Projektgruppe 523

RoSi 3D

Endbericht

Kooperierende Industrieroboter
zum Besprühen und Trocknen von Freiformflächen

26. November 2008

Veranstalter:
Lehrstuhl VII, Graphische Systeme, TU Dortmund
Institut für Roboterforschung, TU Dortmund

Betreuer:
Frank Weichert
Carsten Scheele
Adrian Schyja

Teilnehmer:
Diether Dieckmann
Marian Gröning
Benjamin Johnen
Christopher Keller
Rudolf A. Kerbitz
Christopher Kontis
Thulackshan Mohan
Stephan Ohlmeier
Qing Sun
Matthias Teich
Mohammed Thaifa
Christian Wessel

Inhaltsverzeichnis

Abbildungsverzeichnis

Liste der Algorithmen

Abkürzungsverzeichnis

ANSI	American National Standards Institute
BRep	Boundary Representation
CAD	Computer Aided Design
CAE	Computer Aided Engineering
CSG	Constructive Solid geometry
IGES	Initial Graphics Exchange Specification
ISO	International Organization for Standardization
MST	Minimal Spanning-Tree
MVC	Model-View-Controller
NURBS	Non-Uniform Rational B-Splines
OBJ	Object File Format
OCC	Open CASCADE
OMF	Open CASCADE Mesh Framework
PLY	Polygon File Format
SAT	Seperating Axis Theorem
STEP	STandard for the Exchange of Product model data
STL	StereoLithography oder auch Surface Tesselation Language
SVN	Subversion
TCP	Tool Center Point
TSP	Traveling Salesperson Problem
WKS	Weltkoordinatensystem
XML	Extensible Markup Language

Mathematische Notation

Notation	Bedeutung
$\mathbb{R}$	Die Menge der reellen Zahlen
$\mathbb{N}$	Die Menge der natürlichen Zahlen
$\mathbb{N}_0$	Die Menge der natürlichen Zahlen inklusive der null
$[0, 1]$	Ein geschlossenes Intervall von 0 bis 1
$[0, 1[$	Ein halboffenes Intervall von 0 bis 1
$]0, 1[$	Ein offenes Intervall von 0 bis 1
$\{0, ..., n\}$	Eine Menge von 0 bis n
$\|x\|$	Betrag von x
$\|\mathbf{x}\|$	Euklidische Norm
$\mathbf{a}..\mathbf{z}$	Vektoren
$\binom{n}{i}$	Binomialkoeffizient
$f(x)$	Eine Funktion
$\sum_{i=0}^{n} a_i$	Eine Summe $a_0+, ..., +a_n$
$\mathbf{a} \cdot \mathbf{b}$	Punktprodukt von zwei Vektoren
$\mathbf{a} \times \mathbf{b}$	Kreuzprodukt von zwei Vektoren
$V \longrightarrow \mathbb{R}^d$	Eine Abbildung von einem Definitionsraum V nach einem Bildraum $\mathbb{R}^d$
$x \mapsto x^2$	Eine Zuordnung von x nach x^2
$(\mathbf{a}_{n,m})$	Eine $n \times m$ Matrix
$\mathbf{T}$	Transformation

1. Einleitung

Unter dem Begriff kooperierende Roboter wird im industriellen Umfeld das Zusammenarbeiten von mindestens zwei Robotern an einer Bearbeitungsaufgabe verstanden. Die parallele Bearbeitung bietet die Möglichkeit mehr Arbeitsprozesse zu bewältigen und somit für die gleiche Leistung weniger Zeit zu benötigen. Des Weiteren ist es möglich neuartige Herstellungsverfahren zu unterstützen, wie z.B. die Beschichtung von Oberflächen. Unmittelbar nach dem Auftragen kann verdichtet werden und dadurch werden verbesserte Eigenschaften wie z.B. höhere Härte und damit bessere Verschleißeigenschaften erlangt.

Das Ziel der Projektgruppe bestand darin, ein Softwaresystem zu entwickeln (*RoSi3D*), welches automatisch Bewegungsbahnen auf beliebigen Objekten ermittelt, so dass zwei sechs-achsige Industrieroboter kooperierend ein Werkstück bearbeiten können, ohne dabei miteinander oder mit dem Werkstück zu kollidieren. Angesichts dieser Aufgabe sollen als erstes die verwendeten Robotermodelle genauer beschrieben werden. Daher werden im Kapitel 4 grundlegende Begriffe und Definitionen im Zusammenhang mit der Kinematik des Roboters, seiner Geschwindigkeit und Zielgenauigkeit erläutert. Dabei ist eine mathematische Beschreibung seiner Bewegung notwendig, um festzulegen wie er eine Position optimal anfahren kann, ohne dabei mit anderen Gegenständen in Kollision zu treten.

Folglich ist es die Aufgabe der Bahnplanung eine Bahn zu ermitteln, die der Roboter zum Sprühen bzw. Trocknen abfahren kann. Um dieser Aufgabe gerecht zu werden, wird zuerst das Werkstück in mehrere Segmente unterteilt, auf denen Teilbahnen berechnet werden. Schließlich werden die einzelnen Teilbahnen zu einer Gesamtbahn verbunden, die das gesamte Werkstück umfasst. Die einzelnen Schritte der Bahnberechnung, sowie die angewandten Verfahren werden in Kapitel 5 näher erläutert.

Anhand der so berechneten Bahn wird von der Bewegungsplanung (siehe Kapitel 6) die genaue Bewegung des Roboters geplant. Daher werden Details wie Geschwindigkeit, Orientierung und Art der Bewegung berücksichtigt. Im nächsten Schritt wird die berechnete Bewegung auf Kollisionen getestet. Die dafür verwendeten Ansätze, um mögliche Kollisionen frühzeitig zu erkennen und zu vermeiden werden im Kapitel 7 vorgestellt. Das Ergebnis der Bewegungsplannung dient am Ende als Eingabe für das Programm *RobotStudio*, welches dazu verwendet wird, das von *RoSi3D* berechnete Roboter-Programm zu simulieren.

Bis ein Roboter-Programm letztendlich berechnet wird, erfolgt ein weiterer Schritt, in dem die berechnete Bewegung der Roboter getestet wird. Ein solcher Test findet in einer virtuellen Welt statt – die sogenannte Szene – die der realen Welt entspricht. Dabei werden die Akteure einer Szene und ihre Interaktionen visualisiert, um Kollisionen und Ungenauigkeiten in der Bewegung zu korrigieren. Im Falle der Projektgruppe 523 sind das die Roboter, das zu besprühende Werkstück und der Arbeitstisch. In Kapitel 7.2 wird diese Szene beschrieben und darauf eingegangen, wie die Akteure in der Szenerie dargestellt werden.

Bei erfolgreicher Simulation der Bewegung entlang der berechneten Bahn in *RobotStudio*, erfolgt letztendlich die Übertragung des Roboter-Programms auf die realen Roboter mit Hilfe von *RobotStudio Online*.

Für den realen Versuch konnte die Projektgruppe die, vom Institut für Roboterforschung (IRF), bereitgestellten Roboter ABB IRB 2400 und ABB IRB 4400 verwenden. Als zu bearbeitendes Werkstück wurde ein UFO-ähnliches Objekt verwendet, welches durch eine Gußform mit Gips modelliert wurde (siehe Abbildung 1.1).

Abbildung 1.1.: Versuchsumgebung

2. Grundlagen

Das Kapitel Grundlagen beschreibt eine Auswahl an Methoden und Datenstrukuren, welche für die Lösung einzelner Teilprobleme in der Projektgruppe benötigt werden. Bei der Geometrieverarbeitung werden Oberflächenmodelle in Form von Regel- und Freiformflächen verwendet, welche durch Flächen-Dreiecksnetze approximiert werden. Für ein grundlegendes Verständnis von Freiformflächen, werden diese in Abschnitt 2.1 vorgestellt. Im darauf folgenden Abschnitt 2.2 werden Dreiecksnetze, Datenstrukturen für Dreiecksnetze und ein Verfahren zur Netzverbesserung beschrieben. Die Realisierung einer Kollisionserkennung zwischen Geometrien und die Berechnung von Roboterbewegungsabläufen bedienen sich verschiedener elementarer geometrischer Schnittberechnungen. Diese werden im Abschnitt 2.3 hergeleitet. Die im Rahmen der Projektgruppe entwickelte Software erzeugt Roboter-Programme in der Programmiersprache RAPID (siehe Kapitel 3.8), bei der Rotationen im dreidimensionalen Raum durch Quaternionen beschrieben werden. Quaternionen und das Berechnen von Rotationsabbildungen mit selbigen werden in Abschnitt 2.4 skizziert.

2.1. Freiformflächen

Freiformflächen sind anschaulich Oberflächen, welche keine Regelmäßigkeiten aufweisen müssen. Im Kontext der Projektgruppe dienen sie zur Beschreibung von Oberflächengeometrien wie die eines zu bearbeitenden Werkstücks, der dabei verwendeten Werkzeuge oder einer Halterung. Später werden diese Freiformgeometrien zur Weiterverarbeitung durch Dreiecksnetze approximiert (siehe Kapitel 2.2).

Im Folgenden werden die Bézier und die B-Spline-Technik in jeweils zwei Ausprägungen vorgestellt. Die aus der B-Spline-Technik hervorgegangenen NURBS stellen nicht nur einen De-facto-Standard dar, weil sie durch viele CAD-, CAE- und 3D-Modellierungsanwendungen verwendet werden. Sie finden sich auch als Komponente des ISO X3D Standards [ISO08] sowie im ISO Standard zur Kodierung audiovisueller Objekte [ISO06] und werden von standardisierten Datenaustauschformaten IGES [IGE80] und STEP [ISO94] unterstützt.

2.1.1. Prinzip

Die hier vorgestellten Methoden zur Beschreibung von Freiformflächen gründen auf Methoden zur Beschreibung von beliebig geformten Raumkurven. Die Beschreibung von Raumkurven in parametrisierter Darstellung besteht aus zwei Komponenten:

1. Kontrollpunkte, über die der Verlauf einer Kurve spezifiziert wird.

2. Jeweils eine Mischfunktion zu jedem Kontrollpunkt, welche zu einem gegebenen Parameterwert den Anteil des Kontrollpunktes liefert, der zum resultierenden Kurvenpunkt beigetragen wird.

Im d-dimensionalen Raum sind die Kontrollpunkte eine endliche Folge

$$(\mathbf{p}_i)_{0\leq i\leq n}\,,\ \mathbf{p}_i \in \mathbb{R}^d,\ n \in \mathbb{N}_0. \tag{2.1}$$

und liefern zusammen mit Mischfunktionen $M_i : \mathbb{R} \supseteq U \to \mathbb{R}$ die Raumkurve als Funktion

$$\mathbf{k} : \mathbb{R} \supseteq U \longrightarrow \mathbb{R}^d, \tag{2.2}$$

mit

$$\mathbf{k}\,(u) = \sum_{i=0}^{n} M_i\,(u) \cdot \mathbf{p}_i. \tag{2.3}$$

Ein Freiformflächensegment kann aufgefasst werden als eine Kurve, die durch den Raum bewegt wird und dabei ihre Form ändert. In Bezug auf obige Kurvendefinition wird ein weiterer Parameter $v \in V \subseteq \mathbb{R}$ eingeführt, der intuitiv formuliert, den „Zeitpunkt“ einer Momentaufnahme der Kurve beschreibt. Für obige Kurve $\mathbf{k}$ bedeutet eine Bewegung durch den Raum und eine dabei zusätzlich auftretende Änderung ihrer Form, dass ihre Kontrollpunkte abhängig von v stetig verschoben werden. Die Kontrollpunkte $\mathbf{p}_i$ von $\mathbf{k}$ beschreiben also $n+1$ Raumkurven $\mathbf{l}_i$, welche genau wie $\mathbf{k}$ aus Kontrollpunkten und Mischfunktionen zusammengesetzt werden können. Unter der Annahme, dass alle $\mathbf{l}_i$ dieselbe Anzahl von $m+1$, $m \in \mathbb{N}$ Kontrollpunkten $\mathbf{q}_{i,0}$ bis $\mathbf{q}_{i,m}$ haben, gleich parametrisiert sind und dieselben Mischfunktionen $N_j : \mathbb{R} \supseteq V \to \mathbb{R}$, $0 \leq j \leq m$ verwenden, sind $\mathbf{l}_i$ Funktionen

$$\mathbf{l}_i : \mathbb{R} \supseteq V \longrightarrow \mathbb{R}^d \tag{2.4}$$

mit

$$\mathbf{l}_i\,(v) = \sum_{j=0}^{m} N_i\,(v) \cdot \mathbf{q}_{i,j}.$$

Das Flächensegment wird also konstruiert aus einer Raumkurve $\mathbf{k}$ mit „zeitabhängigen“ Kontrollpunkten $\mathbf{l}_i$, was folgende Funktion ergibt:

$$\mathbf{f} : \mathbb{R}^2 \supseteq U \times V \longrightarrow \mathbb{R}^d \tag{2.5}$$

mit

$$\begin{aligned}\mathbf{f}(u,v) &= \sum_{i=0}^{n} M_i(u) \cdot \mathbf{l}_i(v) \\ &= \sum_{i=0}^{n} M_i(u) \cdot \left(\sum_{j=0}^{m} N_j(v) \cdot \mathbf{q}_{i,j} \right) \\ &= \sum_{i=0}^{n} \sum_{j=0}^{m} M_i(u) \cdot N_i(v) \cdot \mathbf{q}_{i,j}.\end{aligned}$$

Die im nächsten Abschnitt besprochenen Flächensegmente haben alle diesen Aufbau und unterscheiden sich nur in den verwendeten Mischfunktionen.

Es hat sich herausgestellt, dass Polynome eine geeignete Funktionsklasse für Mischfunktionen darstellen. Polynome über einem Körper $(K, +, \cdot)$ bilden zusammen mit der Addition $+$ und der Multiplikation $\cdot$ einen Vektorraum. Interessant für Mischfunktionen sind verschiedene Basisdarstellungen von endlichen Polynomuntervektorräumen, da hiermit alle Polynome bis zu einem bestimmten Grad erzeugt werden können. Unterraumbasen $\{P_0^n, P_1^n, \ldots, P_n^n\}$ eines $(n+1)$ −dimensionalen Unterraumes können als Mischfunktionen für Raumkurven mit $n+1$ Kontrollpunkten verwendet werden.

Die wohl bekannteste Polynom-Basis sind Monome, also

$$\left\{ u \mapsto u^0, u \mapsto u^1, \ldots, u \mapsto u^n \right\}. \tag{2.6}$$

Die entsprechende Kurvenfunktion – ein Polynomkurvensegment – ist definiert als

$$\mathbf{k}_{Polynom} : [0,1] \longrightarrow \mathbb{R} \tag{2.7}$$

mit

$$\mathbf{k}_{Polynom}(u) = \sum_{i=0}^{n} u^i \cdot \mathbf{p}_i \tag{2.8}$$

Da mittels der Kontrollpunkte das Aussehen der Kurve gesteuert wird und dies gegebenenfalls interaktiv, z.B. als Teil eines CAD-Programmes, geschehen soll, hat die Verwendung von Monomen den Nachteil, dass der Zusammenhang zwischen Kontrollpunkten und Kurve wenig intuitiv ist. Mit einer Ausnahme lässt sich der Einfluss der Kontrollpunkte nicht gut einschätzen. Der Kontrollpunkt, welcher auf der Kurve liegt, ist der Koeffizient des konstanten Monoms. Die Kurve beginnt also in diesem Punkt und lässt sich (ohne dabei die Form zu ändern) mit diesem Kontrollpunkt verschieben (siehe Abbildung 2.1). Über den gestrichelt dargestellten Polygonzug werden der Reihe nach aufsteigend die Koeffizienten der Monome des nächst höheren Grades erreicht.

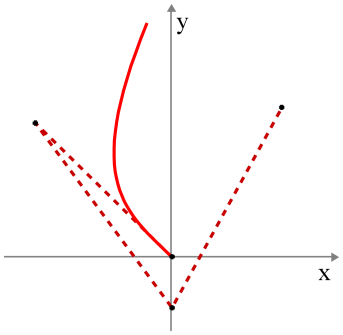


Abbildung 2.1.: Polynomkurvensegment mit Kontrollpunkten (verbunden durch gestrichtelte Linie)

2.1.2. Bézierflächen und rationale Bézierflächen

Bernsteinpolynome

Bernsteinpolynome sind genau wie Monome Basen für Polynom-Vektorräume. Für $n \in \mathbb{N}_0$, $0 \leq i \leq n$ sind Bernsteinpolynome vom Grad n definiert als:

$$B_i^n : \mathbb{R} \longrightarrow \mathbb{R}, \tag{2.9}$$

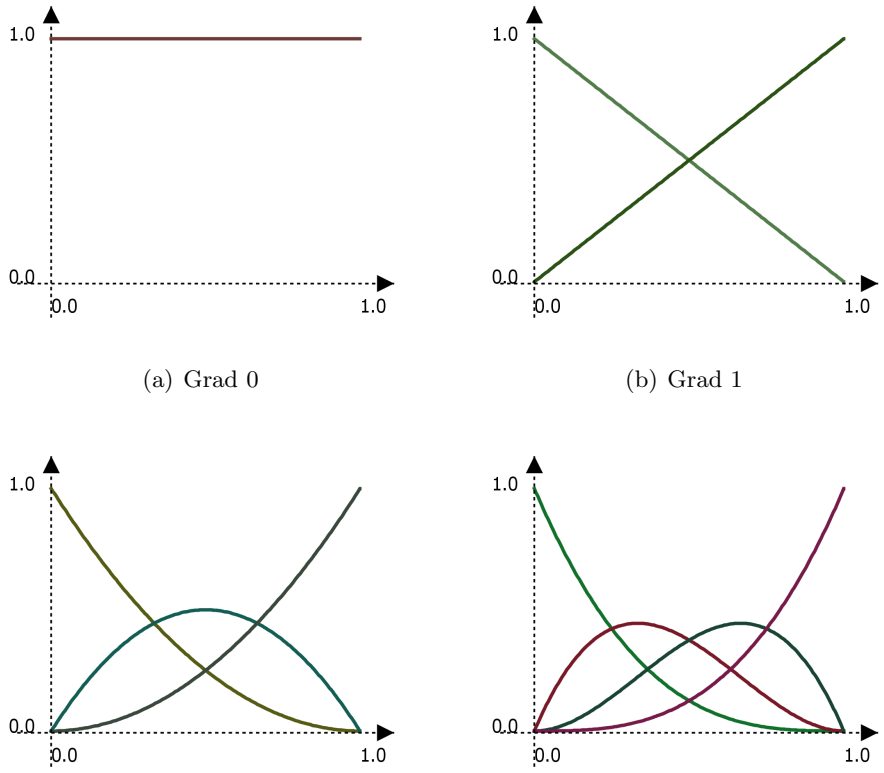
mit

$$B_i^n(x) = \binom{n}{i} \cdot x^i \cdot (1-x)^{n-i}. \tag{2.10}$$

Einige Eigenschaften der Bernsteinpolynome sind:

- Partition der 1: $\sum_{i=0}^{n} B_i^n(x) = 1$ für $x \in [0,1]$.
- Positivität: Für $x \in [0,1]$ ist $B_i^n(x) \geq 0$.
- Maxima: Für $x = \frac{i}{n}$ ist $B_i^n(x)$ lokales Maximum in $[0,1]$.
- Symmetrie: $B_i^n(x) = B_{n-i}^n(1-x)$.
- Basis: $\{B_i^n \mid 0 \leq i \leq n\}$ bilden ein minimales Erzeugendensystem für Polynome vom Grad 0 bis n.

Siehe hierzu [AM91]. Abbildung 2.2 zeigt die Graphen der Bernsteinpolynome vom Grad null, eins, zwei und drei.



(a) Grad 0

(b) Grad 1

(c) Grad 2

(d) Grad 3

Abbildung 2.2.: Bernsteinpolynome mit Grad 0 bis Grad 3

Bézierkurven- und Flächensegment

Eine Bézierkurve zur Kontrollpunktfolge $(\mathbf{b}_i)_{0\leq i\leq n}$, $n \in \mathbb{N}$ ist eine Funktion

$$\mathbf{k}_{\text{Bézier}} : [0,1] \longrightarrow \mathbb{R}^d \tag{2.11}$$

mit

$$\mathbf{k}_{\text{Bézier}}(u) = \sum_{i=0}^{n} B_i^n(u) \cdot \mathbf{b}_i. \tag{2.12}$$

Die Kontrollpunkte werden in diesem Fall auch Bézierpunkte genannt. Bézierkurven haben durch die Bernsteinpolynome einige für die Modellierung günstige Eigenschaften:

1. Der erste und letzte Kontrollpunkt wird jeweils durch die Kurve interpoliert. Für die Bernsteinpolynome gilt:

$$B_i^n(0) = \binom{n}{i} \cdot 0^i \cdot 1^{n-i} = \begin{cases} 1 & \text{, für } i = 0 \\ 0 & \text{, für } i > 0 \end{cases}$$

$$B_i^n(1) = \binom{n}{i} \cdot 1^i \cdot 0^{n-i} = \begin{cases} 1 & \text{, für } i = n \\ 0 & \text{, für } i < n \end{cases}$$

   Also fallen für Parameterwert 0 bzw. 1 alle Summanden der Kurvenformel weg außer $B_0^n(0) \cdot \mathbf{b}_0 = \mathbf{b}_0$ bzw. $B_n^n(1) \cdot \mathbf{b}_n = \mathbf{b}_n$ und es gilt $\mathbf{k}_{\text{Bézier}}(0) = \mathbf{b}_0$ und $\mathbf{k}_{\text{Bézier}}(1) = \mathbf{b}_n$.

2. Das Kurvensegment liegt in der konvexen Hülle seiner Kontrollpunkte. Die konvexe Hülle der Kontrollpunkte $\mathbf{b}_0, \ldots, \mathbf{b}_n$ ist die Menge

$$conv\{\mathbf{b}_0, \ldots, \mathbf{b}_n\} = \left\{ \sum_{i=0}^{m} \delta_i \cdot \mathbf{b}_i \mid \sum_{i=0}^{m} \delta_i = 1,\ \delta_j \geq 0,\ m \in \{0, \ldots, n\},\ j = 0 \ldots n \right\}.$$

   Die Punktmenge der zu den Kontrollpunkten gehörenden Kurve $\mathbf{k}$ enthält die Punkte

$$\{\mathbf{k}(u) \mid u \in [0,1]\} = \left\{ \sum_{i=0}^{n} B_i^n(u) \cdot \mathbf{b}_i \mid u \in [0,1] \right\}$$

   und aufgrund der Teilungseigenschaft und der Positivität der Bernsteinpolynome

gilt:

$$\begin{aligned}
\left\{\sum_{i=0}^{n} B_i^n(u)\cdot\mathbf{b}_i \,|\, u\in[0,1]\right\} &= \left\{\sum_{i=0}^{n} B_i^n(u)\cdot\mathbf{b}_i \,|\, \sum_{i=0}^{n} B_i^n(u)=1,\ , B_j^n(u)\geq 0,\, u\in[0,1],\, j\in J\right\}\\
&\subset \left\{\sum_{i=0}^{n} \delta_i\cdot\mathbf{b}_i \,|\, \sum_{i=0}^{n}\delta_i=1,\, \delta_j>0,\, j\in J\right\}\\
&= \left\{\sum_{i=0}^{m} \delta_i\cdot\mathbf{b}_i \,|\, \sum_{i=0}^{m}\delta_i=1,\, \delta_j>0,\, m\in\{n\},\, j\in J\right\}\\
&\subset \left\{\sum_{i=0}^{m} \delta_i\cdot\mathbf{b}_i \,|\, \sum_{i=0}^{m}\delta_i=1,\, \delta_j>0,\, m\in J,\, j\in J\right\}\\
&= conv\{\mathbf{b}_0,\ldots,\mathbf{b}_n\},
\end{aligned}$$

wobei

$$J := \{0,\ldots,n\}$$

3. Ein Kontrollpunkt $\mathbf{b}_i$ übt seinen größten Einfluss auf eine Kurve mit $n+1$ Kontrollpunkten bei Parameterwert $u=\frac{i}{n}$ aus. Dies geht offensichtlich aus der Maxima-Eigenschaft der Bernsteinpolynome hervor.

Wie in Abschnitt 2.1.1 besprochen, lassen sich aus der Definition der Bézierkurven entsprechende Flächensegmente direkt ableiten. Seien $n,m\in\mathbb{N}$. Ein Bézierflächensegment zu einer $n\times m$ - Kontrollpunktmatrix $(\mathbf{b}_{i,j})$, $\mathbf{b}_{i,j}\in\mathbb{R}^d$ ist eine Funktion

$$\mathbf{f}_{\text{Bézier}} : [0,1]\times[0,1]\longrightarrow\mathbb{R}^d \tag{2.13}$$

mit

$$\mathbf{f}_{\text{Bézier}}(u,v) = \sum_{i=0}^{n}\sum_{j=0}^{m} B_i^n(u)\cdot B_j^m(v)\cdot\mathbf{b}_{i,j}. \tag{2.14}$$

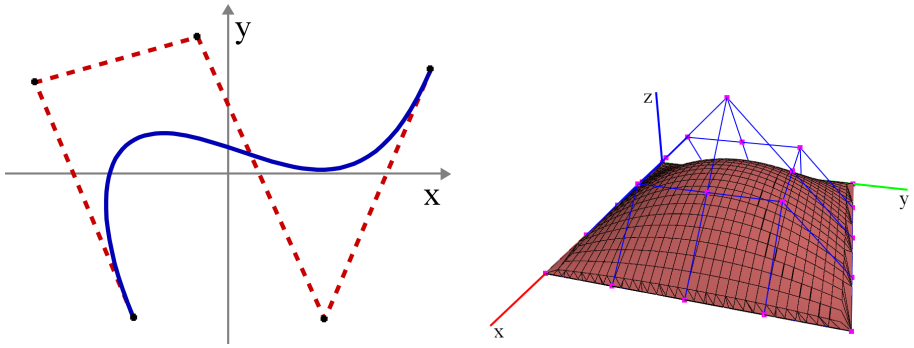
Aufgrund ihrer Konstruktion haben auch Bézierflächensegmente einige für die Modellierung günstige Eigenschaften:

1. Die Eckpunkte des Flächensegments stimmen mit den Eckpunkten des Kontrollpunktgitters überein.

2. Die Punkte des Flächensegments liegen in der konvexen Hülle seines Kontrollpunktgitters.

3. Ein Kontrollpunkt $\mathbf{b}_{i,j}$ übt seinen größten Einfluss auf eine Kurve mit $(n+1)\times(m+1)$ Kontrollpunkten bei Parameterwert-Tupel $(u,v)=\left(\frac{i}{n},\frac{j}{m}\right)$ aus.

4. Die Randkurven der Fläche haben als Kontrollpunkte die Randpunkte des Kontrollpunktnetzes.



(a) Bézierkurve mit Kontrollpolygonzug (b) Bézierfläche mit Kontrollpunktgitter

Abbildung 2.3.: Modellierung mit Bézierkurven/Bézierflächen

Für einen Beweis der Eigenschaften siehe [AM91]. Es ist leicht einzusehen, dass Bézierkurven und -flächen intuitiv zu modellieren sind. Durch die Endpunkteigenschaften der Kurven lassen sich zusammengesetzte Kurven konstruieren. Mittels Differentiation der Kurvenfunktionen lassen sich Übergangsbedingungen zwischen den einzelnen Segmenten formulieren, um z.B. krümmungs-stetige Übergänge zu erhalten. Das Zusammenfügen von Flächensegmenten ist mit etwas mehr Aufwand verbunden. Für adjazente Flächensegmente muss zu diesem Zweck zumindest sichergestellt werden, dass benachbarte Randkurven identisch sind. In diesem Fall sind die Übergänge dann stetig, aber im Allgemeinen nicht stetig differenzierbar. Allerdings lassen sich auch hier mit Mitteln der Analysis Bedingungen formulieren, um „glatte“ Übergänge zu erhalten.

Algorithmus von de Casteljau zur Berechnung von Kurven und Flächen

Der nach seinem Erfinder Paul de Faget de Casteljau benannte de Casteljau Algorithmus ist ein numerisch stabiler Algorithmus zum Lösen von Polynomen in Bernstein-Basis-Darstellung. Zu einer Folge von $n \in \mathbb{N}$ Koeffizienten $(a_i)_{0 \leq i \leq n}$ und einem Parameterwert $x \in \mathbb{R}$ berechnet der Algorithmus die Funktion

$$f(x) = \sum_{i=0}^{n} B_i^n(x) \cdot a_i \tag{2.15}$$

mittels Konvexkombinationen.

Aufgrund der besseren Lesbarkeit ist der de Casteljau Algorithmus so formuliert, dass die gegebenen Koeffizienten überschrieben werden. Der Algorithmus hat, bei k wesentlichen Operationen in der Rechenvorschrift, eine Laufzeit von

$$k \cdot (n + (n-1) + (n-2) + \ldots + 3 + 2) = k \cdot \left(\frac{n \cdot (n+1)}{2} - 1 \right) = \mathcal{O}\left(n^2\right). \tag{2.16}$$

```
Eingabe  : Parameter x, Koeffizienten c[], Anzahl Koeffizienten n
Ausgabe : Zum Parameter x gehörender Punkt c[0]
for i ← 0 to n do
    for j ← 0 to n − i do
        c[j] ← (1-x)*c[j] + x*c[j+1];
    endfor
endfor
return c[0]
```

Algorithmus 1 : De Casteljau Algorithmus zur Berechnung von Kurvenpunkten

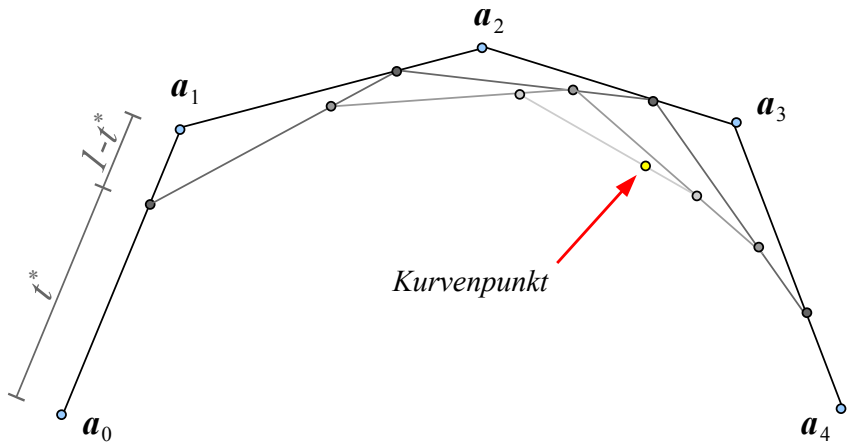


Abbildung 2.4.: Prinzip des Algorithmus von de Casteljau

An dieser Stelle sei erwähnt, dass ein auf dem Horner-Schema basierender Algorithmus mit Laufzeit $\mathcal{O}(n)$ existiert, der allerdings nicht numerisch stabil ist (siehe [AM91]).

In Abbildung 2.4 ist das Prinzip des Algorithmus für fünf Bézierpunkte als Koeffizienten dargestellt. Im ersten Iterationsschritt wird die Konvexkombination jeweils benachbarter Punkte des Bézierpolygonzugs gebildet. Geometrisch interpretiert entspricht dies der Unterteilung der Verbindungsstrecke zwischen den Punkten zu den Teilen x und $1 - x$. Die bei der Unterteilung entstehenden Punkte bilden einen neuen Polygonzug, dessen Segmente analog zum Polygonzug der Bézierpunkte unterteilt werden. Der Algorithmus terminiert, wenn nur noch ein einzelner Punkt übrig ist. Dies ist der Kurvenpunkt zum gegebenen Parameterwert x.

Für eine approximierende Repräsentation einer Bézierkurve müssen nur mehrere Kurvenpunkte berechnet und über einen Polygonzug miteinander verbunden werden.

Analog zur Konstruktion einer Formel für Flächensegmente aus einer Formel für Kurvensegmente kann der de Casteljau Algorithmus für Kurven zur Berechnung von Flä-

chenpunkten verwendet werden:

```
Eingabe : Parameter x und y, Koeffizienten c[][], Anzahl Koeffizienten n und m
Ausgabe : Zu Parametern x, y gehörender Punkt res
temp [n];
for i ← 0 to n do
 |  temp[i] ← deCasteljau(y, c[i], m);
endfor
res ← deCasteljau(x, temp, n);
return res
```

Algorithmus 2 : Algorithmus von de Casteljau zur Berechnung von Flächenpunkten

Zuerst werden über die Zeilen (oder die Spalten) der Koeffizientenmatrix die Kontrollpunkte der Flächenkurve zum Parameterwert von y (x) berechnet. Schließlich werden die parameterabhängigen Kontrollpunkte als Eingabe für den Kurvenalgorithmus verwendet, um den resultierenden Flächenpunkt zum Parameterwert von x (y) zu erhalten. Der Algorithmus hat offensichtlich eine Laufzeit von $n \cdot \mathcal{O}\left(m^2\right) + \mathcal{O}\left(n^2\right) = \mathcal{O}\left(n \cdot m^2 + n^2\right)$. Unter Verwendung des auf dem Horner-Schema basierenden Algorithmus ist die Laufzeit $\mathcal{O}\left(n \cdot m + n\right)$ (siehe [AM91]). Bei unterschiedlich großen n und m sollten die Zeilen und Spalten der Koeffizientenmatrix in getauschten Rollen durchlaufen werden, sodass der $n \cdot m^2$-Term der Laufzeit möglichst gering ist.

Durch Berechnung mehrerer Flächenpunkte kann eine Approximation des Flächensegments mittels Polygonnetz erzeugt werden.

Rationales Bézierkurven- und Flächensegment

Ein rationales Bézierkurvensegment ist eine gebrochen rationale Funktion

$$\mathbf{k}_{r.\text{Bézier}} : [0,1] \longrightarrow \mathbb{R}^d, \tag{2.17}$$

mit

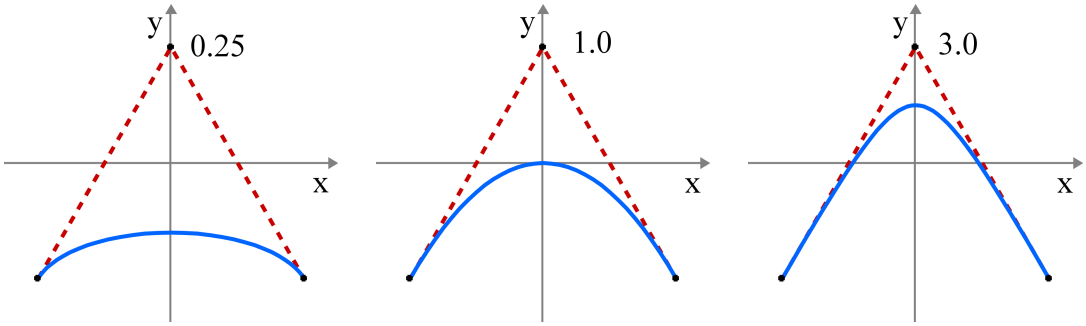
$$\mathbf{k}_{r.\text{Bézier}}(u) = \frac{\sum_{i=0}^{n} B_i^n(u) \cdot \beta_i \cdot \mathbf{b}_i}{\sum_{i=0}^{n} B_i^n(u) \cdot \beta_i}. \tag{2.18}$$

Analog dazu ist ein rationales Bézierflächensegment definiert als

$$\mathbf{f}_{r.\text{Bézier}} : [0,1] \times [0,1] \longrightarrow \mathbb{R}^d, \tag{2.19}$$

mit

$$\mathbf{f}_{r.\text{Bézier}}(u,v) = \frac{\sum_{i=0}^{n}\sum_{j=0}^{m} B_i^n(u) \cdot B_j^m(v) \cdot \beta_{i,j} \cdot \mathbf{b}_{i,j}}{\sum_{i=0}^{n}\sum_{j=0}^{m} B_i^n(u) \cdot B_j^m(v) \cdot \beta_{i,j}}. \tag{2.20}$$



(a) Gewicht 0,25 (b) Gewicht 1,0 (c) Gewicht 3,0

Abbildung 2.5.: Kegelschnitte modelliert mit rationalen Bézierkurven

Hierbei können $\beta_i \in \mathbb{R}_+$ und $\beta_{i,j} \in \mathbb{R}_+$ als Gewichte interpretiert werden, welche die Attraktivität der jeweiligen Kontrollpunkte $\mathbf{b}_i$ bzw. $\mathbf{b}_{i,j}$ steuern. Sind alle Gewichte gleich, so ist ein rationales Segment identisch zum nicht rationalen Segment mit denselben Kontrollpunkten. Erst durch Ungleichheiten der Gewichte untereinander ändert sich die Form des Segments gegenüber der nicht rationalen Form. Abbildung 2.5 zeigt ein rationales Bézierkurvensegment vom Grad 2 mit dem variierten Knotengewicht am mittleren Kontrollpunkt. Von links nach rechts dargestellt sind ein elliptisches, parabolisches und hyperbolisches Kurvensegment. Rationale Béziersegmente stellen allgemein keine Polynome dar und werden verwendet um Kurven und Flächen zu modellieren, die mit reinen Polynomen nicht exakt beschrieben werden können.

