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

Praktikumsbericht

Entwicklung einer CAD-Schnittstelle für die MatlabKK-Visualization Toolbox

Eingereicht am: 07.12.2012

von: Christoph Rasch, geboren am

21.08.1989 in Parchim

Matrikelnummer: 114104

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

Aufgabenstellung

für das Bachelor-Ingenieurpraktikum

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

Thema: **Entwicklung einer CAD-Schnittstelle für die MatlabKK-Visualization Toolbox**

In der FG CEA wurden verschiedene MATLAB Toolboxen zur Steuerungsentwicklung für KUKA und Kawasaki Roboter sowie eine Visualisierungs-Toolbox (MatlabKK-Visualization) für die Robotertypen KUKA-KR3 und KAWASAKI-FS003N entwickelt.
In dieser Arbeit soll eine CAD-Anbindung für die MatlabKK-Visualization Tbx. entwickelt werden. Ein wesentliches Ziel ist die einfachere Entwicklung von Umgebungsszenarien für zu visualisierende Roboterapplikationen sowie die Integration neuer Robotertypen.

Schwerpunkte:

- Einarbeitung in die Grundlagen: Roboterkinematik, MatlabKK-VisualizationTbx., Visualisierung mit surf-Objekten in Matlab
- Analyse von CAD-Formaten und Auswahl eines geeigneten CAD-Formats
- Umsetzung eines Lösungsentwurfs mit Creo Parametric 2.0 am Beispiel
  - der beiden Robotertypen (KR3, FS003N)
  - prototypischer Umweltobjekte und Parts
- Einbindung des Lösungsentwurfs in die MatlabKK-Visualization Tbx.
- Nachweis der Funktionsfähigkeit

Die Arbeit ist nach den vorgegebenen Richtlinien anzufertigen. Neben der geforderten Anzahl schriftlicher Exemplare ist die schriftliche Arbeit (Originaldateien & PDF-Dateien) sowie die entwickelte Software auf einem dokumentierten digitalen Datenträger abzugeben

Betreuer: **MEng. Artur Schmidt, HS Wismar, FIW/MVU, FG CEA**

MEng. Tobias Schwatinski, HS Wismar, FIW/MVU, FG CEA

Prof. Dr. Thorsten Pawletta, HS Wismar, FIW/MVU, FG CEA

Thorsten Pawletta

Zusammenfassung

In der Forschungsgruppe Computational Engineering and Automation (CEA) wurde eine KUKA-KAWASAKI-Visualization-Toolbox für MATLAB entwickelt, in welcher die Visualisierung bei gleichzeitiger Steuerung eines KUKA- oder KAWASAKI-Roboters vorgenommen werden kann. Die grafische Darstellung erfolgt dabei mittels der vordefinierten `surf` Funktion. Ziel ist es, dass CAD-Dateien eingelesen und visualisiert werden können. Schwerpunkte bilden zum einen das Einlesen und grafische Darstellen der CAD-Dateien, als auch das Erweitern der bereits vorhandenen Toolbox mit den entsprechenden Funktionen.

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 MatlabKK-Visualization-Toolbox 5
2.3 Probleme der Visualisierung mit `surf` Objekten 7

3 Typische Formate von CAD-Dateien 10
3.1 Auswahl eines CAD-Formats 10
3.2 Implementierung des CAD-Format-Imports in MATLAB 12
3.2.1 Grundlagen der Implementierung des CAD-Format-Imports 12
3.2.2 Beispiel eines Imports aus Creo Parametric 2.0 14
3.2.3 Beispiel eines Imports aus der 3D-Software Blender 2.64 16

4 Darstellungsmöglichkeiten der Datenpunkte in MATLAB 18
4.1 Vergleich der `surf` und `patch` Funktion 18
4.2 Besonderheiten der Zeichenfunktion 22

5 Realisierung des Lösungsentwurfs mit Creo Parametric 2.0 23
5.1 Modellierung der Roboter 23
5.2 Modellierung der Umweltobjekte und Parts 25

6 Einbindung des Lösungsentwurfs in die MatlabKK-Visualization-Toolbox 26
6.1 Erweiterung der MatlabKK-Visualization-Toolbox 26
6.2 Erweiterung der `env_mdl` und `Part` Funktion 30
6.3 Nachweis der Funktionsfähigkeit 31

7 Zusammenfassung und Ausblick 34

Literaturverzeichnis 35

Anlagenverzeichnis und Anlagen 36

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: Visualisierung des Kawasaki FS003N 7
Abbildung 6: Darstellung der Objekte mit Linie 8
Abbildung 7: Datenpunkte der `surf` Objekte 8
Abbildung 8: `Surf` Darstellung der Datenpunkte 9
Abbildung 9: Beispiel einer STL-Datei im ASCII-Format 13
Abbildung 10: Grundlegende Struktur der Funktion `STL_read` 14
Abbildung 11: Export eines Modells in den STL-Typ 15
Abbildung 12: Export einer ASCII-STL-Datei aus Creo Parametric 2.0 15
Abbildung 13: ASCII-Ausgabewerte in MATLAB 16
Abbildung 14: Export der binären STL-Datei aus Blender 2.64 17
Abbildung 15: Binäre Ausgabewerte in MATLAB 17
Abbildung 16: Funktionsweise von `STL_Surf` 19
Abbildung 17: Boot-Objekt mit Hilfe der `surf` Funktion 20
Abbildung 18: Funktion `STL_Patch` 21
Abbildung 19: Boot-Objekt mit Hilfe der `patch` Funktion 21
Abbildung 20: Koordinatensystem in MATLAB (a) und Creo Parametric 2.0 (b) 23
Abbildung 21: Skizze des Kawasaki-Roboterarms 24
Abbildung 22: Erweiterung der `propertie` Variablen in `VirtualRobot.m` 26
Abbildung 23: Einlesen des Kawasaki-Dreharms aus einer STL-Datei 27
Abbildung 24: Einlesen und Färben der Markierung 28
Abbildung 25: Erweiterung der `Draw` Methode am Beispiel von Arm 1 29
Abbildung 26: Veränderung der `place_env` Methode 30
Abbildung 27: Erweiterung der `env_mdl` und `Part` Funktion 31
Abbildung 28: Test mit ursprünglicher Toolbox 33
Abbildung 29: Test mit erweiterter Toolbox 33

Tabellenverzeichnis

Tabelle 1: DH-Parameter des RV6-12 3

Tabelle 2: Befehlsabfolge des Beispiels 6

Tabelle 3: CAD-Formate im Überblick 10

Tabelle 4: Aufbau der STL-Dateitypen 12

Tabelle 5: Bewertung der Darstellungsmöglichkeiten 18

Tabelle 6: Befehlsfolge des Tests 32

Verzeichnis der Abkürzungen

CEA	Computational Engineering and Automation
TCP	Tool-Center-Point
DH-Parameter	Denavit-Hartenberg-Parameter
PC	Personal Computer
IGES	Initial Graphics Exchange Specification
STL	Surface Tesselation Language
VRML	Virtual Reality Modeling Language
STEP	Standard for the exchange of product model data
VDA-FS	Verband der Automobilindustire - Flächenschnittstelle
ASCII	American Standard Code for Information Interchange
OOP	Objekt-Orientierte Programmierung

1 Einleitung

Die MatlabKK-Visualization-Toolbox kann in Verbindung mit der MatlabKK-Robotic-Toolbox Kuka- und Kawasaki-Roboter sowohl steuern, als auch Visualisieren. Letzteres erfolgt mittels der `surf` Funktion. Dazu müssen alle darzustellenden Objekte vorher manuell in Koordinatenpunkten modelliert werden. Da diese Methode umständlich ist und nicht ohne weiteres auf andere Modelle oder Typen von Roboterarmen übertragen werden kann, soll im Folgenden die MatlabKK-Visualization-Toolbox erweitert werden. Dabei soll die Modellierung mit CAD-Programmen vorgenommen und diese anschließend nach MATLAB importiert werden. Dies erleichtert das grafische Erstellen der Objekte und ermöglicht eine einfachere Umgestaltung.

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-Visualization-Toolbox [1] eingegangen und welche Probleme in den dort verwendeten Darstellungen vorkommen können. Die Grundlage für Kaptiel 2.1 bildet [2].

2.1 Kinematik von Robotern

Sowohl der Kuka KR3 als auch der Kawasaki FS003N weisen je sechs Gelenke auf, welche durch ihre Gelenkwinkel definiert sind. Durch das Einstellen der entsprechenden Winkel kann man so 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 auch als inverse Kinematik bezeichnet 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 veranschaulicht:

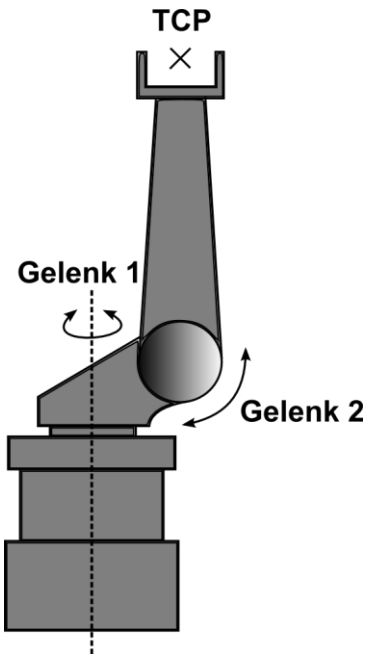

Abbildung 3: Zweigelenkroboter RV6-12

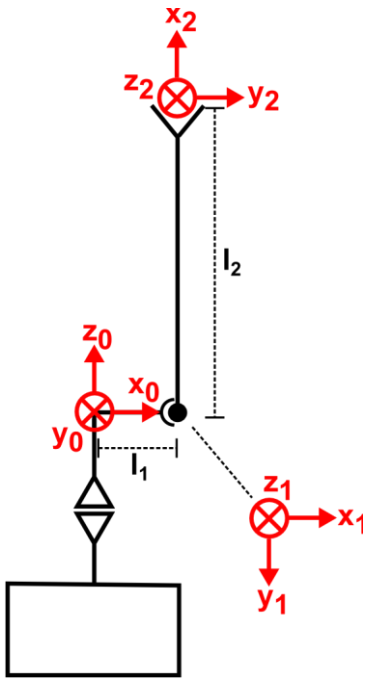

Abbildung 4: Koordinatensysteme des RV6-12

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 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 Koordinaten multipliziert. So ist es möglich, durch Veränderung der Gelenkwinkel die Frames, und damit die Stellung des TCP zu ändern. Die allgemeine Form einer Frame-Matrix wird hier im Folgenden aufgezeigt:

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

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 Gelenk 0 (Roboterbasis) zu Gelenk 2 (TCP) berechnet. Dabei ist die Multiplikation aller Frame-Matrizen der vorrangegangenen Gelenke durchzuführen.

