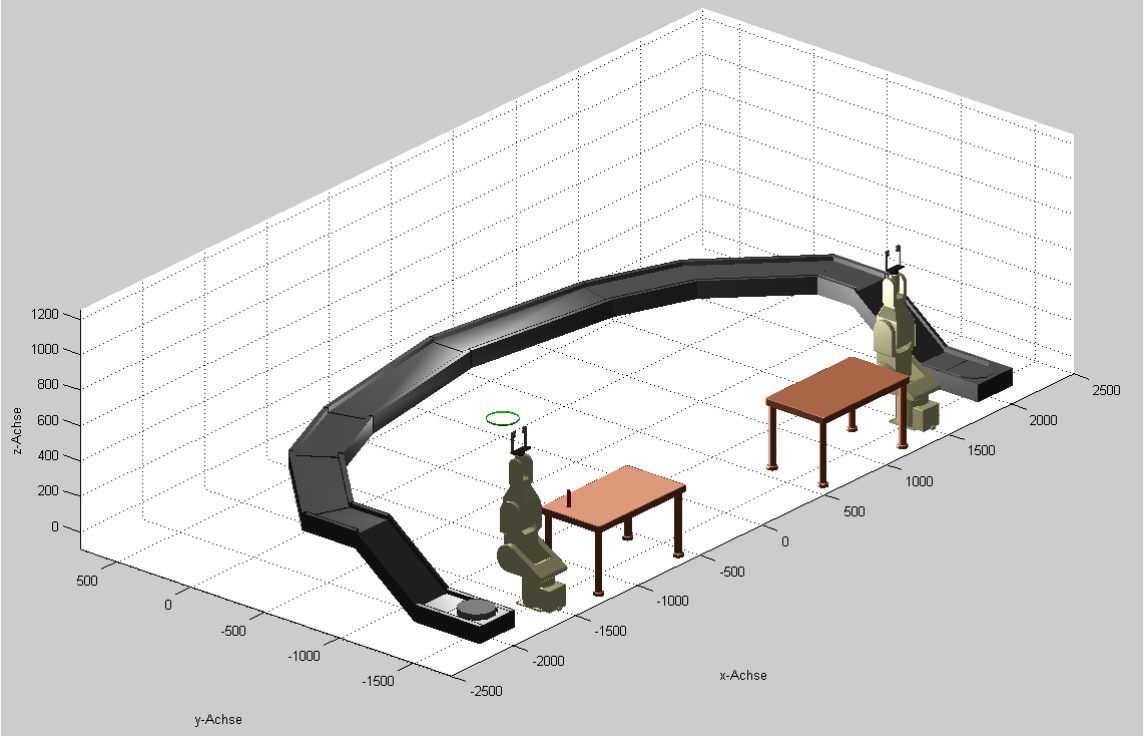

Abbildung 46: Visualisierung der Roboterappikation mit `create_line`

6.2 Beispiel: Roboter mit Conveyer-Toolbox und einem STL-Transportband Objekt

Dieses Beispiel befasst sich mit der Möglichkeit, selbst erstellte Transportbänder durch STL-Import als Umweltobjekt in die Visualisierung einzufügen. Die Transportbänder können ohne visuelle Einschränkungen implementiert werden. Ein Nachteil der `create_line` Methode ist, dass nur achsparallele Rampen definiert werden können. Zudem sind unregelmäßige und zu kleine Formen und individuelle Details nicht generierbar. Das in dem CAD-Programm Creo Parametric 2.0 erstellte, kleine Transportband besitzt schräge Rampen und ein zylinderförmiges Ende. Die Höhe wird in MATLAB als Z-Achse definiert. Bei dem Erstellen der Objekte mit einem CAD-Programm ist diese Orientierung des Koordinatensystems zu beachten, da die importierten Objekte sonst eine Verdrehung haben können. Es ist in Abbildung 47 mit dem Export der STL-Datei dargestellt. Die Befehlsabfolge des Beispiels ist in Tabelle 15 vorgestellt.