Nachteile der Bézier-Technik

Die Bézier-Technik hat zwei wesentliche Nachteile. Einzelne Kurven- oder Flächensegmente dürfen nicht zu komplex werden, da proportional zur Anzahl der Kontrollpunkte der Grad des Segments steigt und der Polynomgrad die Laufzeit für die Berechnung von Kurven- bzw. Flächenpunkten stark beeinflusst. Bezüglich der Modellierung hat die Bézier-Technik die Eigenschaft, dass jeder Kontrollpunkt eines Segments praktisch Einfluss auf das gesamte Segment hat.

2.1.3. B-Spline-Flächen und rationale B-Spline-Flächen

B-Spline Funktionen

(Polynom-) Splines sind Funktionen, welche stückweise aus Polynomen zusammengesetzt sind. Der Definitionsbereich einer (Polynom-) Spline- Funktion vom Grad $d \in \mathbb{N}_0$ ist unterteilt in Teilintervalle mit den Anforderungen, dass die Funktion über jedem

Teilintervall ein Polynom und in den Übergängen zwischen den Teilintervallen $d-1$ mal stetig differenzierbar ist. Die Teilintervalle werden durch eine monoton steigende Zahlenfolge $(x_0, x_1, \ldots, x_k) \in \mathbb{R}^{k+1}$, $k \in \mathbb{N}_0$, genannt Knotenvektor, beschrieben. Analog zum Polynomvektorraum gibt es den Vektorraum der stückweisen Polynome. Eine Basis für endliche Polynom-Spline Untervektorräume sind die Basis-Spline- oder kurz B-Spline Funktionen.

Zu einem Knotenvektor $(x_0, x_1, \ldots, x_k) \in \mathbb{R}^k$, $x_0 \leq x_1 \leq \ldots \leq x_k$ sind die $k-n$ B-Spline Funktionen $N_0^n, \ldots, N_{k-n-1}^n$ vom Grad $n \in \mathbb{N}$, $1 \leq n < k$ rekursiv definiert als

$$N_i^n : [x_0, x_k[\longrightarrow \mathbb{R}, \tag{2.21}$$

mit

$$N_i^0(x) = \begin{cases} 1 & , x \in [x_i, x_{i+1}[\\ 0 & , \text{sonst} \end{cases} \tag{2.22}$$

$$N_i^n(x) = \frac{x - x_i}{x_{i+n} - x_i} \cdot N_i^{n-1}(x) + \frac{x_{i+n+1} - x}{x_{i+n+1} - x_i} \cdot N_{i+1}^{n-1}(x), \tag{2.23}$$

wobei $\frac{0}{0}$ defininiert wird als 0.

Einige Eigenschaften der B-Spline Funktionen sind:

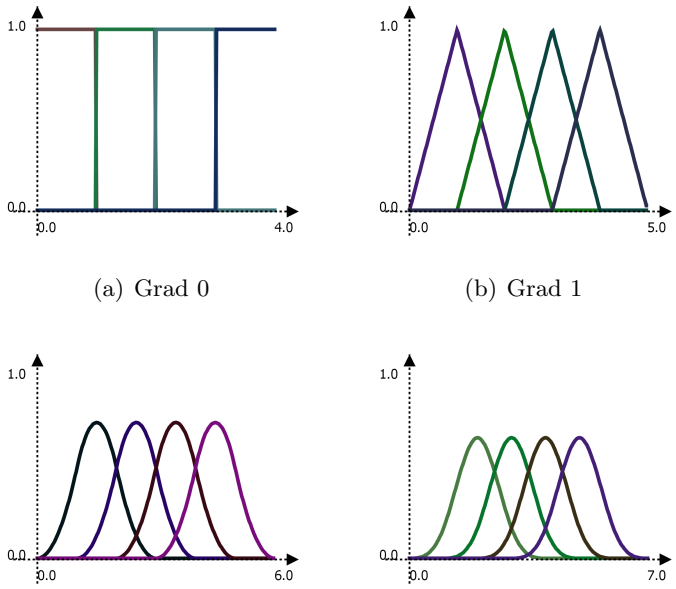
- Partition der 1: $\sum_{i=0}^{k-n-1} N_i^n(x) = 1$ für $x \in [x_n, x_{k-n}[$.
- Positivität: Für $x \in [x_0, x_k[$ ist $N_i^n(x) \geq 0$.
- Trägerintervalle: $[x_i, x_{i+n+1}[$ ist Trägerintervall von N_i^n. Außerhalb des Trägerintervalls, also für $x \in [x_0, x_k[\setminus [x_i, x_{i+n+1}[$, ist $N_i^n(x) = 0$.

Siehe hierzu [AM91]. Ein wichtiges Unterscheidungskriterium von B-Spline Funktionen N_i^n sind die Art der verwendeten Knotenvektoren. Ein Knotenvektor

$$(x_0, x_1, \ldots, x_k) \in \mathbb{R}^k$$

heißt

- *uniform*, falls $x_{i+1} = x_i + \Delta x$ für $0 \leq i < k$ und festem $\Delta x \in \mathbb{R}$,
- *offen uniform*, falls $x_0 = x_1 = \ldots = x_n$ und $x_{k-n} = x_{k-n+1} = \ldots = x_k$ und $(x_{n+1}, \ldots, x_{k-n-1})$ uniform ist,
- *nicht uniform*, falls er keiner Einschränkung außer der Monotonie $x_i \leq x_{i+1}$ unterworfen ist.



(a) Grad 0 (b) Grad 1

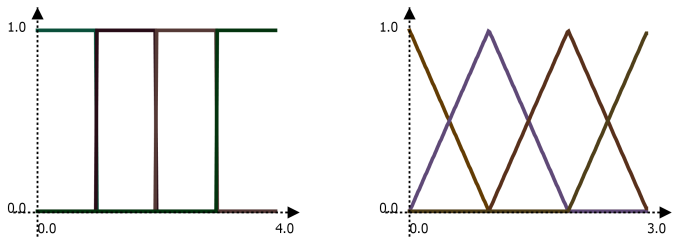
(c) Grad 2 (d) Grad 3

Abbildung 2.6.: Uniforme B-Spline Funktionen

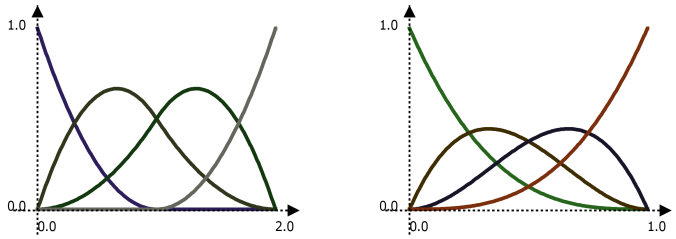
Der Einfachheit halber werden die Begriffe uniform, offen uniform und nicht uniform auch auf Funktionen angewendet. Eine uniforme B-Spline Funktion bezeichnet somit eine B-Spline Funktion mit uniformem Knotenvektor. Spezielle B-Spline Funktionen haben bestimmte Eigenschaften, von denen hier eine Auswahl gelistet sind:

- Für uniforme B-Splines gilt: $N_i^n(x) = N_{i+1}^n(x + \Delta x)$.
- Für offen uniforme B-Splines gilt die Partition der 1 auf dem gesamten Intervall $[x_0, x_k[$.
- Für einen Knotenvektor mit $x_0 = x_1 = \ldots = x_{\lfloor k/2 \rfloor} = 0$ und $x_{\lceil k/2 \rceil} = x_{\lceil k/2 \rceil + 1} = \ldots = x_k = 1$ ist $N_i^n(x) = B_i^n(x)$ für $x \in [0, 1]$, $0 \leq i \leq n$.

Abbildungen 2.6 und 2.7 zeigen die Graphen der uniformen bzw. offen uniformen B-Spline Funktionen vom Grad null, eins, zwei und drei ($\Delta x = 1$) und Abbildung 2.8 zeigt ein Beispiel für nicht uniforme B-Spline Funktionen vom Grad zwei zum Knotenvektor (0, 1, 3, 3, 4, 4.5, 6).



(a) Grad 0 (b) Grad 1



(c) Grad 2 (d) Grad 3

Abbildung 2.7.: Offen uniforme B-Spline Funktionen

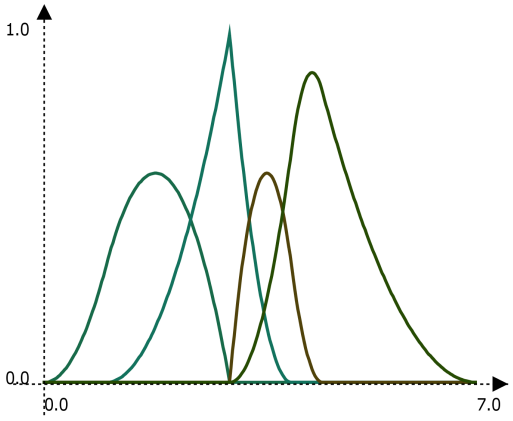


Abbildung 2.8.: Nicht uniforme B-Spline Funktionen vom Grad zwei

B-Spline Kurven- und Flächensegmente

B-Spline Kurven- und Flächensegmente basieren auf B-Splines zu Knotenvektoren von spezieller Gestalt, wodurch sie Eigenschaften ähnlich denen der Bézierkurven und -Flächen erhalten. B-Spline Kurven können im Gegensatz zu Bézierkurven in zwei Gestalten auftreten – als offenes und als geschlossenes Kurvensegment. Letztere werden hier nicht weiter besprochen.

Ein offenes B-Spline Kurvensegment zur Kontrollpunktfolge $(\mathbf{d}_i)_{0\leq i\leq m}$, $m \in \mathbb{N}$ (de Boor Punkte) ist eine Funktion

$$\mathbf{k}_{BS} : [u_0, u_{m+n+1}[\longrightarrow \mathbb{R}^d, \tag{2.24}$$

mit

$$\mathbf{k}_{BS}(u) = \sum_{i=0}^{m} N_i^n(u) \cdot \mathbf{b}_i. \tag{2.25}$$

wobei für den Knotenvektor $(u_0, u_1, \ldots, u_{m+n+1})$ der B-Splines N_i^n vom Grad $n \in \mathbb{N}$, $0 \leq n \leq m$ gilt: $u_0 = u_1 = \ldots = u_n$ und $u_{m+1} = u_{m+2} = \ldots = u_{m+n+1}$. Diese Anforderung an den Knotenvektor ist allgemeiner als offen uniform, stellt aber sicher, dass die Partition der eins auf dem gesamten Intervall $[u_0, u_{m+n+1}[$ erfüllt ist.

B-Spline Kurven weisen einige für die Modellierung günstige Eigenschaften auf – ähnlich denen der Bézierkurven:

1. Der erste und letzte Kontrollpunkt einer offenen Kurve wird jeweils durch die Kurve interpoliert.
   Da die ersten $n+1$ Knoten des Knotenvektors gleich sind, gilt:

$$\begin{aligned}
N_0^n(u_0) &= \frac{u_0 - u_0}{u_n - u_0} \cdot N_0^{n-1}(u_0) + \frac{u_{n+1} - u_0}{u_{n+1} - u_0} \cdot N_1^{n-1}(u_0) \\
&= N_1^{n-1}(u_0) \\
&= \frac{u_0 - u_1}{u_n - u_1} \cdot N_0^{n-2}(u_0) + \frac{u_{n+1} - u_0}{u_{n+1} - u_1} \cdot N_2^{n-2}(u_0) \\
&= N_2^{n-2}(u_0) \\
&\vdots \\
&= N_n^0(u_0) \\
&= 1, \text{ da } u_0 = u_n \in [u_n, u_{n+1}[
\end{aligned}$$

   Analog wird für $N_m^n(u_{m+n+1})$ über die Gleichheit der letzten $n+1$ Knoten des

Knotenvektors argumentiert:

$$\begin{aligned} N_m^n(u_{m+n+1}) &= \frac{u_{m+n+1}-u_m}{u_{m+n}-u_m}\cdot N_m^{n-1}(u_0) + \frac{u_{m+n+1}-u_{m+n+1}}{u_{m+n+1}-u_m}\cdot N_{m+1}^{n-1}(u_0) \\ &= N_m^{n-1}(u_{m+n+1}) \\ &= \frac{u_{m+n+1}-u_m}{u_{m+n-1}-u_m}\cdot N_0^{n-2}(u_0) + \frac{u_{m+n}-u_{m+n+1}}{u_{m+n}-u_m}\cdot N_2^{n-2}(u_0) \\ &= N_m^{n-2}(u_{m+n+1}) \\ &\vdots \\ &= N_m^0(u_{m+n+1}) \\ &= 1, \text{ da } u_{m+n+1} = u_m \in [u_m, u_{m+1}[\end{aligned}$$

An dieser Stelle ist noch nicht klar, dass alle anderen $N_j^n(x)$, $1 \leq j \leq n+1$ bzw. $m-n-1 \leq j \leq m-1$, deren Trägerintervalle $[u_j, u_{j+n+1}[$ das Trägerintervall $[u_0, u_{n+1}[$ von $N_0^n(x)$ bzw. $[u_m, u_{m+n+1}[$ von $N_m^n(x)$ schneiden, für $x = u_0$ bzw. $x = u_{m+n+1}$ null sind. Dies folgt direkt aus der Eigenschaft „Partition der eins“ und „Positivität“.
Also fallen für Parameterwert u_0 bzw. u_{m+n+1} alle Summanden der Kurvenformel weg außer $N_0^n(u_0)\cdot\mathbf{d}_0 = \mathbf{d}_0$ bzw. $N_m^n(u_{n+m+1})\cdot\mathbf{d}_n = \mathbf{d}_n$ und es gilt $\mathbf{k}_{BS}(u_0) = \mathbf{d}_0$ und $\mathbf{k}_{BS}(u_{m+n+1}) = \mathbf{d}_m$.

2. Das Kurvensegment liegt in der konvexen Hülle seiner Kontrollpunkte.
   Diese Eigenschaft kann analog zur konvexen Hülle Eigenschaft von BézierKurven über die Teilung der eins und der Positivität über $[u_0, u_{m+n+1}[$ gezeigt werden.

3. Ein Kontrollpunkt $\mathbf{d}_i$ hat nur lokalen Einfluss auf die Kurve.
   Zum Kontrollpunkt $\mathbf{d}_i$ gehört der B-Spline $N_i^n(u)$, welcher nur auf seinem Trägerintervall, also für $u \in [u_i, u_{i+n+1}[$, ungleich null ist. Falls $n < m$ ist das Trägerintervall eine echte Teilmenge von $[u_0, u_{m+n+1}[$ und der Einfluß des korrespondierenden Kontrollpunktes entsprechend lokal.

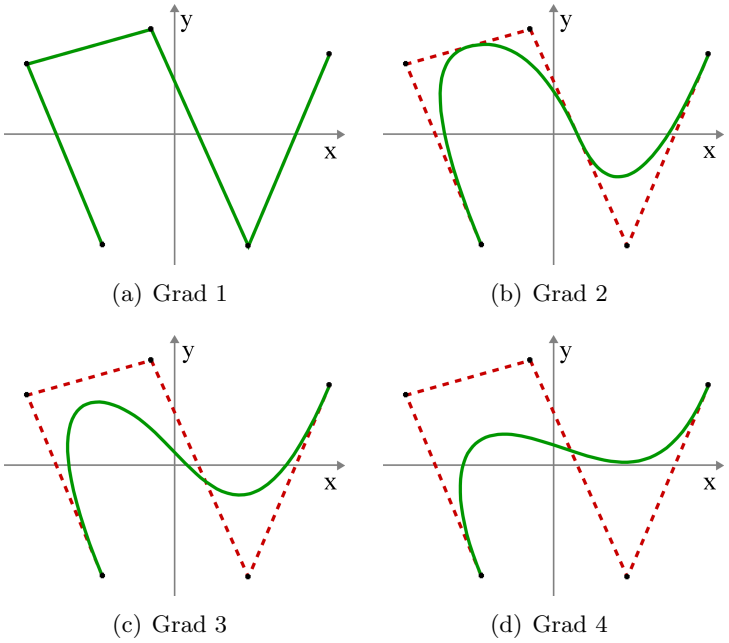
Abbildung 2.9 zeigt die möglichen B-Spline Kurvensegmente vom Grad eins bis vier zu einem festen Kontrollpolygonzug (gestrichelt) mit fünf de Boor Punkten.

Nach dem schon bekannten Muster werden Flächensegmente direkt aus der Definition von B-Spline Kurven abgeleitet. Ein (offenes) B-Spline Flächensegment zu einer $k \times l$ - Kontrollpunktmatrix $(\mathbf{b}_{i,j})$, $\mathbf{b}_{i,j} \in \mathbb{R}^d$ ist eine Funktion

$$\mathbf{f}_{BS} : [u_0, u_{k+n+1}[\times [v_0, v_{l+m+1}[\longrightarrow \mathbb{R}^d, \tag{2.26}$$

mit

$$\mathbf{f}_{BS}(u,v) = \sum_{i=0}^{k}\sum_{j=0}^{l} N_i^n(u)\cdot N_j^m(v)\cdot\mathbf{d}_{i,j}, \tag{2.27}$$



(a) Grad 1 (b) Grad 2

(c) Grad 3 (d) Grad 4

Abbildung 2.9.: B-Spline Kurvensegmente vom Grad eins bis vier

wobei die Knotenvektoren $(u_0, u_1, \ldots, u_{k+n+1}) \in \mathbb{R}^{k+n+1}$ und $(v_0, v_1, \ldots, v_{l+m+1}) \in \mathbb{R}^{l+m+1}$ der B-Splines N_i^n und N_j^m der Grade $n, m \in \mathbb{N}$ dieselben Anforderungen erfüllen wie die Knotenvektoren offener B-Spline Kurven.
Ein (offenes) B-Spline Flächensegment hat folgende Eigenschaften:

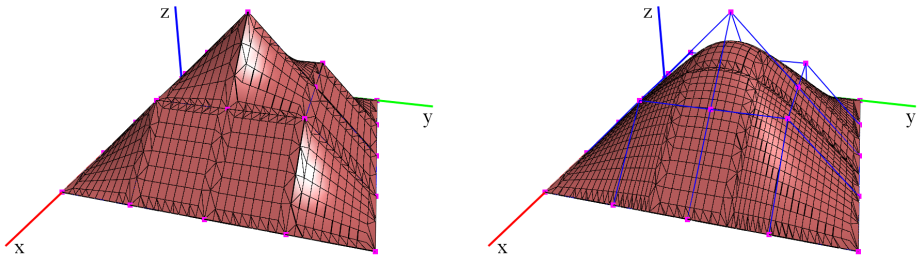
1. Die Eckpunkte des Flächensegments stimmen mit den Eckpunkten des Kontrollpunktgitters überein.

2. Die Punkte des Flächensegments liegen in der konvexen Hülle seines Kontrollpunktgitters.

3. Ein Kontrollpunkt $\mathbf{d}_{i,j}$ übt nur lokal, d.h. für den Parameterbereich $[u_i, u_{i+n+1}[\times [v_i, v_{i+m+1}[$, Einfluß auf die Gestalt des Flächensegments aus.

4. Die Randkurven der Fläche haben als Kontrollpunkte die Randpunkte des Kontrollpunktnetzes.

Siehe hierzu [AM91].



(a) Grad 1 (b) Grad 2

(c) Grad 3 (d) Grad 4

Abbildung 2.10.: B-Spline Flächensegmente, wobei beide B-Splines den selben Grad haben

Algorithmus von de Boor zur Berechnung von Kurven und Flächen

Carl R. de Boor formulierte den nach ihm benannten Algorithmus zur numerisch stabilen Berechnung von stückweisen Polynomen in B-Spline-Basisdarstellung. Zu einer Folge von $m \in \mathbb{N}$ Koeffizienten $(a_i)_{0 \leq i \leq n}$, einem Parameterwert $x \in \mathbb{R}$ und einem Knotenvektor $(x_0, x_1, \ldots, x_{m+n+1})$ berechnet der Algorithmus die aus stückweisen Polynomen vom Grad $n \in \mathbb{N}$ bestehende Funktion

$$f(x) = \sum_{i=0}^{m} N_i^n(x) \cdot a_i \tag{2.28}$$

mittels Konvexkombinationen.

Aufgrund der besseren Lesbarkeit ist der de Boor Algrithmus so formuliert, dass die gegebenen Koeffizienten überschrieben werden. Der Algorithmus hat, bei k wesentlichen Operationen in der Rechenvorschrift, eine Laufzeit von

$$\mathcal{O}(m) + ((n-1) + (n-2) + \ldots + 1) \cdot k \quad = \quad \mathcal{O}(m) + \tfrac{n \cdot (n-1)}{2} \cdot k \quad = \quad \mathcal{O}\left(m + n^2\right). \tag{2.29}$$

Mit anderen Worten wächst die Laufzeit des de Boor Algorithmus linear in der Anzahl der Kontrollpunkte und quadratisch im Grad der verwendeten B-Splines. Da in der

Eingabe : Grad n, Anzahl Koeffizienten m, Parameter u, Koeffizienten c[], Knotenvektor knots[]
Ausgabe : Zu u gehörender Punkt auf der Kurve
$i \leftarrow 0$;
while $i < m$, *knots[i]* $\leq u$ *und* $u \geq$ *knots[i+1]* **do**
  $i \leftarrow i + 1$;
endw
for $j \leftarrow 1$ **to** n **do**
  for $k \leftarrow i$ **downto** $i - n + j$ **do**
    $v \leftarrow \frac{u - knots[k]}{knots[k+n+1-j] - knots[k]}$;
    c[k] $\leftarrow$ (1-v)*c[k-1] + v*c[k];
  endfor
endfor
return *c[i]*

Algorithmus 3 : Algorithmus von de Boor zur Berechnung von Kurvenpunkten

Praxis üblicherweise das Verhältnis von Splinegrad zu Kontrollpunktezahl sehr gering ist, stellt dies einen enormen Vorteil dar.

In Abbildung 2.11 ist das Prinzip des Algorithmus für fünf de Boor Punkte als Koeffizienten zu einem offen uniformen Knotenvektor und B-Splines vom Grad drei dargestellt. Im ersten Schritt des Verfahrens wird bestimmt, welche Kontrollpunkte den Kurvenpunkt beeinflussen. Dies geschieht durch Suchen des Index $i = 3$ des größten Knotens, der kleiner ist als der gegebene Parameterwert t^*. Die Indizes der involvierten de Boor Punkte sind dann $i - n$, $i - n + 1$, ..., i, im Beispiel null, eins, zwei und drei. Sind die Punkte bekannt, funktioniert der Algorithmus vom Prinzip her ähnlich wie der Algorithmus von de Casteljau. Es werden in jedem Iterationsschritt die Konvexkombinationen jeweils benachbarter Punkte des involvierten Teilpolygonzugs gebildet und der Polygonzug durch einen neuen Polygonzug mit genau einem Knoten weniger ersetzt; mit dem Unterschied, dass die Koeffizienten für die Berechnung der Konvexkombination nicht fix sind, sondern in Abhängigkeit von den Knotenvektoren für jede Unterteilung bestimmt werden. Das Vorgehen wird durch die Intervallstreifen angedeutet: In jedem Iterationsschritt werden die Intervallstreifen durch den Parameterwert t^* in zwei Hälften geteilt, die dem Unterteilungsverhältnis der Polygonzugkanten entsprechen.
Der Algorithmus terminiert, wenn nur noch ein einzelner Punkt übrig ist. Dies ist der Kurvenpunkt zum gegebenen Parameterwert u. Wie schon beim Algorithmus von de Casteljau folgt der de Boor Algorithmus für Flächen durch mehrmalige Anwendung des Algorithmus für Kurven unter Ausnutzung des nur lokalen Einflusses der Kontrollpunke:

Dabei bezeichnet k (l) den Index des letzten Elementes einer Zeile (Spalte), *uknots* (*vknots*) den Knotenvektor der Kurve der Koeffizienten einer Zeile (Spalte) der Koeffizientenmatrix und m, n die Grade. Die Funktion *deBoor* bezeichnet hier den zuvor

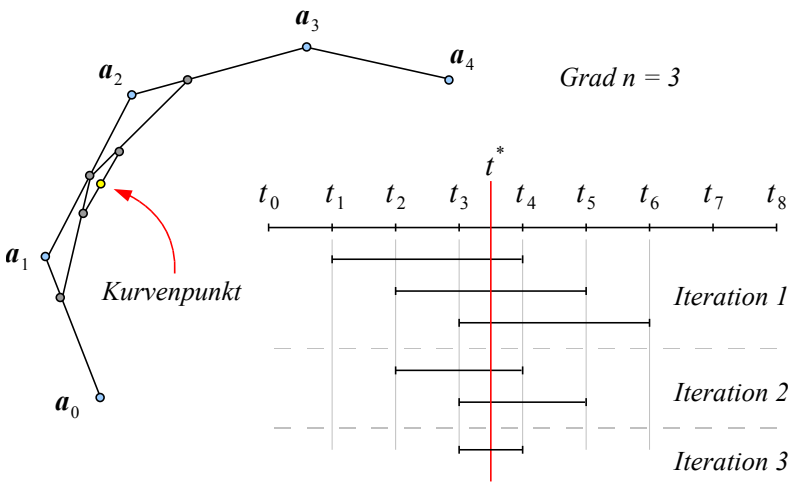


Abbildung 2.11.: Prinzip des de Boor Algorithmus

beschriebenen de Boor-Algorithmus, ohne die anfängliche Suche nach dem Knotenvektorindex. Dieser wird als letzter Parameter übergeben. Damit ist die Laufzeit

$$\mathcal{O}(l) + \mathcal{O}(k) + (n+1) \cdot \mathcal{O}\left(m^2\right) + \mathcal{O}\left(n^2\right) = \mathcal{O}\left(n \cdot m^2 + n^2 + k + l\right). \tag{2.30}$$

Rationales B-Spline Kurven- und Flächensegment

Ein rationales Bézierkurvensegment vom Grad $n \in \mathbb{N}$ zu $m \in \mathbb{N}$ Kontrollpunkten $\mathbf{d}_0, \ldots, \mathbf{d}_m$, zum monoton steigenden Knotenvektor $(u_0, u_1, \ldots, u_{m+n+1}) \in \mathbb{R}$ und mit den Gewichten $\beta_i \in \mathbb{R}_+$ ist eine gebrochen rationale Funktion

$$\mathbf{k}_{r.BS} : [u_0, u_{m+n+1}[\longrightarrow \mathbb{R}^d, \tag{2.31}$$

mit

$$\mathbf{k}_{r.BS}(u) = \frac{\sum_{i=0}^{m} N_i^n(u) \cdot \beta_i \cdot \mathbf{d}_i}{\sum_{i=0}^{m} N_i^n(u) \cdot \beta_i}. \tag{2.32}$$

Analog dazu ist ein rationales Bézierflächensegment zur $(k+1)\times(l+1)$ großen Kontrollpunktmatrix $(\mathbf{d}_{i,j})$, zu den monoton steigenden Knotenvektoren $(u_0, u_1, \ldots, u_{k+n+1})$, $(v_0, v_1, \ldots, v_{l+m+1})$, den Graden $m, n \in \mathbb{N}$ und Gewichten $\beta_{i,j} \in \mathbb{R}_+$ definiert als

$$\mathbf{f}_{r.BS} : [u_0, u_{m+n+1}[\times [v_0, v_{m+n+1}[\longrightarrow \mathbb{R}^d, \tag{2.33}$$

mit

$$\mathbf{f}_{r.BS}(u, v) = \frac{\sum_{i=0}^{k} \sum_{j=0}^{l} N_i^n(u) \cdot N_j^m(v) \cdot \beta_{i,j} \cdot \mathbf{b}_{i,j}}{\sum_{i=0}^{k} \sum_{j=0}^{l} N_i^n(u) \cdot N_j^m(v) \cdot \beta_{i,j}}. \tag{2.34}$$

Eingabe : Grade k und l, Anzahl Koeffizienten n und m, Knotenvektoren uknots[] und vknots[], Parameter u, Parameter v, Koeffizienten c[][]
Ausgabe : Zu u und v gehörender Punkt auf der Fläche
$i \leftarrow 0$;
while $i < k$, *uknots[i]* $\leq u$ *und* $u \geq$ *uknots[i+1]* **do**
  $i \leftarrow i + 1$;
endw
$j \leftarrow 0$;
while $j < l$, *vknots[j]* $\leq v$ *und* $v \geq$ *vknots[j+1]* **do**
  $i \leftarrow j + 1$;
endw
tmp [n];
for $q \leftarrow i - n$ **to** i **do**
  $temp[q] \leftarrow deBoor(v, l, m, vknots, c[k], j)$;
endfor
$res \leftarrow deBoor(u, k, n, uknots, temp, i)$;
return *res*

Algorithmus 4 : Algorithmus von de Boor zur Berechnung von Flächenpunkten

Da an die Knotenvektoren keine weiteren Forderungen gestellt werden als Monotonie, heißen solche Kurven und Flächensegmente nicht uniforme rationale B-Splines (Non Uniform Rational Basis Splines, kurz NURBS).

B-Spline Technik

Wie zuvor besprochen, bietet die B-Spline-Technik ähnlich komfortable Modellierungseigenschaften wie die Bézier-Technik. Allerdings ist die B-Spline-Technik der Bézier-Technik aus Performanzgründen und der besseren Modellierbarkeit überlegen. Neben dem lokalen Einfluss der Kontrollpunkte kommt positiv hinzu, dass über den Knotenvektor ein weiteres Mittel zur Modellierung bereit steht. Des Weiteren, was hier nur am Rande angesprochen wurde, können mit der B-Spline-Technik geschlossene Kurven und Flächen modelliert werden, was mit der Bézier-Technik nur mit zusammengesetzten Kurven- oder Flächensegmenten möglich ist.

2.2. Dreiecksnetze

Die in *RoSi3D* verwendeten Algorithmen zur Bahnplanung (siehe Kapitel 5), zur Kollisionserkennung (siehe Kapitel 7) und zur Bewegungsplanung (siehe Kapitel 6) arbeiten auf Dreiecksnetzen. Ein Dreiecksnetz besteht aus mehreren miteinander verbundenen Dreiecken. Die einzelnen Dreiecke eines Dreiecksnetzes werden als Facetten bezeichnet

und die Eckpunkte einer Facette als Vertices.

Ein regelmäßiges Dreiecksnetz hat folgende Eigenschaften:

- Alle Vertices haben Grad sechs, Randvertices haben Grad vier.
- Die Innenwinkel der Dreiecke sind 60°.

Ein mögliches Maß, mit dem die Regelmäßigkeit gemessen werden kann, ist das Verhältnis von Umkreis zu Inkreis der Dreiecke.

Für die Umwandlung von Freiformgeometrie in Dreiecksnetze wird der frei erhältliche CAD-Kernel OpenCASCADE (OCC, (siehe Kapitel A.5)) verwendet. Dieser ist im Gegensatz zum zusätzlich erhhältlichen und kostenpflichtigen Open CASCADE Mesh Framework (OMF) nicht darauf ausgelegt, regelmäßige Dreiecksnetze zu erzeugen. Die von OpenCASCADE erzeugten Triangulierungen approximieren die zugrunde liegende Geometrie aber ausreichend gut und sind damit für die Visualisierung geeignet.

Um dennoch ein regelmäßiges Netz zu erhalten, werden die Dreiecksnetze in einem weiteren Schritt umvernetzt (siehe Kapitel 2.2.2). Der Algorithmus zur Umvernetzung und zwei zur Speicherung von Dreiecksnetzen geeignete Datenstrukturen werden im folgenden kurz vorgestellt.

2.2.1. Datenstrukturen für Dreiecksnetze

Bei der Auswahl einer Datenstruktur für die Dreiecksnetze in *RoSi3D* sind die Geschwindigkeit der Abfrage der Nachbarschaftsbeziehungen unter den Elementen des Netzes und der benötigte Speicherplatz mögliche Kriterien. Der verwendete Algorithmus zur Netzoptimierung (siehe Kapitel 2.2.2) und die Bahnplanung (siehe Kapitel 5) fragen häufig Nachbarschaftsinformationen ab, daher ist die Geschwindigkeit der Abfrage von Nachbarschaftsinformationen das primäre Auswahlmerkmal.

Naive Datenstrukturen für Polygonnetze wie zum Beispiel die Vertexliste scheiden daher aus.

Zwei bekannte komplexere Datenstrukturen zum Speichern von polygonalen Netzen sind die Winged-Edge Datenstruktur (siehe [Bau72]) und die Half-Edge Datenstruktur (siehe [CKS98]). In beiden Datenstrukturen sind die Konnektivitätsinformationen vor allem in den Kanten gespeichert, es handelt sich also um kantenbasierte Datenstrukturen.

Kante	Start-vertex	Ziel-vertex	Fläche links	Fläche rechts	linker Vorgänger	rechter Vorgänger	linker Nachfolger	rechter Nachfolger
a	X	Y	1	2	d	e	b	c

Tabelle 2.1.: Auszug aus der Kantenliste in der Winged-Edge Datenstruktur (zu Abbildung 2.12)

Winged-Edge Datenstruktur

Bei der Winged-Edge Datenstruktur werden folgende Informationen gespeichert:

- Jeder Vertex speichert eine Referenz auf eine beliebige anliegende Kante.
- Jede Facette speichert eine Referenz auf eine beliebige anliegende Kante.
- Jede Kante speichert eine Referenz auf:
  - Ihren Startvertex.
  - Ihren Endvertex.
  - Die Fläche links von der Kante.
  - Die Fläche rechts von der Kante.
  - Die linke Vorgängerkante.
  - Die rechte Vorgängerkante.
  - Die linke Nachfolgerkante.
  - Die rechte Nachfolgerkante.

Zur Veranschaulichung sind in Abbildung 2.12 die Nachbarschaftsinformationen für eine Kante dargestellt. Tabelle 2.1 zeigt den zugehörigen Eintrag in der Kantenliste.

Mit der Winged-Edge Datenstruktur ist es möglich in Zeit $\mathcal{O}(1)$ abzufragen, welche Ecken oder Facetten zu einer Kante inzident sind. Von einer Facette ausgehende Anfragen sind etwas langsamer. Hier muss entlang der zur Facette inzidenten Kanten iteriert werden, die Anfragen sind also in Zeit $\mathcal{O}(k_f)$ möglich , wobei k_f die Anzahl der zur Facette inzidenten Kanten ist. Die restlichen Anfragen sind deutlich langsamer, da hier die gesamte Kantenliste durchlaufen werden muss. Sie brauchen also Zeit $\mathcal{O}(k)$ für die Bearbeitung, wobei k die Anzahl der Kanten im Netz ist.

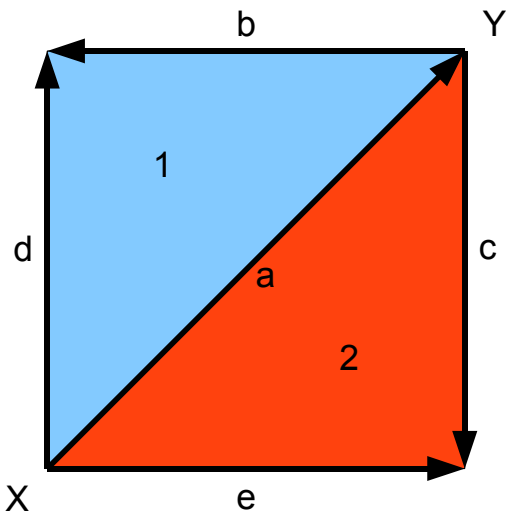


Abbildung 2.12.: Kante in der Winged-Edge Datenstruktur

Half-Edge Datenstruktur

Mit Hilfe der Half-Edge Datenstruktur (siehe [CKS98]) ist es möglich die benötigte Zeit für diese Anfragen von $\mathcal{O}(k)$ auf $\mathcal{O}(k_v)$ zu reduzieren. k_v ist hierbei die Anzahl der Nachbarvertices von Vertex v.

Die Konnektivität des Netzes wird dabei vor allem in den Kanten gespeichert:

- Jeder Vertex speichert eine Referenz auf eine beliebige ausgehende Halbkante.
- Jede Facette speichert eine Referenz auf eine beliebige anliegende Halbkante.
- Jede Halbkante speichert eine Referenz auf:
  - Ihren Zielvertex.
  - Die zugehörige Facette.
  - Die nächste Halbante (geordnet im Gegenuhrzeigersinn).
  - Die gegenüberliegende Halbkante.
  - Die vorherige Halbkante (optional).

Abbildung 2.13 zeigt die Kante aus Abbildung 2.12 in der Half-Edge Datenstruktur. Der entsprechende Eintrag in der Kantenliste ist in Tabelle 2.2 zu sehen.