Anders als bei der Vorwärtskinematik wird bei der inversen Kinematik zuerst die Position des TCP vorgegeben. Diese muss sich im Arbeitsraum des Roboters befinden, welcher den Bereich darstellt, in dem es den Roboter möglich ist, sich zu bewegen. Liegt die Position außerhalb des Arbeitsraums, so ist keine Bewegung dorthin durchführbar. Anschließend werden anhand geometrischer Berechnungen die einzelnen Gelenkwinkel bestimmt, und die Stellung des Roboterarms kann danach durch die Vorwärtskinematik beeinflusst werden.

2.2 MatlabKK-Visualization-Toolbox

Die MatlabKK-Visualization-Toolbox muss in Zusammenarbeit mit der MatlabKK-Robotic-Toolbox [3] verwendet werden. Dabei werden auf einem oder mehreren Rechnern insgesamt zwei MATLAB-Instanzen erstellt. Dies bedeutet, dass MATLAB zweimal ausgeführt wird. Die erste Instanz symbolisiert den Visualisierungs-PC, während die Zweite die Rolle des Control-PCs übernimmt. Nun kann man über den Visualisierungs-Computer die dazustellenden Objekte erzeugen und über den Entwicklungs-Rechner die Visualisierung der Roboterarme steuern. Dabei wurde darauf geachtet, dass die Befehle mit den Kommandos übereinstimmen, die man auch an den realen Roboter weitergibt. Der Nutzer kann zwischen zwei vordefinierten Roboterarmen wählen, den Kawasaki FS003N und den Kuka KR3, welche in den Abbildungen 1 und 2 veranschaulicht sind. Es sind somit für beide Roboter sowohl die DH-Parameter, als auch Datenpunkte für die `surf` Funktion vorhanden. Die `surf` Funktion ist ein Befehl für die grafische Visualisierung von Koordinaten in MATLAB, mit dem in der Toolbox die Objekte dargestellt werden. Der Nutzer kann außerdem der Visualisierung Umweltobjekte und Parts hinzufügen. Erstere sind starre, unbewegliche Körper. Parts hingegen 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 [4] beschrieben.

Zuerst wird die Datenstruktur des entsprechenden Roboters 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 den Kawasaki- oder den Kuka-Roboter handelt. Die nächste Variable definiert den Server-Port und `Pose` die Position der Basis des Roboterarms. `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 kann der Roboterarm vom Control-Rechner durch den Befehl `r1=robot(Key, Type, Connection, IP-Adress, Port)` erzeugt werden. Damit eine Steuerung des Roboters nach dem Erstellen möglich ist, wird dieser mit dem Handle `r1` gekennzeichnet. Die `Key` Variable gibt an, ob die Verbindung zum Roboter geöffnet oder geschlossen ist, während `Type` und `Port` ihre vorrangegangenen Wertzuweisungen behalten. `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` vom Visualisierungs-PC startet die grafische Darstellung. Der Nutzer kann nun vom Control-Rechner aus den Roboterarm steuern. Als Beispiel wird `rmove(r1, 'home2')` verwendet, welcher den Roboter mit der Kennzeichnung `r1` in die vordefinierte Position `home2` steuert. Schlussendlich wird dieser 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, 'home2')
5		robot(r1, 'close')
6	VirtualRobot.delete_all	

Als Ergebnis der Eingabe visualisiert MATLAB, wie in Abbildung 5 gezeigt, den Kawasaki-Roboter in der Position `home2`.

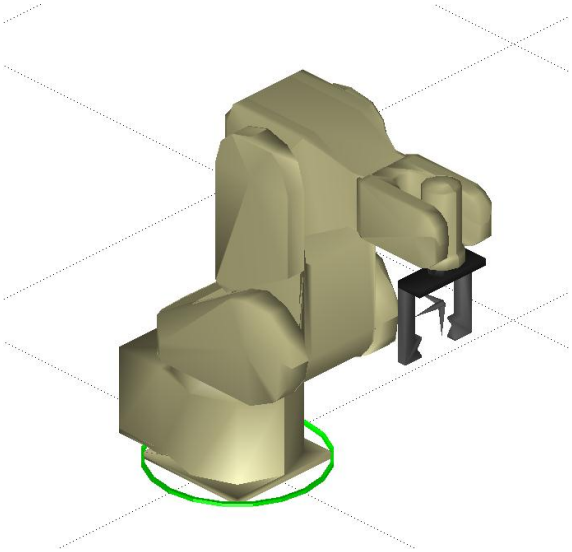

Abbildung 5: Visualisierung des Kawasaki FS003N

2.3 Probleme der Visualisierung mit `surf` Objekten

Die in der Toolbox verwendete Funktion `surf(x,y,z)` bietet die Möglichkeit zum Visualisieren dreidimensionaler Objekte in MATLAB. Eingabewerte sind die x-, y- und z-Werte der Koordinaten des darzustellenden Körpers. Dabei müssen diese in einer bestimmten Reihenfolge und Dimensionierung definiert werden. Um die Vorgehensweise einer Visualisierung mit der `surf` Funktion zu verdeutlichen, wird an dieser Stelle ein Beispiel vorgestellt:

Dazu soll ein Würfel mit einem Loch in der Mitte modelliert werden. Zuerst werden die x-, y- und z-Eingabewerte definiert. Der `Surf` Befehl zeichnet im Beispiel eine Linie aus den Koordinaten der ersten Zeilen der drei Achswerte. Dabei ist darauf zu achten, dass diese Linie geschlossen wird, indem die ersten und letzten Werte einer Zeile identisch sind. Anschließend wird die nächste Zeile bearbeitet und eine Fläche zwischen den

beiden Linien in Richtung der Z-Achse gespannt. Mit dieser Vorgehensweise lassen sich die Mantelflächen erzeugen. Um die Grund- und Deckfläche zu modellieren, vereint man alle x- und y-Koordinaten an einem Punkt. Da es in MATLAB nicht möglich ist, Flächen oder Abschnitte eines `surf` Objekts zu entfernen, muss das Beispiel, wie in Abbildung 6 gezeigt, aus zwei zusammengesetzten Körpern bestehen.

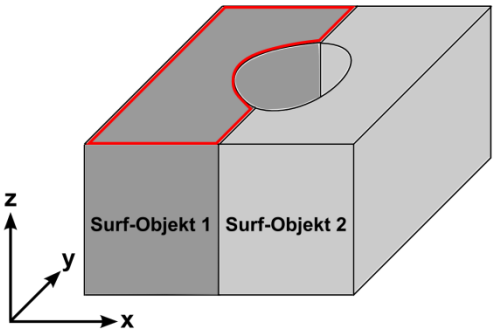

Abbildung 6: Darstellung der Objekte mit Linie

Je genauer komplexe Linien, wie Rundungen oder Kurven, dargestellt werden sollen, umso mehr Datenpunkte müssen definiert werden. In dem Beispiel wurde die Linie aus zehn Punkten zusammengesetzt.

```
x= [-0.8 -0.8 -0.8 -0.8 -0.8 -0.8 -0.8 -0.8 -0.8 -0.8;
    -1 0 0 -0.35 -0.5 -0.35 0 0 -1 -1;
    -1 0 0 -0.35 -0.5 -0.35 0 0 -1 -1;
    -0.8 -0.8 -0.8 -0.8 -0.8 -0.8 -0.8 -0.8 -0.8 -0.8];

y= [0 0 0 0 0 0 0 0 0 0;
    -1 -1 -0.5 -0.25 0 0.25 0.5 1 1 -1;
    -1 -1 -0.5 -0.25 0 0.25 0.5 1 1 -1;
    0 0 0 0 0 0 0 0 0 0];

z= [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1;
    -1 -1 -1 -1 -1 -1 -1 -1 -1 -1;
     1  1  1  1  1  1  1  1  1  1;
     1  1  1  1  1  1  1  1  1  1];

 surf(x,y,z)
 hold on

 x= [0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8;
    1 0 0 0.35 0.5 0.35 0 0 1 1;
    1 0 0 0.35 0.5 0.35 0 0 1 1;
    0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8 0.8];

 surf(x,y,z)
```

Abbildung 7: Datenpunkte der `surf` Objekte

Ergebnis des in Abbildung 7 gezeigten MATLAB-Codes ist der in Abbildung 8 dargestellte Würfel.

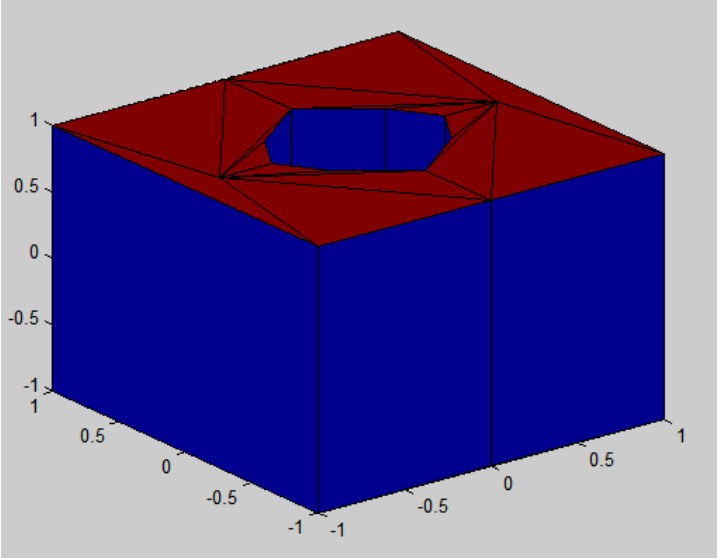

Abbildung 8: `Surf` Darstellung der Datenpunkte

Ein Nachteil der Darstellung ist der mit der Erstellung der Datenpunkte verbundene Aufwand. Zwar können Objekte wie Zylinder, Kugeln oder Quader durch vorgegebene Funktionen relativ leicht erzeugt werden, doch gerade mit unregelmäßigen Formen oder Bohrungen und Löchern ist extremer Arbeitsaufwand verbunden. Zudem sind komplexe Körper mit vielen Details und Löchern, wie zum Beispiel Fräsmaschinen oder Karosserieteile, mit der `surf` Funktion nahezu unmöglich darzustellen.