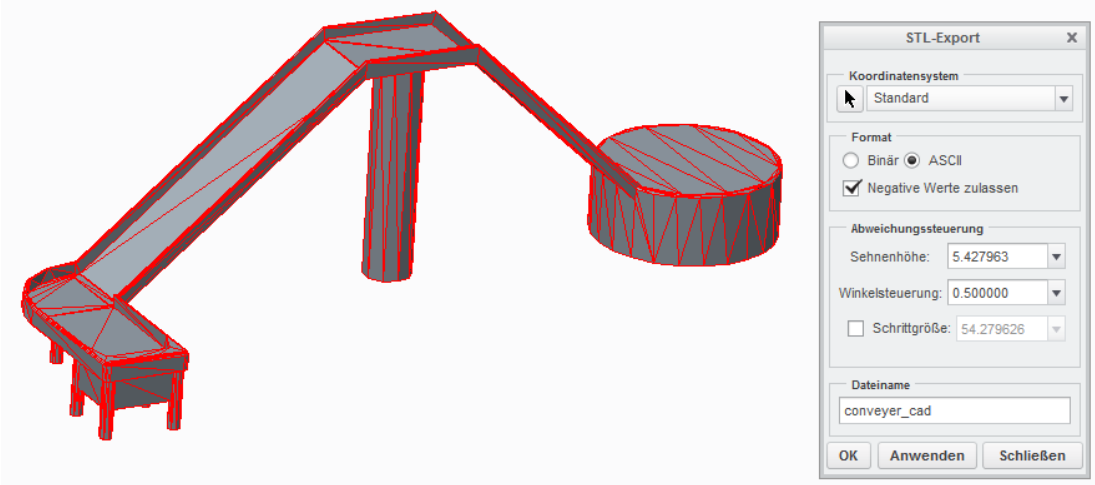

Abbildung 47: CAD-Transportband mit STL-Export

Tabelle 15: Befehlsfolge der Applikation mit STL-Transportband

Befehlsabfolge	Visualisierungs-Rechner	Control-Rechner
1	VirtualRobot.create('Cart', 40000, [-175, 0, 110, 0, 0, 0],[-1500, 1500, -2000, 1500, 0, 1250],100)	
2		c=conveyer('open', 'tcpip', 'localhost', 40000)
3	VirtualRobot.create('Kawasaki', 40001, [700, -1200, 0, 0, 0, 0], [-1500 1500, 1500, -2000, 0, 1250])	
4		r=robot('open', 'Kawasaki', 'tcpip', 'localhost', 40001)
5	VirtualRobot.start_all	
6	VirtualRobot.place_env('conveyer_cad.stl', [0 0 0 0 0 0],[0.2 0.2 0.2]);	
7	VirtualRobot.place_env('table.stl', [600 -1700 0 0 0 0]);	
8	VirtualRobot.place_part('test_tube.stl', [-175 -0 135 0 0 0], [1,0,0]);	
9		c_example2;
10	VirtualRobot.delete_all	

Die Eingabe der Befehle lässt das Cart das Reagenzglas über das Transportband zum Roboter transportieren. Dieser greift das Part und positioniert es auf dem Tisch. Die Bewegungsbefehle sind im Anhang in dem MATLAB Skript `c_example2.m` gespeichert. Es ist darauf zu achten, dass der Control-Rechner auf die MatlabKK-Robotic-Toolbox und die Conveyer-Toolbox zugreifen kann.

Als Ergebnis wird der in Abbildung 48 gezeigte Aufbau visualisiert. Die Bewegungsabläufe werden entsprechend der Eingaben ohne Fehler umgesetzt und die Funktionsfähigkeit ist somit nachgewiesen.