Kante	Ziel-vertex	Fläche	Nachfolger	Vorgänger	Gegen-stück
a_1	X	2	c_1	e_1	a_2
a_2	Y	1	d_2	b_2	a_1

Tabelle 2.2.: Auszug aus der Kantenliste in der Half-Edge Datenstruktur (zu Abbildung 2.13)

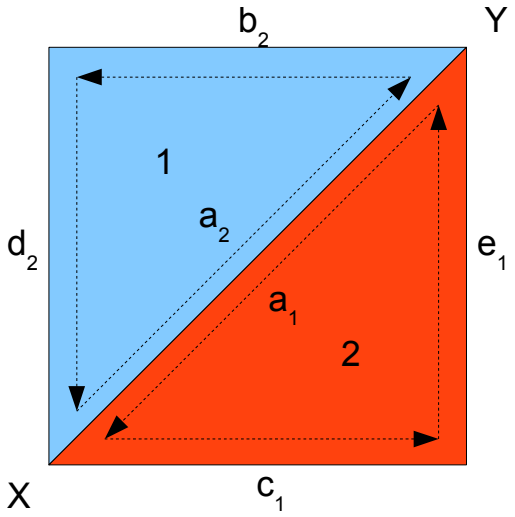


Abbildung 2.13.: Kante in der Half-Edge Datenstruktur

Mit diesen Konnektivitätsinformationen ist es möglich, alle Knoten, Kanten oder Nachbarfacetten zu einer gegebenen Facette in Zeit $\mathcal{O}(A_f)$ zu finden, wobei A_f die Anzahl der an die Facette angrenzenden Halbkanten ist.

Von einem Vertex ausgehend können alle benachbarten Kanten, Facetten und Vertices in Zeit $\mathcal{O}(A_v)$ gefunden werden. A_v bezeichnet hierbei die Anzahl der zu einem Vertex gehörenden ausgehenden Halbkanten.

Die Nachbarschaftsinformationen der Halbkanten sind in Zeit $\mathcal{O}(1)$ abfragbar, da diese direkt in der Halbkante gespeichert werden.

Da die Abfragegeschwindigkeit bei der Half-Edge Datenstruktur höher ist als bei der Winged-Edge Datenstruktur, wird im Rahmen der Projektgruppe *OpenMesh* verwendet, eine von der *Computer Graphics Group* der RWTH Aachen entwickelte Implementierung

der Half-Edge Datenstruktur.

2.2.2. Umvernetzung

Ziel der Umvernetzung ist ein gegebenes Netz in ein regelmäßiges Netz umzuformen, wobei die durch das Netz beschriebene Gestalt gar nicht oder nur geringförmig geändert wird. Es existieren bereits eine Vielzahl von Algorithmen die dieses Problem lösen. Im Kontext der Projektgruppe werden bestimmte Anforderungen an die verwendeten Netze gestellt: Entartete Dreiecke rufen bei der Kollisionserkennung zwischen Netzen, genauer dem Schnittest zwischen Dreiecken, numerische Fehler hervor. Diese Fehler können das Ergebnis des Schnittests verfälschen. Desweiteren verursachen sehr feine Netze lange Berechnungszeiten.

Eine pragmatische Anforderung an den Umvernetzungsalgorithmus ist der benötigte Aufwand für die Implementierung. Die Wahl ist deswegen auf den Umvernetzungsalgorithmus nach Kobbelt [KBS00] gefallen, welcher durch sukzessive Anwendung folgender drei Operationen ein Netz verbessert:

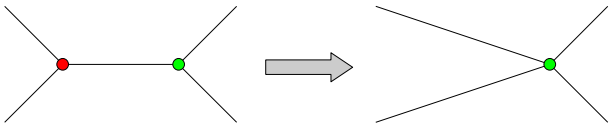
1. Kantenkontraktion (edge_collapse)
2. Kantenunterteilung (edge_split)
3. Umklappen von Kanten (edge_flip)

Die drei Operationen sind schematisch in Abbildung 2.14 dargestellt.

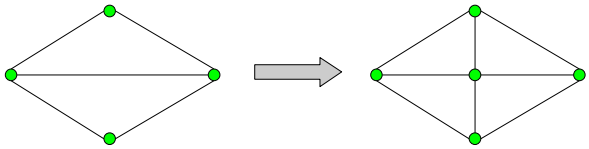
Der Algorithmus stellt sicher, dass keine Kante existiert, die länger ist als eine gegebene Schranke l_{max} und keine Kante kürzer ist als l_{min}. Desweiteren wird versucht, für alle Knoten einen Grad von sechs zu erreichen. Durch letzeres werden die Winkel, der zu einem Vertex inzidenten Dreiecke, beschränkt und betragen im Idealfall 60°.

Pro Iterationsschritt durchläuft der Algorithmus drei Phasen (vergleiche Algorithmus 5):

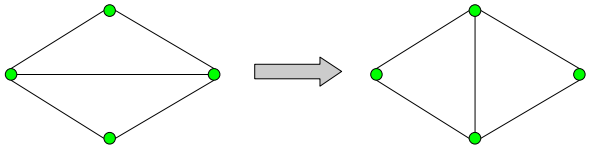
Zunächst werden alle Kanten durchlaufen und diejenigen entfernt, die kürzer sind als l_{min}. Dies wird durch Halbkantenkontraktion erreicht, bei der der Vertex mit geringerem Grad in den Vertex mit höherem Grad verschoben wird. Die Vertices werden dabei verschmolzen und die verbindende Kante, sowie die zu ihr inzidenten Dreiecke werden entfernt. Bei der Halbkantenkontration ist zu beachten, dass formgebene Kanten, d.h. Kanten, welche besondere charakteristische Merkmale der Fläche beschreiben, und Flächenränder nur bedingt entfernt werden dürfen. Andernfalls droht die Flächengestalt stark verwändert zu werden.



(a) Kantenkontraktion



(b) Kantenunterteilung



(c) Umklappen von Kanten

Abbildung 2.14.: Schematische Darstellung der Operationen für den Umvernetzungsalgorithmus

Auf das Entfernen zu kurzer Kanten folgt eine Halbierung aller Kanten, die länger sind als l_{max}. Dabei wird ein neuer Vertex erzeugt, der mittig zwischen den Endpunkten der Kante liegt und die zur Kante inzidenten Dreiecke entsprechend geteilt. Damit die beiden erzeugten Kantenhälften in der nächsten Iteration nicht wieder entfernt werden, muss $l_{max} > 2 \cdot l_{min}$ gelten. An dieser Stelle wurde der ursprüngliche Algorithmus von Kobbelt leicht abgeändert. Falls es gewünscht ist, kann die Krümmung der Fläche, die durch das Netz beschrieben wird bei der Kantenunterteilung berücksichtigt werden. Anstatt die Kante zu halbieren, wird sie durch eine kubische Polynomkurve ersetzt, für die mittig bezüglich der Parametrisierung der neue Punkt berechnet wird (siehe Abbildung 2.15). Auch hier gilt wieder besondere Rücksicht auf formgebende Kanten und Ränder zu nehmen.

Als letztes werden Kanten des Netzes umgeklappt, um möglichst einen Knotengrad von sechs zu erreichen. Zu diesem Zweck wird für benachbarte Dreiecke $\triangle(A, B, C)$ und $\triangle(B, C, D)$ geprüft, ob der totale Knotengrad

$$\sum_{v \in \{A,B,C,D\}} (d(v) - 6)^2 \tag{2.35}$$

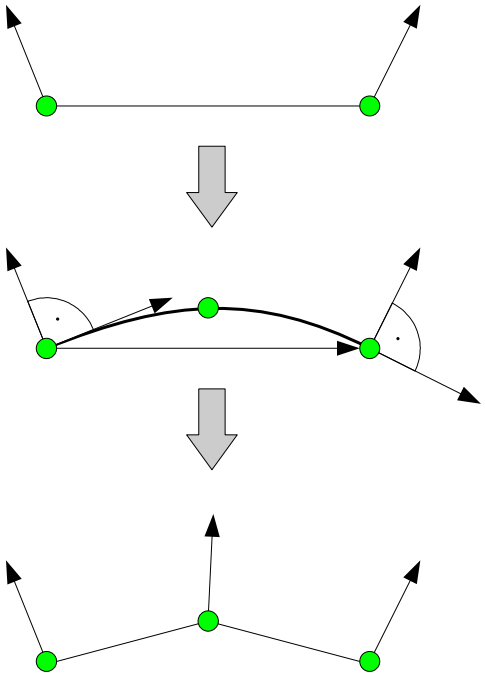


Abbildung 2.15.: Unterteilung mittels kubischer Polynomkurve

durch Vertauschen der Kante $\overline{BC}$ mit $\overline{AD}$ geringer wird. Hierbei bezeichnet $d(v)$ den Grad des Vertex v.

Auch beim Umklappen von Kanten sind einige Besonderheiten zu beachten. Alle vier Punkte müssen auf ein und derselben Ebene liegen. Desweiteren muss das ebene Viereck konvex sein. Ansonsten wird durch das Umklappen die durch das Netz beschriebene Fläche in ihrer Gestalt geändert bzw. sich überlappende Dreiecke erzeugt. Formgebende Kanten dürfen auf gar keinen Fall umgeklappt werden.

2.3. Schnittberechnungen

In diesem Kapitel werden die mathematischen Verfahren zu den verschiedenen Schnittberechnungen erläutert, die in *RoSi3D* Anwendung finden. Die Algorithmen der Bahnplanung (siehe Kapitel 5) und der Bewegungsplanung (siehe Kapitel 6) greifen auf Funktionen zurück, die auf den hier beschriebenen Berechnungen und Verfahren aufbauen.

2.3.1. Strahl-Kugel

Ein beliebiger Punkt auf der Oberfläche einer Kugel lässt sich mit Hilfe des Mittelpunkts und des Radius einer Kugel bestimmen. Sei $\mathbf{o}$ der Mittelpunkt einer Kugel und r ihr Radius. Sei $\mathbf{g}$ ein Strahl definiert durch

$$\mathbf{g}(t) = \mathbf{p} + t \cdot \mathbf{d}. \tag{2.36}$$

Dabei ist $\mathbf{g}(t)$ ein möglicher Schnittpunkt, $\mathbf{p}$ der Ortsvektor, t ein Skalar und $\mathbf{d}$ der Richtungsvektor des Strahls. Ein Schnitttest eines Strahls mit einer Kugel kann einen, zwei oder keine Lösung besitzen. Bei der Berechnung muss die folgende Bedingung erfüllt sein:

$$\|\mathbf{p} + t \cdot \mathbf{d} - \mathbf{o}\| = r. \tag{2.37}$$

Nach dieser Gleichung gibt es mindestens einen Schnittpunkt, wenn der Strahl durch den Mittelpunkt der Kugel verläuft und für das Skalar t ein Wert existiert, so dass der Strahl die Länge des Kugelradius besitzt. Um das Skalar t berechnen zu können, werden die X-,Y- und Z-Werte der Vektoren betrachtet, dabei gilt für die folgenden Gleichungen $\mathbf{a} = \mathbf{p} - \mathbf{o}$:

$$\sqrt{(\mathbf{a}_x + t \cdot \mathbf{d}_x)^2 + (\mathbf{a}_y + t \cdot \mathbf{d}_y)^2 + (\mathbf{a}_z + t \cdot \mathbf{d}_z)^2} = r. \tag{2.38}$$

Nach diversen Umformungen ergibt sich für t :

$$t_{12} = -\frac{\mathbf{a} \cdot \mathbf{d}}{\mathbf{d}^2} \pm \sqrt{\left(\frac{\mathbf{a} \cdot \mathbf{d}}{\mathbf{d}^2}\right)^2 - \frac{\mathbf{a}^2 - r^2}{\mathbf{d}^2}}. \tag{2.39}$$

Gibt es für t keine Lösung, existiert folglich auch kein Schnittpunkt zwischen dem Strahl und der Kugel; bei einer Lösung für t tangiert der Strahl die Kugel. Existieren zwei Lösungen, so verläuft der Strahl durch die Kugel und es gibt sowohl einen Eintritts- und einen Austrittspunkt.

2.3.2. Strahl-Dreieck

Die Strahl-Dreieck-Schnittberechnung, erfolgt in zwei Schritten. In einem ersten Schritt wird für das Dreieck die Ebene berechnet, in der sich das Dreieck befindet und festgestellt, ob der Strahl die Ebene schneidet. Ist dies der Fall, lässt sich für den Strahl

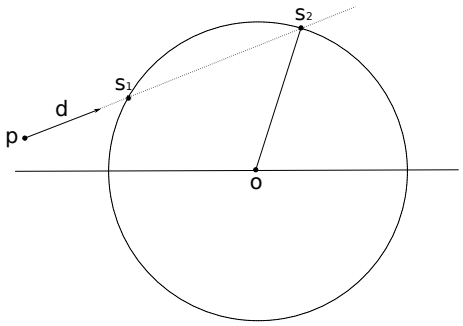


Abbildung 2.16.: Schnitt zwischen einem Strahl und einer Kugel

und die Ebene ein Schnittpunkt berechnen. Im zweiten Schritt wird überprüft, ob der berechnete Schnittpunkt innerhalb des Dreiecks liegt.

Eine Ebene lässt sich durch folgende Formel beschreiben:

$$\mathbf{p} \cdot \mathbf{n} + \mathbf{d} = 0, \tag{2.40}$$

$\mathbf{p}$ ist ein Punkt der Ebene, $\mathbf{n}$ die Normale der Ebene und $\mathbf{d}$ der Ortsvektor. Seien $\mathbf{a}$, $\mathbf{b}$ und $\mathbf{c}$ die Punkte des Dreiecks. Dann ergibt sich $\mathbf{n}$ aus:

$$\mathbf{n} = (\mathbf{b} - \mathbf{a}) \times (\mathbf{c} - \mathbf{a}) . \tag{2.41}$$

Ein Strahl ist definiert durch:

$$\mathbf{p} = \mathbf{o} + t\mathbf{r}. \tag{2.42}$$

Dabei ist $\mathbf{p}$ der errechnete Punkt auf dem Strahl, $\mathbf{o}$ ist der Ursprungspunkt des Strahls, t ein Skalar und $\mathbf{r}$ der Richtungsvektor des Strahls.

Zur Ermittlung des Schnittpunktes des Strahls mit der Ebene wird die Strahlengleichung in die Ebenengleichung eingesetzt und die Gleichung nach t aufgelöst:

$$0 = (\mathbf{o} + t\mathbf{r}) \cdot \mathbf{n} + \mathbf{d} \tag{2.43}$$

$$\Leftrightarrow t = -(\mathbf{o} \cdot \mathbf{n} + \mathbf{d}) / (\mathbf{r} \cdot \mathbf{n}) . \tag{2.44}$$

Um den Schnittpunkt zwischen dem Strahl und der Ebene auszurechnen, wird t in die Strahlengleichung eingesetzt. Das Ergebnis der Strahlengleichung ergibt den Schnittpunkt des Strahls mit der Ebene.

Für den berechneten Schnittpunkt muss schließlich überprüft werden, ob er sich innerhalb des Dreiecks befindet. In der Ebene lässt sich das Dreieck als Schnitt von drei Halbräumen beschreiben. Diese Halbräume entstehen, wenn für jede Kante des Dreiecks eine Ebene berechnet wird, die orthogonal zur Ebene liegt, in der sich das Dreieck befindet. Nun kann für den Schnittpunkt der Abstand zu jeder dieser drei Ebenen folgendermaßen berechnet werden:

$$\lambda_1 = [(\mathbf{a} - \mathbf{s}) \times (\mathbf{b} - \mathbf{s})] \cdot \mathbf{n} \tag{2.45}$$
$$\lambda_2 = [(\mathbf{b} - \mathbf{s}) \times (\mathbf{c} - \mathbf{s})] \cdot \mathbf{n} \tag{2.46}$$
$$\lambda_3 = [(\mathbf{c} - \mathbf{s}) \times (\mathbf{a} - \mathbf{s})] \cdot \mathbf{n}. \tag{2.47}$$

Dabei repräsentieren λ_1, λ_2, und λ_3 die drei Distanzen des Schnittpunkts zu den drei Ebenen der Kanten, $\mathbf{a}$, $\mathbf{b}$ und $\mathbf{c}$ die Punkte des Dreiecks, $\mathbf{n}$ die Normale der Ebene und $\mathbf{s}$ den zu testenden Schnittpunkt.

Das Vorzeichen der berechnten Distanzen gibt Aufschluss, in welchem von jeder Ebene erzeugtem Halbraum sich der Schnittpunkt befindet. Dabei haben alle Halbräume, die auch das Dreieck beinhalten, immer das gleiche Vorzeichen. Besitzen demnach λ_1, λ_2 und λ_3 das gleiche Vorzeichen, befindet sich der Schnittpunkt innerhalb des Dreiecks und es existiert ein Schnitt zwischen dem Strahl und dem Dreieck (siehe [Rei05]).

2.3.3. Strahl-Quader

Sei Q ein Quader definiert durch seine Eckpunkte $\mathbf{q}_i$, mit $i \in [0..7]$. Auf der Oberfläche des Quaders können mit Hilfe der Eckpunkte insgesamt 12 Dreiecke definiert werden, so dass der Quader vollständig durch die Dreieke beschrieben wird. Dabei bestimmen die Indizes der Eckpunkte ein Dreieck eindeutig. Beispielsweise sei D ein Dreieck auf der Oberfläche des Quaders definiert durch die Indizes 0, 1 und 2. Zur Durchführung des Schnittests sind dann lediglich alle Dreiecke des Quaders auf Schnitt mit einem Strahl zu testen. Dieses Vorgehen wurde bereits in Kapitel 2.3.2 hinreichend beschrieben.

2.3.4. Dreieck-Dreieck 2D

Der zweidimensionale Dreieck-Dreieck Schnittest überprüft als erstes, ob sich eine Kante des einen Dreiecks mit einer Kante des anderen Dreiecks überkreuzt. Ist dies der Fall,

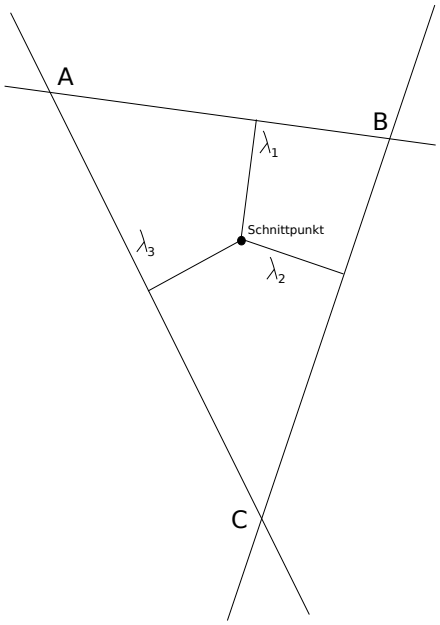


Abbildung 2.17.: Distanzen des Schnittpunkts zu den drei Ebenen der Kanten

liegt ein Schnitt zwischen den beiden Dreiecken vor.

Jede Kante ist begrenzt durch zwei Punkte, jedoch kann ein Strahl konstruiert werden, auf dem die Punkte der Kante liegen. Für zwei Kanten k_1 und k_2 mit den Punkten $\mathbf{p}_1$, $\mathbf{p}_2$ und $\mathbf{q}_1$, $\mathbf{q}_2$ lassen sich die entsprechenden Strahlengleichungen folgendermaßen beschreiben:

$$\mathbf{p}_0 = \mathbf{p}_1 + s\,(\mathbf{p}_2 - \mathbf{p}_1) \tag{2.48}$$

$$\mathbf{q}_0 = \mathbf{q}_1 + t\,(\mathbf{q}_2 - \mathbf{q}_1)\,. \tag{2.49}$$

Dabei sind $\mathbf{p}_0$ und $\mathbf{q}_0$ die errechneten Punkte der Strahlengleichungen, $\mathbf{p}_1$ und $\mathbf{q}_1$ zwei beliebige Punkte auf der Geraden, s und t zwei Skalare und $(\mathbf{p}_2 - \mathbf{p}_1)$ und $(\mathbf{q}_2 - \mathbf{q}_1)$ ergeben die Richtungsvektoren der Strahlen.

Um den Schnittpunkt der beiden Strahlen ermitteln zu können, werden beide Strahlengleichungen jeweils für den X- und Y-Anteil jedes Punktes gleichgesetzt:

$$x_{p_1} + s\,(x_{p_2} - x_{p_1}) = x_{q_1} + t\,(x_{q_2} - x_{q_1}) \tag{2.50}$$

$$y_{p_1} + s\,(y_{p_2} - y_{p_1}) = y_{q_1} + t\,(y_{q_2} - y_{q_1})\,. \tag{2.51}$$

Anschließend werden beide Gleichungen nach s und t umgestellt und s und t ausgerechnet. Wenn dabei der Nenner für beide Gleichungen null ergibt, liegen die beiden Kanten parallel zueinander und es existiert kein Schnittpunkt. Ergeben zusätzlich beide Zähler ebenfalls null, sind beide Kanten koinzident, liegen also übereinander. Dementsprechend ist ein Schnitt zwischen den beiden Kanten vorhanden. Ansonsten existiert ein Schnittpunkt zwischen den beiden Strahlen und es muss überprüft werden, ob sich der Schnittpunkt der beiden Strahlen auf den beiden Kanten befindet. Dies ist der Fall, wenn sich die Werte der ausgerechneten Skalare s und t im Intervall von $[0, 1]$ befinden.

Auch wenn sich keine Kanten der Dreiecke überkreuzen besteht noch die Möglichkeit, dass ein Dreieck das andere Dreieck komplett enthält. Um dies auszuschließen, wird für jeweils einen Punkt eines Dreiecks überprüft, ob er sich innerhalb des anderen Dreiecks befindet. Ist dies nicht der Fall, kann ein Schnitt der beiden Dreiecke ausgeschlossen werden.

2.3.5. Dreieck-Dreieck 3D

Der Dreieck-Dreieck Schnittest erfolgt ebenfalls in zwei Schritten. Da sich für jedes Dreieck eine Ebene bestimmen lässt, in der sich das Dreieck befindet, wird in einem ersten Schritt überprüft, ob eins der Dreiecke die Ebene, in der das andere Dreieck liegt, schneidet. Ist dies nicht der Fall liegt kein Schnitt zwischen den beiden Dreiecken vor.

Gegeben seien das Dreieck t_1 mit den Punkten $\mathbf{p}_1$, $\mathbf{p}_2$, $\mathbf{p}_3$ und das Dreieck t_2 mit den Punkten $\mathbf{q}_1$, $\mathbf{q}_2$, $\mathbf{q}_3$.

Um nun für das Dreieck t_2 zu überprüfen, ob es die Ebene des Dreiecks t_1 schneidet, wird für t_2 zuerst die Ebenengleichung aufgestellt und der Normalenvektor sowie der Ortsvektor der Ebene bestimmt:

$$\mathbf{n}_2 \cdot \mathbf{x} + \mathbf{d}_2 = 0. \tag{2.52}$$

$\mathbf{n}_2$ ist der Normalenvektor der Ebene, die von t_2 aufgespannt wird, $\mathbf{x}$ ein beliebiger Punkt auf der Ebene und $\mathbf{d}_2$ der Ortsvektor.

Der Normalenvektor $\mathbf{n}_2$ und der Ortsvektor $\mathbf{d}_2$ werden wie folgt berechnet:

$$\mathbf{n}_2 = (\mathbf{q}_2 - \mathbf{q}_1) \times (\mathbf{q}_3 - \mathbf{q}_1) \tag{2.53}$$

$$\mathbf{d}_2 = -\mathbf{n}_2 \cdot \mathbf{q}_1. \tag{2.54}$$

Anschließend werden alle drei Punkte von t_1 in die Ebenengleichung von t_2 eingesetzt und somit die Distanzen der Punkte zur Ebene bestimmt:

$$d_{p_i} = \mathbf{n}_2 \cdot \mathbf{p}_i + \mathbf{d}_2, i \in \{0, 1, 2\}\,. \tag{2.55}$$

Die gleichen Berechnungen werden ebenfalls für das Dreieck t_2 und der vom Dreieck t_1 aufgespannten Ebene durchgeführt.

Jede Ebene teilt einen Raum in zwei Halbräume. Besitzen demnach alle drei Punkte eines Dreiecks eine positive oder negative Distanz zur Ebene des anderen Dreiecks, bedeutet dies, dass sich alle drei Punkte in demselben Halbraum befinden und somit das Dreieck die Ebene nicht schneidet.

Haben alle Distanzen den Wert null bedeutet dies, dass sich beide Dreiecke in derselben Ebene befinden und deshalb mittels eines zweidimensionalen Dreieck-Dreieck Schnittests auf Schnitt getestet werden müssen. (siehe Kapitel 2.3.4).

Sind die Vorzeichen der berechneten Distanzen unterschiedlich, schneidet ein Dreieck die Ebene des anderen Dreiecks. Demnach schneiden sich auch die beiden Ebenen der Dreiecke und der Schnitt der beiden Ebenen ergibt eine Gerade.

Auf dieser Geraden befinden sich ebenfalls die Schnittpunkte der Dreiecke mit den Ebenen.

Im zweiten Schritt werden nun diese Schnittpunkte näher untersucht, um einen Schnitt der Dreiecke zu bestätigen oder auszuschließen. Die Schnittpunkte lassen sich berechnen, indem die Kanten der Dreiecke betrachtet werden. Jede Kante besteht aus zwei Punkten und wenn ein Punkt eine negative Distanz zur Ebene des anderen Dreiecks aufweist und der andere Punkt eine positive Distanz zu der gleichen Ebene besitzt, schneidet die Kante die Ebene. Die entsprechenden Distanzen wurden bereits im ersten Schritt bestimmt. Für alle Kanten die eine Ebene schneiden, werden schließlich die Schnittpunkte folgendermaßen berechnet:

Aus den zwei Punkten einer Kante mit den Punkten $\mathbf{p}_1$ und $\mathbf{p}_2$ lässt sich ein Strahl konstruieren der am Punkt $\mathbf{p}_0$ eine Ebene mit dem Normalenvektor $\mathbf{n}$ und dem Ortsvektor $\mathbf{d}$ schneidet:

$$\mathbf{p}_0 = \mathbf{p}_1 + t\,(\mathbf{p}_2 - \mathbf{p}_1)\,. \tag{2.56}$$

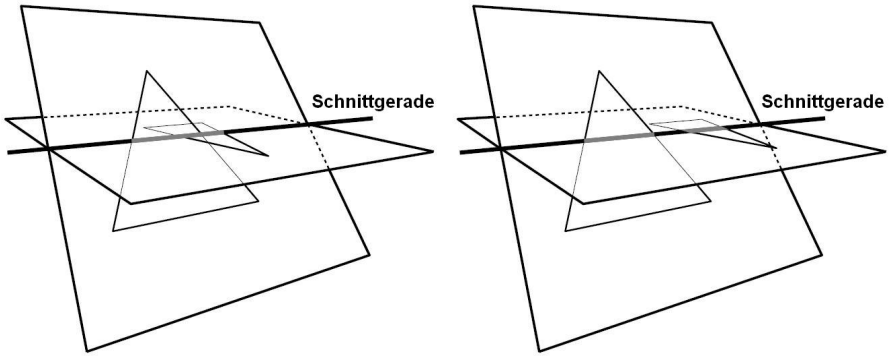


Abbildung 2.18.: Überlappende Intervalle und nicht überlappende Intervalle (aus [Möl97])

Die Strahlengleichung wird in die Ebenengleichung eingesetzt und die Gleichung so umgestellt, dass das Skalar t bestimmt wird:

$$t = \frac{\mathbf{d} - \mathbf{n} \cdot \mathbf{p}_1}{\mathbf{n} \cdot (\mathbf{p}_2 - \mathbf{p}_1)}. \tag{2.57}$$

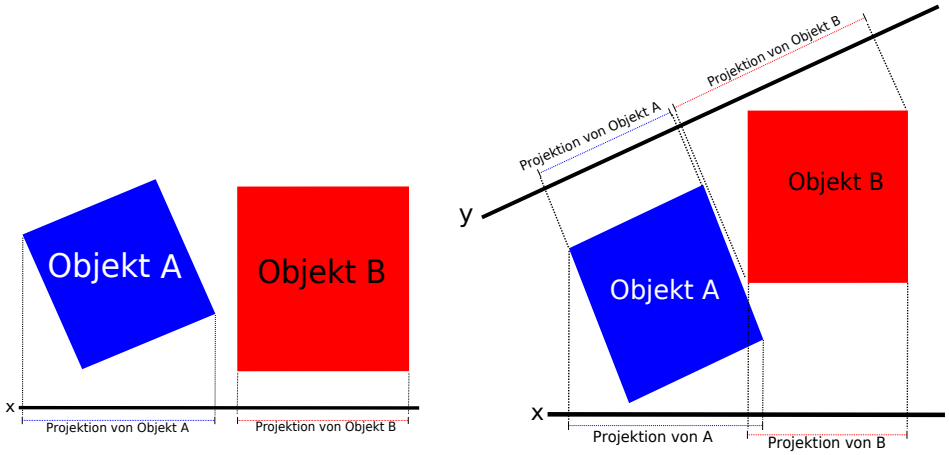
Der Schnittpunkt wird schließlich errechnet, indem das berechnete Skalar t wieder in die Strahlengleichung eingesetzt wird.

Nachdem die Schnittpunkte berechnet wurden und bekannt ist, dass sie alle auf einer Geraden liegen, muss noch überprüft werden, ob sich die Schnittpunkte eines Dreiecks mit den Schnittpunkten des anderen Dreiecks auf der Geraden überschneiden. Es können für jedes Dreieck ein oder zwei Schnittpunkte existieren. Nur einen Schnittpunkt gibt es, wenn ein einzelner Eckpunkt eines Dreiecks genau auf der Ebene des anderen Dreiecks liegt.

Gibt es lediglich einen Schnittpunkt für jedes Dreieck muss überprüft werden, ob die beiden Schnittpunkte identisch sind. Sind es zwei Schnittpunkte für ein Dreieck und lediglich ein Schnittpunkt für das andere Dreieck gilt es zu überprüfen, ob der einzelne Punkt im Intervall der beiden anderen Punkte liegt. Besitzen beide Dreiecke zwei Schnittpunkte müssen dementsprechend zwei Intervalle auf Überlappung getestet werden (siehe [Möl97]).

2.3.6. Quader-Quader

Der Test auf einen Schnitt zwischen zwei Quadern wird anhand des „Separating Axis Theorem (SAT)“ durchgeführt. Das SAT besagt, dass zwischen zwei konvexen Objekten



(a) Eine Achse benötigt (b) Zwei Achsen benötigt

Abbildung 2.19.: Separating Axis Theorem (Kein Schnitt zwischen den Objekten)

genau dann kein Schnitt existiert, falls es eine Achse gibt, auf derer die Projektionen der beiden Objekte sich nicht schneiden (siehe Abbildung 2.19).

Der Test funktioniert dabei nur für konvexe Objekte in allen Fällen, bei nicht konvexen Objekten kann es vorkommen, dass ein Schnitt erkannt wird, obwohl die Objekte sich nicht schneiden (siehe Abbildung 2.20).

Im Fall des Quader-Quader Schnitttests, kann dieser Fall allerdings nicht vorkommen, da Quader per Definition konvexe Objekte sind. Die trennende Achse ist dabei entweder orthogonal zu einer Fläche eines Quaders oder orthogonal zu einer Kante von jedem Quader. Da jeder Quader drei verschiedene im Koordinatensystem orientierte Flächen und Kanten besitzt, existieren fünfzehn mögliche Trennachsen, bestehend aus den sechs Achsen der verschiedenen Flächen sowie aus den neun Achsen aller Kantenkombinationen der beiden Quader. Um für eine der fünfzehn Achsen zu überprüfen, ob es eine Trennachse ist, werden sowohl die Schwerpunkte (siehe Kapitel 7.3.2) der Quader als auch die Radien der Quader auf die Achse projiziert. Danach werden die Längen des Intervalls der Schwerpunkte mit den addierten Längen der Intervalle der Radien verglichen. Wenn die Länge des Intervalls der Schwerpunkte größer ist als die addierten Längen der Intervalle der Radien, ist die Achse eine Trennachse. Ist eine Trennachse gefunden worden, haben die Quader keinen Schnitt und es müssen keine weiteren Achsen überprüft werden (siehe [GLM96]).

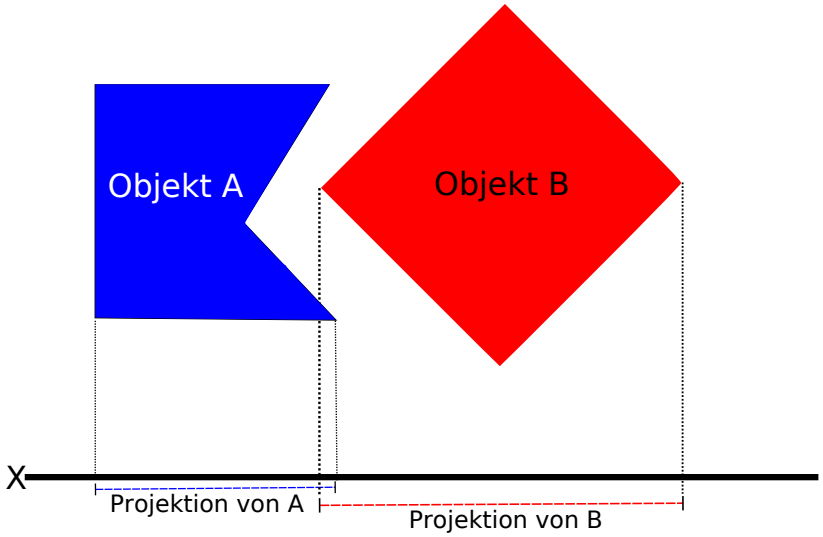


Abbildung 2.20.: Schnittest versagt für nicht konvexe Objekte

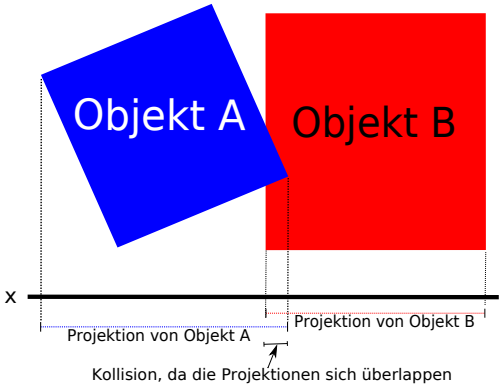


Abbildung 2.21.: Erkennung kollidierender Objekte mittels Separating Axis Theorem

2.3.7. Ebene-Dreiecksnetz

Die Schnittberechnung einer vorgegebenen Ebene mit einem Dreiecksnetz erfolgt, indem alle Kanten des Netzes auf Schnitt mit der Ebene getestet werden. Die Vorgehensweise bei diesem Geraden-Ebenen-Schnitttest ist, dass zuerst überprüft wird, ob einer oder beide Punkte der Kante in der Ebene liegen. Dazu wird der zu überprüfende Punkt in die gegebene Ebenengleichung eingesetzt:

$$\mathbf{p} \cdot \mathbf{n} - \mathbf{d} = 0. \tag{2.58}$$

Dabei ist $\mathbf{p}$ der zu überprüfende Punkt, $\mathbf{n}$ der Normalenvektor der Ebene und $\mathbf{d}$ der Ortsvektor der Ebene. Ergibt das Punktprodukt von $\mathbf{p}$ und $\mathbf{n}$ abzüglich des Ortsvektors $\mathbf{d}$ null, so befindet sich $\mathbf{p}$ in der Ebene und es liegt dann trivialerweise bereits ein Schnitt vor.

Ansonsten bleibt zu überprüfen, ob die Kante die Ebene schneidet. Das ist genau dann der Fall, wenn die Punkte nicht beide auf einer Seite der Ebene liegen. Die Ebene teilt dabei den Raum in zwei Halbräume und wenn die Punkte der Kante in verschiedenen Halbräumen liegen, muss dementsprechend ein Schnitt vorliegen. Um dies zu überprüfen wird aus den beiden Punkten der Kante ein Strahl konstruiert:

$$\mathbf{p}_0 = \mathbf{p}_1 + s \cdot (\mathbf{p}_2 - \mathbf{p}_1)\,. \tag{2.59}$$

Der Punkt $\mathbf{p}_0$ ist der errechnete Punkt des Strahls, $\mathbf{p}_1$ ist ein Punkt auf dem Strahl, s ein Skalar und $\mathbf{p}_2 - \mathbf{p}_1$ repräsentiert den Richtungsvektor des Strahls.

Als nächstes muss festgestellt werden, ob der Strahl die Ebene schneidet. Deshalb wird die Strahlengleichung in die Ebenengleichung eingesetzt:

$$(\mathbf{p}_1 + s \cdot \mathbf{p}_2 - \mathbf{p}_1) \cdot \mathbf{n} - \mathbf{d} = 0 \tag{2.60}$$

Ergibt sich aus der Gleichung, dass ein Schnitt zwischen Strahl und Ebene vorliegt wird die Gleichung nach s umgestellt und s ausgerechnet.

$$s = \frac{\mathbf{d} - \mathbf{p}_1 \cdot \mathbf{n}}{\mathbf{p}_2 - \mathbf{p}_1 \cdot \mathbf{n}} \tag{2.61}$$

Da der Richtungsvektor aus den Punkten der Kante konstruiert worden ist, muss bei einem Schnitt des Strahls mit der Ebene das Skalar s im Intervall von $[0, 1]$ liegen, damit nicht nur der Strahl, sondern auch die Kante die Ebene schneidet.