Zusätzlich ist es nachteilig, dass komplexe Objekte mit Hilfe eines CAD-Programms genau modelliert werden können. Deren Koordinaten der Datenpunkte müssen vor der Visualisierung noch vom Nutzer manuell an die `surf` Funktion übergeben werden. Eine Übertragung der Datenpunkte aus einem CAD-System und deren Visualisierung wäre demnach sinnvoll.

3 Typische Formate von CAD-Dateien

Die Betrachtungen zur der Visualisierung mit `surf` Objekten haben gezeigt, dass es sinnvoll ist, die Modellierung mit einem CAD-Programm vorzunehmen und die erstellten Dateien nach MATLAB zu importieren. Es gilt nun zu klären, welche Daten-Formate zur Auswahl stehen. Anschließend wird die begründete Wahl für ein Format getroffen und die Implementierung des Imports in MATLAB vorgestellt.

3.1 Auswahl eines CAD-Formats

Aufgrund der Anforderungen an die Visualisierung kommen nur CAD-Formate in Frage, welche dreidimensionale Objekte darstellen können. Häufig verwendete Formate werden in Tabelle 3 vorgestellt.

Tabelle 3: CAD-Formate im Überblick

CAD-Format	Beschreibung
IGES (.igs)	**Initial Graphics Exchange Specification** - Kann als ASCII- oder Binär-Format gespeichert werden, wobei Letzteres nicht mehr unterstützt wird - Daten werden in einzelnen Entities dargestellt - Neben den Datenpunkten werden auch die Zuordnungen der Flächen benötigt
STL (.stl)	**Surface Tesselation Language** - Kann als ASCII- oder Binär-Format gespeichert werden, wobei Ersteres zusätzlich noch die Farbe der Objekte wiedergibt - Datenpunkte (Vertices) werden in einzelnen Dreiecken dargestellt - Jedem Dreieck werden noch die Anordnung der Flächen (Faces) zugeordnet

VRML (.wrl)	**Virtual Reality Modeling Language** - Sind im ASCII-Format gespeichert, welche zusätzlich noch Farbe, Ausleuchtung usw. übernehmen können - Datenpunkte werden als Knoten dargestellt, welche für bestimmte Geometrieobjekte vordefiniert sind - Kann vom Webbrowser gelesen werden
STEP (.stp)	**Standard for the exchange of product model data** - Standard zum Beschreiben von Produktdaten - Leichter Austausch zwischen verschiedenen Programmen möglich - Je nach Anwendungsbereich stehen diverse Applikationsmodelle zur Verfügung
VDA-FS (.vda)	**Verband der Automobilindustrie – Flächenschnittstelle** - Ermöglicht Austausch von Geometrieinformationen zwischen CAD-Programmen - Zeichnungen können nicht übertragen werden

Theoretisch ist es möglich, alle genannten Formate nach MATLAB zu importieren. Sowohl das an der Hochschule Wismar verwendete CAD-Programm Creo Parametric 2.0 [5], als auch andere CAD-Programme wie Catia [6] oder SolidWorks [7] können einen Export für alle Formate durchführen. Zwar ist es möglich mit Hilfe der Simulink 3D Animation Toolbox [8] VRML-Dateien einzulesen und zu visualisieren, jedoch ist diese Erweiterung nicht im Standart-Umfang von MATLAB vorhanden und muss finanziell erworben werden. Im Rahmen der Betrachtung der Programme wurde auch kostenlos erhältliche 3D-Grafiksoftware berücksichtigt, mit welcher ebenfalls, wenn auch nach anderer Vorgehensweise, dreidimensionale Objekte erstellt werden können. Vor allem Nutzer ohne Zugriff auf ein CAD-Programm kann damit die einfache Modellierung von Körpern ermöglicht werden. Die sich durch intuitive Bedienung und großer Community auszeichnende 3D-Software Blender 2.64 [9] führt beispielsweise nur den Export in das Format STL durch.

Ein Vorteil von STL-Dateien gegenüber den anderen Formaten ist, dass es für das Rapid Prototyping eingesetzt wird. Als Rapid Prototyping bezeichnet man Fertigungsverfahren, deren Ziel es ist, mit vorhandenen CAD-Dateien Werkstücke herzustellen. Ein Beispiel für dieses Verfahren bildet das 3D-Drucken von Objekten.

Zusammenfassend wird der STL-Datentyp von kommerziellen und Open Source CAD-Programmen, wie beispielsweise Blender, unterstützt. Es ist ein typisches Datenformat im Bereich des Rapid Prototyping und der Import nach MATLAB ist ohne zusätzliche Toolboxen möglich. Aufgrund dessen wurde der STL-Datentyp gewählt und in einer MATLAB-Implementierung umgesetzt.

3.2 Implementierung des CAD-Format-Imports in MATLAB

3.2.1 Grundlagen der Implementierung des CAD-Format-Imports

Eine STL-Datei kann im ASCII- oder Binärformat abgespeichert werden. Der Unterschied besteht darin, dass ASCII-Dateien zwar größer sind, dafür aber auch Parameter für die im CAD-Programm verwendete Färbung der Objekte enthalten können. Es gilt also zuerst festzustellen, welcher Datentyp vorliegt. Dies kann anhand der ersten Zeile des STL-Files erfasst werden. Ist das erste Wort, welches sich in der Datei befindet, „solid“, so handelt es sich um das ASCII-Format. Für jedes der beiden Formate ist eine andere Implementierungsvariante erforderlich, da sie sich im Aufbau unterscheiden, wie Tabelle 4 dargestellt.

Tabelle 4: Aufbau der STL-Dateitypen

Zeile	ASCII-Format	Binär-Format
1	solid „Name“	„Name“
2	facet normal [Normalenvektor]	Anzahl der Dreiecke
3	outer loop	foreach triangle
4	vertex [Punkt 1]	[Normalenvektor]
5	vertex [Punkt 2]	[Punkt 1]
6	vertex [Punkt 3]	[Punkt 2]
7	endloop	[Punkt 3]
8	endfacet	Interner Zähler
Ende	endsolid „Name“	end

Die mit STL-Dateien definierten Objekte bestehen aus mehreren Dreiecken mit jeweils einem Face. Ein Face, auch Normalenvektor genannt, gibt an, welche Fläche welchen Punkten zugeordnet wird.

Beim ASCII-Format beginnt das Definieren eines Dreiecks mit dem Normalenvektor. Dieser wird durch den Ausdruck „facet normal“ gekennzeichnet. Anschließend werden zwischen den Wörten „outer loop“ und „endloop“ die Koordinaten der Eckpunkte implementiert. Jeder Punkt wird mit dem Befehl „vertex“ definiert. „Endfacet“ schließt die Modellierung eines Dreiecks ab. Ein Beispiel ist in Abbildung 9 aufgezeigt.

```
solid Beispiel
  facet normal 0.000000e+00 0.000000e+00 -1.000000e+00
    outer loop
      vertex 5.000000e+00 -5.000000e+00 -5.000000e+00
      vertex -5.000000e+00 -5.000000e+00 -5.000000e+00
      vertex -5.000000e+00 5.000000e+00 -5.000000e+00
    endloop
  endfacet

  ...

  facet normal 0.000000e+00 0.000000e+00 1.000000e+00
    outer loop
      vertex -5.000000e+00 -5.000000e+00 5.000000e+00
      vertex 5.000000e+00 -5.000000e+00 5.000000e+00
      vertex 5.000000e+00 5.000000e+00 5.000000e+00
    endloop
  endfacet
endsolid BEISPIEL
```

Abbildung 9: Beispiel einer STL-Datei im ASCII-Format

Beim Binär-Format steht nach dem Namen der Datei die Anzahl der modellierten Dreiecke. Nach dem Ausdruck „foreach triangle“ sind der Normalenvektor und die drei Koordinatenpunkte ohne zusätzliche Kennzeichnung beschrieben. Der interne Zähler ist eine Nummerierung der modellierten Dreiecke. Jeder Punkt besteht dabei aus 12 Bytes, die Nummerierung jedoch nur aus zwei Bytes.

Für die Umsetzung des Datenimports wurde die Funktion

```
[Typ, Koordinaten, Faces, Color] = STL_read(filename)
```

implementiert, welche den Namen des STL-Files als Eingabewert besitzt. Die Dateinamenserweiterung „.stl“ muss mit angehängt werden. Es ist zu beachten, dass die Funktion die ASCII-Dateien von CAD-Programmen, als auch die binären Dateien von 3D-Software lesen kann. Als Ausgabewert erhält der Nutzer den Typ der Datei, die Koordinaten (Vertices), die Faces und die Farbe (Color). Die Faces definieren die Reihenfolge, in der die Koordinaten der Eckpunkte visualisiert werden. Falls eine binäre Datei vorliegt oder keine Farbe definiert wurde, wird das Objekt standardisiert rot eingefärbt.