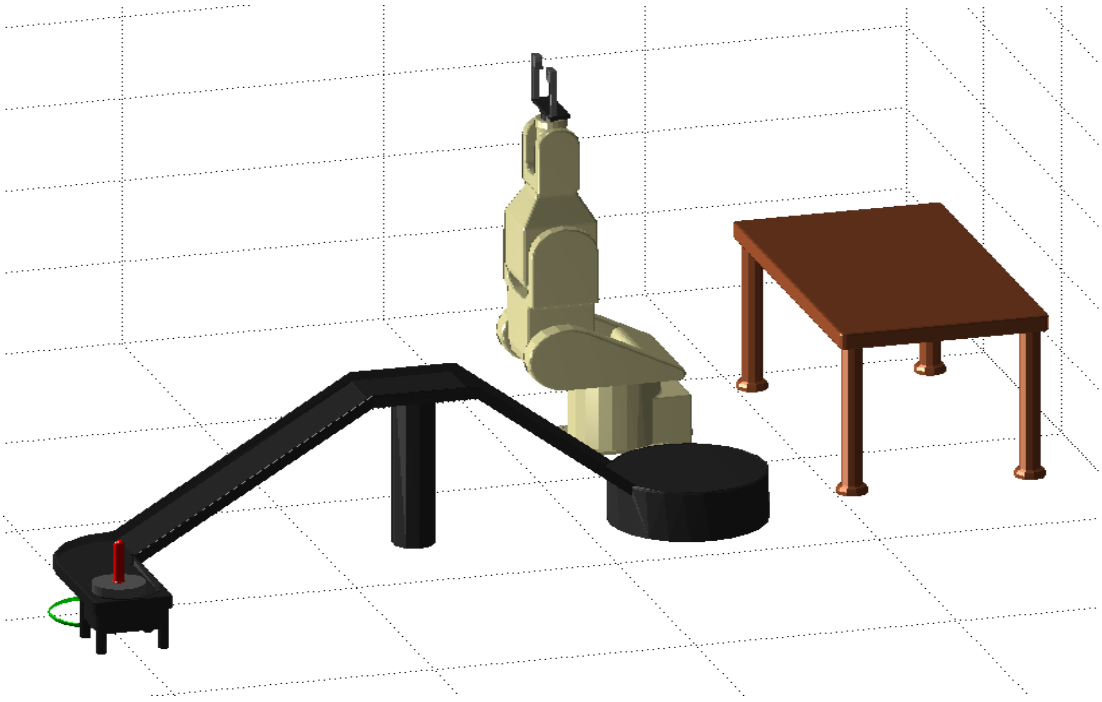

Abbildung 48: Visualisierung der Roboterapplikation mit CAD-Transportband

7 Zusammenfassung und Ausblick

Zusammenfassend ist zu sagen, dass das Ziel der Implementierung einer koordinierten Roboterapplikation erreicht wurde. Es ist ein Transportwagen implementiert, welcher Parts greifen und bewegen kann. Große Distanzen können somit trotz begrenzter Arbeitsräume der Roboter überwunden werden. Das Aussehen des Carts lässt sich durch den `New_Robot` Typ an die individuellen Bedürfnisse und Aufgabenbereiche des Anwenders anpassen. Zudem können mit der Methode `create_line` automatisch Transportbänder durch Vorgabe von Punkten visualisiert werden. Da die MatlabKK-Visualization-Toolbox über eine CAD-Schnittstelle verfügt, kann der Nutzer auch selbst erstellte STL-Modelle als Transportbänder visualisieren. Diese werden als Umweltobjekte definiert. Die aufgezeigten Beispiele verdeutlichen die unkomplizierte Bedienung des Carts und Erstellung der Transportbänder. Zusätzlich wurde die Erweiterung der Toolbox um den neuen Robotertyp `Cart` dargestellt. Diese Vorgehensweise kann auf andere, in Zukunft zu implementierende Robotertypen angewendet werden. Die Implementierung einer Kollisionserkennung könnte in Zukunft auch integriert werden.

Literaturverzeichnis

[1] SCHMIDT, ARTUR; CHRISTERN MICHAEL : Entwicklung einer Matlab- und Scilab-Kawasaki-AS-Toolbox mit dazugehörigem AS-Interpreter. Wismar, Hochschule Wismar, Fachhochschule für Technik, Wirtschaft und Gestaltung, Fachbereich Maschinenbau und Verfahrens- und Umwelttechnik, Bachelor-Thesis, 2009

[2] OTTO, JOHANNES : Entwicklung einer Visualisierungstoolbox für kooperierende Kawasaki-FS003N Roboter. Wismar, Hochschule Wismar, Fachhochschule für Technik, Wirtschaft und Gestaltung, Fachbereich Maschinenbau und Verfahrens- und Umwelttechnik, Bachelor-Thesis, 2011

[3] RASCH, CHRISTOPH : Entwicklung einer CAD-Schnittstelle für die MatlabKK-Visualization Toolbox. Wismar, Hochschule Wismar, Fachhochschule für Technik, Wirtschaft und Gestaltung, Fachbereich Maschinenbau und Verfahrens- und Umwelttechnik, Praktikums-bericht, 2012

[4] WEBER, WOLFGANG : Industrieroboter, Methoden der Steuerung und Regelung. 2., neu bearbeitete Auflage, Hanser Verlag, 2009

[5] J. OTTO, T. SCHWATINSKI, T.PAWLETTA (2011). MatlabKK-Robotic-Visualization-Toolbox [Online]; http://www.mb.hs-wismar.de/cea/MatlabKK_Robotic-and-Visualization_Tbx /MatlabKK_Robotic_Visualization_Tbx.html (30.11.2012)

[6] OTTO, JOHANNES : Erweiterung der MatlabKK-Visualization-Toolbox, Wismar, Hochschule Wismar, Fachhochschule für Technik, Wirtschaft und Gestaltung, Projektarbeit, 2011

Anlagenverzeichnis und Anlagen

Alle Anlagen befinden sich auf der beigefügten CD. Das Anlagenverzeichnis und eine Beschreibung der einzelnen Dateien sind dort noch einmal in Anhang.txt formuliert.

Anlagenverzeichnis Matlab-Dateien:

- VirtualRobot.m
- conveyer.m
- cpoint.m
- cmove.m
- cset.m
- cget.m
- ccallback.m
- cis.m
- cwait.m
- cbrake.m
- cstop.m
- ckill.m
- crun.m
- creset.m
- cprocess.m
- cgrab.m
- crelease.m
- c_example1.m
- c_example2.m

Anlagenverzeichnis STL-Dateien:

- conveyer_line.stl
- conveyer_start.stl
- conveyer_cart.stl

- conveyer_car.stl
- conveyer_cad.stl

Anlagenverzeichnis CAD-Dateien:

- conveyer_line.prt.1
- conveyer_start.prt.1
- conveyer_cart.prt.1
- conveyer_car.prt.1
- conveyer_cad.prt.1

Selbstständigkeitserklärung

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

Wismar, 23.04.2013 Christoph Rasch