2.4. Quaternionen

Neben 3×3-Matrizen sind Quaternionen ein mathematisches Objekt mit der Rotationen im $\mathbb{R}^3$ dargestellt werden können. Sie werden in der Computer-Grafik gerne als Ersatz für Matrizen verwendet. Im Gegensatz zu Matrizen benötigen Quaternionen weniger Speicherplatz und ihre Multiplikation kann mit weniger elementaren arithmetischen Operationen durchgeführt werden. Da Quaternionen einfach zu interpolieren sind, stellen sie ein beliebtes Hilfsmittel für die Berechnung weicher Animationen dar. Einen pragmatischen aber dennoch formalen Einstieg in die Thematik der Quaternionen liefert [Len02].

In der im Kapitel 3.8 beschriebenen Programmiersprache RAPID werden Quaternionen verwendet, um ein Target, genauer die Orientierung in einem Bahnpunkt, zu beschreiben. Quaternionen werden zu diesem Zweck durch vier reellwertige Zahlen beschrieben, welche für den skalaren und vektoriellen Teil stehen.

Die Menge der Quaternionen $\left\{[s, \mathbf{v}] \mid s \in \mathbb{R},\ \mathbf{v} \in \mathbb{M}^3\right\}$ bilden mit Multiplikation und Addition einen nicht kommutativen Ring mit einselement. Sie besitzt zusammengefasst folgende Operationen und Eigenschaften:

- Addition: $[s_1, \mathbf{v}_1] + [s_2, \mathbf{v}_2] = [s_1 + s_2,\ \mathbf{v}_1 + \mathbf{v}_2]$
- Multiplikation: $[s_1, \mathbf{v}_1] \cdot [s_2, \mathbf{v}_2] = [s_1 \cdot s_2 - \mathbf{v}_1 \cdot \mathbf{v}_2,\ s_1 \cdot \mathbf{v}_2 + s_2 \cdot \mathbf{v}_1 + \mathbf{v}_1 \times \mathbf{v}_2]$
- Neutrales Element bzgl. Addition: $[0, \mathbf{0}]$
- Neutrales Element bzgl. Multiplikation: $[1, \mathbf{0}]$
- Additives Inverses: $-[s, \mathbf{v}] = [-s, -\mathbf{v}]$
- Konjugiert: $\overline{[s, \mathbf{v}]} = [s, -\mathbf{v}]$
- Betrag: $|[s, \mathbf{v}]| = \sqrt{s^2 + |\mathbf{v}|^2}$
- Multiplikatives Inverses: $[s, \mathbf{v}]^{-1} = \frac{\overline{[s,\mathbf{v}]}}{\left|\overline{[s,\mathbf{v}]}\right|^2}$

Eine Rotation mit Winkel $\theta \in [0, 2\pi[$ um eine beliebige Achse beschrieben durch den normierten Richtungsvektor $\mathbf{v} \in \mathbb{R}^3$, $\|\mathbf{v}\| = 1$, ist gegeben durch die Quaternion

$$q_r = \left[cos\left(\frac{\theta}{2}\right), sin\left(\frac{\theta}{2}\right) \cdot \mathbf{v}\right] \tag{2.62}$$

Sie hat als Betrag den Wert eins, da laut Vorraussetzung $|\mathbf{v}_r| = 1$ ist und für beliebige

Winkel $\alpha \in \mathbb{R}$ gilt: $cos\,(\alpha) + sin\,(\alpha) = 1$.

$$
\begin{aligned}
|\overline{q_r}| &= \left|\overline{\left[cos\left(\frac{\theta}{2}\right), sin\left(\frac{\theta}{2}\right) \cdot \mathbf{v}\right]}\right| && (2.63)\\
&= \left|\left[cos\left(\frac{\theta}{2}\right), -sin\left(\frac{\theta}{2}\right) \cdot \mathbf{v}\right]\right| && (2.64)\\
&= \sqrt{cos^2\left(\frac{\theta}{2}\right) + \left(-sin^2\left(\frac{\theta}{2}\right)\right) \cdot \mathbf{v}^2} && (2.65)\\
&= \sqrt{cos^2\left(\frac{\theta}{2}\right) + sin^2\left(\frac{\theta}{2}\right)} && (2.66)\\
&= 1 && (2.67)
\end{aligned}
$$

Anschaulich liegt eine Quaternion mit Betrag eins auf einer vierdimensionalen Einheits-Hyper-Sphäre.

Um eine Rotation $\mathbf{R}$ auf einen Vektor $\mathbf{x} \in \mathbb{R}$ anzuwenden, wird der Vektor zunächst in den Quaternionenraum abgebildet:

$$\mathbf{x} \longmapsto q_{\mathbf{x}} = [0, \mathbf{x}]\,. \tag{2.68}$$

Danach wird der so entstandene Repräsentant $q_{\mathbf{x}}$ durch folgende Abbildung transformiert:

$$q_{\mathbf{x}} \longmapsto q_r \cdot q_{\mathbf{x}} \cdot q_r^{-1}. \tag{2.69}$$

Da der Betrag der Rotation gleich eins ist, gilt:

$$[s, \mathbf{v}]^{-1} = \overline{[s, \mathbf{v}]}. \tag{2.70}$$

Damit kann die Abbildung dann vereinfacht werden zu:

$$q_{\mathbf{x}} \longmapsto q_r \cdot q_{\mathbf{x}} \cdot \overline{q_r} = q_{\mathbf{x}\text{'}}. \tag{2.71}$$

Abschließend kann der rotierte Vektor durch Abbildung aus dem Quaternionenraum bestimmt werden:

$$q_{\mathbf{x}\text{'}} = [0, \mathbf{x}\text{'}] \longmapsto \mathbf{x}\text{'}. \tag{2.72}$$

Für einen Beweis der Korrektheit der Rotation siehe [Len02].

Eingabe : Minimale l_{min} und maximale l_{max} Kantenlänge. Dreiecksnetz mesh in Form einer Datenstruktur welche die Operationen edge_collapse, edge_split, edge_flip und Bestimmung des Knotengrads valence unterstützt.
Ausgabe : Umvernetztes Dreiecksnetz
repeat
  `/*Alle zu kurzen Kanten entfernen */`
  foreach $e \in$ *edges(mesh)* **do**
    if $|e| < l_{min}$ **then**
      if *is_collapse_possible(e)* **then**
        edge_collapse(e);
      endif
    endif
  endfch
  `/*Alle zu langen Kanten unterteilen */`
  foreach $e \in$ *edges(mesh)* **do**
    if $|e| > l_{max}$ **then**
      if *is_split_possible(e)* **then**
        edge_split(e);
      endif
    endif
  endfch
  `/*Knotengrad reduzieren */`
  foreach $e \in$ *edges(mesh)* **do**
    $\{t_l, t_r\}$ = incident_triangles(e);
    $\{v_1, v_2\}$ = incident_vertices(e);
    $\{v_0\}$ = incident_vertices(t_l) $\setminus\{v_1, v_2\}$;
    $\{v_3\}$ = incident_vertices(t_r) $\setminus\{v_1, v_2\}$;
    $(c_0, ..., c_3)$ = (1,-1,-1,1);
    total_valence = $\sum_{i=0}^{3} (valence(v_i) - 6)^2$;
    total_valence_flipped = $\sum_{i=0}^{3} (valence(v_i) - 6 + c_i)^2$;
    if *total_valence_flipped* $<$ *total_valence* **then**
      if *is_flip_possible(e)* **then**
        edge_flip(e);
      endif
    endif
  endfch
until *(max_iterations_reached* $\vee \neg$ *mesh_changed)* ;
return *mesh*

Algorithmus 5 : Umvernetzungsalgorithmus nach Kobbelt

3. Datenformate

Dieses Kapitel gibt einen kurzen Überblick über die im Rahmen der Projektgruppe relevanten Datenformate.

Allein für Geometriedaten gibt es eine Vielzahl an Datenformaten. Eine (unvollständige) Liste ist zum Beispiel in [MvR96] zu finden. Um möglichst viele verschiedene Modelle verarbeiten zu können, unterstützt *RoSi3D* verschiedene CAD-Datenformate. Die Unterstützung von IGES und STEP basiert auf dem CAD-Kernel OpenCASCADE (siehe Kapitel A.5). Dieser liest die Freiformgeometrien ein und trianguliert sie, so dass sie von *RoSi3D* weiterverarbeitet werden können. BRep, das native Datenformat von OpenCASCADE, wird ebenfalls unterstützt.

OBJ- und STL-Dateien werden mit Hilfe von OpenMesh (siehe Kapitel A.4) verarbeitet, welches in *RoSi3D* dazu verwendet wird, die benutzten Dreiecksnetze zu speichern.

Mit der Geometrie des Werkstücks werden noch zusätzliche Eigenschaften gespeichert. Dies geschieht mit Hilfe der Boost-Bibliothek (siehe Kapitel A.8) im Ply- bzw. Gply-Format.

Die Auszeichnungssprache XML wird im Rahmen von *RoSi3D* zum Speichern und Laden von vollständigen Projekten benutzt, für den Im- und Export von Teilkomponenten, sowie zum Datenaustausch zwischen *RoSi3D* und *RobotStudio*.

RAPID schließlich ist die Programmiersprache, mit der die Roboter von ABB angesteuert werden.

3.1. IGES

Die erste Version des IGES Standards entstand 1981 als American National Standard (siehe [Ame81]). 1996 wurde die Weiterentwicklung dann eingestellt.

Mittlerweile ist IGES ein etablierter Standard im CAD-Umfeld. Das Format ist zwar nicht proprietär, aber sehr komplex. Die Spezifikation von Version 4 umfasst über 500 Seiten. Daher wird es meistens nur von kommerziellen CAD-Programmen unterstützt. *RoSi3D* unterstützt das IGES-Format durch den OpenCASCADE CAD-Kernel.

Der Datenaustausch geschieht hier in Form von sogenannten Entities. Ein Entity kann sowohl eine einfache Linie als auch eine komplexe Bézierfläche sein. Aufgrund der bereits erwähnten Komplexität des Standards unterstützen viele CAD-Programme nur eine Teilmenge aller im IGES-Standard definierten Entities. Die nicht unterstützen Entities werden ignoriert, wodurch die Eignung von IGES als Datenaustauschformat eingeschränkt wird. Ein weiteres Problem des IGES-Formats sind numerische Ungenauigkeiten.

Die positive Seite von IGES ist seine Ausdruckskraft, denn mit IGES können Freiformflächen (Bézier und BSpline) gespeichert werden. Außerdem hat IGES eine Reihe zusätzlicher Features wie zum Beispiel die Unterstützung von Constructive Solid Geometry. Bei Constructive Solid Geometry (CSG) handelt es sich um eine Modelliertechnik, bei der Objekte mit Hilfe boolescher Operationen aus anderen Objekten erzeugt werden können. Weitere Features wie die Gruppierung von Objekten in einer Hierarchie oder in Layern sind für die Projektgruppe nicht relevant.

3.2. STEP

STEP ist das Neueste der beschriebenen Formate und hat sich weitestgehend als Nachfolger von IGES durchgesetzt (siehe [ISO94]). Dateien im STEP-Format sollen nicht nur die Geometrie eines Objekts bzw. Produkts speichern, sondern ein ganzes Produktmodell. Dazu gehören ganz allgemein alle Daten, die im Laufe des Lebenszyklus des Produkts benötigt werden könnten. STEP gliedert sich in viele Unterprotokolle für viele unterschiedliche Bereiche. Für die Projektgruppe interessant sind die Protokolle AP203 und AP214, da diese auch vom frei verfügbaren CAD-Kernel OpenCASCADE unterstützt werden.

Die Speicherung der Geometrie geschieht hier auch wieder in Form von Entities. Zum Funktionsumfang von IGES kommen noch Erweiterungen wie Stücklisten, Versionierung, Änderungsmanagement und vieles andere hinzu, welches im Rahmen der Projektgruppe allerdings nicht von Relevanz ist.

3.3. Polygon File Format

Das Polygon-File-Format ist auch bekannt als Stanford-Triangle-Format. Wie der Name schon andeutet, können mit Dateien im Ply-Format nur Polygone gespeichert werden. Zusätzlich können auch Farben, Texturkoordinaten und Normalen gespeichert werden. Das Ply-Format ist leicht erweiterbar, so dass auch benutzerdefinierte Attribute gespeichert werden können. In *RoSi3D* werden zum Beispiel bestimmte Kantentypen des CAD-Modells des Werkstücks markiert und im Ply-Format abgespeichert.

3.4. STL File Format

Mit Hilfe des STL-Formats (STL: „StereoLithography“) können Dreiecksnetze gespeichert werden (siehe [3D 89]). Es gibt ein binäres STL-Format und ein STL-Format in ASCII-Codierung.

Für jede Facette des Dreiecksnetzes werden bei STL die Koordinaten der Eckpunkte und der Normalenvektor abgespeichert. Da jeder Eckpunkt in mehreren Facetten vorkommen kann, werden die Koordinaten der Eckpunkte mehrfach abgespeichert, wodurch STL-Dateien größer werden als eigentlich nötig. Aufgrund der Tatsache, dass nur Dreiecksnetze zur Verfügung stehen, können Objekte mit Rundungen nur angenähert werden.

Trotzdem ist das STL-Format sehr weit verbreitet und gilt als Industriestandard. Außerdem können STL-Modelle mit Hilfe von Rapid Prototyping Maschinen direkt in ein echtes Modell umgesetzt werden.

3.5. Wavefront OBJ

Das OBJ-File-Format (siehe [MvR96]) von Wavefront ist ein weiteres gebräuchliches Datenformat. Wie beim STL-Format gibt es hier sowohl eine binäre Variante als auch eine Variante in ASCII-Codierung. Obwohl mit dem OBJ-Format auch Freiformgeometrie gespeichert werden kann, wird das OBJ-Format oft nur zum Speichern von polygonalen Netzen gebraucht. Für OBJ-Dateien gibt es zusätzliche „Material Library Files“ in denen Materialeigenschaften wie Farben (diffus, ambient und spekular), Transparenz und Brechungsindex gespeichert werden können. Texturkoordinaten werden direkt in der OBJ-Datei gespeichert.

In *RoSi3D* wird das OBJ-Format für polygonale Netze unterstützt. Freiformgeometrien können mit dem OBJ-Format in *RoSi3D* nicht geladen werden.

3.6. BRep

Bei BRep handelt es sich um das native Datenformat von OpenCASCADE. BRep steht hierbei für Boundary Representation. Da OpenCASCADE S.A. keine Formatbeschreibung veröffentlicht hat, gibt es hierzu wenige Informationen.

3.7. XML

XML (Extensible Markup Language) ist eine hierarchisch strukturierte Auszeichnungssprache. Ein XML-Dokument besteht aus Elementen und ein Element wiederum besteht aus einem Tag am Anfang, dem Inhalt und einem Tag am Schluss. Tags beschreiben den Inhalt der Daten in einem XML-Dokument. Jedes Element muss ein Anfangstag und ein Endtag haben:

```
<Tag>Inhalt</Tag>
```

Tags können ineinander geschachtelt werden. Hierdurch entsteht die Hierarchie in XML-Dokumenten. Unter der XML-Deklaration folgt das Wurzelelement, unter das alle anderen Elemente gehängt werden.

Zum Parsen der XML-Dateien wird im Rahmen der Projektgruppe Qt (siehe Kapitel A.2) verwendet. Qt bietet einen DOM-Parser, welcher die komplette zu ladende XML-Datei in einer Baumstruktur in den Speicher lädt und einen SAX-Parser, welcher sequentiell durch die XML-Datei traversiert. In *RoSi3D* wird der DOM-Parser verwendet.

3.8. RAPID

RAPID ist die Programmiersprache für Industrieroboter von ABB. Ein mit RAPID geschriebenes Programm besteht typischerweise aus verschiedenen Modulen. Es gibt zwei Typen von Modulen: Programm- und Systemmodule. Die Systemmodule enthalten vor allem vordefinierte Systeminformationen wie Geschwindigkeits- und Zonendeklarationen (siehe Kapitel 3.8.1), während die Programmmodule die benutzerdefinierten Datendeklarationen und Prozeduren enthalten.

Eine Moduldefinition wird über das Schlüsselwort MODULE eingeleitet und per Schlüsselwort ENDMODULE beendet. Innerhalb eines Moduls stehen zunächst die globalen Datendeklarationen wie Robotertargets, Werkzeuge, Werkobjekte, Strings, boolesche Variablen, numerische Variablen und so weiter. Unter den Datendeklarationen stehen die Prozeduren, die zwischen den Schlüsselwörtern PROC und ENDPROC zu definieren sind und eine Hauptprozedur enthalten, die die anderen Prozeduren in der entsprechenden Reihenfolge aufruft:

```
MODULE Module1
      CONST robtarget Target_10:=[...];

PROC MeinPfad()
      MoveL Target_10,[...];
```

```
ENDPROC

PROC main()
      MeinPfad;
ENDPROC

ENDMODULE
```

Prozeduren selbst bestehen aus einem Satz von Instruktionen, Funktions- und Prozeduraufrufen und liefern im Gegensatz zu Funktionen keinen Wert an den Aufrufer zurück. Da der RAPID-Befehlssatz recht umfangreich ist, werden im Folgenden nur die Instruktionen und Datentypen erläutert, die im Rahmen der Projektgruppe benötigt werden. Die verwendeten Instruktionen lassen sich in Bewegungs-, Logik-, Warte- und Synchronisationsinstruktionen aufteilen. Weitere Instruktionen wie zum Besipiel If-Then-Else-Konstrukte, Goto-Instruktionen, While- und For-Schleifen werden hier nicht beschrieben, da sie bei der Erzeugung der RAPID-Programme im Rahmen der Projektgruppe nicht verwendet werden.

Mehr Informationen zur Programmiersprache RAPID sind im offiziellen RAPID Referenzhandbuch von ABB [ABB07] zu finden.

3.8.1. Datentypen

Neben den bekannten Standarddatentypen für Zahlen, Zeichenketten und booleschen Werten gibt es in RAPID ein paar spezielle Datentypen, die einer kurzen Erläuterung bedürfen.

Robtarget

Die Definition eines Bahnpunktes wird in RAPID mit dem Schlüsselwort `robtarget` eingeleitet. Folgende Parameter müssen bei der Definition eines Bahnpunktes übergeben werden:

- Ein eindeutiger Name.
- Die Koordinaten des Robtargets.
- Die Orientierung ausgedrückt durch ein Quaternion.
- Die zu verwendende Konfiguration.

Die Syntax für die Definition eines Bahnpunktes sieht somit folgendermaßen aus:

```
CONST robtarget Target_1:=[[x,y,z],[q0,q1,q2,q3],[cf1,cf4,cf6,cfx],ex];
```

Dabei sind `x`, `y` und `z` die Koordinaten des Bahnpunktes und `q0` bis `q3` geben die Drehung des Bahnpunktes in Quaternionen an. `cf1`, `cf4`, `cf6` und `cfx` beschreiben die Konfiguration (siehe Kapitel 4.4.1). Der Parameter `ex` betrifft externe Achsen und wird im Rahmen der Projektgruppe nicht verwendet.

Sie werden daher auf ihren Standardwert [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09] gesetzt.

Jointtarget

Ein `jointtarget` beschreibt eine bestimmte Achsstellung des Roboters. Neben dem eindeutigen Namen für die Achsstellung müssen auch die Werte der Achsen in Grad angegeben werden. Das folgende Beispiel setzt alle Achsen, ausser der fünften, auf 0 Grad. Achse fünf wird auf 30 Grad gesetzt und die externen Achsen werden wieder auf den Standardwert [9E+09,9E+09,9E+09,9E+09,9E+09,9E+09] gesetzt, was im Beispiel mit `ex` angedeutet sein soll:

```
CONST jointtarget jointtarget_10 :=[[0,0,0,0,30,0],ex];
```

Tooldata

Wenn ein Roboter ein Werkzeug verwenden soll, muss sich im jeweiligen Programmmodul eine Definition der Werkzeugdaten befinden.

Eine Werkzeugdefinition beginnt mit dem Schlüsselwort `tooldata`. Die wichtigsten Angaben die hierbei gemacht werden müssen, sind die Position und Orientierung des TCPs:

```
PERS tooldata Spraygun:=[TRUE,[[x,y,z],[q0,q1,q2,q3]],...];
```

Wie bei der Definition eines Bahnpunktes stehen `x`, `y` und `z` hier für die Koordinaten und `q0` bis `q3` für die Orientierung. Der boolsche Wert am Anfang gibt an, ob das Werkzeug am Roboter montiert ist oder ob es sich um ein stationäres Werkzeug handelt. Die restlichen Parameter des Werkzeugs betreffen unter anderem das Gewicht und den Schwerpunkt des Werkzeugs und sind im Rahmen der Projektgruppe nicht von Bedeutung. Sie werden daher auf den Standardwerten belassen.

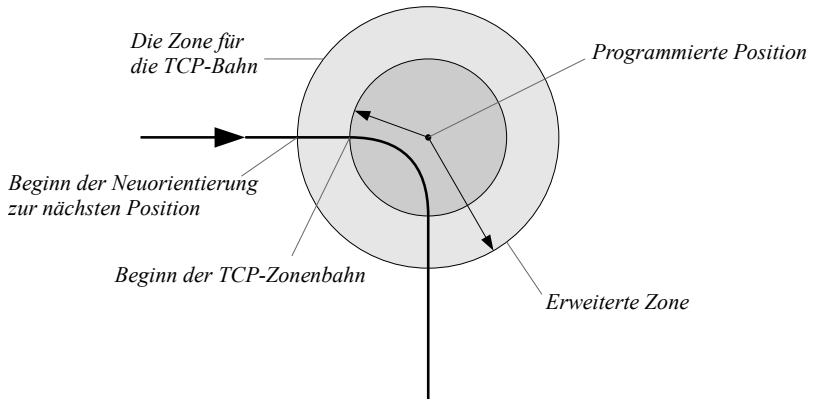


Abbildung 3.1.: Die TCP-Zonenbahn (aus [ABB07])

Zonedata

Bei Zoneninformationen besteht die Möglichkeit, auf vordefinierte Werte zurückzugreifen oder eigene Werte zu spezifizieren. Die Zoneninformationen beschreiben wie die Zielposition einer Bewegung erreicht werden soll, indem die Zielposition der Bewegungsinstruktion entweder als Stoppunkt oder aber als sogenannter Fly-By-Punkt deklariert wird. Wird die Zielposition als Stoppunkt festgelegt, was der vordefinierten Zoneninformation `fine` entspricht, dann muss der TCP die Position exakt erreichen, bevor die darauffolgende Position angefahren werden kann. Wird andererseits die Zielposition als Fly-By-Punkt deklariert, was beispielsweise über die vorgegebene Zoneninformation `z10` bewirkt werden kann, beginnt der Prozess des Anfahrens des nächsten Targets vor Erreichen der vorangegangenen Position, wobei der Beginn dieses Prozesses mit Hilfe der Zoneninformation, die auch manuell definiert werden kann, bestimmt wird. Dieser Prozess besteht im Wesentlichen aus zwei Teilen. Beim Erreichen der äußeren Zone beginnt die Umorientierung für die nächste Zielposition, sodass beim Verlassen der äußeren Zone die Orientierung des Roboters der Orientierung entspricht, wie sie vorläge, wenn an dem Punkt des Verlassens ein Stoppunkt festgelegt worden wäre. Beim Erreichen der inneren Zone wird eine parabelförmige Bahn (TCP-Zonenbahn) zwischen Eintritts- und Austrittspunkt der Zone generiert und abgefahren, sodass die eigentliche Zielposition der Bewegungsinstruktion nicht exakt erreicht wird, bevor die nächste Position angefahren wird (siehe Abbildung 3.1).

Der wesentliche Vorteil bei der Verwendung von Zoneninformationen besteht darin, dass durch die fließende Umorientierung die Wahrscheinlichkeit der Geschwindigkeitsreduktion gegebenenfalls reduziert werden kann, wobei die Umorientierung fließender erfolgt je größer die Zone gewählt wird [ABB07].

3.8.2. Bewegungsinstruktionen

Bewegungsinstruktionen beginnen mit einem entsprechenden Move-Befehl, wobei drei Arten der Roboterbewegung zu unterscheiden sind. MoveL-Befehle („Move Linear“) lassen den Roboter, beziehungsweise den Arbeitspunkt des Roboters, das Ziel auf einer linearen Bahn erreichen. Demgegenüber stehen die MoveJ-Befehle („Move Joint“), die den Roboter das Ziel auf einer nicht linearen Bahn erreichen lassen. Der Roboter bewegt sich bei dieser Bewegungsart achsweise, so dass die Bahn nicht intuitiv vorraussagbar ist. Die dritte Art der Bewegungsinstruktion steht in Form von MoveC-Befehlen („Move Circular“) bereit, die dafür sorgen, dass das Ziel auf einer bogenförmigen Bahn erreicht wird. Eine Besonderheit von MoveC-Befehlen besteht darin, dass ein vollständiger Kreisbogen immer aus zwei MoveC-Befehlen bestehen muss, wobei der erste für das Erreichen des Durchgangspunktes sorgt und der zweite für das Erreichen der Endposition zuständig ist.

MoveL

Eine Anweisung zu einer linearen Bewegung beginnt mit dem Schlüsselwort `MoveL` und benötigt folgende Parameter:

- Die Zielposition, die durch den Datentyp `robtarget` repräsentiert wird.
- Der Geschwindigkeitswert, der durch den Datentyp `speeddata` repräsentiert wird.
- Die sogenannte Zoneninformation, die durch den Datentyp `zonedata` repräsentiert wird.
- Das für die Roboterbewegung verwendete Werkzeug, das durch den Datentyp `tooldata` repräsentiert wird und die Lage des TCP beschreibt, der an die angegebene Zielposition bewegt wird.
- Das Koordinatensystem (Werkobjekt), das durch den Datentyp `wobjdata` repräsentiert wird und bezüglich dessen die Zielposition zu interpretieren ist.

In dem beispielhaften Befehl

```
MoveL    Target_10, v100, z100, AW_Gun\Wobj:=wobj0;
```

beschreibt der Parameter `Target_10` die Zielposition, `v100` die vordefinierte Geschwindigkeit mit der sich der Roboter bewegen soll, `z100` die vordefinierte Zoneninformation, `AW_Gun` das zu verwendende Werkzeug und der Parameter `Wobj:=wobj0` das verwendete Koordinatensystem. `wobj0` ist hierbei das Weltkoordinatensystem.

MoveC

Das Schlüsselwort `MoveC` leitet eine zirkuläre Bewegung ein. Die Syntax ist der des MoveL-Befehls sehr ähnlich. Es kommt nur ein weiteres `robtarget` hinzu:

```
MoveC    Target_10, Target_20, v100, z100, AW_Gun\Wobj:=wobj0;
```

Der implizite Startpunkt der zirkulären Bewegung ist hierbei die aktuelle Position des TCPs vor Ausführung der Instruktion.
`Target_10` ist der Durchgangspunkt und `Target_20` ist der Endpunkt der Bewegung.

MoveJ

Der Befehl `MoveJ` bewegt den Roboter in eine bestimmte Achsposition, die durch ein `jointtarget` definiert wurde. Weitere Parameter die übergeben werden müssen, sind auch hier wieder die Geschwindigkeit und der Zonenwert:

```
MoveJ jointtarget_10,v100, z100;
```

3.8.3. Logikinstruktionen

Logikinstruktionen können im Gegensatz zu Bewegungsinstruktionen Seiteneffekte haben, indem zum Beispiel das Ändern eines Output-Signals ein Ereignis anstößt, das wiederum für die Ausführung einer oder mehrerer Aktionen sorgt. Im Rahmen von *RoSi3D* werden diese Signale zum Ein- und Ausschalten der Werkzeuge benutzt. Digitale Output-Signale können mit dem Befehl `SetDO` gesetzt werden. Zum Setzen analoger Ausgänge gibt es den Befehl `SetAO`. Im folgenden Beispiel wird das digitale Output-Signal `DO1_1` auf 0 gesetzt:

```
SetDO DO1_1, 0;
```

Das Setzen eines analogen Signals funktioniert äquvalent.

3.8.4. Warteinstruktionen

In der Programmiersprache RAPID gibt es verschiedene Warteinstruktionen. Die einfachste Warteinstruktion sorgt dafür, dass der Roboter eine bestimmte Zeit wartet.

```
WaitTime 10;
```

sorgt zum Beispiel dafür, dass der Roboter zehn Sekunden wartet und erst danach mit der Abarbeitung des restlichen Programms fortfährt. Das kann nützlich sein, wenn ein Werkzeug manuell ein- oder ausgeschaltet werden muss. Der andere verwendete Wartebefehl wird im Kontext der Synchronisationsbefehle erklärt.

3.8.5. Synchronisationsinstruktionen

Die Synchronisation beschränkt sich im Rahmen der Projektgruppe im Wesentlichen darauf, dass der trocknende Roboter mit der Abarbeitung seines Programms wartet, bis der sprühende Roboter sein Programm abgefahren hat. Alle Roboter, deren Bewegungen synchronisiert ablaufen sollen, müssen in ihrem Programmmodul eine Liste mit allen zu synchronisierenden Tasks aufweisen. Da bei *RoSi3D* genau zwei Roboter verwendet werden, könnte diese Liste zum Beispiel so aussehen:

```
PERS tasks tasklist1{2} := [["T_ROB1"],["T_ROB2"]];
```

Außerdem müssen für jeden Synchronisationspunkt Synchronsisationsbezeichner definiert werden. Dies geschieht auf die gleiche Art und Weise wie bei anderen Variablen. Der zugehörige Datentyp heißt `syncident`. Eine Definition könnte dann wie folgt aussehen, wobei `sync0` der Name des Synchronsisationsbezeichners ist:

```
VAR syncident sync0;
```

Für die eigentliche Synchronisation muss in jedem Roboterprogramm eine Warteinstruktion eingefügt werden. Alle Roboter arbeiten ihr Programm dann bis zu der Stelle ab, an der der Wartebefehl steht. Wenn alle Programme an dieser Stelle angekommen sind, wird die Abarbeitung fortgesetzt. Das folgende Beispiel zeigt die Syntax des Befehls:

```
WaitSyncTask \InPos,sync0,tasklist1;
```

`sync0` ist hierbei der oben definierte Synchronsisationsbezeichner und `tasklist1` die Liste der zu synchronisierenden Tasks.

4. Roboter

In diesem Kapitel wird zunächst auf Grundlagen und Definitionen eingegangen, die für das Verständnis des Kapitels notwendig sind. Des Weiteren werden die, im Rahmen der PG verwendeten Roboter, vorgestellt und miteinander verglichen. Anschließend wird ein Verfahren vorgestellt, welches das Problem der inversen Kinematik löst. Die genauen Berechnungsschritte dieses Verfahrens erfolgen am Ende dieses Kapitels.

4.1. Einleitung

Nach der VDI Richtlinie 2860 [VDI08] sind Industrieroboter universell einsetzbare Bewegungsautomaten mit mehreren Achsen, deren Bewegungen hinsichtlich Bewegungsfolge und Wegen bzw. Winkeln frei programmierbar (d.h. ohne mechanischen Eingriff vorzunehmen) und gegebenenfalls sensorgeführt sind. Sie sind mit Greifern, Werkzeugen oder anderen Fertigungsmitteln ausrüstbar und können Handhabungs- oder andere Fertigungsaufgaben ausführen. Um eine funktionsfähige Steuerung zu entwickeln und effektiv einzusetzen, stehen Lagebeschreibung, Bewegungssteuerung, Programmierung, Beschreibung der Dynamik und Bewegungsregelung im Vordergrund. Heutige Industrieroboter sind positionsgeregelte Handhabungsgeräte, die präzise eine vorgegebene Bahn im Raum abfahren. Dadurch werden bei vielen Fertigungsaufgaben, die mit Hilfe des Roboters automatisiert werden sollen, hohe Anforderungen an die Genauigkeit der Zuführung und Positionierung von Objekten gestellt. Die Projektgruppe 523 befasst sich mit der Programmierung und Simulation zweier kooperierend arbeitender Industrieroboter bei einem zweiteiligen Arbeitsprozess (Lackieren und Trocknen). Zur Bewältigung dieser Aufgabe sind effiziente und bahngenaue Roboter notwendig.

In der Automobilindustrie werden schon länger kooperierende Roboter eingesetzt. Beispielsweise hat Daimler-Chrysler mit dem Projekt Kooperierende Roboter [Rob08] erfolgreich die Implementierung für eine Punktschweißaufgabe unternommen. Dabei wird das sogenannte Master-Slave-Verfahren eingesetzt. Ein Roboter (der Master) gilt als Hauptroboter, an dem sich die anderen Roboter orientieren. Solch ein Verfahren wird als roboterzentrierter Lösungsansatz bezeichnet.

Bei bahnorientierten Aufgaben, wie Bahnschweißen, Klebeauftrag oder Nahtabdichten wird ein Verfahren eingesetzt, welches prozesszentriert ist. Hierbei wird die Bearbei-

tungsaufgabe über das Werkstück definiert und die Geometrie- und Technologiedaten werden roboterunabhängig festgelegt [Rob08].

4.2. Grundlagen

Im Folgenden werden Definitionen und Begriffe eingeführt, die für das weitere Verständnis des Kapitels notwendig sind.

4.2.1. Effektor

Der Effektor (siehe Abbildung 4.1) kann ein Greifer, ein Schraubendreher, eine Sprühpistole oder ein anderes Bearbeitungswerkzeug sein, das an der letzen Achse eines Industrieroboters, an dem sogenannten Flansch, angekoppelt werden kann, um Werkstücke aufzunehmen oder zu bearbeiten. Ein charakteristischer Punkt des Effektors ist der Tool Center Point oder auch TCP genannt, welcher an geeigneter Stelle am Werkzeug, beispielsweise an dessen Spitze, gewählt wird. Von dieser Stelle aus werden die Werkzeugskoordinaten angegeben. Ein Industrieroboter hat die Aufgabe, einen Effektor geeignet entlang einer Bewegungsbahn zu führen.

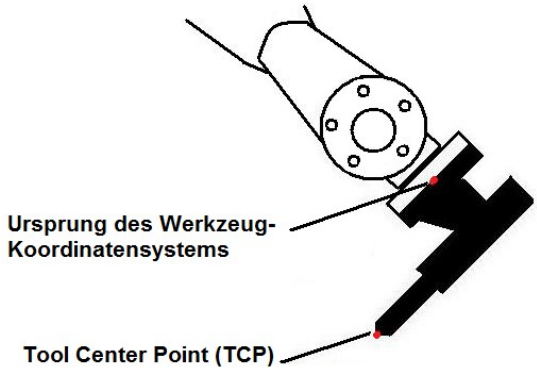


Abbildung 4.1.: Der Effektor

4.2.2. Freiheitsgrad

Unter dem Begriff des Freiheitsgrades ist die die Anzahl der möglichen unabhängigen Bewegungen eines Körpers gegenüber eines Bezugssystems zu verstehen. Ein Körper, der sich im Raum frei bewegen kann, hat maximal den Freiheitsgrad $F = 6$. F entspricht der Anzahl der Angaben, die die Lage eines Körpers im Raum vollständig beschreibt.

Die Lage, auch Stellung oder Pose genannt, eines Roboters kann durch drei Positions- und drei Drehkwinkelangaben, bezogen auf ein Bezugskoordinatensystem, festgelegt. So werden für die Orientierung eines sechs-achsigen Roboters im Raum Drehgelenke benötigt, da Lineargelenke die Orientierung nicht ändern können. Für einen Freiheitsgrad $F > 3$ muss es also mindestens ein Drehgelenk geben.

4.2.3. Arbeitsraum

Der Arbeitsraum (siehe Abbildung 4.2) ist derjenige Raumbereich, der mit dem Roboterflansch angefahren werden kann.

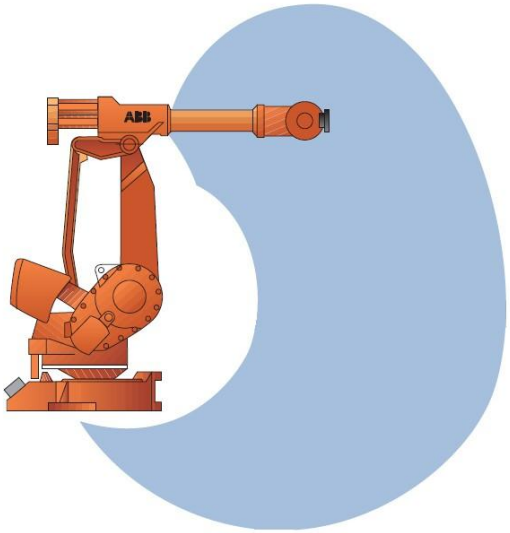


Abbildung 4.2.: Der Arbeitsbereich von IRB 4400 [ABB08a]

4.2.4. Singularität

Eine Roboter-Singularität kann als eine Stellung im Roboter-Arbeitsbereich definiert werden, wo zwei oder mehrere Gelenke keine unabhängige Kontrolle der Position und Orientierung des Werkzeuges mehr haben. Diese tritt auf wenn Roboterachsen redundant sind, d.h. wenn mehr Achsen als nötig verwendet werden, um die gleiche Bewegung auszuführen.