Die grundlegende Struktur der Funktion ist in Abbildung 10 aufgezeigt:

```
if erste Wort == 'solid'
    Typ = 'ASCII'
    Prüfe, ob Wörter mit c oder v beginnen
    => Punkte nach c = Color
    => Punkte nach v = Koordinaten (Vertices)
    => Ordne Faces den Koordinaten zu
    if Color = nicht definiert -> Color = [1, 0, 0] (rot)
else
    Typ = 'Binär'
    Übernehme erste 12 Bytes als Face
    Übernehme nächsten 36 Bytes als Vertices
    Fange nach 2 zusätzlichen Bytes erneut an
    Color = [1, 0, 0] (rot)
end
```

Abbildung 10: Grundlegende Struktur der Funktion `STL_read`

3.2.1 Beispiel eines Imports aus Creo Parametric 2.0

Zum Verständnis wird nun ein Beispiel für einen in Creo Parametric 2.0 modellierten Würfel gezeigt, welcher nach MATLAB als ASCII-File exportiert wird. Der Würfel besitzt die Seitenlänge 10 und das Koordinatensystem befindet sich in seinem Mittelpunkt. Nachdem ein Würfel in dem CAD-Programm modelliert wurde, kann dieser in einer STL-Datei exportiert werden. Dazu muss man diesen unter einer neuen Kopie speichern und als Typ „Stereolithographie“ wählen.

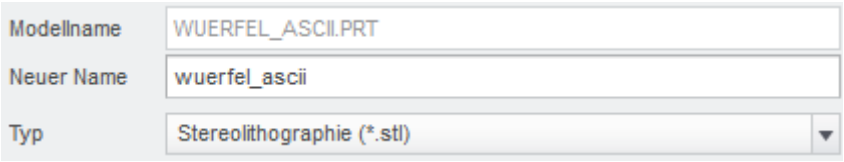

Abbildung 11: Export eines Modells in den STL-Typ

Da STL-Files die Objekte in Dreiecke zerlegen, wird jede Seite des Würfels zweigeteilt. Der Körper ist somit durch insgesamt 12 Dreiecke definiert, wovon jedes aus drei Punkten mit je einen Wert für deren Position auf der x-, y-, und z-Achse besteht. Zusätzlich besitzt jede Fläche ein Face, welches ebenfalls aus drei Punkten besteht. Da in diesem Fall keine Farbe im CAD-Programm vorgegeben wurde, wird die Variable `Color` automatisch auf [1, 0, 0] gesetzt.

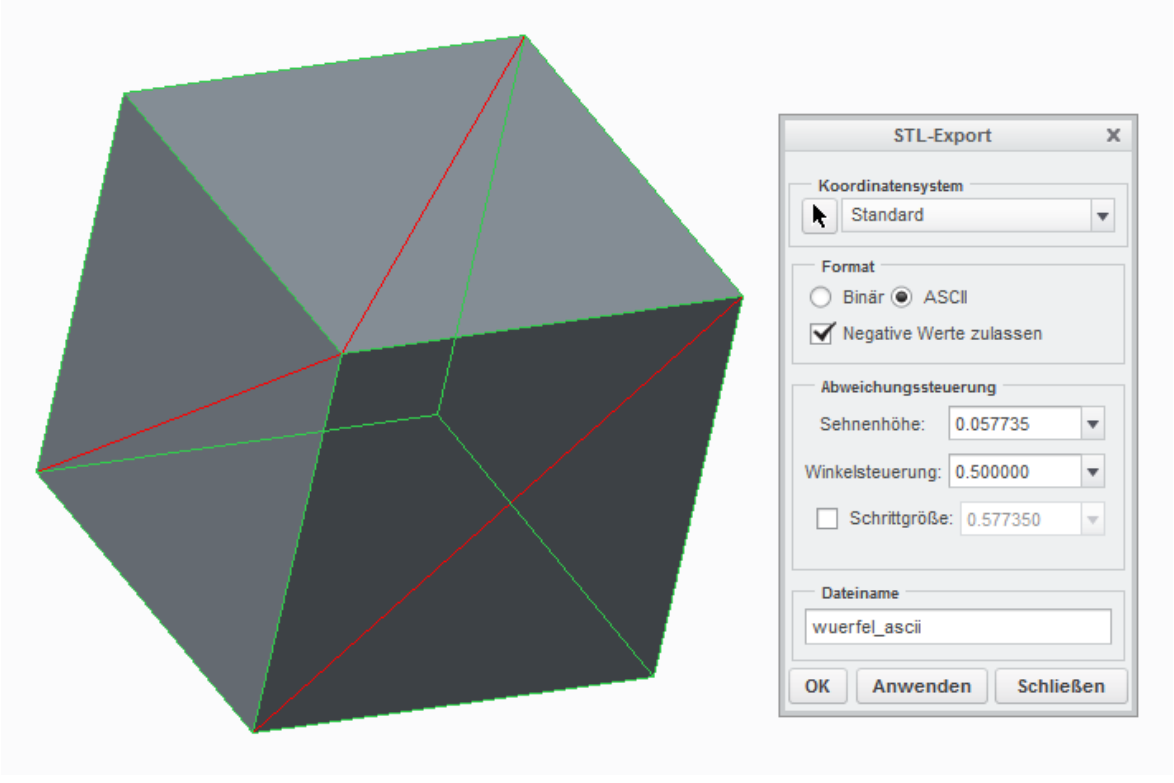

Abbildung 12: Export einer ASCII-STL-Datei aus Creo Parametric 2.0

Beim Export öffnet sich wie in Abbildung 4 gezeigt ein neues Fenster. Darin wählt der Nutzer das gewünschte Format. Nach der Wahl müssen noch die verwendeten Objekte markiert werden, was mit dem Drücken auf den Button „Anwenden“ erfolgt.

Nun ist der Import der Datei nach MATLAB möglich. Hierfür nutzt man die Funktion `STL_read`, welche wie folgt benutzt werden kann:

```
[Typ, Koordinaten, Faces, Color] = STL_read('wuerfel_ascii.stl')
```

Wie zu erkennen darf die Namenserweiterung „.stl” nicht fehlen, da MATLAB sonst nach Datenbanken gleichen Namens sucht.

Im Beispiel erhält der Nutzer alle möglichen Ausgabewerte. In Abbildung 5 erkennt man, dass der Import der Daten korrekt ausgeführt wurde. Sowohl die Farbe als auch der Typ wurden richtig zugewiesen. Außerdem wurde für jedes der 12 Dreiecke je ein Normalenvektor und drei Koordinatenpunkte übernommen. Am minimalen und maximalen Wert der Koordinaten lässt sich die Seitenlänge 10, wie sie in Creo Parametric 2.0 definiert wurde, feststellen.

Workspace

Name	Value	Min	Max
Color	[1 0 0]	0	1
Faces	<12x3 double>	1	36
Koordinaten	<36x3 double>	-5	5
Typ	'ASCII-Format'		

Abbildung 13: ASCII-Ausgabewerte in MATLAB

3.2.2 Beispiel eines Imports aus der 3D-Software Blender 2.64

Da CAD-Programme nicht für jeden Nutzer zugänglich sind, wird noch einmal der Datenimport aus der kostenlosen 3D-Software Blender 2.64 vorgestellt. Hierzu wird ein Würfel mit der Seitenlänge 2 modelliert.

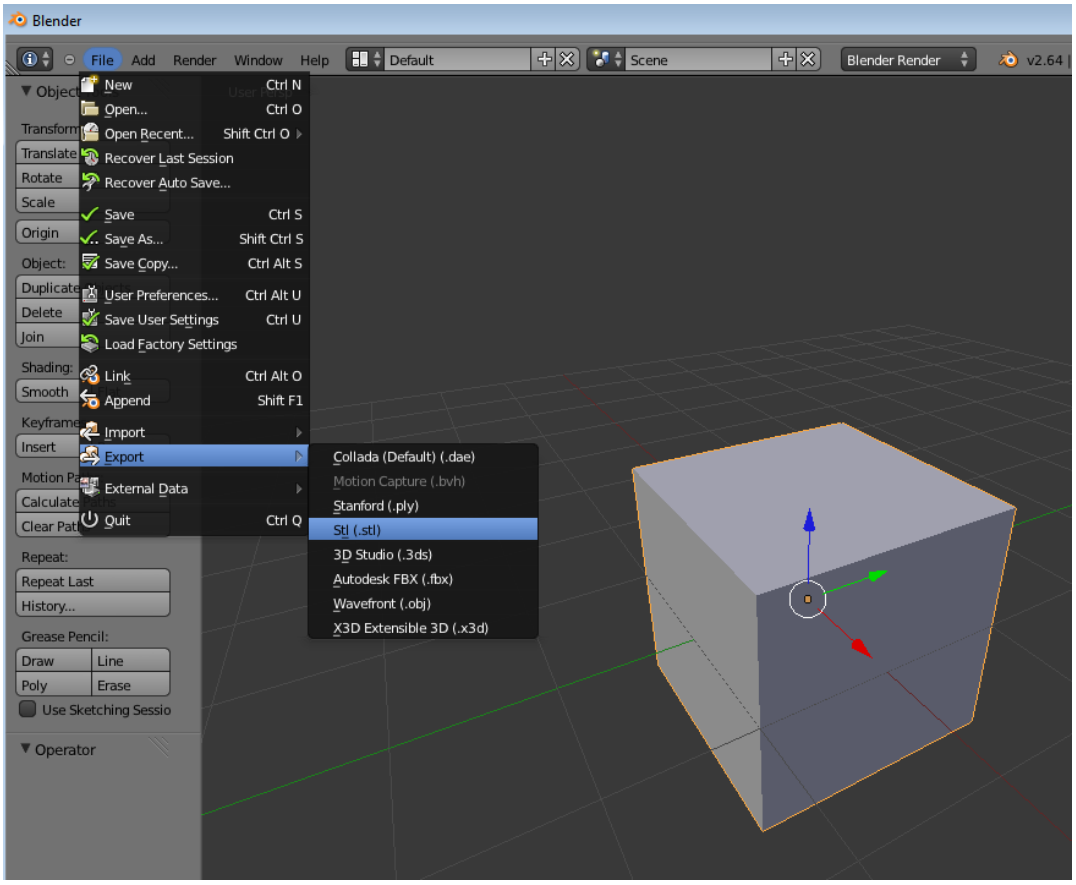

Abbildung 14: Export einer binären STL-Datei aus Blender 2.64

Für den Export muss der Körper vom Nutzer markiert werden und der Export standardisiert im Binär-Format vorgenommen werden. Der Import nach MATLAB kann dann analog zu dem vorangegangenen Kapitel erfolgen:

```
[Typ, Koordinaten, Faces, Color] = STL_read('wuerfel_bin.stl')
```

Anhand von Abbildung 7 wird verdeutlicht, dass auch hier alle Werte korrekt importiert wurden.

Workspace

Name	Value	Min	Max
Color	[1 0 0]	0	1
Faces	<12x3 double>	1	36
Koordinaten	<36x3 double>	-1.0000	1.0000
Typ	'Binaer-Format'		

Abbildung 15: Binäre Ausgabewerte in MATLAB

4 Darstellungsmöglichkeiten der Datenpunkte in MATLAB

MATLAB bietet diverse Möglichkeiten zur räumlichen Darstellung von Objekten, welche in Tabelle 5 zusammengefasst sind:

Tabelle 5: Bewertung der Darstellungsmöglichkeiten

Funktion	Bewertung
plot3	- Visualisiert Linien - Ungeeignet, da nur Linien dargestellt werden können
fill3	- Visualisiert Flächen - Ungeeignet, da nur Flächen dargestellt werden können
meshgrid	- Visualisiert 3D-Gitternetz - Ungeeignet, da Flächen ausgefüllt werden müssen
surf	- Visualisiert Gitternetz und füllt dieses mit Flächen - Geeignet, da 3D-Objekte modelliert werden können
patch	- Visualisiert Dreiecke - Geeignet, da 3D-Objekte aus den Dreiecken zusammengesetzt werden können

Da für die Visualisierung nur die `surf` und die `patch` Funktion sinnvoll sind, werden diese hier gegenübergestellt und einer der beiden Befehle ausgewählt. Nachdem die Auswahl der entsprechenden Funktion getroffen ist, wird deren Implementierung näher erläutert.

4.1 Vergleich der `surf` und `patch` Funktion

Die surf- und die patch-Funktion eignen sich beide zur Visualisierung dreidimensionaler Objekte. Jedoch gibt es zwischen den zwei Befehlen Unterschiede, was die Eingabewerte und die damit verbundene Kompatibilität mit dem STL-Format betrifft.

Um diese Unterschiede zu beschreiben, wird an dieser Stelle die im Anhang befindliche Datei „bootganz.stl“ visualisiert. Dieses Beispiel wurde gewählt, da es besonders komplex und ohne die Hilfe des CAD-Imports nahezu unmöglich in MATLAB zu modellieren ist.

Damit die eingelesenen Punkte mit der `surf` Funktion visualisiert werden können, müssen diese zuerst neu geordnet werden. Hierzu wurde die Funktion `STL_Surf` implementiert, welche im Anhang beigefügt ist. Deren Funktionsweise ist in Abbildung 16 an einem Beispiel erklärt:

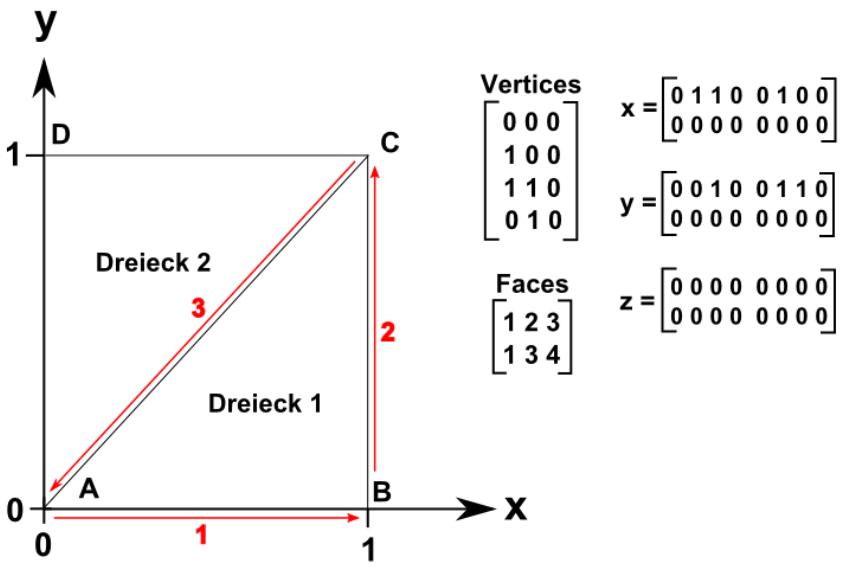

Abbildung 16: Funktionsweise von `STL_Surf`

Hier soll eine Fläche dargestellt werden, welche aus zwei Dreiecken besteht und auf der x-y-Ebene bei einer Höhe z=0 liegt. Nach dem STL-Import sind nur die Vertices und Faces gegebene Größen. Die Vertices sind die Koordinaten der Eckpunkte der Dreiecke, während die Faces die Reihenfolge der darzustellenden Vertices angeben. Anhand derer sollen die Matrizen für x, y und z erstellt werden, mit denen sich die `surf` Funktion ausführen lässt. Die Dreiecke werden dabei nacheinander modelliert. Die vier darzustellenden Punkte sind mit Buchstaben, die Reihenfolge der Definition der Punkte mit roten Zahlen gekennzeichnet. Die Funktion `STL_Surf` ordnet die Koordinaten entsprechend der Faces den x-, y- und z-Spalten der Vertices zu. Dabei besteht jede Matrix eines Dreiecks aus zwei Zeilen und vier Spalten. Nun bildet jede Zeile der Matrix

`Faces` ein Dreieck, wobei der erste Wert am Ende erneut definiert werden muss. So wird sichergestellt, dass die Linien geschlossen sind und somit eine Fläche bilden. Anschließend werden die Vertices in die erste Zeile der x-, y- und z-Matrizen eingetragen, die laut `Faces` vorgegeben werden. So besteht im Beispiel das Dreieck ABC aus den Vertices 1, 2, 3 und 1. Die Definition erfolgt also durch die Punkte A, B, C und A. Diese Koordinaten werden in die Matrizen übernommen. Die zweite Zeile der Matrizen besteht dabei aus jeweils dem ersten x-, y- oder z-Wert. Dieser wird für alle vier Spalten übernommen. Danach lassen sich die Objekte durch die Funktion `surf(x,y,z)` visualisieren. `STL_Surf` kann nach dieser Vorgehensweise beliebig dimensionierte STL-Dateien transformieren. Das in Abbildung 17 visualisierte Boot wurde durch das Kommando

```
[Zeit_0, Zeit_1] = Surf_STL('bootganz.stl')
```

definiert. Zeit_0 misst unter Verwendung des `tic-toc` Befehls die Dauer der Umstellung der Matrizen, während Zeit_1 die Dauer der eigentlichen Visualisierung wiedergibt.

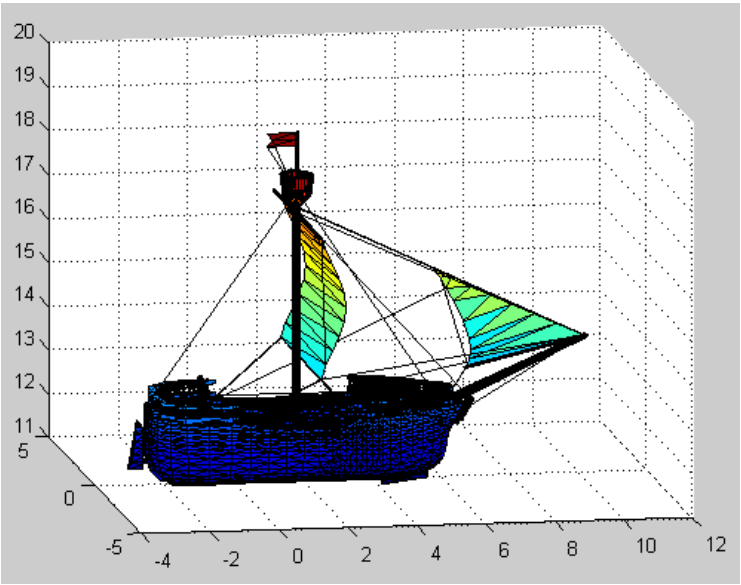

Abbildung 17: Boot-Objekt mit Hilfe der `surf` Funktion

Es ist zu erkennen, dass das Boot nicht korrekt visualisiert wurde, da an einigen Stellen zusätzliche Linien gezeichnet wurden. Die für die Berechnung benötigte Zeit beträgt 1.5214s und das grafische Darstellen benötigt 0.025s.

Anders als bei `surf` sind bei der `patch` Funktion keine zusätzlichen Berechnungen notwendig. Alle Eingabewerte liegen nach dem Import vollständig und richtig angeordnet vor. Die in Abbildung 18 gezeigte Funktion `STL_Patch(filename)` stellt die Koordinaten und die Faces mit Hilfe des `patch` Befehls nach dem Einlesen dar und misst zusätzlich die für die Visualisierung benötigte Zeit:

```
Zeit = Patch_STL('bootganz.stl')
```

```
function Zeit = STL_Patch(filename)

...

[Typ, Koordinaten, Faces, Color] = STL_read(filename);

tic                %Beginn der Zeitmessung

patch('Vertices',Koordinaten,'Faces',Faces,'Facecolor',Color);
grid on

Zeit = toc;        %Ende der Zeitmessung
```

Abbildung 18: Funktion STL_Patch

Das Ergebnis ist das exakt dargestellte Boot, wie es in Abbildung 19 aufgezeigt wird:

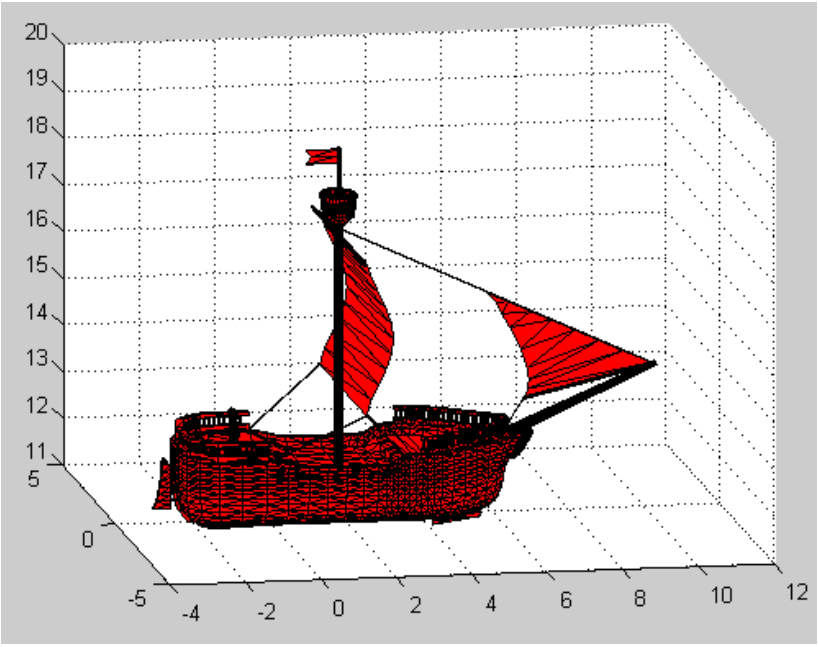

Abbildung 19: Boot-Objekt mit Hilfe der patch-Funktion

Die `patch` Funktion visualisiert in 0.0081s das Boot nicht nur schneller, sondern stellt es im Gegensatz zu der `surf` Funktion auch korrekt dar. Aufgrund dieser Vorteile wurde die `patch` Funktion für die grafische Darstellung der importierten STL-Dateien gewählt.

4.2 Besonderheiten der Zeichenfunktion

Beim Einlesen einer STL-Datei werden durch die Funktion `STL_read` sowohl die Koordinatenpunkte, als auch die Faces und gegebenenfalls die Farben importiert. Beim Visualisieren ist darauf zu achten, dass diese einmal mit der `patch` Funktion dargestellt werden, danach aber der `set` Befehl zu verwenden ist. So wird sichergestellt, dass die Objekte nicht mehrmals gezeichnet werden. Zusätzlich muss die grafische Darstellung einem Handle, hier `p`, zugewiesen werden, welches diese kennzeichnet. Bei der ersten Darstellung wird das Kommando

```
p= patch('Vertices',Koordinaten,'Faces',Faces,'Facecolor',Color)
```

verwendet. Sollen neue Koordinaten des Objekts visualisiert werden, bedient man sich des `set` Befehls. Anstatt immer wieder neue Objekte zu erzeugen, werden die Koordinaten der schon vorhandenen Körper angepasst. Eine Anpassung der Faces ist dabei nicht erforderlich. Mit

```
set(p,'Vertices',Neue_Koordinaten)
```

lässt sich die vorhandene Position und Drehung des Objekts anpassen. Durch die Benutzung des Handles `p` erkennt MATLAB, dass es das mit dem Handle gekennzeichnete Objekt verändern soll. Es ist außerdem möglich noch andere Parameter, wie beispielsweise die Farbe, mit dem `set` Befehl zu beeinflussen.

5 Realisierung des Lösungsentwurfs mit Creo Parametric 2.0

In diesem Kapitel wird das Erzeugen von Objekten mit einem CAD-Programm am Beispiel von Creo Parametric 2.0 erläutert. Es kann zwischen drei Fällen von Modellen, welche man mit der MatlabKK-Visualization-Toolbox visualisieren kann, unterschieden werden.

Zum einen Roboter, welche aus Bauteilen zusammengesetzte Körper darstellen, die sich im Rahmen einer vordefinierten Kinematik frei im Raum bewegen können. Eine weitere Möglichkeit bieten Umweltobjekte, welche starre, unbewegliche Körper darstellen und demzufolge keine eigene Kinematik besitzen. Zudem ist es möglich, Parts zu implementieren. Parts sind starre Objekte, welche von einem Roboter gegriffen und anhand der Kinematik des Roboters von diesem bewegt werden können.

5.1 Modellierung der Roboter

Beim Erzeugen von Objekten in Creo Parametric 2.0 muss auf die Umstellung der SI-Einheiten auf das metrische System geachtet werden. Zusätzlich sind die Ausrichtung der verwendeten Koordinatensysteme von MATLAB und Creo Parametric 2.0 verschieden. Während in MATLAB die Höhe mit der Achse z und die Breite mit y beschrieben wird, sind diese Vektoren in dem CAD-Programm vertauscht.

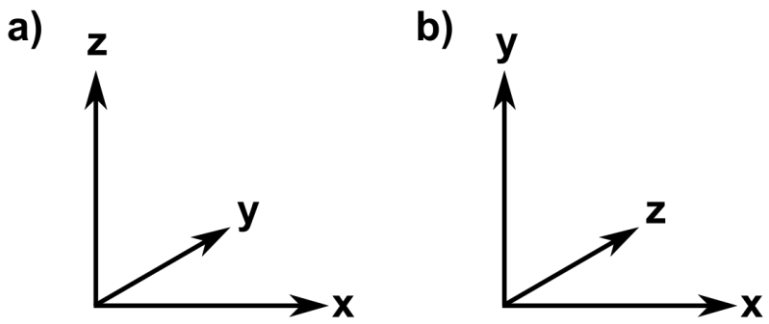

Abbildung 20: Koordinatensystem in MATLAB (a) und Creo Parametric 2.0 (b)

Es gibt drei vordefinierte Bezugsebenen, welche die Ansicht auf das Koordinatensystem in ihren Bezeichnungen „RIGHT“-, „TOP“- und „FRONT“-Ebene wiedergeben. Allerdings ist dabei die Vertauschung der y- mit der z-Achse zu beachten. Somit stellt beispielsweise nicht die TOP-Ebene, sondern die FRONT-Ebene die Draufsicht auf das Koordinatensystem in Bezug auf MATLAB dar. Modelliert wird demzufolge nicht anhand der Ebenen, sondern anhand des MATLAB-Koordinatensystems.

Beim Modellieren der Roboter ist die Zerlegung des Roboters in einzelne CAD-Dateien zu beachten. Es gilt also für jedes Gelenk und der Basis eine eigene Modellierung vorzunehmen. Entsprechend der verwendeten 6-Achs-Roboter werden demnach sieben Dateien benötigt. Entscheidend ist die Positionierung der Gelenke im Bezug zum Basiskoordinatensystem des CAD-Programms. Während die Roboterbasis auf der Fläche steht, welche die x- und die y-Achse bilden, sollte der Ursprung bei den Gelenken immer dort positioniert sein, wo diese Rotationen vornehmen sollen.

Der in Abbildung 21 skizzierte Kawasaki FS003N verdeutlicht die Zerlegung eines Roboters in Teilabschnitte:

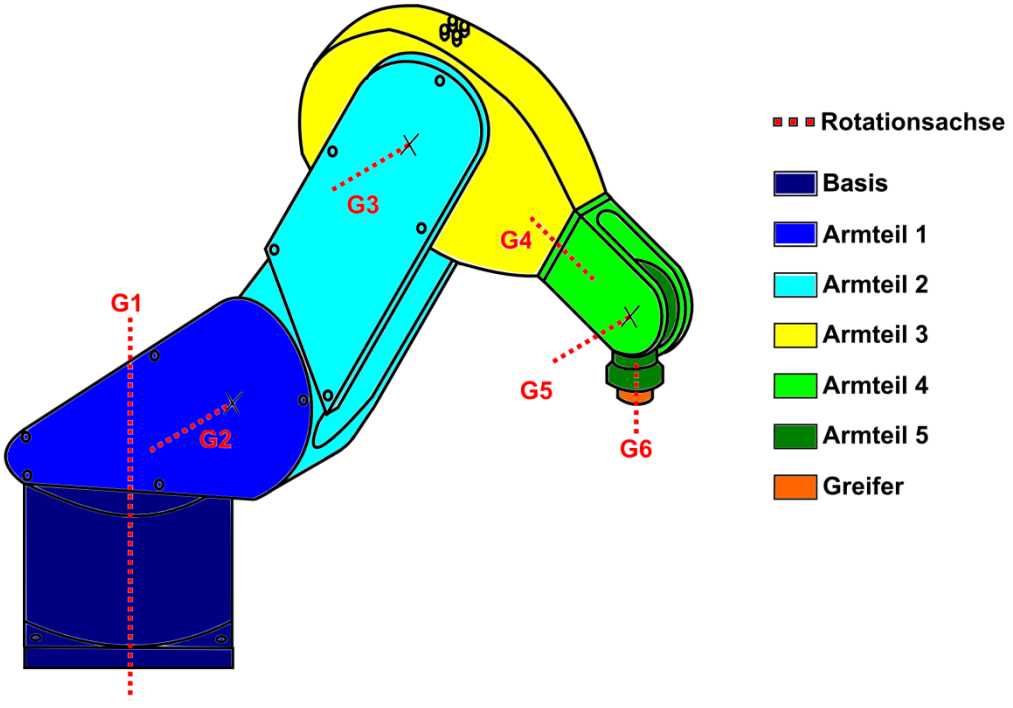

Abbildung 21: Skizze des Kawasaki-Roboterarms

Die Zuweisung G bezeichnet die Gelenke, welchen je eine Rotationsachse zugeordnet ist. Der Roboter besitzt insgesamt sieben einzeln zu modellierende Abschnitte. Jeder ist in einer STL-Datei abzuspeichern. Der Ursprung der Koordinatensysteme der Roboterabschnitte ist dorthin zu legen, wo sich dieser um seine Rotationsachse dreht. So ist beispielsweise der Koordinatenursprung im CAD-Programm für Armteil 3 auf der Rotationsachse von Gelenk 3 zu legen.

5.2 Modellierung der Umweltobjekte und Parts

Anders als bei der Modellierung der Roboter werden Umweltobjekte und Parts nicht aus zusammengesetzten Abschnitten, sondern als einzelnes Objekt konstruiert. Da Umweltobjekte keine eigene Kinematik besitzen, kann die Position des CAD-Koordinatensystems beliebig zu gewählt werden. Sie können somit ohne spezielle Vorgehensweise in CAD modelliert und in MATLAB eingelesen werden.