Die schematische Darstellung in Abbildung 4.3 stellt eine Singularität dar. In diesem Beispiel sind die letzten drei Gelenkwinkel eines ABB Roboters beteiligt. So wenn der Winkel der Achse fünf gleich Null ist, gibt es für die Achsen vier und sechs unendlich viele Lösungen, die alle zur gleichen Stellung des Effektors führen.

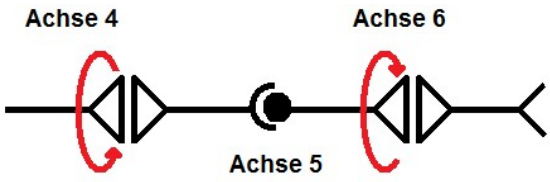


Abbildung 4.3.: Singularität [Yos90]

4.3. Robotermodelle

In diesem Abschnitt werden die von *RoSi3D* unterstützten Robotermodelle (ABB IRB 2400/16, ABB IRB 4400/60 und ABB IRB 6620) vorgestellt. Die technischen Angaben stammen aus [ABB08a].

4.3.1. Roboter im Vergleich

Die von Rosi3D verwendeten Roboter lassen sich in zwei Klassen einteilen: Zu der ersten Klasse gehören die Robotertypen IRB 2400 und IRB 4400 (siehe Abbildung 4.4). Die beiden Roboter verfügen über einen sogenannten Backlink, dieser dient als kinematische Stabilisierung, die den Arbeitsraum des Roboters auf den vorderen Bereich begrenzt . Das Einsatzgebiet dieser Roboter umfasst vor allem Aufgaben, bei denen es darauf ankommt, einen Arbeitsprozess mit großer Präzision zu erledigen. Sie verfügen über eine relativ geringe Handhabungskapazität von max. 16kg bzw. 60kg. Sie sind aber schnell (150°/s für die erste Achse) und haben eine Positionswiederholgenauigkeit von $0,06mm$ bzw. $0,07mm$.

Zu der zweiten Klasse gehört der IRB 6620 (siehe Abbildung 4.5), dieser hat eine größere Handhabungskapazität von max. 150kg, und findet daher bei schwerer Materialhandhabung mehr Einsatz. Im Vergleich zu den beiden ersten Robotern verfügt der IRB 6620 über einen deutlich größeren Arbeitsbereich durch den fehlenden Backlink.

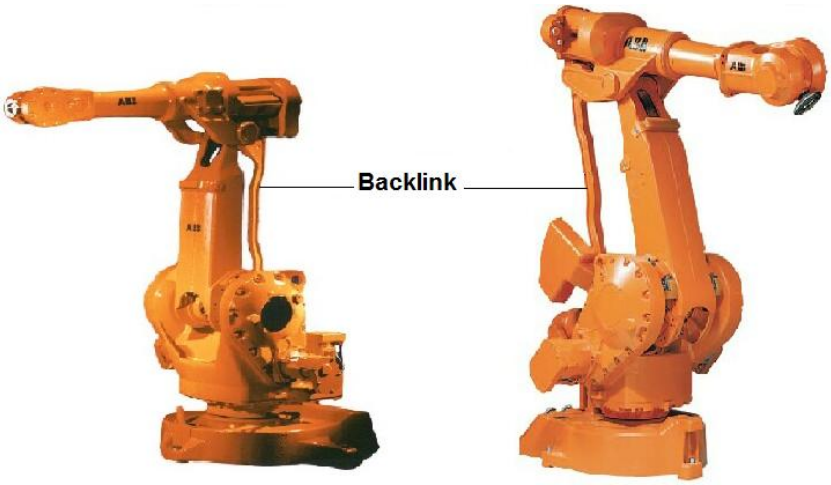


Abbildung 4.4.: ABB IRB 2400 und ABB IRB 4400 [ABB08b]

Abbildung 4.5.: ABB IRB 6620 [ABB08c]

4.3.2. Technische Daten

Roboterversion	Handhabungskapazität	Reichweite
IRB 2400/16	16kg	1.5m
IRB 4400/60	60kg	1.96m
IRB 6620	150kg	2.2m

Tabelle 4.1.: Handhabungskapazität und Reichweite [ABB08a]

Arbeitsbereich	IRB 2400/16	IRB 4400/60	IRB 6620
Achse 1	+180° bis -180°	+165° bis -165°	+170° bis -170°
Achse 2	+110° bis -100°	+95° bis -70°	+140° bis -65°
Achse 3	+65° bis -60°	+65° bis -60°	+70° bis -180°
Achse 4	+200° bis -200°	+200° bis -200°	+300° bis -300°
Achse 5	+120° bis -120°	+120° bis -120°	+130° bis -130°
Achse 6	+400° bis -400°	+400° bis -400°	+360° bis -360°

Tabelle 4.2.: Arbeitsbereich der Robotermodelle [ABB08a]

4.4. Inverse Kinematik

Die Aufgabe der inversen Kinematik besteht darin, anhand einer vorgegeben Position und Orientierung des TCPs die entsprechenden Gelenkkoordinaten zu berechnen, die aufgrund der Bewegungsfreiheit des Roboters nicht eindeutig bestimmt sind. Der Lage des Effektors in Weltkoordinaten können je nach Freiheitsgrad des Roboters, mehrere Lösungen für die Gelenkwinkel entsprechen. Für die von *RoSi3D* unterstützten Roboter mit einem Freiheitsgrad $F = 6$ gibt es, wenn man die Lösungen, die sich durch die vielfachen Drehungen der Gelenke ergeben vernachläßigt, acht relevante Lösungen. In diesem Fall muss die Steuerung, die die Bewegung und Aktionen des Industrieroboters überwacht, sich für eine Lösung entscheiden. Zur Lösung des inversen kinematischen Problems gibt es drei Lösungsvarianten:

- Geometrisches Lösungsverfahren
- Numerisches Lösungsverfahren
- Kombination aus geometrischem und numerischem Lösungsverfahren

Das verwendete Lösungsverfahren hängt von der kinematischen Struktur des Roboters ab, wobei mit zunehmender Achsenanzahl des Roboters, die Berechnung der Gelenkwinkel aufwendiger wird. Für die im Rahmen der Projektgruppe verwendeten Roboter (ABB 2400/16, ABB 4400/60 und ABB 6620) hat sich die Projektpruppe 523 für das geometrische Lösungsverfahren entschieden, da es im Vergleich zu den anderen Verfahren alle exakten Lösungen liefert.

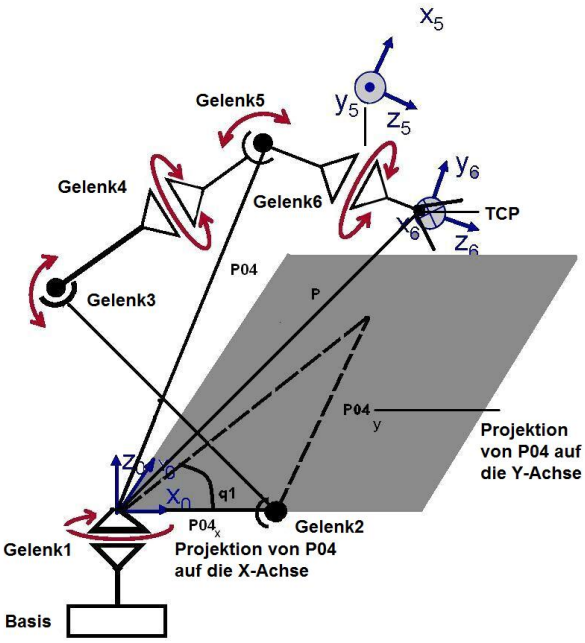
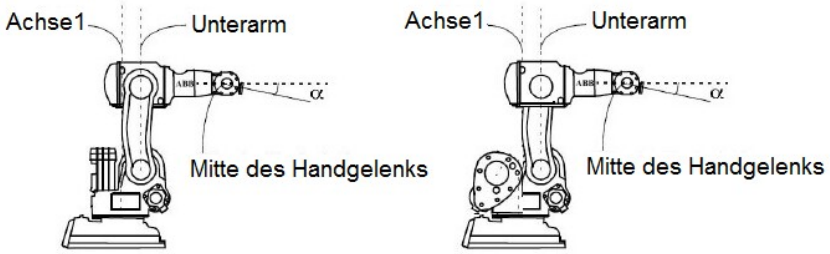


Abbildung 4.6.: Schematische Darstellung des geometrischen Lösungsverfahrens [Web02]

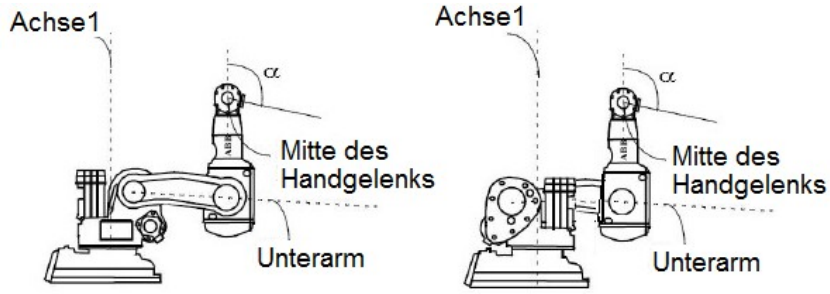
4.4.1. Roboterkonfigurationen

Roboterkonfigurationen beschreiben eine Klasse möglicher Lösungen, welche verschiedene Roboterstellungen definieren, die ein Roboter annehmen kann um eine Position anzufahren.

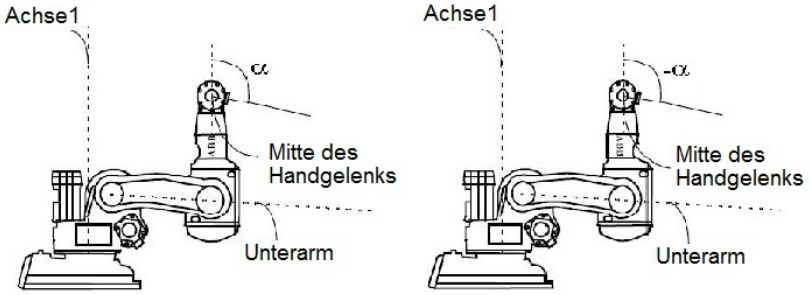
Für die oben vorgestellten Robotermodelle mit einem maximalen Freiheitsgrad $F = 6$ gibt es acht Möglichkeiten (siehe Abbildung 4.7) einen vorgegebenen TCP anzufahren. Liegt eine Singularität vor, führt dies dazu, dass mindestens ein Freiheitsgrad verloren geht, und die Bewegung des Effektors eingschränkt wird. Der Effektor kann sich bei einer Singularität nicht mehr in allen Richtungen des Arbeitsraumes bewegen, und bleibt daher immer auf die aktuell gültige Konfiguration beschränkt. Dieser ist der Fall, da die von Rosi3D unterstützten sechs-achsigen Roboter nicht durch eine lineare Bewegung aus der Singularität können. Zur Beschreibung der Konfigurationen wurden die von ADEPT-Robotern [ADE08] verwendeten Notationen benutzt. Die Abbildungen von 4.7(a) bis 4.7(d) veranschaulichen, wie derselbe TCP mit den acht verschiedenen Konfigurationen erreicht werden kann.



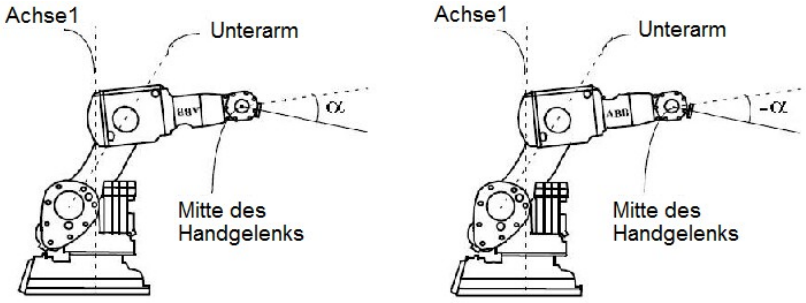
(a) Beispiel der Roboterkonfigurationen Lefty-above und Righty-above [ABB07]



(b) Beispiel der Roboterkonfigurationen Lefty-below und Righty-below [ABB07]



(c) Beispiel der Roboterkonfigurationen Lefty-flip und Lefty-noflip [ABB07]



(d) Beispiel der Roboterkonfigurationen Righty-flip und Righty-noflip [ABB07]

Abbildung 4.7.: Roboterkonfigurationen

4.4.2. ABB Konfigurationen

Bei der ABB Konfiguration handelt es sich um eine Notation, die den Wertebereich, in dem die Roboterachsen liegen bzw. liegen sollen (siehe [ABB07]) beschreibt. Die Roboterkonfiguration wird mit vier Achsenwerten angegeben, um eine der möglichen Konfigurationen eindeutig festzulegen.

Für eine Rotationsachse definiert der entsprechende Konfigurationswert den aktuellen Quadranten der Roboterachse (siehe Abbildung 4.8). Die Nummer des Quadranten gibt an, in welchem Wertebereich sich der Gelenkwinkel befindet. Für jede Achse ist Quadrant 0 die erste Viertelumdrehung, 0 bis 90° in positiver Richtung ab der Nullposition; Quadrant 1 ist die nächste Umdrehung, 90 bis 180° usw. Quadrant -1 ist die Umdrehung 0° bis (-90°) usw.

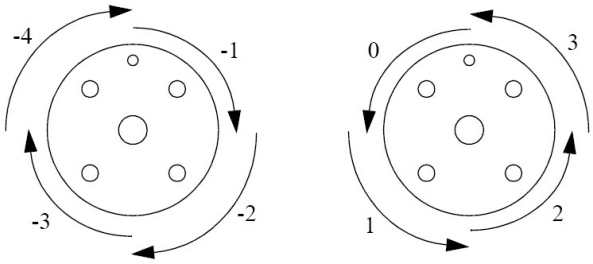


Abbildung 4.8.: Die ABB Konfigutationsquadranten [ABB07]

Für eine lineare Achse (siehe auch 4.9) definiert der Wert ein Intervall von einem Meter für die Roboterachse. Für jede Achse bedeutet der Wert 0 eine Position zwischen 0 und 1 Meter; 1 bedeutet eine Position zwischen 1 und 2 Metern. Ein negativer Wert von -1 bedeutet eine Position zwischen -1 und 0 Metern usw.

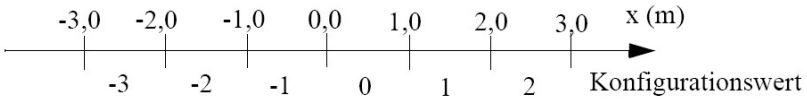


Abbildung 4.9.: Konfigurationswerte für eine lineare Achse [ABB07]

Für die oben vorgestellten Roboter (siehe Kapitel 4.3) werden nur drei Konfigurationsparameter verwendet, um die Konfiguration des Roboters eindeutig festzulegen. Diese Konfiguratinsparameter, bezeichnet mit cf1, cf4 und cf6 bestimmen den Wertebereich der drei Rotationsachsen $q1$, $q4$ und $q6$, welche die Winkel der Achsen eins, vier und sechs definieren. Allgemein werden maximal vier Parameter benötigt, um die Konfiguration für alle von Rosi3D verwendeten Roboter eindeutig zu bestimmen. Daher wird

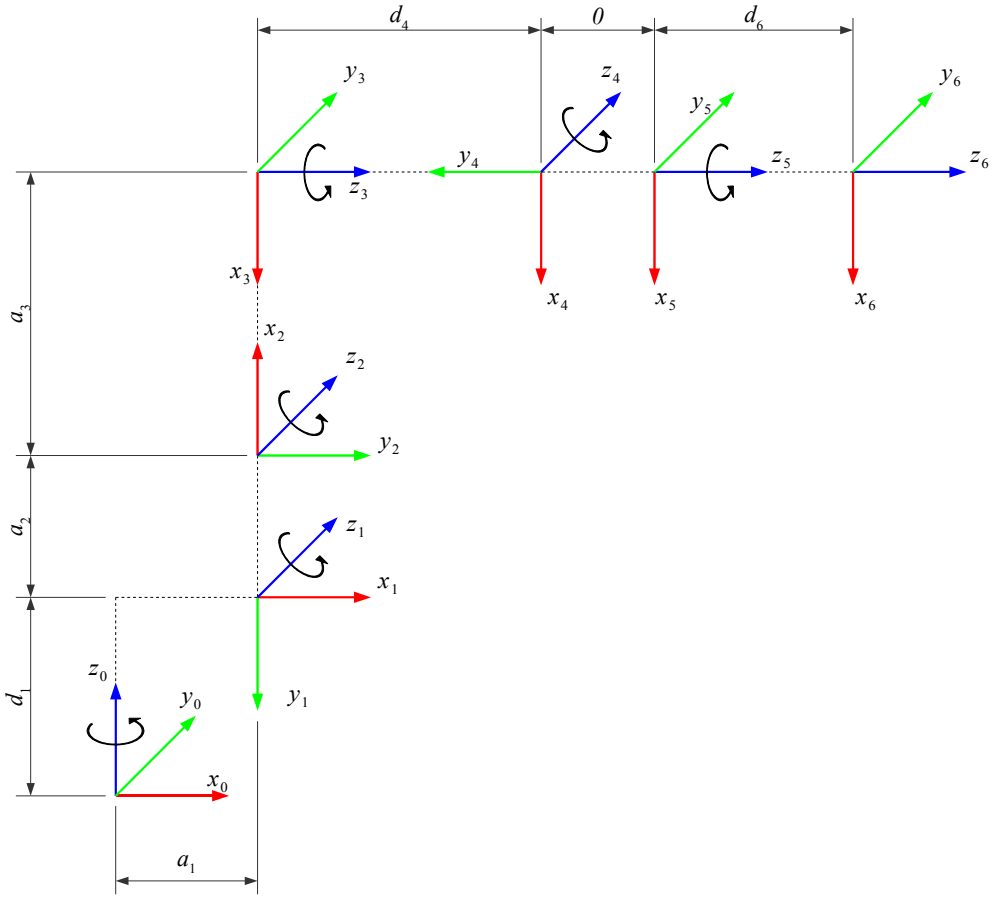


Abbildung 4.10.: Schematische Darstellung der kinematischen Kette für Sechs-Achs-Knickarmroboter mit Zentralhand für verschiedene Achsabstände.

in unserem Fall der vierte unbenutzte Parameter in der Konfiguration konstant mit 0 belegt. Die Konfiguration hätte dann die folgende Form: [cf1, cf4, cf6, 0].

4.4.3. Berechnung

Als Grundlage für die hier durchgeführte Berechnung der inversen Kinematik dient die kinematische Kette für Sechs-Achs-Knickarmroboter mit Zentralhand aus Abbildung 4.10. Die kinematische Kette, bestehend aus den Transformationen $\mathbf{T}_0, \ldots, \mathbf{T}_6$, abstrahiert von den tatsächlichen Robotermodellen und beschreibt die Orientierung und Position der einzelnen Gelenkachsen untereinander. Die Transformation $\mathbf{T}_0$ ist dabei die Basis des Roboters. Die durch die Transformationen beschriebenen Koordinatensysteme sind entsprechend der Denavit-Hartenberg-Konvention [DH55] so angeordnet, dass ihre

Z-Achse jeweils im entsprechenden Rotationsgelenk liegt. Das letzte Koordinatensystem beschreibt das Zentrum des Flansches des Roboters, an dem der Endeffektor montiert werden kann. Da es innerhalb der Berechnung nötig wird, einzelne Transformationen relativ zueinander anzugeben, wird zur Unterscheidung folgende Notation verwendet: $\mathbf{T}_{i:j}$ ist die Transformation i relativ zum Koordinatensystem j. Die Transformationen $\mathbf{T}_{i:i-1}$, $i = 1 \ldots 6$ werden anhand von Denavit-Hartenberg-Parametern, kurz DH-Parameter, $(\theta_i, d_i, \alpha_i, a_i)$ beschrieben.

- θ_i beschreibt die Rotation R_{θ_i} um die Gelenkachse z_{i-1},
- d_i die Translation $\mathbf{T}_{d_i}$ entlang der Gelenkachse z_{i-1},
- α_i die Rotation R_{θ_i} um die z_i- Achse, $z_i = (R_{\theta_i}\mathbf{T}_{d_i}) \cdot z_{i-1}$ und
- a_i die Translation $\mathbf{T}_{a_i}$ um die z_i- Achse

Damit ist

$$\mathbf{T}_{i:i-1} = R_{\theta_i} \cdot \mathbf{T}_{d_i} \cdot R_{\theta_i} \cdot \mathbf{T}_{a_i} \tag{4.1}$$

Die DH-Parameter für die Transformationen der in Abb. 4.10 abgebildeten kinematischen Kette sind in Tabelle 4.3 aufgelistet.

θ_i	d_i	α_i	a_i
$0°$	d_1	$-90°$	a_1
$-90°$	0	$0°$	a_2
$180°$	0	$90°$	$-a_3$
$0°$	d_4	$-90°$	0
$0°$	0	$90°$	0
$0°$	$d6$	$0°$	0

Tabelle 4.3.: DH-Parameter zu Abbildung 4.10

Allgemein ist ein Roboter und damit seine kinematische Kette Teil eines komplexeren Systems, welches bezüglich eines Welt-Koordinatensystems, kurz WKS, beschrieben wird. Seine Basis $\mathbf{T}_0$ ist nicht zwingend identisch mit dem Ursprung des WKS und er besitzt meist einen Endeffektor, dessen TCP nicht im Flansch des Roboters liegt. Dennoch soll im Folgenden die inverse Kinematik für beliebig positionierte Roboter mit verschiedensten Endeffektoren berechnet werden.
Gegeben sei die kinematische Kette in Form von Transformationen $\mathbf{T}_{i:i-1}$ und eine Transformation $\mathbf{T}_{TCP:WKS}$, welche die Position und Orientierung des Roboter-TCP im WKS beschreibt. Zudem sind $\mathbf{T}_{TCP:6}$, die Transformation von Flansch zum TCP, und $\mathbf{T}_{0:WKS}$ die Referenztransformation bezüglich des Weltkoordinatensystems bekannt.
Zur Lösung des inversen Kinematik-Problems, muss folgendes nichtlineares Gleichungs-

system nach θ_i, $i = 1, \ldots, 6$ gelöst werden:

$$\mathbf{T}_{TCP:WKS} = \mathbf{T}_{0:WKS} \cdot \mathbf{T}_{1:0} \cdot \mathbf{T}_{2:1} \cdot \mathbf{T}_{3:2} \cdot \mathbf{T}_{4:3} \cdot \mathbf{T}_{5:4} \cdot \mathbf{T}_{6:5} \cdot \mathbf{T}_{TCP:6} \tag{4.2}$$

Die übrigen Variablen α_i, a_i und d_i sind fest, wobei letztere zwei abhängig vom Robotermodell festgelegt werden. Es ist leicht zu sehen, dass dies identisch ist zur Lösung des Gleichungssystems

$$\mathbf{T}_{0:WKS}^{-1} \cdot \mathbf{T}_{TCP:WKS} \cdot \mathbf{T}_{TCP:6}^{-1} = \mathbf{T}_{1:0} \cdot \mathbf{T}_{2:1} \cdot \mathbf{T}_{3:2} \cdot \mathbf{T}_{4:3} \cdot \mathbf{T}_{5:4} \cdot \mathbf{T}_{6:5} \tag{4.3}$$

Es ist also gerechtfertigt anzunehmen, dass anstatt der Transformation $\mathbf{T}_{TCP:WKS}$ die den TCP relativ zum WKS beschreibt, die Transformation

$$\mathbf{T}_{6:0} = \mathbf{T}_{0:WKS}^{-1} \cdot \mathbf{T}_{TCP:WKS} \cdot \mathbf{T}_{TCP:6}^{-1} \tag{4.4}$$

des Flansch-Koordinatensystems relativ zur Roboterbasis gegeben ist. Wie zu allen anderen Transformationen auch lassen sich aus $\mathbf{T}_{6:0}$ ihre Position $\mathbf{p}_{6:0}$, sowie ihre Basis-Richtungsvektoren $\mathbf{x}_{6:0}$, $\mathbf{y}_{6:0}$ und $\mathbf{z}_{6:0}$ relativ zur Roboterbasis bestimmen. Die Notation ist hier analog zur Notation für Transformationen zu verstehen.

Die Position $\mathbf{p}$ und Richtungsvektoren $\mathbf{x}$, $\mathbf{y}$ und $\mathbf{z}$ des durch eine Transformation $\mathbf{T}$ beschriebenen Koordinatensystems, können allgemein wie folgt berechnet werden:

$$\mathbf{p} = \mathbf{T} \cdot \mathbf{0}, \mathbf{x} = \mathbf{T} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} - \mathbf{p}, \mathbf{y} = \mathbf{T} \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} - \mathbf{p}, \mathbf{z} = \mathbf{T} \cdot \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} - \mathbf{p} \tag{4.5}$$

Im Falle von 4x4-Matrizen als Darstellungsform für Transformationen in Verbindung mit homogenen Koordinaten können Richtungsvektoren und Position an den Spalten der Matrix direkt abgelesen werden. Die erste Spalte liefert die X-, die zweite die Y-, die dritte die Z-Achse und die vierte die Position.

Hierauf aufbauend wird unter der Verwendung der geometrischen Gegebenheiten der kinematischen Kette das Gleichungssystem explizit gelöst.

Die durch die Kette beschriebenen Roboter haben eine Zentralhand, ihre letzten drei Rotationsachsen schneiden sich in einem Punkt. Das bedeutet, dass die ersten drei Achswinkel die Position und die letzten drei die Orientierung des Flansch-Koordinatensystems bestimmen. In dieser Reihenfolge werden im Folgenden exemplarisch einzelne Teillösungen berechnet mit Hilfe derer sich weitere Lösungen bestimmen lassen.

Wie in 4.4.2 besprochen, können Sechs-Achs-Knickarmroboter einen bestimmten TCP in verschiedenen Konfigurationen erreichen. Maximal sind dies diejenigen acht Konfigurationen, die entsprechend der KUKA-Bezeichnung, durch Kombination von Werten

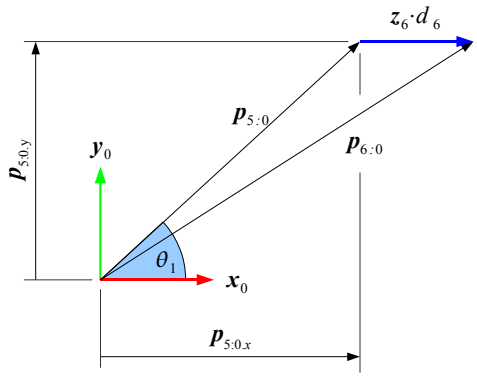


Abbildung 4.11.: Skizze zur Berechnung von θ_1

aus $\{righty, above, flip\}$ und ihren Negationen entstehen. Innerhalb einer Konfiguration gibt es zudem theoretisch abzählbar unendlich viele Lösungen, die durch Summation von Vielfachen von 360° auf einen festen Satz Achswinkel entstehen.

In der Praxis jedoch wird der Lösungsraum durch die Achswinkelbegrenzung des jeweiligen Roboters beschränkt. Somit ergibt sich für einen bestimmten Roboter die Lösungsmenge als Schnitt des Definitionsbereichs für die Achswinkel mit der Menge der theoretisch möglichen Lösungen. Falls eine bestimmte Lösung gefunden wurde, ist es möglich diese Lösung durch Addition von Vielfachen von 360° in den Bereich der Achswinkelbegrenzungen zu bringen. Lässt sich dies nicht durchführen, dann ist die theoretische Lösung keine praktisch realisierbare.

Berechnung von θ_1: Zur Bestimmung des ersten Winkels θ_1 wird $\mathbf{p}_{4:0}$ benötigt, welches aus

$$\mathbf{T}_{4:0} = \mathbf{T}_{6:0} \cdot (\mathbf{T}_{5:4} \cdot \mathbf{T}_{6:5})^{-1} \tag{4.6}$$

abgeleitet werden kann.

Der erste Winkel ist gleich dem Winkel, der von $\mathbf{x}_0$ und dem Vektor $\mathbf{p}_{4:0}$ projiziert auf die Ebene durch $\mathbf{x}_0$ und $\mathbf{y}_0$ eingeschlossen wird (siehe Abbildung 4.11). Falls Achse vier und Achse drei bzw. fünf gleich sind, dann existieren überabzählbar unendlich viele Lösungen für θ_1, θ_5 und θ_6.

$$\theta_1 = \begin{cases} atan2(\mathbf{p}_{4:0.y}, \mathbf{p}_{4:0.x}) & \text{,lefty} \\ atan2(\mathbf{p}_{4:0.y}, \mathbf{p}_{4:0.x}) + 180° & \text{,righty} \end{cases} \tag{4.7}$$

Berechnung von θ_2 und θ_3: Genau wie θ_1 ergeben sich die Winkel θ_2 und θ_3 allein aus der Position des vierten Koordinatensystems $\mathbf{p}_{4:1} = \mathbf{p}_{4:0} - \mathbf{p}_1$. Unter Zuhilfenahme von

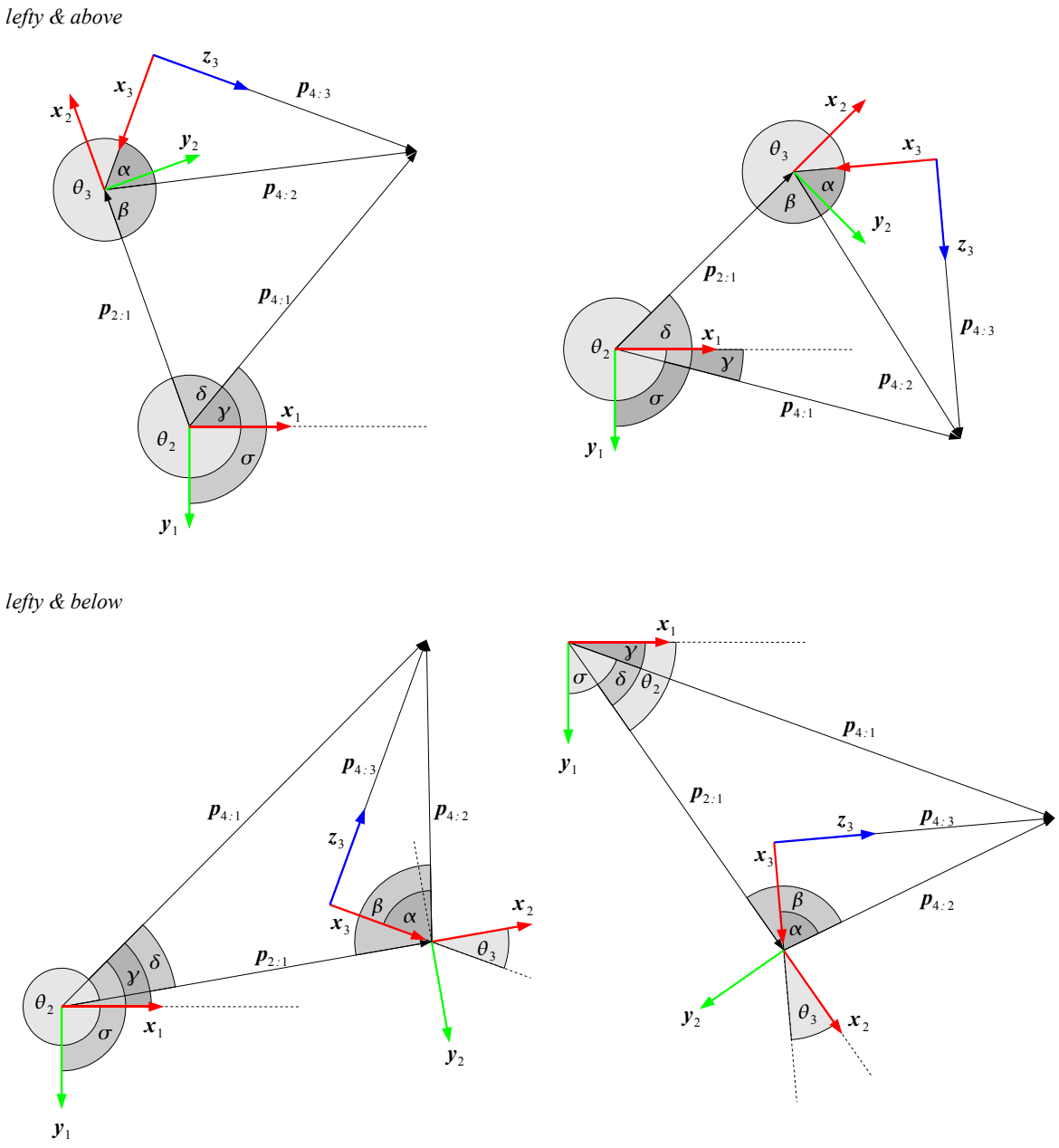
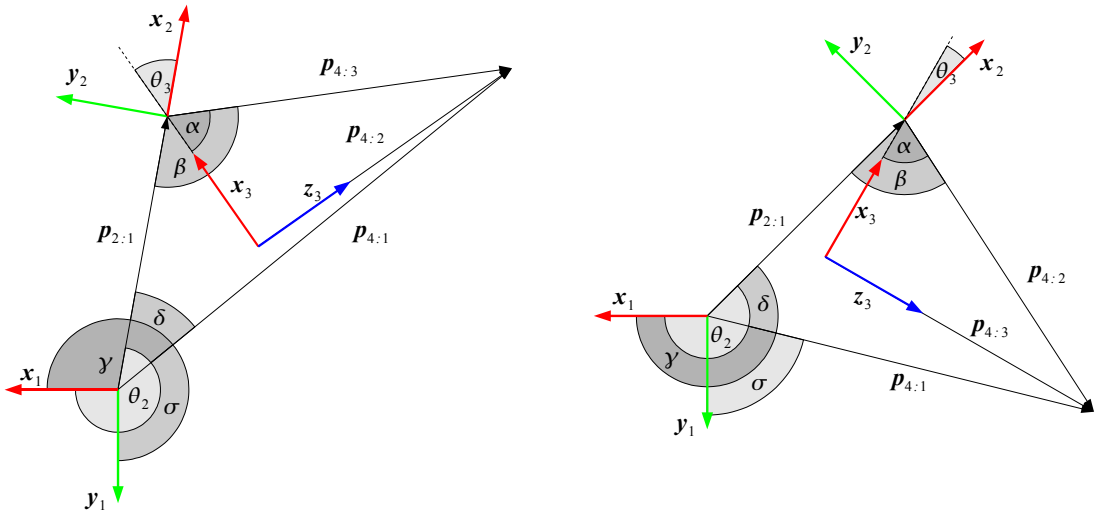


Abbildung 4.12.: Skizze zur Berechnung von θ_2 und θ_3 für „lefty“-Konfigurationen

righty & above



righty & below

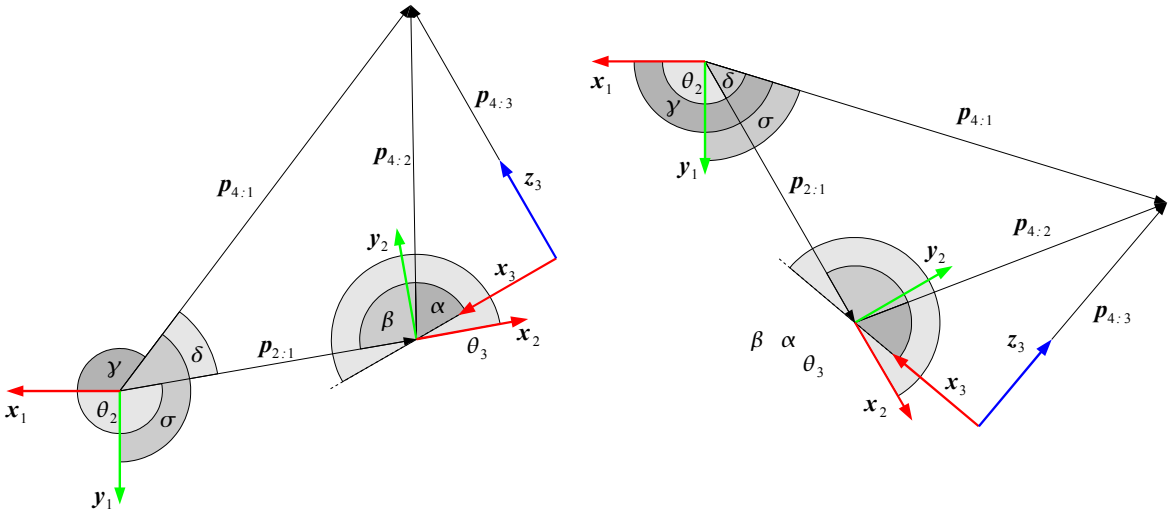


Abbildung 4.13.: Skizze zur Berechnung von θ_2 und θ_3 für „righty“-Konfigurationen