Ähnlich verhält es sich mit den Parts. Zwar können diese von den Roboterarmen bewegt werden, jedoch besitzen sie keine eigene Rotationsachse. Die Rotationsachse wird erst vom Greifer an den Punkt definiert, an dem die Greiferbacken das Part-Objekt greifen. Deshalb kann auch hier die Position des Koordinatenursprungs beliebig gewählt werden.

6 Einbindung des Lösungsentwurfs in die MatlabKK-Visualization-Toolbox

An dieser Stelle wird die Implementierung der vorher erwähnten Erweiterungsmöglichkeiten in die MatlabKK-Visualization-Toolbox erläutert. Außerdem werden die zwei eigenständigen Funktionen `env_mdl.m` und `Part.m` auf die Erweiterungen angepasst. Der Nutzer benötigt zudem die Funktion `STL_read.m`.

6.1 Erweiterung der MatlabKK-Visualization-Toolbox

Die MatlabKK-Visualization Toolbox ist in dem MATLAB-File `VirtualRobot` beschrieben und bedient sich der Objekt-Orientierten Programmierung (OOP). Die verwendeten Variablen werden dabei unter `properties` zusammengefasst. Da die Koordinaten der Roboter nicht mehr über die Tabellen Modell_Kawasaki.mat und Modell_Kuka.mat eingelesen werden, sondern über die einzelnen STL-Dateien, ist es notwendig, die Variablen anzupassen. So ist für jedes Gelenk und die Basis je eine Modell- und eine Handle-Variable zu definieren, wie in Abbildung 22 gezeigt:

```
% Handles zu den Patch Objekten in der Figure
mdl     %Roboter
mdl1    %Greifer
mdl2    %Greiferzange
mdlring    %Farbige Markierung an der Roboterbasis
mdl_basis              %Einzelne Handles des Roboterarms
mdl_arm1
mdl_zwischenarm
mdl_arm2
mdl_dreharm
mdl_hand

% Modell für den Kawasaki-oder Kukaroboter
Modell0     % Basis
Modell1     % Arm1
Modell2     % Zwischenarm
Modell3     % Arm2
Modell4     % Dreharm
Modell5     % Hand
Modell6     % Greifer
Modell7     % Greiferbacke
Modell8     % Gegriffenes Teil
```

Abbildung 22: Erweiterung der `propertie` Variablen in `VirtualRobot.m`

Zusätzlich können in der OOP interne Funktionen, sogenannte Methoden, implementiert werden. Um die CAD-Schnittstelle herzustellen, müssen insgesamt sechs Methoden angepasst werden:

- `VirtualRobot(Type,port,pose,s_axis)`
- `start(obj)`
- `rstop(obj)`
- `Draw(obj)`
- `place_env(name,pose,fc)`
- `place_part(name,pose,fc)`

Beim Starten der Toolbox wird `VirtualRobot(Type,port,pose,s_axis)` aufgerufen. In dieser Methode erfolgt das Einlesen der Datenpunkte der Objekte. Anhand der Variable `Type` wird der verwendete Roboter gewählt. Anschließend erfolgt der Import der entsprechenden Tabelle. Für den Kawasaki-Roboter wird beispielsweise der Befehl

```
obj.Modell=load('Modell_Kawasaki.mat')
```

verwendet. Dieser Befehl ist durch `STL_read` zu ersetzen, wie es in Abbildung 23 für den Kawasaki-Dreharm dargestellt ist.

```
%Laedt Dreharm
[Typ ,Koordinaten4, Faces4,Color4] = STL_read('kawasaki_dreharm.stl');
obj.Modell4.Faces = Faces4;
obj.Modell4.Vertices = Koordinaten4;
obj.Modell4.Color = Color4;
```

Abbildung 23: Einlesen des Kawasaki-Dreharms aus einer STL-Datei

Es ist jeder einzelne Abschnitt des gewählten Roboters in dieser Form einzulesen. Die Ausgabevariablen werden anschließend als Handles des Objekts definiert, um deren Aufruf in anderen Methoden zu ermöglichen.

In der Methode `start(obj)` ist der Import der Statusmarkierung zu ergänzen. Vor der Erweiterung der Toolbox mit der CAD-Schnittstelle wurden die Koordinaten für die Markierung mit der `surf` Funktion visualisiert. Anschließend ist die Farbe definiert worden. Für Letzteres verwendet man wie in Kapitel 4.2 erläutert den `set` Befehl. Nach der Erweiterung wird, wie in Abbildung 24 gezeigt, der Import der CAD-Daten mit `STL_read` ausgeführt. Die Darstellung erhält das Handle `obj.mdlring`, wird mit der `patch` Funktion visualisiert und die Flächen des Objekts werden grün eingefärbt.

```
%Einlesen von robot_ring.stl als Statusmarkierung
[Typ ,Koordinaten_ring, Faces_ring] = STL_read('robot_ring.stl');
obj.mdlring = patch('faces', Faces_ring, 'vertices' ,Koordinaten_ring);
set(obj.mdlring,'FaceColor','green','EdgeColor','none')
```

Abbildung 24: Einlesen und Färben der Markierung

Außerdem muss die Variablenzuweisung der Markierung in `rstop(obj)` ersetzt werden. Sobald der Roboterarm stoppt, färbt sich die Statusmarkierung rot. Hierzu wurde in dem Befehl

```
set(obj.mdlring,'FaceColor','red')
```

die Variable `obj.mdl3` zu `obj.mdlring` geändert.

Die umfangreichste Erweiterung ist die der Zeichenmethode `Draw(obj)`. Hier erfolgt zunächst die Entscheidung, ob der Roboter vorher schon visualisiert wurde. Falls dies der Fall ist, so werden die eingelesenen Koordinaten zunächst in neue x-, y- und z-Werte unterteilt. Anschließend werden die einzelnen Gelenkabschnitte mit Hilfe der `surf` Funktion visualisiert und farblich angepasst. Andernfalls werden die bereits vorhandenen Objekte mit dem `set` Befehl verändert, was ein wiederholtes Darstellen eines Objekts verhindert. Zwar ist die Vorgehensweise unverändert, doch die einzelnen Befehle sind wie in Abbildung 25 zu erkennen modifiziert.

```
%Teil2 Arm1
[x2, y2, z2]=obj.transf_achse(F(2).Frames,...
obj.Modell1.Vertices(:,1), obj.Modell1.Vertices(:,2), obj.Modell1.Vertices(:,3));

...

%Weist die neuen Koordinaten zu
arm1 = [x2, y2, z2];

...

%Beim ersten Plot werden die Patchobjekte erstellt, sowie ihre
%Eigenschaften gesetzt. Bei allen nachfolgenden Darstellungen
%werden jeweils nur die neu berechneten Punktewolken uebergeben.
obj.mdl_arm1 = patch('faces', obj.Modell1.Faces, 'vertices' ,arm1);

...

%Verändert die Farbe der Handles beim ersten Plotten
set(obj.mdl_arm1,'FaceColor',fc,'EdgeColor',...
'none','FaceLighting','gouraud','AmbientStrength',2);

...

%Verändert die Koordinaten
set(obj.mdl_arm1,'Vertices',arm1);
```

Abbildung 25: Erweiterung der `Draw` Methode am Beispiel von Arm1

Wie zu erkennen erfolgt zuerst die Transformation der Koordinaten in ihre neuen x-, y- und z-Werte. Damit die Objekte richtig positioniert sind, werden die Werte mit der entsprechenden Frame-Matrize multipliziert. Dies geschieht in der Methode `transf_achse(T,x,y,z)`, wobei die Variable `T` für das Frame und `x`, `y` und `z` für die ursprünglich eingelesenen Koordinaten stehen. Anschließend werden die Koordinaten unter einer neuen Variablen zusammengefasst, um diese in der `patch` Funktion verwenden zu können.

Das Programm prüft, ob der Roboter zum ersten Mal dargestellt wird. Ist dies der Fall, werden die Objekte mit der `patch` Funktion visualisiert und die Farbe mit Hilfe des `set` Befehls angepasst. Die Variable `fc` definiert im Beispiel die hellbraune Färbung des Kawasaki-Roboterarms. Ist schon eine Visualisierung der Objekte erfolgt, werden die neuen, mit den Frame-Matrizen multiplizierten Koordinaten durch den `set` Befehl übernommen.

Die Methoden `place_env(name,pose,fc)` und `place_part(name,pose,fc)` werden analog zueinander verändert. Das Laden der Koordinaten geschah ursprünglich mit dem Befehl `load(name)` und wurde durch `STL_read` ersetzt. In beiden Methoden ist das Einlesen der STL-Dateien und unterteilen der Koordinaten in ihre x-, y- und z-Werte vorzunehmen, wie es in Abbildung 26 für die Methode `place_env` zu sehen ist. Damit keine Variable mehrfach definiert ist, werden diese mit den Endungen „`_env`“ oder „`_part`“ versehen. Zusätzlich sind die Eingabewerte der Funktionen `env_mdl` und `Part` um die Variable `Faces` zu erweitern.

```
[Typ ,Koordinaten_env, Faces_env,Color_env] = STL_read(name);
...
x = Koordinaten_env(:,1);
y = Koordinaten_env(:,2);
z = Koordinaten_env(:,3);
Faces = Faces_env;

environment=[environment env_mdl(x,y,z,frame,fc,Faces)];
```