Abb. 4.12 und Abb. 4.13 ist ersichtlich, dass

$$\theta_2 = \begin{cases} 360° - \delta - \gamma & ,\text{lefty} \wedge \text{above} \wedge \sigma > 90° \\ 360° - \delta + \gamma & ,\text{lefty} \wedge \text{above} \wedge \sigma \leq 90° \\ 360° - \gamma + \delta & ,\text{lefty} \wedge \text{below} \wedge \sigma > 90° \\ \gamma + \delta & ,\text{lefty} \wedge \text{below} \wedge \sigma \leq 90° \\ 360° + \delta - \gamma & ,\text{righty} \wedge \text{above} \wedge \sigma > 90° \\ \delta + \gamma & ,\text{righty} \wedge \text{above} \wedge \sigma \leq 90° \\ 360° - \delta - \gamma & ,\text{righty} \wedge \text{below} \wedge \sigma > 90° \\ \gamma - \delta & ,\text{righty} \wedge \text{below} \wedge \sigma \leq 90° \end{cases} \tag{4.8}$$

wobei die beiden Hilfswinkel δ, γ und σ wie folgt berechnet werden können:

$$\delta = acos\left(\frac{|\mathbf{p}_{2:1}|^2 + |\mathbf{p}_{4:1}|^2 - |\mathbf{p}_{4:2}|^2}{2 \cdot |\mathbf{p}_{2:1}| \cdot |\mathbf{p}_{4:1}|}\right) \tag{4.9}$$

$$\gamma = acos\left(\frac{\mathbf{p}_{4:1} \cdot \mathbf{x}_1}{|\mathbf{p}_{4:1}|}\right) \tag{4.10}$$

$$\sigma = acos\left(\frac{\mathbf{p}_{4:1} \cdot \mathbf{y}_1}{|\mathbf{p}_{4:1}|}\right) \tag{4.11}$$

Da die Berechnung von θ_3 nur abhängig von der Art der Konfiguration variiert, fällt die Fallunterscheidung hierfür entsprechend kürzer aus:

$$\theta_3 = \begin{cases} 360° - \alpha - \beta & ,\text{lefty} \wedge \text{above} \\ \beta - \alpha & ,\text{lefty} \wedge \text{below} \\ \beta - \alpha & ,\text{righty} \wedge \text{above} \\ 360° - \beta - \alpha & ,\text{righty} \wedge \text{below} \end{cases} \tag{4.12}$$

Die Hilfswinkel α und β sind genau wie die Hilfswinkel zur Herleitung von θ_2 unabhängig von der Konfiguration:

$$\alpha = atan\left(\frac{|\mathbf{p}_{3:2}|}{|\mathbf{p}_{4:3}|}\right) \tag{4.13}$$

$$\beta = acos\left(\frac{|\mathbf{p}_{4:2}|^2 + |\mathbf{p}_{2:1}|^2 - |\mathbf{p}_{4:1}|^2}{2 \cdot |\mathbf{p}_{2:1}| \cdot |\mathbf{p}_{4:2}|}\right) \tag{4.14}$$

Bis zu dieser Stelle sind die zwei benötigten Werte $|\mathbf{p}_{4:2}|$ und $|\mathbf{p}_{2:1}|$ noch nicht bestimmt. Jedoch beim Vergleich von Abb. 4.12 oder 4.13 mit Abb. 4.10 wird klar, dass diese sich aus dem fixen Parameter a_2 für das Robotermodell sowie dem bereits berechneten Wert $|\mathbf{p}_{4:1}|$ herleiten lassen:

$$|\mathbf{p}_{2:1}| = a_2 \tag{4.15}$$

$$|\mathbf{p}_{4:2}| = \sqrt{a_2^2 + |\mathbf{p}_{4:1}|^2 - 2 \cdot a_2 \cdot |\mathbf{p}_{4:1}| \cdot cos\,(\delta)} \tag{4.16}$$

$$\delta = acos\left(\frac{\mathbf{p}_{4:1} \cdot \mathbf{p}_{2:1}}{|\mathbf{p}_{4:1}| \cdot |\mathbf{p}_{2:1}|}\right) \tag{4.17}$$

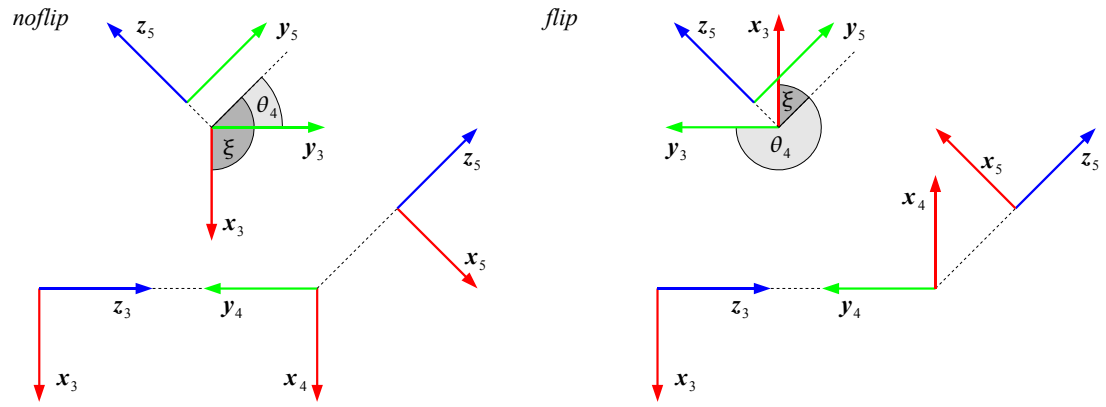


Abbildung 4.14.: Skizze zur Berechnung von θ_4

Da nun Werte für die ersten drei Winkel bekannt sind, kann die Position und Orientierung des Koordinatensystems zu Achse drei bestimmt werden, indem die ersten drei Transformationen anhand ihrer DH-Parameter aufgestellt und aufmultipliziert werden.

$$\mathbf{T}_{3:0} = \mathbf{T}_{1:0} \cdot \mathbf{T}_{2:1} \cdot \mathbf{T}_{3:2} \tag{4.18}$$

Hierauf aufbauend werden im weiteren Verlauf die letzten drei Winkel bestimmt.

Berechnung von θ_4: Der vierte Winkel wird aus dem Winkel hergeleitet, welcher durch x_3 und dem Vektor $\mathbf{v} = \mathbf{z}_5 \times \mathbf{z}_3$ senkrecht auf $\mathbf{z}_5 = \mathbf{z}_6$ und $\mathbf{z}_3$ eingeschlossen wird. Falls $\mathbf{v} = \mathbf{0}$, also z_5 und z_3 parallel sind, gibt es überabzählbar unendlich viele Lösungen für θ_4 und θ_6. Andernfalls lässt sich analog zu den vorherigen Winkeln ein Wert für θ_4 berechnen, der bis auf das Addieren von Vielfachen von 360° eindeutig ist. Wie in Abb. 4.14 zu erkennen, ist

$$\theta_4 = \begin{cases} acos\left(\mathbf{y}_3 \cdot (\mathbf{z}_5 \times \mathbf{z}_3)\right) & , \text{noflip} \wedge \xi \geq 90° \\ 360° - acos\left(\mathbf{y}_3 \cdot (\mathbf{z}_5 \times \mathbf{z}_3)\right) & , \text{noflip} \wedge \xi < 90° \\ 180° + acos\left(\mathbf{y}_3 \cdot (\mathbf{z}_5 \times \mathbf{z}_3)\right) & , \text{flip} \wedge \xi \geq 90° \\ 180° - acos\left(\mathbf{y}_3 \cdot (\mathbf{z}_5 \times \mathbf{z}_3)\right) & , \text{flip} \wedge \xi < 90° \end{cases}, \tag{4.19}$$

wobei für den Hilfswinkel ξ gilt:

$$\xi = acos\left(\mathbf{x}_3 \cdot (\mathbf{z}_5 \times \mathbf{z}_3)\right). \tag{4.20}$$

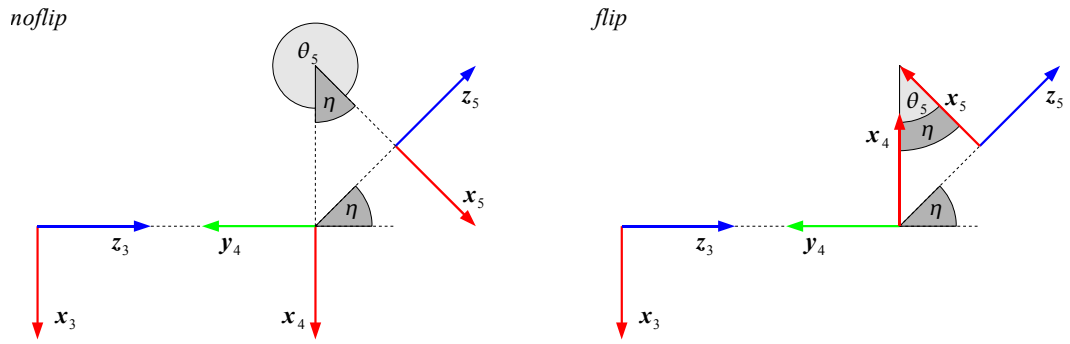


Abbildung 4.15.: Skizze zur Berechnung von θ_5

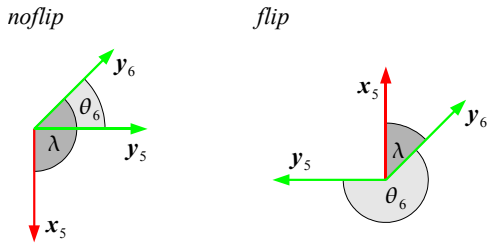


Abbildung 4.16.: Skizze zur Berechnung von θ_6

Berechnung von θ_5: (siehe Abbildung 4.15)

Der fünfte Winkel lässt sich, wie in Abb. 4.15 dargestellt, direkt ableiten aus dem von $\mathbf{z}_3$ und $\mathbf{z}_5 = \mathbf{z}_6$ eingeschlossenen Winkel η:

$$\theta_5 = \begin{cases} \eta & \text{flip} \\ 360° - \eta & \text{noflip} \end{cases} \tag{4.21}$$

Für η gilt:

$$\eta = acos\left(\mathbf{z}_3 \cdot \mathbf{z}_6\right). \tag{4.22}$$

Berechnung von θ_6 (siehe Abbildung 4.16)

Mit Hilfe von θ_4 und θ_5 kann zunächst die Transformation $\mathbf{T}_{5:0}$ berechnet werden.

$$\mathbf{T}_{5:0} = \mathbf{T}_{3:0} \cdot \mathbf{T}_{4:3} \cdot \mathbf{T}_{5:4} \tag{4.23}$$

Aus $\mathbf{T}_{5:0}$ können $\mathbf{x}_5$ und $\mathbf{y}_5$ hergeleitet und mit ihnen der Winkel θ_6 bestimmt werden (siehe Abbildung 4.16):

$$\theta_6 = \begin{cases} acos\left(\mathbf{y}_6 \cdot \mathbf{y}_5\right) & \lambda \geq 90° \\ 360° - acos\left(\mathbf{y}_6 \cdot \mathbf{y}_5\right) & \lambda < 90° \end{cases} \tag{4.24}$$

Für den Hilfswinkel λ gilt:

$$\lambda = acos\left(\mathbf{y}_6 \cdot \mathbf{x}_5\right). \tag{4.25}$$

Sobald eine Lösung für $\theta_1 \ldots \theta_6$ für eine bestimmte Konfiguration gefunden ist, lassen sich weitere Lösungen wie zuvor besprochen durch Aufaddieren von Vielfachen herleiten.

Nach der Berechnung der Inversen Kinematik war ein Testmodul notwendig, um die berechneten Winkel nach Korrektheit und Genauigkeit zu überprüfen. Zu diesem Zweck wurden anhand einer Testumgebung mehrere TCPs ausgegeben, die die Roboter angefahren haben. Die entsprechend von *RoSi3D* berechneten Winkel wurden in einem weiteren Schritt mit den Winkeln von *RobotStudio* verglichen, indem in *RobotStudio* die gleichen TCP-Positionen angefahren wurden. Erst nach Überseinstimmung der beiden Ergebnisse, wurde die Berechnung der Inversen Kinematik als korrekt angenommen.

5. Bahnplanung

Der Begriff Bahnplanung bezeichnet im Kontext der Projektgruppe die dreidimensionale, kollisionsfreie Bewegungsplanung für Industrieroboter. Der Begriff Bahn bezieht sich dabei auf den Pfad, auf dem sich der TCP (siehe Kapitel 4.2.1) des Roboters während der Roboterbewegung befindet. Dieser Pfad muss zwei wesentliche Bedingungen erfüllen: Zum einen müssen Kriterien und Nebenbedingungen des auszuführenden Arbeitsprozesses berücksichtigt werden, wie beispielsweise die Beschaffenheit des Werkstücks oder die Eigenschaften des Werkzeugs, zum anderen müssen alle Punkte des Pfades gültige Roboterpositionen ergeben. Gültige Positionen sind solche, die aufgrund der Kinematik des Roboters erreichbar sind und nicht zu Kollisionen führen. Bei der flächendeckenden Bahnplanung muss der Roboter zusätzlich noch eine spezifizierte Region vollständig abdecken bzw. bearbeiten. Vollständig bedeutet hierbei, dass jeder Punkt der Oberfläche in der Region durch das Werkzeug bearbeitet wird. Ein mögliches Anwendungsgebiet ist zum Beispiel das Schweißen oder Lackieren von Autoteilen in der Automobilindustrie.

Um die Kriterien und Nebenbedingungen des auszuführenden Arbeitsprozesses zu berücksichtigen und gültige Positionen zu gewährleisten, bieten sich mehrere Planungsansätze an. Diese lassen sich generell in zwei Klassen einteilen. In der ersten Klasse wird lediglich eine Planungsphase verwendet, in der gleichzeitig die Kriterien des Arbeitsprozesses und die gültigen Roboterpositionen berücksichtigt werden. Zur zweiten Klasse der Planungsansätze gehören Verfahren, die zwei getrennte Planungsphasen verwenden, wobei zunächst Pfade generiert werden, die das Werkstück vollständig abdecken, und erst in einer zweiten Planungsphase kollisionsfreie Roboterbewegungen über die zuvor erstellten Pfade gesucht werden.

Die Projektgruppe 523 hat sich für den zweiten Ansatz mit zwei separaten Planungsphasen entschieden. Dieses Kapitel befasst sich mit der Planung von Pfaden, die, noch unabhängig von den Roboterbewegungen, das Werkstück vollständig abdecken. In Kapitel 6 wird dann die Planung kollisionsfreier Roboterbewegungen, basierend auf den zuvor berechneten Pfaden, beschrieben.

Die automatische vollständige Oberflächenbearbeitung beliebig geformter, komplexer Werkstücke ist bisher wenig erforscht und es existieren nur wenige Systeme für diese Aufgabe, wie der in [BL07] vorgestellte Systementwurf zur automatischen Lackierung. Weiter verbreitet sind Verfahren zur Bearbeitung von regulären, flachen oder nur leicht gekrümmten Bauteilen. Zu dieser Gruppe gehören beispielsweise die in [ACG+04]

und [CXSC05a] vorgestellten Systeme, die speziell für die Lackierung von Karosserieteilen geeignet sind. Die Betrachtung flacher oder annähernd flacher Werkstücke, bzw. Werkstückteile, weist ebenfalls eine große Ähnlichkeit zu den Lösungsstrategien von Abdeckungsalgorithmen von mobilen Robotern auf, wie im Anwendungsfall von Reinigungsrobotern in [OCPZ04].

Der von der Projektgruppe verwendete Ansatz orientiert sich an der Arbeit von [CXSC05b]. Bei diesem wird zur Bearbeitung komplexer Geometrien eine gegebene Freiformfläche zunächst in Teilstücke, sogenannte Patches, unterteilt, die nur leicht gekrümmten Teilstücken des Werkstücks entsprechen. In [CXSC05a] wird ein Partitionierungsalgorithmus ausgehend von einem Dreiecksnetz beschrieben, der als Grundlage für die weitere Bahnplanung dient.

Wie bereits erwähnt müssen bei der Berechnung des Pfades Kriterien und Nebenbedingungen des auszuführenden Arbeitsprozesses und die Geometrie des Werkstücks berücksichtigt werden, um einen adäquaten Pfad für den jeweiligen Arbeitsprozess zu erhalten. Im konkreten Fall der Projektgruppe 523, bei dem ein Roboter das Werkstück mit Farbe besprühen und ein anderer dieses trocknen soll, sind übliche Optimierungskriterien, die bei der Bahnplanung berücksichtigt werden müssen, zum Beispiel ein gleichmäßiger Farbauftrag, eine Minimierung des Materialverlustes und Zeiteffizienz. In Bezug auf die kollisionsfreien Roboterbewegungen sind insbesondere Kollisionen mit dem Werkstück, zwischen den Robotern, mit dem Sprühstrahl und zwischen den Werkzeugen zu beachten.

Die Berechnung des Pfades wird auf den dem Werkstück zugehörigen Dreiecksnetzen durchgeführt(siehe Kapitel 2.2). Für die Pfadgenerierung wird das Werkstück anhand seiner Geometrie in mehrere Segmente, sogenannte Patches, eingeteilt. Die Einteilung in Patches erfolgt in drei Bearbeitungsschritten: Ein (optionales) Meanshift-Verfahren als Vorverarbeitungsschritt, ein Bereichswachstumsverfahren zur eigentlichen Segmentierung und ein Optimierungsverfahren. Diese Verfahren werden im Kapitel 5.1 genauer vorgestellt.

Nachdem das Werkstück in mehrere Patches eingeteilt wurde, wird für jeden dieser Patches eine eigene Bahn berechnet, die den Patch vollständig abdecken soll. Für einige Arbeitsprozesse, wie zum Beispiel das Aufsprühen von Farbe, ist es notwendig, dass das Werkzeug nicht direkt auf der Werkstückoberfläche ansetzt, sondern einen bestimmten Abstand, den sogenannten tool stand-off, einhält. Daher wird jeder Patch vor der eigentlichen Pfadberechnung so skaliert, dass der auf ihm geplante Pfad um den benötigten Abstand (tool stand-off) von der Werkstückoberfläche extrahiert ist. Abbildung 5.1 zeigt das Modell eines Werkstücks mit einem extrahierten Patch. Auf diesen skalierten Patches werden dann unter Anwendung des Bounding-Box-Algorithmus aus [CXM+04] Linearbahnen berechnet. Um eine optimale Abdeckung des Patches zu gewährleisten und um möglichst lange Bahnstücke zu erhalten, gehen in die Berechnung der Linearbahnen unter anderem der Werkzeugradius r_w und die Geometrie des Patches ein.

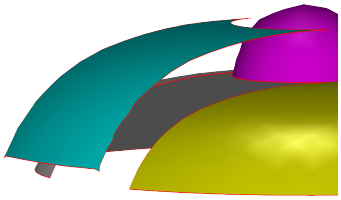


Abbildung 5.1.: Extrahierter Patch

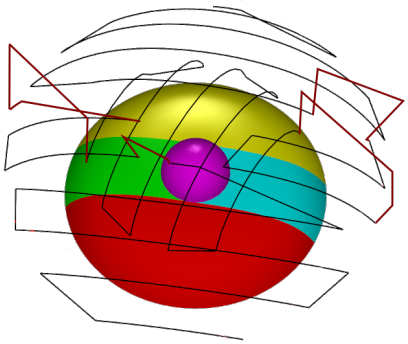


Abbildung 5.2.: Interpatchverbindungen über Werkstück

Nach erfolgreicher Berechnung der Bahnen auf den einzelnen Patches, müssen die Patches so miteinander verbunden werden, dass sich eine durchgängige Bahn über das gesamte Werkstück ergibt. Hierzu werden sogenannte Interpatchverbindungen berechnet, welche die Verbindung zwischen je zwei Patches herstellen. Um die einzelnen Patches zu verbinden, wurden mehrere Strategien implementiert. In Abbildung 5.2 ist ein Werkstück dargestellt, dass sowohl die Pfade auf den einzelnen Patches (schwarz), als auch die Interpatchverbindungen (rot) zeigt. Um die Bahnlänge klein zu halten, ist eine günstige Wahl der Reihenfolge, in der die Patches von den Robotern angefahren werden, notwendig. Das Festlegen der Reihenfolge, in der die Patches angefahren werden, gleicht im Wesentlichen dem Traveling-Salesman Problem (TSP). Die Pfadberechnung für einen Patch und die Pfadberechnung zwischen den Patches werden im Kapitel 5.2 genauer beschrieben.

5.1. Segmentierung

Polygonale Netze und insbesondere Dreiecksnetze werden in vielen Bereichen (zum Beispiel dem Ingenieurwesen und der Medizin) eingesetzt, um im Rahmen der Bildverarbeitung Oberflächen zu repräsentieren. Da es für viele Anwendungen nötig ist, die vorliegendenen Netze in semantisch zusammengehörige Segmente (bezüglich eines durch den jeweiligen Anwendungsfall gegebenen Kriteriums) zu unterteilen, gibt es bereits viele wissenschaftliche Arbeiten, die sich damit beschäftigen, diese Unterteilung zu automatisieren. Ein Vergleich einiger dieser Ansätze findet sich in [AKM+06], weitere interessante, in genannter Arbeit nicht berücksichtigte Verfahren sind Wasserscheiden-basierte Algorithmen (zum Beispiel beschrieben in [PKA03]), Meanshift-basierte Algorithmen (zum Beispiel beschrieben in [YLL+05]) und Bereichswachstumsverfahren (zum Beispiel beschrieben in [SWG+03]). Im Folgenden wird die Motivation, das Werkstück zu segmentieren, das dafür verwendete Homogenitätskriterium und die konkrete Vorgehensweise kurz erläutert; in den jeweiligen Unterkapiteln folgt eine detailliertere Darstellung der verwendeten Verfahren.

Der erste Ansatz der Projektgruppe 523 eine geeignete Segmentierung des Werkstücks zu erzeugen, war das in [YLL+05]) beschriebene Meanshift-Verfahren (siehe Kapitel 5.1.1). Aufgrund von Problemen bei der technischen Umsetzung und der Tatsache, dass die daraus resultierenden Segmentierungen nicht zufriedenstellend waren im Hinblick auf die weitere Verarbeitung im Rahmen der Pfadberechnung, wurde dieser Ansatz aber wieder verworfen. Daher wurde ein Bereichswachstumsverfahren (siehe Kapitel 5.1.2) implementiert, das mehr auf die Anforderungen der Projektgruppe 523 zugeschnitten ist. Um Pfade planen zu können, die von den Robotern in einer möglichst gleichmäßigen Bewegung abgefahren werden können, ist es notwendig, das Werkstück zunächst in Segmente zu zerlegen, die ein Homogenitätskriterium bezüglich ihrer Oberflächenkrümmung erfüllen. Andernfalls bestünde die Gefahr, Pfade zu generieren, die aufeinanderfolgende Bahnpunkte enthalten, die eine so große Abweichung bezüglich der Orientierung des TCPs (siehe Kapitel 4.2.1) aufweisen, dass eine flüssige Bewegung des Roboters nicht möglich ist. Es liegt also nahe, als Homogenitätskriterium die Abweichung der Normalen von adjazenten Facetten des Dreiecksnetzes zu verwenden. Daher wird das Werkstück mit Hilfe des Bereichswachstumsverfahrens in Segmente zerlegt, für die gilt, dass die Normalen, die zu den Dreiecksmittelpunkten von je zwei zueinander adjazenten Dreiecken gehören, nicht mehr als einen, vom Benutzenden festgelegten Wert, voneinander abweichen. Damit zwei Dreiecke in diesem Zusammenhang als adjazent gelten, müssen sie eine gemeinsame Kante besitzen. Das bereits erwähnte Meanshift-Verfahren könnte prinzipiell als Vorverarbeitungsschritt eingesetzt werden, um die Ergebnisse des Bereichswachstumsverfahrens zu verbessern. Wie sich das Meanshift-Verfahren mit dem Bereichswachstumsverfahren kombinieren ließe, wird am Ende des Kapitels 5.1.1 beschrieben.

Die durch das Bereichswachstumsverfahren erzeugte Segmentierung ist aber nicht notwendigerweise als Eingabe für den Pfadberechnungsalgorithmus (siehe Kapitel 5.2) ge-

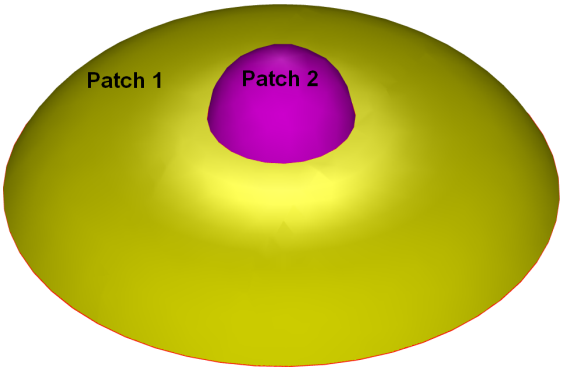


Abbildung 5.3.: Beispiel für ein Loch innerhalb eines Patches in Form eines weiteren Patches

eignet, da in diesem mit Hilfe mehrerer Schnittebenen, die parallel zur Oberseite einer um den Patch definierten Bounding Box (siehe Kapitel 7.3.2) verlaufen, die Pfade von einem Rand des Patches zum anderen ermittelt werden. Daher ist es notwendig, dass eine solche Schnittebene nur zwei Rand-Dreiecke des Patches schneidet. Wenn dies nicht gewährleistet ist (weil beispielsweise ein Patch innerhalb eines anderen Patches liegt, wie in Abbildung 5.3 dargestellt), wird dies als Loch im Patch interpretiert. Nach der Segmentierung ist folglich noch ein Post-Processing nötig, um solche Löcher zu beseitigen. Wie dies bewerkstelligt wird, wird im Kapitel Patch-Optimierung (siehe Kapitel 5.1.3) beschrieben.

Das Meanshift-Verfahren, das Bereichswachstumsverfahren und die Patch-Optimierung werden im Folgenden vorgestellt. Die Beschreibung des Meanshift-Verfahrens basiert auf [YLL+05], die verwendeten Formeln stammen ebenfalls aus dieser Quelle. Das Bereichswachstumsverfahren ist angelehnt an das in [CXM+04] beschriebene Verfahren zur Erzeugung von Patches.

5.1.1. Meanshift

Meanshift ist ein nicht-parametrisches Clusteringverfahren, d.h. die Anzahl der Cluster muss nicht vorher festgelegt und somit nicht als Parameter übergeben werden. Nicht-parametrisch bedeutet in diesem Zusammenhang aber nicht, dass es keine Parameter gibt – so wird die Anzahl der durch das Meanshift erzeugten Cluster maßgeblich durch eine zuvor definierte Bandbreite h beeinflusst. Grundsätzlich basiert das Meanshift-Verfahren auf dem Gradientenverfahren, bei dem sogenannte Kernel-Dichte-Schätzer eingesetzt werden, um anhand einer Menge diskreter Beobachtungen eine unbekannte Verteilungsdichte innerhalb des Merkmalsraums abzuschätzen. Für eine gegebene Menge von Merkmalsvektoren $X = \{\mathbf{x}_1, ..., \mathbf{x}_N\} \subset \mathbb{R}^d$ ist der Kernel Dichte-Schätzer $\widehat{f}$ am Punkt $\mathbf{x}$ gegeben durch:

$$\widehat{f}(\mathbf{x}) = \frac{1}{Nh^d} \sum_{i=1}^{N} K\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right). \tag{5.1}$$

Dabei ist h die bereits erwähnte Bandbreite und $K(\mathbf{x})$ eine Kernel-Funktion, die folgendermaßen definiert ist:

$$K(\mathbf{x}) = c \cdot k(|\mathbf{x}|^2). \tag{5.2}$$

$k(\mathbf{x})$ ist das Profil des Kernels und c eine Normalisierungskonstante. In [YLL+05] wird der sogenannte Epanechnikov-Kernel mit folgendem Profil verwendet:

$$k_E(x) = \begin{cases} 1 - x & \text{für } 0 \leq x < 1 \\ 0 & \text{für } x > 1 \end{cases}. \tag{5.3}$$

Um die Menge der Beobachtungen zu clustern, bietet es sich an, die lokalen Maxima von $\widehat{f}(\mathbf{x})$ als Zentren der Cluster zu verwenden. Um diese lokalen Maxima zu finden, wird der Kernel entlang des Gradienten der Verteilungsdichte verschoben. Für die Dichteschätzung ergibt sich unter Verwendung des Kernel Dichte-Schätzers aus Gleichung 5.1 schließlich folgende Gleichung:

$$\nabla \widehat{f}(\mathbf{x}) = \frac{2c}{Nh^{d+2}} \sum_{i=1}^{N} g\left(\left|\frac{\mathbf{x} - \mathbf{x}_i}{h}\right|^2\right) m(\mathbf{x}), \tag{5.4}$$

wobei $g(\mathbf{x}) = -k'(\mathbf{x})$ und

$$m(\mathbf{x}) = \frac{\sum_{i=1}^{N} \mathbf{x}_i g\left(\left|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right|^2\right)}{\sum_{i=1}^{N} g\left(\left|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right|^2\right)} - \mathbf{x}. \tag{5.5}$$

Die eigentliche Meanshift-Prozedur besteht nun darin, dass über die Punkte des Eingabevektors iteriert und ein Fenster mit Radius h um den gerade betrachteten Punkt als Zentrum definiert wird. Anschließend wird der Meanshiftvektor (gegeben durch Gleichung 5.5) berechnet und das Fenster in dessen Richtung verschoben. Die Bandbreite h bestimmt folglich welche Merkmalsvektoren in der Berechnung berücksichtigt werden. Der resultierende Punkt bildet das neue Zentrum für die Berechnung des nächsten Meanshiftvektors. Dies geschieht so lange, bis das Verfahren konvergiert, der Meanshiftvektor also die Länge null hat. In dem Fall wurde das lokale Maximum gefunden, dem der

Ausgangspunkt zugeordnet wird. Indem für jeden Punkt auf diese Weise das zugehörige Maximum bestimmt wird, lässt sich eine Einteilung in disjunkte Cluster vornehmen.
Wie bereits in der Einleitung erwähnt, könnte das beschriebene Verfahren als Pre-Processing Schritt für das Bereichswachstumsverfahren eingesetzt werden, indem das Dreiecksnetz anhand der Normalenvektoren der Dreiecksmittelpunkte mittels Meanshift in disjunkte Cluster eingeteilt wird. Die Dreiecke der Cluster-Zentren könnten dann als potentielle Start-Dreiecke für das Bereichswachstumsverfahren verwendet werden. Das Start-Dreieck für den ersten Patch würde zufällig aus dieser Menge gewählt werden und das Start-Dreieck jedes weiteren Patches würde aus der Menge der noch nicht verwendeten Cluster-Zentren gewählt werden, indem das Cluster-Zentrum bestimmt wird, dessen Normale die größte Abweichung zum zuvor verwendeten Cluster-Zentrum aufweist. Auch könnte eine Variante des Bereichswachstumsverfahrens implementiert werden, die die durch das Meanshift entstandenen Cluster als Eingabe verwendet und miteinander verschmelzt. Der prinzipielle Ablauf des implementierten Bereichswachstumsverfahren wird im folgenden Kapitel erläutert.

5.1.2. Bereichswachstumsverfahren

Das auf die Anforderungen der Projektgruppe 523 zugeschnittene Bereichswachstumsverfahren lässt sich wie folgt skizzieren:

1. Wähle das erste Dreieck, das noch keinem Patch zugeordnet wurde als Start-Dreieck s für den zu erzeugenden Patch.

2. Durchlaufe die zu den drei Seiten von s adjazenten Dreiecke s_i, $i \in \{1..3\}$ und berechne die Abweichung Δ der Normalen der Dreiecksmittelpunkte.

3. Wenn die Abweichung Δ unter einem vom Benutzenden angegebenen Grenzwert ϵ liegt, füge das zu s adjazente Dreieck s_i dem Patch hinzu.

4. Wiederhole (2) und (3) für die neu hinzugefügten Dreiecke bis kein weiteres adjazentes Dreieck mehr hinzugefügt wird.

5. Führe (1)-(4) durch, um den nächsten Patch zu erzeugen, bis alle Dreiecke einem Patch zugeordnet wurden.

Das Hauptkriterium bildet also die Abweichung der Normalen zwischen den zueinander adjazenten Dreiecken – der Grund dafür ist, dass Patches mit einer stetigen Krümmung und die daraus resultierende Bahn bezüglich des Abfahrens der Bahn durch den Roboter günstiger sind als solche, die grosse Abweichungen zwischen adjazenten Dreiecken aufweisen, wodurch zwei aufeinanderfolgende Bahnpunkte große Unterschiede bezüglich der Orientierung des TCPs (siehe Kapitel 4.2.1) zur Folge haben können.

Da der Algorithmus zur Berechnung der Bahnen auf den einzelnen Patches (siehe Kapitel 5.2) nur funktioniert, wenn jeder Patch einen Rand besitzt, wäre es problematisch, wenn beispielsweise eine Kugel aufgrund der gleichbleibenden stetigen Krümmung einen einzelnen Patch bilden würde. Daher wird in Schritt (3) des oben skizzierten Algorithmus als weiteres Kriterium die Abweichung zwischen der Normalen des betrachteten Dreiecks und der Normalen des Start-Dreiecks berücksichtigt. Liegt die Abweichung über 90°, wird das Dreieck nicht zu dem Patch hinzugefügt – eine Kugel würde demnach im Idealfall (zwei sich gegenüberliegende Dreiecke als Start-Dreiecke) in zwei Halbkugeln segmentiert.

Nachdem mit Hilfe des Bereichswachstumsverfahrens eine Segmentierung des Werkstücks erzeugt wurde, ist noch eine Post-Processing Phase notwendig, um sicherzustellen, dass die einzelnen Patches eine geeignete Eingabe für den Pfadberechnungsalgorithmus für einzelne Patches (siehe Kapitel 5.2.1) bilden. Welche Schritte diese Post-Processing Phase umfasst, wird im folgenden Kapitel erläutert.

5.1.3. Patch Optimierung

Nach der Segmentierung des CAD-Modells des Werkstücks können in Patches Löcher vorhanden sein, aufgrund der Lage anderer Patches oder der Geometrie des Werkstückes selbst. Im Rahmen der Pfadberechnung können diese zu Problemen führen, da in diesem Fall gegebenenfalls Teile der Geometrie bei der Planung nicht berücksichtigt werden. Aus diesem Grund werden für jeden Patch vorhandene Löcher, wie im Nachfolgenden beschrieben, entfernt.

In einem ersten Schritt wird eine Oriented Bounding Box (siehe Kapitel 7.3.2) für den vollständigen Patch berechnet. Dabei wird im Folgenden die Seite der Bounding Box, die dem gewichteten Durchschnittsnormalenvektor des Patches gegenüber liegt, als Vorderseite bezeichnet. Die anderen Seiten werden Abbildung 5.4 entsprechend benannt. Der gewichtete Durchschnittsnormalenvektor n_a aller N Dreiecke des Patches berechnet sich anhand Gleichung 5.6, wobei n_i die Normale und S_i die Fläche des i-ten Dreiecks bezeichnet.

$$n_a = \frac{\sum_{i=1}^{N} S_i n_i}{\left\| \sum_{i=1}^{N} S_i n_i \right\|} \tag{5.6}$$

Anschließend wird eine Schnittebene aus der rechten oder oberen Seite der Bounding Box erzeugt, abhängig davon, welche der beiden Seiten eine längere angrenzende Kante zur Vorderseite der Bounding Box besitzt. Diese Schnittebene wird von der rechten zur linken Seite bzw. von der oberen zur unteren Seite der Bounding Box im Abstand d

über den Patch verschoben. Der Abstand d bestimmt dabei die Genauigkeit und muss geringer als der Werkzeugradius sein. Dabei wird die Anzahl der Schnitte der Ebene mit den äußeren Kanten des Netzes dieses Patches berechnet. Unter der Annahme, dass das Werkstück entsprechend der im Kapitel Segmentierung (siehe Kapitel 5.1) beschriebenen Verfahren segmentiert wurde, würden für einen lochfreien Patch für jeden Schnitttest zwei Schnittpunkte errechnet.

Ausgehend von dieser Voraussetzung können anhand der steigenden oder fallenden Anzahl von Schnittpunkten mit äußeren Kanten die Grenzen von Löchern in dem Netz des Patches erkannt werden (siehe Abbildung 5.5). Wird eine solche Grenze erkannt, wird das Netz mit der aktuellen Ebene zerschnitten (siehe Kapitel 2.3.7) und der Teil des Netzes rechts bzw. oberhalb der Schnittebene einem neuen Patch zugeordnet. Auf dem übrigen Teil des Netzes wird das Verfahren iterativ fortgesetzt. Bei diesem Verfahren können neue Patches entstehen, die keine vollständige Zusammenhangskomponente bilden. Daher wird im Anschluss ein Bereichswachstumsverfahren, ähnlich dem in Kapitel 5.1.2 beschriebenen angewendet, um diese Bereiche zu separieren.

Dazu wird für jeden Patch über die in ihm enthaltenen Dreiecke iteriert und ausgehend von dem ersten gefundenen Dreieck, dessen Patchnummer mit der Nummer des aktuell betrachteten Patches übereinstimmt (zu Beginn für alle Dreiecke der Fall), durch rekursives Umnummerieren aller adjazenten Dreiecke eine Zusammenhangskomponente erzeugt. Dieses wird so lange wiederholt, bis kein Dreieck mehr gefunden wird, das die gleiche Patchnummer wie der Ausgangspatch hat.

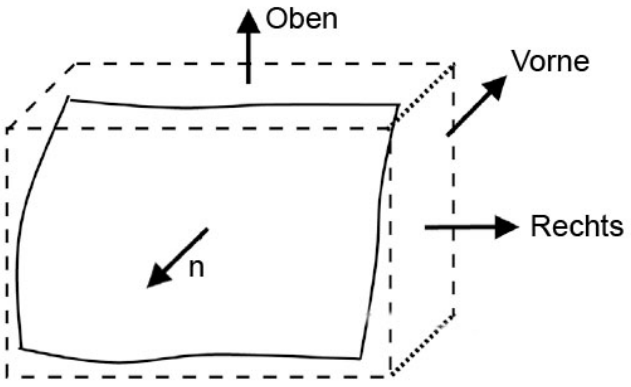


Abbildung 5.4.: Ein Patch mit seiner Bounding-Box

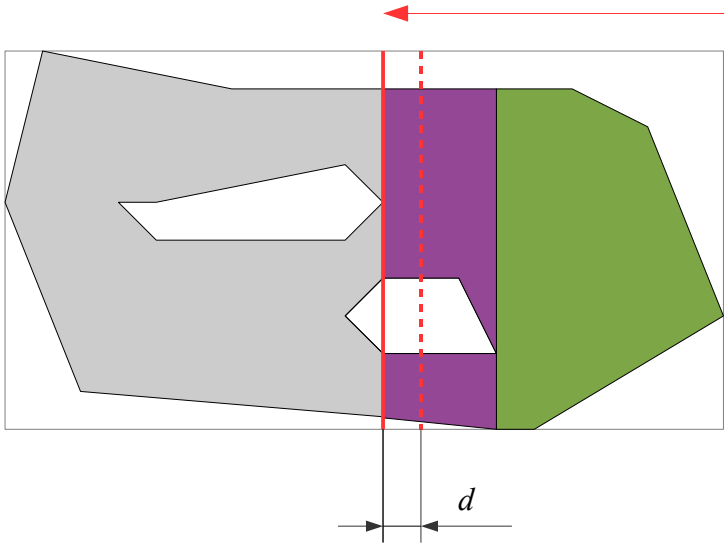


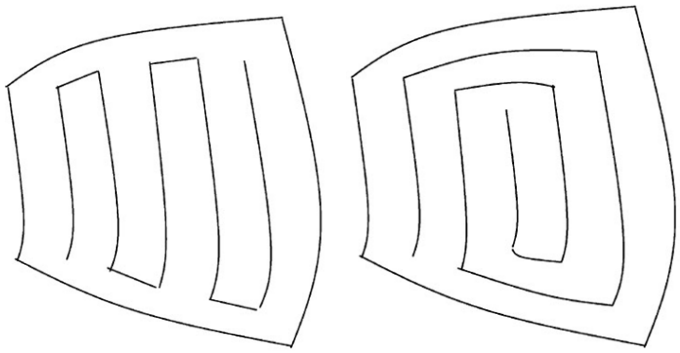
Abbildung 5.5.: Detektion von Lochgrenzen

5.2. Pfadberechnung

5.2.1. Pfadberechnung für einzelne Patches

Nachdem das Werkstück in Patches eingeteilt wurde, müssen für die einzelnen Patches Abdeckungsbahnen berechnet werden. Es wird für jeden Patch ein Pfad gesucht, auf dem sich der TCP (siehe Kapitel 4.2.1) des Roboters bewegt, sodass die zu bearbeitende Oberfläche des Patches durch den Wirkradius r_w des Werkzeugs vollständig abgedeckt wird. Ein Pfad besteht aus einer Menge von Punkten, die nacheinander vom Roboter angefahren werden. Um eine vollständige Abdeckung zu erreichen, bieten sich zwei Arten von Bahnen an. Zum einen kann der Patch mit einer Bahn abgedeckt werden, die aus geodätisch parallelen Linien besteht, die an den Enden verbunden sind, zum anderen kann eine spiralartige Bahn benutzt werden (siehe Abbildung 5.6). Die Projektgruppe 523 hat sich an dem Ansatz von [CXM+04] orientiert, bei dem ein Bounding Box-Algorithmus angewendet wird, um die zuerst genannte Bahn zu berechnen.

Der erste Schritt bei der Pfadberechnung ist das Skalieren des Dreiecksnetzes des Patches. Dies ist notwendig, da viele Arbeitsprozesse Werkzeuge verwenden, die einen bestimmten Abstand zum Werkstück, den sogenannten tool stand-off, einhalten müssen. Der Faktor beim Skalieren des zu bearbeitenden Patches wird so gewählt, dass der Pfad auf der Oberfläche des skalierten Patches genau dem um den tool stand-off extrahierten Pfad auf dem ursprünglichen Patch entspricht (siehe Abbildung 5.1).



(a) Geodätisch parallele Bahn (b) Spiralartige Bahn

Abbildung 5.6.: Mögliche Abdeckungsbahnen

Die im Folgenden vorgestellten Schritte der Pfadberechnung werden auf dem skalierten Patch ausgeführt. Die in *RoSi3D* eingesetzte Pfadberechnung für einzelne Patches orientiert sich an dem Bounding Box-Algorithmus aus [CXM+04]. Die Bounding Box umschließt den gesamten Patch, wobei die Oberseite der Box dem gewichteten Durchschnittsnormalenvektor des Patches gegenüber liegt (siehe Abbildung 5.7). Anschließend wird eine Schnittebene aus der vorderen oder linken Seite der Bounding Box erzeugt, abhängig davon, welche der beiden Seiten eine längere angrenzende Kante zur Oberseite der Bounding Box besitzt. Diese Schnittebene wird von der linken zur rechten Seite bzw. von der vorderen zur hinteren Seite der Bounding Box im Abstand d_p über den Patch verschoben. Der Abstand d_p einer Ebene zu ihrer benachbarten Ebene, legt den Abstand zwischen zwei benachbarten Pfadsegmenten fest (siehe Abbildung 5.8) und ergibt sich aus Formel 5.7. Die Variable r_w entspricht dem Wirkradius des Werkzeugs und die Überlagerungssdistanz d_o gibt an, wie weit sich die Wirkradien zweier benachbarter Pfadsegmente überlagern sollen.

$$d_p = 2r_w - d_o \tag{5.7}$$

Für jede Ebene werden alle Schnitte mit den Dreiecken des Dreiecksnetzes der Patchoberfläche berechnet (siehe Kapitel 2.3.7). Gleichzeitig werden die Normalenvektoren in den Schnittpunkten berechnet, da durch diese die Orientierung des TCPs (siehe Kapitel 4.2.1) bestimmt wird, die bei der Bewegungsplanung benötigt wird (siehe Kapitel 6). Das Ergebnis der Schnittberechnung ist eine Menge von Schnittlinien auf dem Patch, die als Grundlage für den Pfad auf dem Patch dienen (siehe Abbildung 5.9).

Da die Ebene so gewählt ist, dass sie stets die längere Seite des Patches schneidet, wird gewährleistet, dass die Schnittlinien möglichst lang werden. Dadurch wird die Anzahl der Richtungsänderungen, die der Roboter beim Abfahren der Patchbahn ausführt, minimiert.

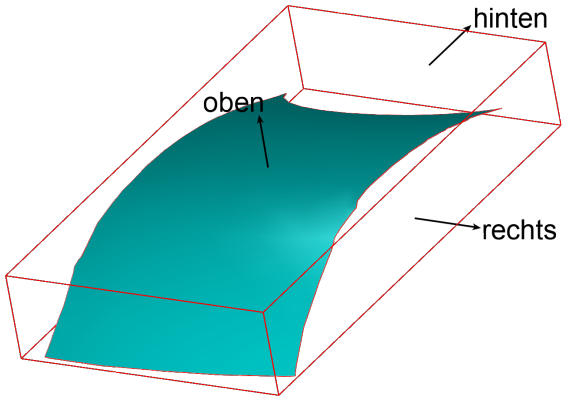


Abbildung 5.7.: Ein Patch mit seiner Bounding Box

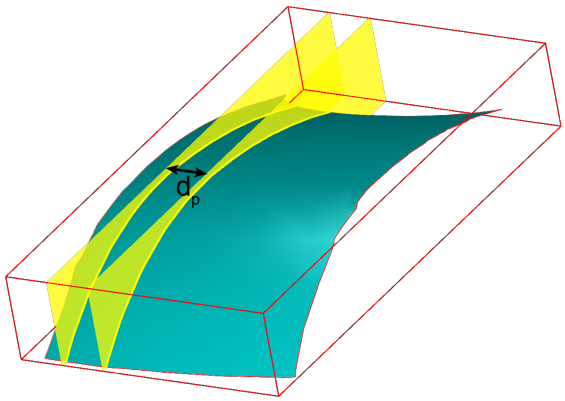


Abbildung 5.8.: Patch mit Schnittebenen

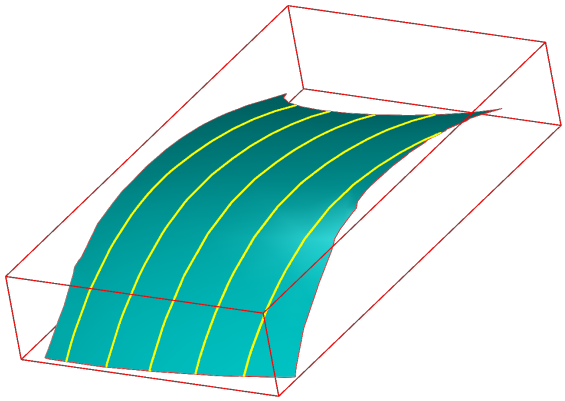


Abbildung 5.9.: Schnittlinen auf einem Patch

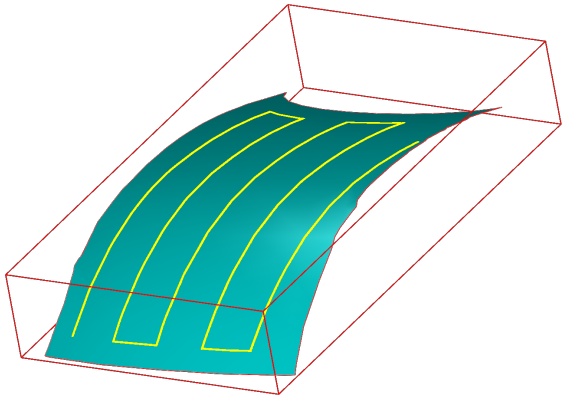


Abbildung 5.10.: Pfad auf einem Patch

Damit der Roboter die Bahn abfahren kann, müssen die einzelnen Schnittlinien miteinander verbunden werden. Dazu wird zunächst der Abstand zu den Patchaußenkanten korrigiert. Alle Pfadpunkte, die näher als die Hälfte des Werkzeugradius r_w an der Patchaußenkante liegen werden entfernt. Dies hat zugleich den Effekt, dass die Nahtstelle zweier aneinanderliegender Patches nicht als solche zu erkennen ist. Danach werden die Schnittlinien so verbunden, dass ein durchgängiger Pfad entsteht. Hierzu werden gemäß Abbildung 5.10 abwechselnd jeweils zwei aufeinanderfolgende Schnittlinien an ihren Enden direkt verbunden. Diese Verbindungen sind so ausgerichtet, dass sie parallel zur Außenkante des Patches verlaufen. Das Ergebnis dieses Verfahrens ist ein Pfad, der unter Berücksichtigung des tool stand-off, des Werkzeugradius r_w und der Überlagerungssdistanz d_o den Patch lückenlos abdeckt.

5.2.2. Pfadberechnung zwischen den Patches mittels TSP

Nach erfolgreicher Berechnung der Bahnen innerhalb der einzelnen Patches müssen nun die Patches zu einer Gesamtbahn miteinander verbunden werden. Für den Sprühvorgang ist diese Verbindung der Patches irrelevant, da auf diesen Verbindungsstücken, im folgenden mit Interpatchverbindung bezeichnet, das Werkzeug abgeschaltet wird. Es gilt daher einen optimierten Pfad über die Patches nur aus Zeitgründen zu bestimmen. Es liegt eine endliche Menge von Punkten vor, bestimmt durch die Lage der Patches, auf der eine bestmögliche Verbindung der Punkte untereinander hergestellt werden soll, so dass der Gesamtpfad kostenminimal sein soll. Die Kosten von Punkt zu Punkt ergeben sich durch die jeweils zuvor gewählte Interpatchverbindungstrategie (siehe Kapitel 5.2.3). Es handelt sich hierbei also um das bekannte Traveling-Salesman Problem (TSP). Die Aufgabe besteht darin, eine Reihenfolge verschiedener Orte, die es zu besuchen gilt, zu ermitteln, so dass die Gesamtreisekosten minimal sind, und der Reisende nach vollendeter Reise an den Ausgangsort zurückkehrt. Die Orte und deren Verbindungen untereinander werden auf einen Graphen reduziert, wobei die Knoten die Orte und die Kanten die Verbindun-

gen repräsentieren. Allerdings gehört das TSP der Klasse der NP-vollständigen Probleme an. Für eine optimale Lösung wäre es notwendig, alle möglichen Rundreisen zu berechnen und die minimale zu wählen. Bei n Knoten gibt es $\frac{(n-1)!}{2}$ mögliche Reisen. Bei nur 100 Knoten ergeben sich bereits $4,666311 \cdot 10^{155}$ Reisen. Eine optimale Lösung für viele Patches ist folglich nicht mit einer akzeptablen Laufzeit in Einklang zu bringen. Bei bis zu fünf Patches wird eine optimale Rundreise berechnet, indem jede mögliche Rundreise berechnet wird und die minimale ausgewählt wird. Es müssen folglich 181.440 verschiedene Rundreisen berechnet werden. Diese sind noch in wenigen Sekunden bestimmbar. Für mehr als fünf Patches wird zur Approximation der Lösung des TSPs die sogenannte Minimal-Spanning-Tree-Heuristik sowie die 2-Post-Optimization-Heuristik eingesetzt. Das Ergebnis der MST-Heuristik ist nicht immer optimal, aber auch nicht schlechter als das Doppelte der Kosten der optimalen Lösung (siehe 5.2.2). Die als Post-Processing dienende 2-OPT-Heuristik optimiert die zuvor berechnete Rundreise. Die Rechenzeit auf 100 Knoten ist mit Hilfe der Heuristiken auf wenige Sekunden reduzierbar und kann daher hier als guter Kompromiss zwischen Laufzeit und Güte der Lösung verwendet werden.

Minimal-Spanning-Tree Heuristik

Die MST Heuristik benötigt als Eingabe einen zuvor berechneten minimalen Spannbaum. Dieser wird vorab nach *Prim* berechnet. Nun folgen vier weitere Schritte um eine gültige Rundreise zu erhalten, wobei nur unmarkierte Knoten weiter betrachtet werden.

- Schritt 1: Verdoppeln aller Kanten und Wahl eines Startpunktes, der markiert wird
- Schritt 2: Wahl der nächsten Kante und Markieren neuer Knoten
- Schritt 3: Rücklaufen aus einer erreichten Sackgasse
- Schritt 4: Wiederhole Schritt 2 und 3 solange bis alle Knoten markiert sind

Im ersten Schritt werden alle Kanten verdoppelt, so dass sie doppelt gerichtet bzw. ungerichtet sind. Ferner wird ein Startknoten S ausgewählt und markiert. Nun werden alle adjazenten Kanten dieses Knotens betrachtet und die Kante mit den niedrigsten Kosten gewählt. Führt diese zu einem noch nicht markierten Knoten, so wird dieser markiert und es wird von dort aus weiter gesucht. Dies wird solange wiederholt bis ein Knoten P erreicht wird, an dem kein weiterer unmarkierter Knoten gefunden werden kann. Dieser Knoten befindet sich in einer sogenannten Sackgasse. Von hier aus wird solange zurück gelaufen, bis ein Knoten Q gefunden wird, an dem eine Kante zu mindestens einem nicht markierten Knoten führt. Es wird erneut nach einem kürzesten Pfad zu einem nicht markierten Knoten R gesucht. Dieser Knoten wird markiert. Anschließend wird die Kante zwischen R und Q entfernt und eine Kante zwischen P und R gezogen. Nun wird so lange iteriert bis alle Knoten markiert sind. Die dabei entstandene Eulertour beschreibt

einen Weg, der maximal doppelt so lang ist wie der Kürzeste. Dies lässt sich damit begründen, dass jede Kante des minimalen Spannbaums genau zweimal benutzt wird. Jeder Knoten wiederum wird genau einmal aufgesucht (mit Ausnahme von Start- und Endknoten). Daher wird an einigen Stellen nicht die Kante aus dem minimalen Spannbaum sondern eine nicht im Spannbaum enthaltene Kante gewählt. Aufgrund der Erfüllung der Dreiecksungleichung kann die so berechnete Reise nicht teurer als das Doppelte des minimalen Spannbaums und somit nicht teurer als die optimale Reise sein.

2-OPT Heuristik

Die 2-OPT Heuristik ist ein Spezialfall der k-OPT Heuristiken. Diese gehören zu den sogenannten Post-Optimization-Algorithmen für das TSP und sind dazu ausgelegt eine zuvor durch beispielsweise eine Heuristik gefundene Lösung zu optimieren. Die Grundidee besteht darin k Kanten aus einer zuvor bestimmten Tour zu entfernen und durch k andere Kanten zu ersetzen, mit dem Ziel dass eine kürzere Tour entsteht.
Für den Fall $k = 2$ werden zwei Kanten entfernt und kreuzweise wieder eingefügt. Es werden zum Beispiel zwei Kanten $A \rightarrow B$ und $C \rightarrow B$ entfernt. Der Graph zerfällt dabei in zwei Teilstücke von B nach C und von D nach A. Es werden nun die Kanten $A \rightarrow C$ und $D \rightarrow B$ eingefügt. Führt dies zu einer günstigeren Reise, so werden diese Änderungen beibehalten, andernfalls werden sie rückgängig gemacht.
In Bezug auf die zu verbindenden Patches stellt sich folgendes Problem: Ein Patch besteht aus einem Startpunkt, einem Endpunkt und diversen Punkten zwischen diesen Beiden. Es ist also nicht möglich einen Patch durch einen einzelnen Punkt für das TSP zu repräsentieren, sondern immer nur durch ein Knotenpaar: einen Start- und Endknoten. Die Aufgabe besteht nun darin einen Endpunkt eines Patches mit einem Startpunkt des nächsten zu verbinden. Jedoch würde dieser naive Ansatz nicht ausreichen um eine optimale Patchverbindung zu garantieren. Je nach Lage der Patches kann es u.U. nötig sein einen Endpunkt mit einem anderen Endpunkt zu verbinden, wobei der angeschlossene Patch nun “verkehrt” herum abgearbeitet werden muss. Dies soll bedeuten, dass die einzelnen Punkte innerhalb des zu betrachtenden Patches in entgegengesetzter Richtung betrachtet werden. Beim Bearbeiten eines Patches muss sichergestellt werden, dass dieser vom Anfang bis zum Ende bzw. vom Ende bis zum Anfang abgearbeitet werden muss. Folglich ist eine TSP-Rundreise nur gültig, genau dann, wenn sich in ihr diese Überlegungen widerspiegeln. Um dies zu verwirklichen wird die Verbindung zwischen Startknoten und Endknoten eines Patches durch eine Kante mit negativen Kosten repräsentiert.
Es folgt eine kurze, graphische Übersicht der Heuristiken, angewandt auf zehn Punkten. In Abbildung 5.11 ist der vollständige Graph über alle Punkte dargestellt. Aus diesem wird, wie in Abbildung 5.12 zu sehen ist, ein minimaler Spannbaum berechnet. Nach Anwendung beider Heuristiken ergibt sich eine kostengünstige Rundreise, wie sie Abbildung 5.14 zu entnehmen ist.

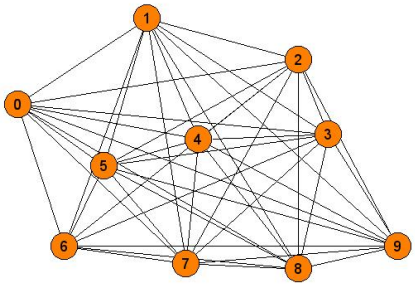


Abbildung 5.11.: vollständiger Graph

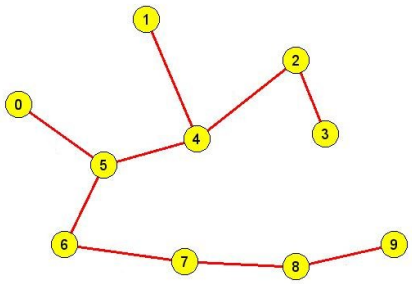


Abbildung 5.12.: minimaler Spannbaum

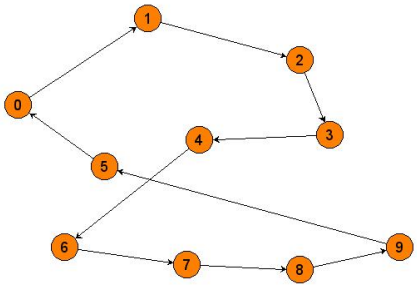


Abbildung 5.13.: Nach Anwednung der MST-Heuristik

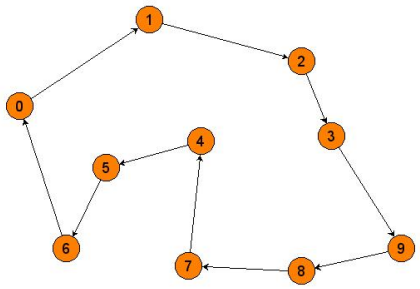
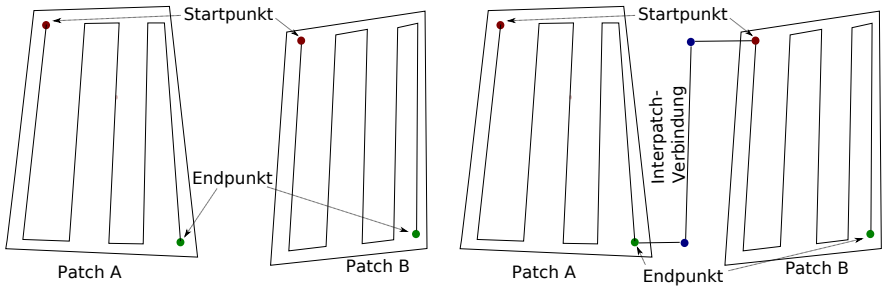


Abbildung 5.14.: Nach Anwendung der 2-Opt Heuristik



(a) Nicht verbundene Patches (b) Verbundene Patches

Abbildung 5.15.: Schema der Interpatchverbindungen

5.2.3. Interpatchverbindungen

Durch den TSP wurde eine Rundreise über alle Patches berechnet. Das heißt es liegt eine Ordnung vor in welcher Reihenfolge die einzelnen Patches vom Roboter angefahren werden sollen. Um einen durchgängigen Pfad zu generieren, ist es noch nötig die Patches miteinander zu verbinden (siehe Abbildung 5.15). Dazu werden zusätzliche Bahnpunkte eingefügt, die einen Patch mit einem anderen verbinden. Die Projektgruppe hat für diesen Arbeitsschritt zwei Verfahren entwickelt, die in den nachfolgenden Abschnitten behandelt werden.

Bounding Box-Methode

Bei der Bounding Box-Methode wird um das Objekt, für welches Teilbahnen über die Patches generiert wurden, eine Axis Aligned Bounding Box (AABB) berechnet. Die Be-

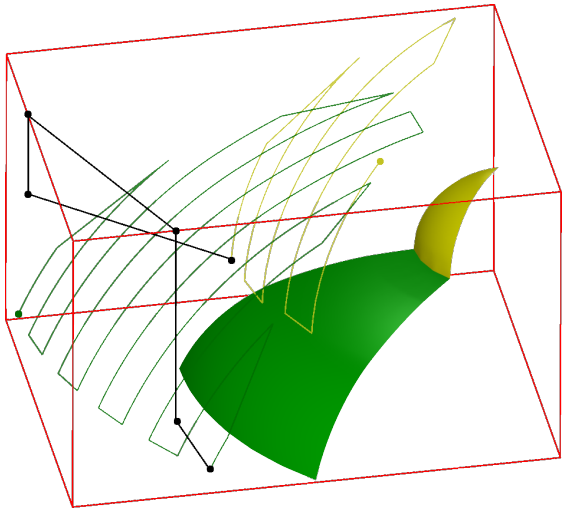


Abbildung 5.16.: Schema der Bounding Box Interpatchverbindungen

rechnung der AABB erfolgt um die extrahierten Patches, da ansonsten Interpatchpunkte berechnet würden, die genau auf dem Werkstück liegen und somit zu einer Kollision führen würden. Vom Endpunkt eines Patches wird mit Hilfe der gemittelten Patch-Normalen ein Strahl definiert. Danach wird ein Strahl-Quader Schnittest (siehe Kapitel 2.3.3) durchgeführt, um den Schnittpunkt auf der AABB zu ermitteln, welcher den ersten Interpatchpunkt darstellt. Das gleiche Verfahren wird mit dem Startpunkt des anderen Patches durchgeführt. Somit existieren zwei Interpatchpunkte, welche beide auf Flächen der AABB liegen. Falls beide Schnittpunkte auf der gleichen Fläche der AABB liegen, werden diese direkt miteinander verbunden und eine vollständige Interpatchverbindung liegt vor. Falls beide Punkte auf unterschiedlichen Flächen der AABB liegen, werden für die Punkte, die nicht bereits auf der Oberseite der AABB liegen, Zwischenpunkte auf der Oberseite eingefügt. Hierzu werden für die Punkte, die nicht auf der Oberseite liegen, Kopien erzeugt und diese entlang der Z-Achse verschoben, bis die Punkte auf der Oberseite liegen. Anschließend liegen alle Punkte auf der Oberseite der AABB und können direkt verbunden werden. Abbildung 5.16 verdeutlicht das Vorgehen.

Hemisphären-Methode

Die Berechnung von Interpatchverbindungen unter Verwendung einer Hemisphäre funktioniert ähnlich wie die Bounding Box-Methode. Dabei sei angemerkt, dass dieses Verfahren nur anwendbar ist, wenn das Objekt eine nicht zu bearbeitende Seite hat, wie zum Beispiel eine Fläche auf der es steht.

Statt eines Hüllquaders wird einmalig eine umhüllende Halbkugel berechnet, die die extrahierten Patches vollständig enthält und deren flache Seite parallel zu der nicht

zu bearbeitenden Seite des Objekts liegt. Der Einfachheit halber wird angenommen, dass diese unerreichbare Seite parallel zur XY-Ebene und unterhalb der extrahierten Patches liegt. Zunächst werden in einem ersten Durchlauf durch die Patch-Vertices die minimale Z-Koordinate z_{min} sowie der Schwerpunkt $\mathbf{g}$ dieser Punktwolke berechnet. Ihr Schwerpunkt wird daraufhin auf die Ebene projiziert, welche parallel zur XZ-Ebene ist und die Z-Achse in Koordinate z_{min} schneidet. Der resultierende Punkt $\tilde{c}$ ist der Ursprung der Hemisphäre. In einem zweiten Durchlauf durch die Patch-Vertices wird nun der maximale Abstand eines Patch-Vertex zum Hemisphärenursprung berechnet. Dies ist der Radius der Hemisphäre.

Die Berechnung der Interpatchverbindung funktioniert nun wie folgt: Die zwei zu verbindenden Pfadpunkte $\mathbf{p}$ und $\mathbf{q}$ werden auf die Hemisphäre projiziert. Die resultierenden Punkte seien $\tilde{p}$ und $\tilde{\mathbf{q}}$. Um ein Kreisbogensegment für die Interpatchverbindung auf der Hemisphäre beschreiben zu können, wird zudem die Konvexkombination $\frac{1}{2} \cdot (\tilde{\mathbf{p}} + \tilde{\mathbf{q}})$ auf die Hemisphäre projiziert. Dies sei $\tilde{\mathbf{r}}$. Der Polygonkantenzug durch die Punktfolge

$$(\mathbf{p}, \tilde{\mathbf{p}}, \tilde{\mathbf{r}}, \tilde{\mathbf{q}}, \mathbf{q}) \tag{5.8}$$

beschreibt eine Verbindung zwischen $\mathbf{p}$ und $\mathbf{q}$.

Noch nicht geklärt ist die Art und Weise, wie genau die Projektionen durchgeführt werden. Hierfür sind zwei mögliche Vorgehensweisen implementiert worden, die je nach Gestalt der verwendeten Geometrie unterschiedlich gute Ergebnisse liefern. Eine kürzere Interpatchverbindung gilt hierbei als besser.

Beim ersten Ansatz wird jeweils ein Strahl ausgehend vom Schwerpunkt $\mathbf{g}$ durch die Pfadpunkte $\mathbf{p}$ und $\mathbf{q}$ mit der Halbkugel geschnitten (siehe Kapitel 2.3.1). Analog wird mit der Konvexkombination der Projektionen verfahren. Beim zweiten Ansatz ist der jeweilige Projektionsstrahl durch den Punkt und der Oberflächennormalen des Objekts in diesem Pfadpunkt definiert. In diesem Fall wird die Konvexkombination entlang der gemittelten Oberflächennormalen von Start- und Endpunkt projiziert. Abbildung 5.17 zeigt eine Interpatchverbindung, die nach dem ersten Ansatz berechnet wurde.

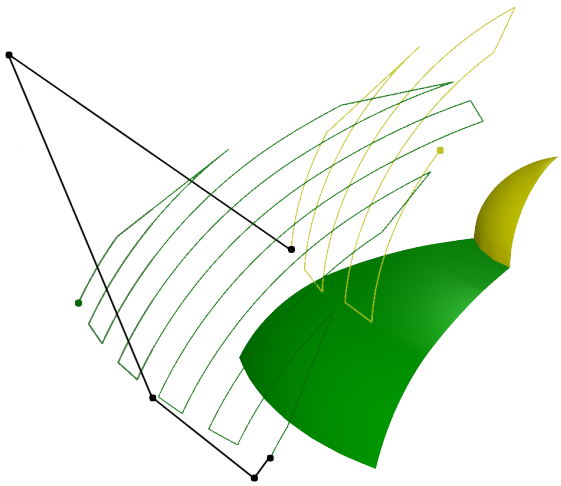


Abbildung 5.17.: Hemisphären Interpatchverbindung

6. Bewegungsplanung

Nachdem in Kapitel 5 erläutert wurde, wie die Pfade auf dem Werkstück berechnet werden, wird in diesem Kapitel beschrieben, wie aus den reinen Pfaden Roboterprogramme erzeugt werden. Mit Roboterprogrammen sind hier primär Programme in *RoSi3D* gemeint, also keine RAPID-Programme. Da sich die interne Repräsentation in *RoSi3D* stark an der Programmiersprache RAPID (siehe Kapitel 3.8) orientiert, können aus diesen in einem weiteren optionalen Schritt relativ einfach konkrete RAPID-Programme generiert werden, die sowohl in RobotStudio als auch direkt in die reale Robotersteuerung geladen werden können. Der einzige wesentliche Unterschied der zwischen der erwähnten internen Repräsentation der Programme und den resultierenden RAPID-Programmen besteht darin, dass die Home-Position in den abgeleiteten RAPID-Programmen mittels eines MoveJ-Befehls (siehe Kapitel 3.8.2) angefahren wird. Die Home-Position ist dabei in *RoSi3D* – in Anlehnung an *RobotStudio* – als die Position definiert, die sich ergibt, wenn alle Achsen der im Rahmen der Projektgruppe verwendeten Roboter (siehe Kapitel 4.3.1) auf null stehen bis auf die fünfte Achse, die auf 30° steht, und stellt die Grundstellung der Roboter dar. Da sich der Roboter in der realen Zelle nicht unbedingt in der Nähe dieser Grundstellung befinden muss, besteht die Möglichkeit, dass eben diese Position nicht auf einer linearen Bahn erreicht werden kann, was durch Verwendung eines MoveJ-Befehls umgangen wird. Allerdings sollte sich der Roboter dabei nicht in der Nähe von Hindernissen befinden, weil er nicht linear zwischen der aktuellen Position und der Home-Position interpoliert.
Das Verfahren zur Bewegungsplanung, das im Rahmen der Projektgruppe 523 erarbeitet wurde, lässt sich grob in die folgenden Schritte einteilen, wobei als Targets solche Pfadpunkte verstanden werden, denen ein TCP (siehe Kapitel 4.2.1) zugeordnet wurde:

1. Reduktion der Punktmenge eines Pfades (siehe Kapitel 6.1)
2. Orientierung der Targets (siehe Kapitel 6.2)
3. Konfigurationsauswahl je Target (siehe Kapitel 6.3)
4. Einfügen zusätzlicher Bewegungsinstruktionen (siehe Kapitel 6.4)
5. Einfügen von Steuerungsinstruktionen für die Werkzeuge (siehe Kapitel 6.5)
6. Synchronisation einzelner Roboterbahnen (siehe Kapitel 6.6)

Die einzelnen Schritte werden nachfolgend genauer erläutert.

6.1. Reduktion der Punktmenge eines Pfades

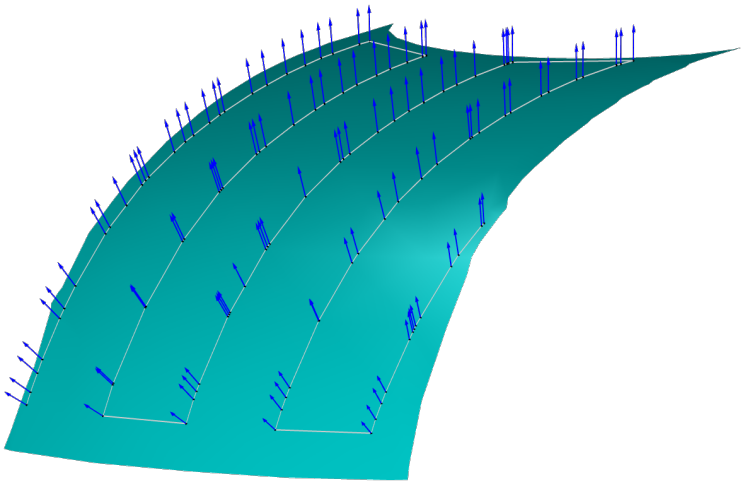
Die innerhalb der Bahnplanung generierten Pfade bestehen in der Regel aus sehr vielen Bahnpunkten. Dies führt unter Umständen zu zahlreichen unerwünschten Umorientierungen des Roboters zwischen den Bahnpunkten, was wiederum zu einer ungleichmäßigen Bewegung des TCPs (siehe Kapitel 4.2.1) führen kann. Ein weiteres Problem besteht darin, dass die Stuerung der Roboter nur eine begrenzte Anzahl an Bahnpunkten verarbeiten kann. Aus diesen Gründen besteht der erste Schritt der Bewegungsplanung darin, die Menge der Bahnpunkte zu reduzieren. Dies geschieht in *RoSi3D* in zwei Schritten, wobei der zweite Schritt im Gegensatz zum Ersten optional ist. Zunächst werden Bahnpunkte, die sich auf einer gemeinsamen Ausgleichsgeraden befinden, zu linearen Segmenten zusammengefasst. Anschließend wird in einem zweiten Schritt auf diesen linearen Segmenten nach kreisbogenförmigen Segmenten gesucht. Hierdurch wird die Bahn geglättet und die Anzahl der Bahnpunkte reduziert. Ein Beispiel für diese Glättung ist in Abbildung 6.1 zu sehen, wobei kreisbogenförmige Segmente den gelben Verbindungen entsprechen, während lineare Segmente rosa hervorgehoben sind.

6.1.1. Lineare Segmente

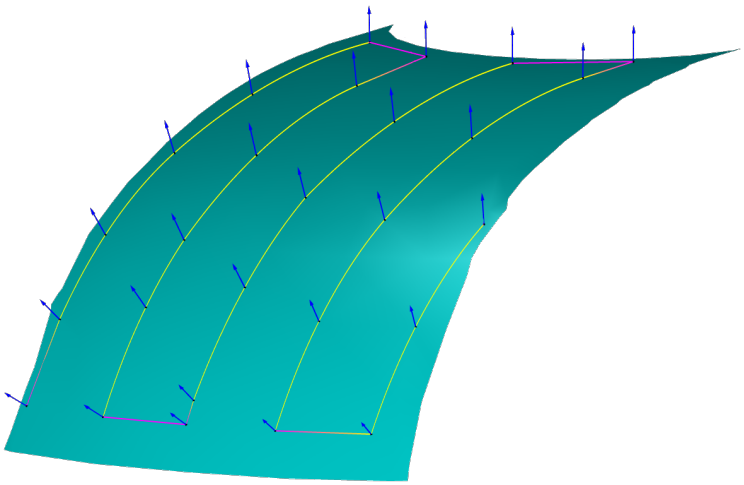
Die Einteilung von Bahnpunkten in lineare Segmente erfolgt, indem die Bahnpunkte, die auf einer gemeinsamen Ausgleichsgeraden liegen bzw. einen vorher festgelegten maximalen Abstand von dieser haben, zusammengefasst werden.
Hierzu wird zuerst eine Gerade durch die ersten beiden Punkte des Pfades gelegt. Im Anschluss daran wird sukzessiv ein weiterer Punkt der Bahn betrachtet und eine temporäre Ausgleichsgerade als Verbindungsvektor zwischen dem festen Startpunkt des linearen Segments und eben diesem Punkt bestimmt. Sofern für alle bisherigen Punkte des aktuellen Segments gilt, dass sie nicht weiter als das oben erwähnte Maximum von der temporären Ausgleichsgeraden entfernt sind, wird diese als aktuelle Ausgleichsgerade übernommen und weiter über die restlichen Punkte des Pfades iteriert. Falls die Distanz des Punktes von der temporären Ausgleichsgeraden zu groß ist, wird das aktuelle lineare Segment nicht um den betrachteten Punkt erweitert, sondern abgeschlossen und analog ein Neues begonnen.