Abbildung 26: Änderung der `place_env` Methode

6.2 Erweiterung der `env_mdl` und `Part` Funktion

Auch in den zwei eigenständigen Funktionen gilt es, zuerst die `properties` um die notwendige Variable `Faces` zu erweitern. Die in Kapitel 6.1 ergänzte Variable `Faces` in den `place_env` und `place_part` Methoden muss auch in der Methode `env_mdl` und `Part` übernommen werden. Außerdem sind die Variablen dem Objekt als Handle zuzuweisen. In `Draw(obj)` erfolgt die Prüfung, ob das Objekt zum ersten Mal visualisiert wird. Im Anschluss wird der Körper entweder mit der `patch` Funktion visualisiert und dessen Farbe angepasst, oder das bereits dargestellte Objekt mit dem `set` Befehl verändert. Alle Änderungen sind in Abbildung 27 dargestellt:

```
properties
    ...
    Faces
end

methods
    function obj=env_mdl(x,y,z,frame,fc,Faces)
        obj.x=x;
        obj.y=y;
        obj.z=z;
        obj.Frame=frame;
        obj.Faces=Faces;
    end

    function Draw(obj)
        ...
        if isempty(obj.mdl)
            obj.mdl = patch('faces', obj.Faces, 'vertices' ,[obj.xc,obj.yc,obj.zc]);
            set(obj.mdl,'FaceColor',obj.fc,'EdgeColor','none')
        else
            set(obj.mdl,'vertices' ,[obj.xc,obj.yc,obj.zc]);
        end
    end
end
```

Abbildung 27: Erweiterung der `env_mdl` und `Part` Funktion

6.3 Nachweis der Funktionsfähigkeit

Zum Testen der erweiterten Toolbox wurde ein in STL modellierter Kawasaki-Roboter geladen, welcher zwei anschließend hinzugefügte Reagenzgläser auf einem Tisch neu positioniert. Das Beispiel ist an das auf der Homepage der Forschungsgruppe CEA [4] veröffentlicht und ist für die ursprüngliche Toolbox ausgeführt worden. Um die Ergebnisse der Erweiterung zu veranschaulichen, wurden die Befehle des Beispiels auf die erweiterte Toolbox angepasst und die Tabellen durch STL-Dateien ersetzt. Tabelle 6 verdeutlicht die verwendeten Befehle für den Visualisierungs- und Control-Rechner am Beispiel der veränderten MatlabKK-Visualization-Toolbox. Die Befehle stimmen mit dem Beispiel auf der CEA Homepage überein, nur dass hinter dem Namen jedes eingelesenen Objekts die Endung „.stl“ angefügt wird.

Tabelle 6: Befehlsfolge des Tests

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	VirtualRobot.place_env('table**.stl**', [0 -500 0 0 0 0]);
	VirtualRobot.place_env('base**.stl**', [200 -450 400 0 0 0], [0,1,0]);
	VirtualRobot.place_env('base**.stl**', [-200 -450 400 0 0 0], [0,1,0]);
	VirtualRobot.place_env('base**.stl**', [0 -450 400 0 0 0], [0,1,0]);
	VirtualRobot.place_part('test_tube**.stl**', [200 -450 425 0 0 0], [1,0,0]);
	VirtualRobot.place_part('test_tube**.stl**', [-200 -450 425 0 0 0], [1,0,0]);
	view(45,45);
	zoom(2);	

5		load waypoints;
6		rmove(r1, waypoints);
7		robot(r1, ‚close‘)
8	VirtualRobot.delete_all	

Die Ergebnisse der Durchläufe sind in den Abbildungen 28 und 29 dargestellt. In beiden Fällen führt der Roboterarm die gleichen Bewegungen aus. Die Implementierung der Kinematik ist demnach unverändert. Verändert hat sich lediglich die Darstellung der Roboter, Umweltobjekte und Parts. Hier sind die Vorteile einer CAD-Schnittstelle deutlich zu erkennen. Zum einen sind die Kanten der Körper abgerundet und sehen damit realistischer aus. Zum anderen können die Objekte detaillierter dargestellt werden. So sind bei dem Tisch runde Standfüße und in den Reagenzgläsern Hohlräume modelliert. Vor allem die Hohlräume und die Abrundungen der Kanten sind mit der vorher verwendeten `surf` Funktion schwer zu modellieren. Des Weiteren sind Flächen nun gewollt einfarbig visualisiert. Der Fehler, dass einzelne Flächen in

mehrere Verschiedenfarbige unterteilt werden, wie es beim Tisch zu sehen ist, wurde behoben. Zudem sind die Objekte besser schattiert, sodass einzelne Abschnitte des Roboters besser zu erkennen sind.

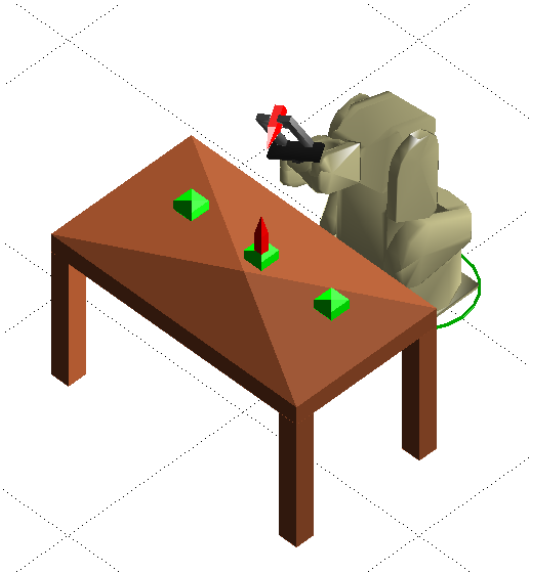

Abbildung 28: Test mit ursprünglicher Toolbox

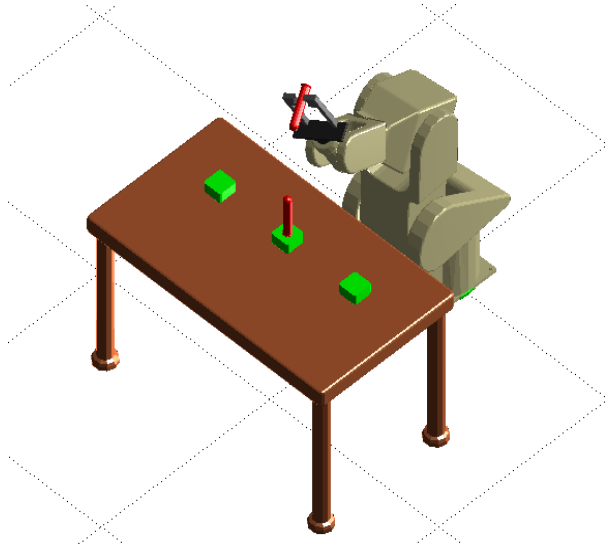

Abbildung 29: Test mit erweiterter Toolbox

Das Ziel der Implementierung einer CAD-Schnittstelle ist erreicht und deren Funktionsfähigkeit nachgewiesen. Importierte STL-Dateien lassen sich visualisieren und detailreicher darstellen als mit der `surf` Funktion. Eventuelle Hindernisse, wie das Modellieren von Löchern oder Hohlräumen, sind demzufolge beseitigt.

7 Zusammenfassung und Ausblick

Zusammenfassend lässt sich sagen, dass das Ziel der Implementierung einer CAD-Schnittstelle erreicht wurde. STL-Objekte können mit einem CAD-Programm erstellt und ohne zusätzliche Nachbearbeitung durch den Nutzer mit Hilfe der MatlabKK-Visualisation-Toolbox visualisiert werden. Die verwendete `patch` Funktion stellt Objekte schneller dar als die vorher benutzte `surf` Funktion. Für Anwender ohne Zugang zu einem CAD-Programm wurde der Export von STL-Dateien aus dem kostenlosen 3D-Programm Blender vorgestellt. Durch die Erweiterung der Toolbox können die dargestellten Objekte wesentlich einfacher und detailreicher modelliert und modifiziert werden. In Zukunft könnte man noch weitere Roboter oder ganze Produktionsstraßen für die Toolbox hinzufügen oder den Import anderer Dateitypen ermöglichen.

Literaturverzeichnis

[1] 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

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

[3] 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

[4] 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)

[5] PARAMETRIC TECHNOLOGY CORPORATION (2012). PTC Creo [Online]; http://de.ptc.com /product/creo/ (30.11.2012)

[6] DASSAULT SYSTÈMES (2012). 3DS CATIA [Online]; http://www.3ds.com/de/products/catia/ (30.11.2012)

[7] DASSAULT SYSTÈMES (2012). 3DS SolidWorks [Online] http://www.solidworks.de/ (30.11.2012)

[8] THE MATHWORKS, INC (2012). Simulink 3D Animation [Online] http://www.mathworks.de/ products/3d-animation/ (30.11.2012)

[9] Blender [Online]; http://www.blender.org/ (30.11.2012)

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

[11] CAD [Online]; http://de.wikipedia.org/wiki/CAD (30.11.2012)

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
- env_mdl.m
- Part.m
- STL_read.m
- STL_Surf.m
- STL_Patch.m

Anlagenverzeichnis STL-Dateien:

- kawasaki_basis.stl
- kawasaki_arm1.stl
- kawasaki_zwischenarm.stl
- kawasaki_arm2.stl
- kawasaki_dreharm.stl
- kawasaki_hand.stl
- kawasaki_greifer.stl
- kawasaki_greiferbacke.stl
- kawasaki_greiferbacke_zu.stl
- robot_ring.stl
- bootganz.stl
- wuerfel_ascii.stl
- kuka_basis.stl
- kuka_arm1.stl
- kuka_zwischenarm.stl
- kuka_arm2.stl
- kuka_dreharm.stl
- kuka_hand.stl
- kuka_greifer.stl
- kuka_greiferbacke.stl
- table.stl
- base.stl
- test_tube.stl
- wuerfel_bin.stl

Anlagenverzeichnis CAD-Dateien:

- kawasaki_basis.prt
- kawasaki_arm1.prt
- kawasaki_zwischenarm.prt
- kawasaki_arm2.prt
- kawasaki_dreharm.prt
- kawasaki_hand.prt
- kawasaki_greifer.prt
- kawasaki_greiferbacke.prt
- kawasaki_greiferbacke_zu.prt
- robot_ring.prt
- bootganz.blend
- wuerfel_ascii.prt
- kuka_basis.prt
- kuka_arm1.prt
- kuka_zwischenarm.prt
- kuka_arm2.prt
- kuka_dreharm.prt
- kuka_hand.prt
- kuka_greifer.prt
- kuka_greiferbacke.prt
- table.prt
- base.prt
- test_tube.prt
- wuerfel_bin.blend

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, 07.12.2012 Christoph Rasch