6.1.2. Bogenförmige Segmente

Nachdem die Punktmenge, die im Rahmen der Bahnplanung berechnet wurde, durch lineare Segmente (siehe Kapitel 6.1.1) approximiert und damit reduziert wurde, soll in



(a) Ergebnis der Bahnplanung



(b) Pfad nach Reduktion der Punkte

Abbildung 6.1.: Punktreduktion im Rahmen der Bewegungsplanung

diesem Kapitel erläutert werden, wie zur weiteren Reduktion und Glättung auf diesen linearen Segmenten innerhalb von *RoSi3D* nach kreisbogenförmigen Segmenten gesucht wird. In Anlehnung an die Programmiersprache RAPID (siehe Kapitel 3.8) werden bogenförmige Segmente dabei durch einen Durchgangs- und Endpunkt definiert (siehe Kapitel 3.8.2). Die konkrete Suche nach bogenförmigen Segmenten umfasst die folgenden Schritte.
Zunächst werden jeweils die Punkte gesucht, die durch dieselbe Schnittebene (siehe Kapitel 5.2.1) entstanden sind. Sofern mehr als drei Kandidaten vorhanden sind, werden die Punkte, die auf einem gemeinsamen Kreisbogen liegen, zu einem kreisbogenförmigen Segment zusammengefasst.
Hierzu werden die Kandidaten in die oben bereits erwähnte gemeinsame zweidimensionale Schnittebene projiziert. Anschließend erfolgt die Bestimmung des Kreismittelpunktes, indem überprüft wird, ob sich die Mittelsenkrechten M_1 und M_2 zwischen den Verbindungsstrecken der ersten drei Kandidaten P_1, P_2, P_3 in einem Punkt C, der dann den Kreismittelpunkt festlegt, schneiden (siehe Abbildung 6.2). Sofern sich die beiden Geraden nicht schneiden, wird die Verbindungsstrecke zwischen dem potentiellen Startpunkt P_1 und dem potentiellen Durchgangspunkt P_2 endgültig als lineares Segment klassifiziert und eine entsprechende lineare Bewegungsinstruktion der internen Repräsentation des Roboterprogramms hinzugefügt. Ansonsten ist für die restlichen Kandidaten lediglich zu testen, ob sie ebenfalls auf dem durch die ersten drei Kandidaten determinierten Kreisbogen liegen oder nicht. Im positiven Fall können die entsprechenden Kandidaten ebenfalls durch das berechnete Kreisbogensegment repräsentiert werden und es kann eine kreisbogenförmige Bewegungsinstruktion zur Repräsentation des Roboterprogramms hinzugefügt werden. Im negativen Fall wird die Suche vom letzten Kandidaten aus, der auf dem Kreisbogensegment lag, fortgeführt. An dieser Stelle ist insbesondere zu bemerken, dass das Vorgehen auf die spezielle Art und Weise zugeschnitten ist, mit der in *RoSi3D* die Pfade (siehe Kapitel 5.2.1) berechnet werden. Das heißt also, dass der Ansatz zum Finden von Kreisbogensegmenten im Allgemeinen Fall nicht funktionieren muss, da insbesondere die Ebene, in die die Punkte projiziert werden, nicht eindeutig bestimmt ist.

6.2. Orientierung der Targets

Im Rahmen der Bahnplanung wurde die Z-Achse des TCPs bereits festgelegt: Sie zeigt in Richtung des zugehörigen Punktes auf dem Werkstück, ist also die invertierte Normale in diesem Punkt. Bei der Bewegungsplanung müssen also nur noch X- und Y-Achse gewählt werden. Um zusätzliche Drehungen der Achsen des Roboters für unnötige Umorientierungen zu vermeiden, werden die Orientierungen so gewählt, dass die X-Achse des TCPs am ersten Bahnpunkt P_0 auf dem Werkstück tangential in Richtung Bahnpunkt P_1 zeigt. Für alle folgenden Bahnpunkte P_i mit $i \geq 1$, die für das gewählte Werkstück berechnet wurden, wird diese Ausrichtung der X-Achse lediglich übernommen, indem

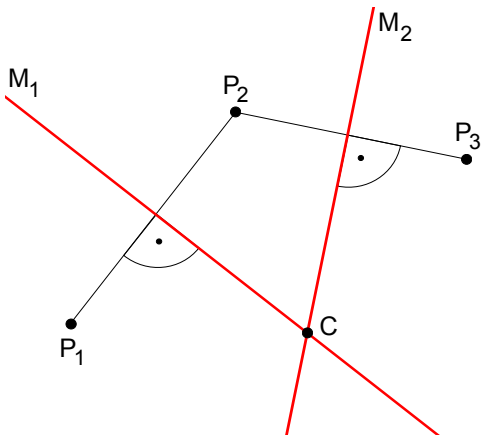
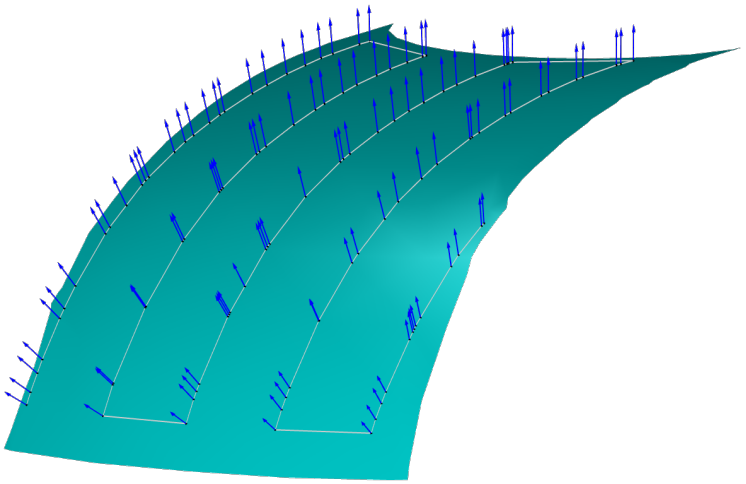


Abbildung 6.2.: Schematische Darstellung der Bestimmung des Kreismittelpunktes

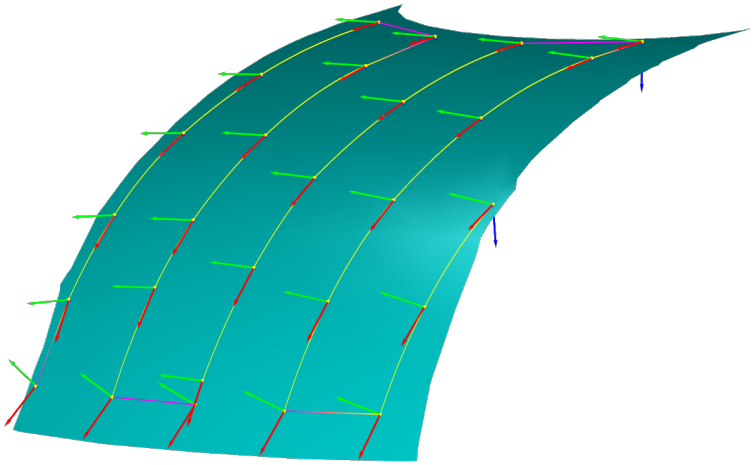
der Punkt, der durch Addition des Ortsvektors P_i und der normalisierten tangential ausgerichteten X-Achse entsteht, in die Ebene projiziert wird, die sich durch P_i und der Normalen in P_i ergibt. Die X-Achse für P_i ist dann der Verbindungsvektor zwischen P_i und dem projizierten Punkt. Die Y-Achse entsteht schließlich jeweils als Kreuzprodukt der berechneten X- und Z-Achsen (siehe Abbildung 6.3).

6.3. Konfigurationsauswahl je Target

In der Regel gibt es für die Roboter verschiedene Möglichkeiten eine Position anzufahren (siehe Kapitel 4.4.1). Bei der Konfigurationsauswahl wird versucht, eine im Sinne der Bewegungsplanung gute Konfiguration für die jeweiligen Bewegungsinstruktionen zu finden. Eine Konfiguration für ein Target wird dabei als gut klassifiziert, falls sie einerseits kollisionsfrei ist. Andererseits sollte die Konfiguration die Änderungen der Achswinkel in Bezug zum direkten Vorgänger des betrachteten Targets minimieren, sodass kollisionsfrei und gleichmäßige Bewegungen des Roboters resultieren. Die Konfigurationen legen bei den im Rahmen der Projektgruppe verwendeten Robotern dabei fest, in welchem Quadranten sich die Achswinkel der Achsen eins, vier und sechs befinden (siehe Kapitel 4.4.2).
Zur Bestimmung einer im Sinne der Bewegungsplanung günstigen Konfigurationswahl der Targets, werden zunächst die Konfigurationen entfernt, die zu einer Kollision führen. Sofern auf diese Weise keine gültigen Konfigurationen gefunden werden können, wird das jeweilige Target um die Z-Achse rotiert. Da dies einer Drehung der letzten Achse, die bei den im Rahmen der Projektgruppe verwendeten Robotern am wenigsten eingeschränkt ist, entspricht, kann dies gegebenenfalls zu einer insgesamt abfahrbaren Bewegung führen.
Sollten auf diese Weise gar keine gültigen Konfigurationen gefunden werden, wird die



(a) Pfad mit Normalen der Pfadpunkte



(b) Bahn mit orientierten TCPs

Abbildung 6.3.: Orientierung von Pfadpunkten im Rahmen der Bewegungsplanung

zugehörige Bewegungsinstruktion als nicht erreichbar und gegebenenfalls als nicht kollisionsfrei klassifiziert. Ansonsten wird aus der Liste der gefundenen Konfigurationen diejenige ausgewählt, die die Summe der Achswinkeländerungen im Vergleich zum letzten erreichbaren – im Idealfall dem direkt vorangegangenen – Target minimiert. Falls kein vorangegangenes Target erreichbar bzw. im Falle des ersten Targets vorhanden war, so wird die Konfiguration mit minimalem Abstand von der Standardkonfiguration (0,0,0,0) ausgewählt.
Die Frage, ob eine Bahn, die aus erreichbaren Targets zusammengesetzt ist, von den verwendeten Robotern abzufahren ist, kann in letzter Konsequenz nur die Simulation der Bewegung (insbesondere in *RobotStudio*) bestätigen oder widerlegen, da die Erreichbarkeit aller einzelnen Targets nicht gewährleistet, dass die Bewegung insgesamt abgefahren werden kann.

6.4. Einfügen zusätzlicher Bewegungsinstruktionen

Die Roboterprogramme beschreiben zunächst lediglich – wie in den Kapiteln 6.1.1 und 6.1.2 beschrieben – lineare und zirkuläre Bewegungsinstruktionen auf dem Werkstück. Damit die Roboter nicht schon zu Beginn des Arbeitsprozesses kollidieren, werden beide Roboter in ihre jeweilige Home-Position gefahren, bevor sie mit der Abarbeitung der eigentlichen zum Arbeitsprozess gehörigen Bahnen beginnen. Hierzu wird, wie bereits in der Einleitung von Kapitel 6 angesprochen, ein MoveJ-Befehl (siehe Kapitel 3.8.2) verwendet. Auch am Ende der jeweiligen Pfade auf dem Werkstück ist es sinnvoll, den jeweiligen Roboter zurück zur Home-Position zu fahren. Auf diese Weise wird dafür gesorgt, dass sich die Roboter nicht gegenseitig behindern bzw. miteinander kollidieren. Die Bewegung von der Home-Position zum jeweiligen ersten Punkt des Pfades auf dem Werkstück wird zusätzlich noch weiter äquidistant in mehrere lineare Teilstücke unterteilt, um Umorientierungen zu ermöglichen und so Fehler bzw. Einschränkungen des Roboters beim Umorientieren zu vermeiden.

6.5. Einfügen von Steuerungsinstruktionen für die Werkzeuge

Auf jeder Roboterbahn gibt es bestimmte Bereiche, in denen das jeweilige Werkzeug, sofern es diese Funktionalität unterstützt, ausgeschaltet sein sollte:

- Auf dem initialen Weg zur Home-Position
- Auf dem Weg zum ersten Target auf dem Werkstück
- Auf Interpatchverbindungen (siehe Kapitel 5)

- Auf dem finalen Weg zur Home-Position

Das Ein- und Ausschalten der Werkzeuge geschieht mit Hilfe von digitalen Ausgängen auf der Robotersteuerung. Vor den oben genannten Bahnabschnitten wird also das jeweilige Signal auf *false* gesetzt. Wenn es danach mit einem Abschnitt auf dem Werkstück weiter geht, werden die Signale dementsprechend wieder auf *true* gesetzt. Die passenden Signalnamen müssen beim Laden eines Werkzeugs in *RoSi3D* angegeben werden, damit das Roboterprogramm während der Bewegungsplanung automatisiert entsprechend erzeugt werden kann.

6.6. Synchronisation

Wie bereits in Kapitel (siehe Kapitel 1) erwähnt, bestand eines der Ziele der Projektgruppe darin, Roboterbahnen zu bestimmen, die von zwei sechs-achsigen Industrierobotern kooperierend und kollisionsfrei abgearbeitet werden können. Die Kollisionsfreiheit zwischen den Robotern wird letztlich in *RoSi3D* dadurch gewährleistet, dass die jeweiligen Bahnen sequentiell abgefahren werden.
Hierbei hat die Synchronisation der Roboterbahnen die Aufgabe, sicherzustellen, dass die Home-Positionen sequentiell eingenommen und die Roboterbahnen in sequentieller Reihenfolge abgefahren werden. Die Reihenfolge hängt in diesem Zusammenhang vom Arbeitsprozess ab, den jeder einzelne Roboter ausführt. Im Kontext der Projektgruppe bedeutet dies, dass der Roboter, der für das Besprühen des Werkstückes eingesetzt werden soll, mit der Ausführung des RAPID-Programms beginnt, während der Roboter, der für das Trocknen des Werkstückes eingesetzt werden soll, solange in seiner Home-Position wartet, bis der andere Roboter sein RAPID-Programm vollständig abgearbeitet hat und sich wieder in seiner Home-Position befindet.
Diese Synchronisation geschieht mit Hilfe von WaitSyncTask-Befehlen (siehe Kapitel 3.8), die an den folgenden Stellen hinzugefügt werden.
Das Programm des sprühenden Roboters beginnt mit einem MoveJ-Befehl zu seiner Home-Position, während das Programm des trocknenden Roboters mit einem WaitSyncTask-Befehl startet. Die Synchronisation wird anschließend erreicht, indem dem Programm des ersten Roboters ein WaitSyncTask-Befehl hinzugefügt wird, der denselben Synchronisationsbezeichner aufweist, wohingegen dem Programm des zweiten Roboters analog ein entsprechender MoveJ-Befehl angefügt wird. Der dritte Befehl ist bei beiden Programmen ein WaitSyncTask-Befehl, der sicherstellt, dass der sprühende Roboter solange in seiner Home-Position wartet, bis der andere Roboter ebenfalls seine Home-Position erreicht hat. Die vierte Instruktion stellt für den sprühenden Roboter dann den Beginn des eigentlichen Sprüh-Programms dar. Sein Programm endet mit einem WaitSyncTask-Befehl. Demgegenüber steht das Programm des trocknenden Roboters, welches vor dem Beginn des eigentlichen Programms, das das Trocknen realisiert, einen weiteren WaitSyncTask-Befehl enthält mit demselben Synchronsisationsbezeichner des

zuvor erwähnten WaitSyncTask-Befehls.
Ein konkretes synchronisiertes RAPID-Programm könnte dabei zum Beispiel wie folgt aussehen, wobei zur besseren Lesbarkeit lediglich die Prozeduren angegeben sind und nicht sämtliche Deklarationen.
Für den sprühenden Roboter könnte ein kurzes RAPID-Programm also die folgende Form haben:

```
PROC Path_1()
MoveJ Target_0,v100,fine,Spraygun\WObj:=wobj0;
WaitSyncTask \InPos,sync0,tasklist1;
WaitSyncTask \InPos,sync1,tasklist1;
MoveL Target_1,v100,fine,Spraygun\WObj:=wobj0;
SetDO DO10_9, 1;
MoveC Target_2,Target_3,v100,z5,Spraygun\WObj:=wobj0;
MoveL Target_4,v100,fine,Spraygun\WObj:=wobj0;
SetDO DO10_9, 0;
MoveL Target_5,v100,z1,Spraygun\WObj:=wobj0;
WaitSyncTask \InPos,sync2,tasklist1;
ENDPROC
```

Spraygun stellt dabei den Namen eines Sprühwerkzeugs dar, das über die Robotersteuerung ein- und ausgeschaltet werden kann (siehe Kapitel 6.4), was durch die SetDO-Befehle (siehe Kapitel 3.8.3) geschieht. Für eine ausführliche Erläuterung der übrigen RAPID-Befehle sei auf Kapitel 3.8 verwiesen.
Das RAPID-Programm des trocknenden Roboters, dessen Werkzeug den Namen Dryer trägt und ebenfalls über die Logikinstruktion SetDO de- und aktiviert werden kann, könnte wie nachfolgend aussehen:

```
PROC Path_1()
WaitSyncTask \InPos,sync0,tasklist1;
MoveJ Target_0,v100,fine,Dryer\WObj:=wobj0;
WaitSyncTask \InPos,sync1,tasklist1;
WaitSyncTask \InPos,sync2,tasklist1;
MoveL Target_1,v100,z5,Dryer\WObj:=wobj0;
SetDO DO10_16, 1;
MoveC Target_2,Target_3,v100,z10,Dryer\WObj:=wobj0;
MoveC Target_4,Target_5,v100,z10,Dryer\WObj:=wobj0;
SetDO DO10_16, 0;
MoveL Target_4,v100,z5,Dryer\WObj:=wobj0;
ENDPROC
```

7. Kollisionsdetektion

Nachdem im Kapitel Bahnplanung (siehe Kapitel 5) die Berechnung eines Pfades über einem Werkstück erläutert wurde, beschäftigt sich dieses Kapitel mit Algorithmen und Datenstrukturen zur Feststellung von Kollisionen zwischen den aus Dreiecksnetzen (siehe Kapitel 2.2) aufgebauten Objekten der Szene, die an der Simulation eines Sprüh- oder Trocknungsvorgangs beteiligt sind. Anwendung findet die Kollisionserkennung insbesondere in der Bewegungsplanung (siehe Kapitel 6), welche für die beiden Roboter aus entsprechenden Pfaden Bewegungen erzeugt. Diese Bewegungen müssen kollisionsfrei sein, das heißt, kein Objekt der Szene darf mit einem anderen Objekt während eines Bewegungsablaufs in Berührung kommen oder ein anderes durchdringen. Um dieses zu verhindern wird jede Bewegung eines Roboters auf mögliche Kollisionen überprüft.

Um das von der Projektgruppe realisierte Verfahren zur Kollisionserkennung zu erläutern, werden folgende Punkte im Detail betrachtet:

- Motivation
- Szenenbeschreibung
- Hüllquader (Bounding Boxes)
- Die Datenstruktur Octree
- Erzeugen von Octrees in *RoSi3D*
- Schnittest zwischen Objekten einer Szene

7.1. Motivation

Prinzipiell können zwei Objekte einer Szene auf Kollision getestet werden, indem für jedes Dreieck des einen Objekts überprüft wird, ob es sich mit einem Dreieck des anderen Objekts überschneidet. Dieser Ansatz hat den Nachteil, dass die Laufzeit der Kollisionserkennung quadratisch zu der Anzahl der Dreiecke wächst. Aus diesem Grund hat sich die Projektgruppe dazu entschieden einen Ansatz zu verfolgen, bei dem nicht alle möglichen Dreieckspaare der Objekte auf Kollision getestet werden müssen. Um die-

ses Ziel zu erreichen, werden die Geometriedaten jedes Objekts um einen Octree (siehe [FvDF+94]), ergänzt, welche die Anzahl der nötigen Schnittests der Kollisionsdetektion durch den zusätzlichen Einsatz von Hüllquadern verringern soll. Mit Geometriedaten eines Objekts sind in diesem Zusammenhang und im Folgenden die Dreiecke eines Objekts gemeint. Bevor die eigentlichen Algorithmen und Datenstrukturen zur Feststellung von Kollisionen vorgestellt werden, wird zunächst die Szenenbeschreibung in *RoSi3D* erläutert.

7.2. Szenenbeschreibung

RoSi3D bietet die Möglichkeit, die zuvor beschriebenen Roboter virtuell darzustellen, so dass die räumliche Lage der Roboter in der graphischen Oberfläche repräsentiert wird und der Benutzende diese genau spezifizieren kann. Somit besteht die Möglichkeit, die reale Umgebung der Roboter in *RoSi3D* genau abzubilden. Dies ist wichtig, damit *RoSi3D* eine korrekte Berechnung der Roboterbewegung durchführen kann und der Sprühvorgang innerhalb der Simulation dem realen Sprühvorgang realitätsnah ist.
Unter einer Szene ist im Allgemeinen eine dreidimensionale Beschreibung von Objekten und die Position sowie Blickrichtung eines virtuellen Betrachters zu verstehen. Als Objekte fungieren hier die Roboter, das Werkzeug, das Werkstück, sowie ein Werkstückhalter, in diesem Fall ein Tisch. Diese werden in einen Szenengraphen (siehe 7.2.1) geladen, so dass ein direkter Zugriff auf die Objekte erfolgen kann, um diese räumlich zu positionieren. Sie befinden sich nun in einer virtuellen Welt und haben dieser bezüglich geometrische Koordinaten - die sogenannten Weltkoordinaten. Es besteht die Möglichkeit, pro Szene bis zu zwei Roboter, zwei Werkzeuge, genau ein Werkstück und ein Werkstückhalter in die Szene zu integrieren. Bei einem ungewollten Hinzufügen ist das Entfernen eines oder mehrerer Objekte möglich. Nach einem Hinzufügen, Entfernen oder Positionieren eines Objektes muss die graphische Oberfläche dementsprechend aktualisiert werden. Dies wird durch das Model-View-Controller Prinzip (siehe Anhang B) ermöglicht, wobei die Szene ein Element der View ist, und die Objekte durch die Models abgebildet werden.

7.2.1. Szenengraph

Um eine Szene aufzubauen, hat sich aus mehreren Gründen die objektorientierte Datenstruktur Szenengraph angeboten, da mit ihrer Hilfe ermöglicht wird, die räumlichen Anordnungen sowie die Abhängigkeiten der einzelnen Objekte untereinander als Hierarchie darzustellen.
Der Szenengraph entsteht durch die Entwicklung eines Graphen in einer gerichteten, azyklischen Baumstruktur, wobei die Knoten mittels verketteten Listen miteinander adjaziert werden. Die inneren Knoten beschreiben jeweils eine innere Wurzel für weitere Teilbäume, für zusammengesetzte Objekte. Die Wurzel des Baumes beschreibt die ge-

samte Welt. In den Blättern werden die in die Szene zu integrierenden Objekte, in ihren Einzelstücken, mit ihren Daten abgelegt. Ein Objekt steht dabei stets mit seinem Elterobjekt räumlich in Verbindung und bietet das Bezugskoordinatensystem für die Kinder. Die Wurzel besitzt die Teilbäume der Roboter und des Werkstückhalters als direkte Kinder. Bei einer Verschiebung eines Objekts in der Welt, wird diese Bewegung ebenfalls auf allen Kinderobjekten ausgeübt. Die Bewegung wird durch Ändern der Koordinaten einer Wurzel realisiert, daher beinhalten die Knoten eine Transformationsmatrix, um die korrekten Koordinaten bezüglich der Weltkoordinaten bestimmen zu können. Eine Transformationsmatrix wird hierbei durch Quaternionen (siehe Kapitel 2.4) beschrieben, mit denen es ermöglicht wird, die lokalen Koordinaten des Objekts in die Weltkoordinaten zu überführen. Mit Hilfe der Quaternionen kann eine Rotationsvorschrift und eine Translationsvorschrift in eine Vorschrift zu vereint werden, die sonst einzelnen ausgeübt werden müssten. Jedes Objekt befindet sich zu Beginn in einem eigenen Koordinatensystem. Wird ein Knoten bzw. ein Objekt durch Ändern seiner Transformationsmatrix in der Welt verschoben, so werden alle Transformationsmatrizen aller Kinder entsprechend aktualisiert und somit erfolgt die gewünschte Verschiebung der Kinder. Das Verschieben des Tisches soll somit eine passende Verschiebung des Werkstücks mit sich führen, jedoch nicht der Roboter. In der Szene ist das Werkstück beispielsweise ein Kind des Tisches beziehungsweise des Werkstückhalters. Ebenso verhält es sich mit den Robotern, da sich diese aus vielen Einzelteilen zusammensetzen. In Abbildung 7.1 wird der resultierende Szenengraph veranschaulicht, wobei die rot gefärbten Knoten Wurzeln von Teilbäumen darstellen und die gelben die dazugehörigen Blätter.

7.3. Bounding Boxes

Eine Bounding Box ist ein Hüllquader, der ein beliebig geformtes Objekt umschließt. Dabei ist das Volumen der Bounding Box für die zu umhüllenden Objekte größer, als das Volumen der Objekte selbst. Eine Ausnahme davon bildet jedoch eine Bounding Box für einen Quader, da in diesem Fall die Volumina gleich sind. In *RoSi3D* dienen Bounding Boxes dem Zweck, die Kollisionserkennung zu beschleunigen. Diese werden eingesetzt, um sowohl einzelne Teile der Geometriedaten von Szenenobjekten als auch komplette Szenenobjekte selbst zu umhüllen. Bei einem Test auf Kollision zwischen einem, von einer Bounding Box umhüllten, Objekt mit einem anderen Objekt, muß daher in einem ersten Schritt lediglich der Test zwischen der Bounding Box des Objekts und dem anderen Objekt getestet werden. Erst bei einem positiven Testergebnis ist dann ein weiterer Schritt nötig, in dem das tatsächliche Objekt mit dem anderen Objekt auf eine Kollision getestet wird. Im Folgenden werden zwei Arten von Bounding Boxes näher betrachtet, die Axis Aligned Bounding Box und die Oriented Bounding Box. Die Axis Aligned Bounding Box wird in *RoSi3D* im Bereich der Bahnplanung verwendet (siehe Kapitel 5.2.3). Beim Entwurf der Kollisionsdetektion hat sich die Projektgruppe zugunsten des Einsatzes der Oriented Bounding Box entschlossen, da diese die Objekte aufgrund der

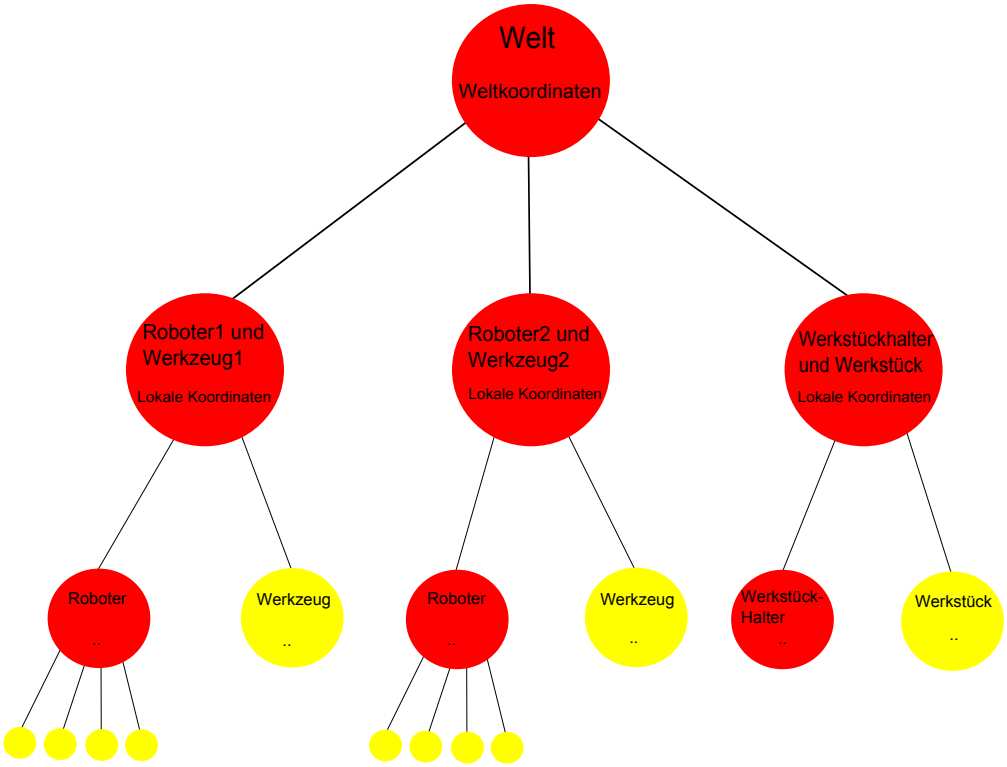
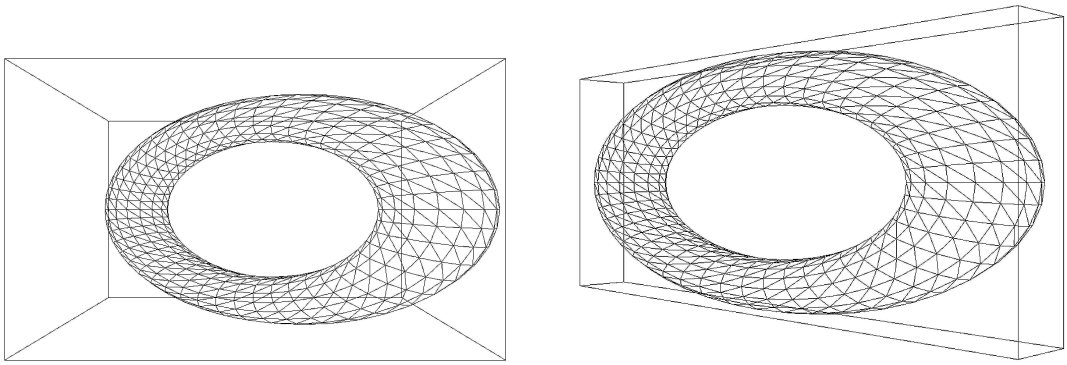


Abbildung 7.1.: Szenengraph



(a) Axis Aligned Bounding Box (b) Oriented Bounding Box

Abbildung 7.2.: Axis Aligned Bounding Box und Oriented Bounding Box im Vergleich

Art ihrer Konstruktion enger umshüllen kann als eine Axis Aligned Bounding Box (siehe Abbildung 7.3.1). Diese Eigenschaft hat den Vorteil, dass beispielsweise für zwei Objekte die mit Axis Aligned Bounding Boxes umhüllt sind, eine Kollision erkannt wird, die bei den gleichen Objekten mit zugeghörigen Oriented Bounding Boxes nicht auftritt. Da jede Kollision von Bounding Boxes im Verlauf der Kollisionserkennung weitere Schnittests nach sich zieht, kann der Einsatz von Oriented Bounding Boxes zu einer Reduzierung der Anzahl der benötigten Schnittests führen.

7.3.1. Axis Aligned Bounding Box

Axis Aligned Bounding Boxes sind Hüllquader, deren Kanten an den Achsen des gegebenen Weltkoordinatensystems ausgerichtet sind. Um entsprechend die Start- und Endpunkte der einzelnen Kanten einer Axis Aligned Bounding Box für ein zu umhüllendes Szeneobjekt festzulegen, muss die Lage der Vertices des Objekts im Raum betrachtet werden. Ein Vertex ist ein dreidimensionaler Punkt im Koordinatensystem und enthält eine X-, Y-, und eine Z-Koordinate. Um die maximale Ausdehnung des Szenenobjektes auf jeder der drei Achsen des Koordinatensystems zu bestimmen, werden die minimalen sowie maximalen X-, Y-, und Z-Koordinaten aller Vertices berechnet. Aus diesen berechneten Koordinaten ergeben sich schließlich die die Start- und Endpunkte der einzelnen Kanten und somit die acht Eckpukte einer Axis Aligned Bounding Box für das Szenenobjekt.

7.3.2. Oriented Bounding Box

Wie Axis Aligned Bounding Boxes sind auch Oriented Bounding Boxes ebenfalls Hüllquader, deren Kanten im Unterschied zur Axis Aligned Bounding Box nicht an den Achsen des gegebenen Koordinatensystems ausgerichtet werden, sondern an der Lage des Objekts im dreidimensionalen Raum ausgerichtet werden. Zu diesem Zweck wird die Verteilung der Vertices eines zu umhüllendes Objekts im Raum betrachtet. Wird eine Achse durch den Schwerpunkt des Objekts gelegt, ist zu erkennen, dass sich eine gewisse Anzahl an Vertices in der Nähe der Achse befindet. Dieser Zusammenhang lässt sich als Streuung der Vertices beschreiben, wobei die Anzahl der Vertices in der Nähe der Achse variiert, je nachdem in welche Richtung die Achse ausgerichtet ist. Demnach kann eine Oriented Bounding Box für ein Objekt ausgerichtet werden, indem zwei Achsen mit folgenden Eigenschaften bestimmt werden:

- Die zwei Achsen sind orthogonal zueinander
- Eine Achse weist eine maximale Streuung in ihre Richtung auf
- Die andere Achse weist eine minimale Streuung in ihre Richtung auf

Um diese Achsen zu finden wird folgendes Verfahren angewandt: Als erstes wird der Schwerpunkt des zu umhüllendes Objekts mit folgender Formel berechnet:

$$\mu = \frac{1}{3n} \sum_{i=0}^{n} (\mathbf{p}_i + \mathbf{q}_i + \mathbf{r}_i) \tag{7.1}$$

wobei μ der Schwerpunkt des Objekts ist, n die Anzahl der Dreiecke des Objekts beschreibt und $\mathbf{p}$, $\mathbf{q}$ sowie $\mathbf{r}$ die Vertices eines Dreiecks repräsentieren.

Danach wird ein in [GLM96] und [DH73] beschriebenes Verfahren verwendet, welches aus den Positionen der Vertices im dreidimensionalen Raum die Achsen berechnet. Als Eingabe benötigt dieses Verfahren eine sogenannte Kovarianzmatrix. Eine Kovarianz ist ein Maß für die Richtung des Zusammenhangs zweier Merkmale X und Y (siehe [Sac07]). Für Wertepaare (x_1, x_n) und (y_1, y_n) ist diese definiert als:

$$Cov\,(X, Y) = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})\,(y_i - \bar{y}) \tag{7.2}$$

wobei n die Anzahl der Wertepaare repräsentiert und $\bar{x}$ sowie $\bar{y}$ für die Mittelwerte aller x und y stehen.

Für die Berechnung der gesuchten Achsen werden die Kovarianzen für alle Wertepaarkombinationen der X-, Y- und Z-Koordinaten der Vertices mit dem Schwerpunkt als Mittelwert in die Kovarianzmatrix eingetragen:

$$(\mathbf{c}_{j,k}) = \frac{1}{3n} \sum_{i=0}^{n} \left(\left(\mathbf{p}_j^i - \mu\right) \left(\mathbf{p}_k^i - \mu\right) + \left(\mathbf{q}_j^i - \mu\right) \left(\mathbf{q}_k^i - \mu\right) + \left(\mathbf{r}_j^i - \mu\right) \left(\mathbf{r}_k^i - \mu\right) \right), 1 \leq j, k \leq 3 \tag{7.3}$$

Dabei beschreiben j und k die Indizes der Matrix, n die Anzahl der Dreiecke des Objekts, μ der Schwerpunkt und schließlich repräsentieren $\mathbf{p}$, $\mathbf{q}$ und $\mathbf{r}$ entsprechende Spaltenvektoren für die Vertices des Objekts.

Die Kovarianzmatrix entspricht einem linearen Gleichungssystem mit n Gleichungen und n Unbekannten, daher wird auf diese Matrix das Jacobi-Verfahren (siehe [New08]) angewendet um die Eigenvektoren zu bestimmen. Eigenvektoren sind vom Nullvektor verschiedene Vektoren, deren Richtung durch eine Abbildung zum Beispiel

$$f : V \longrightarrow V, V \in \mathbb{R}^3 \tag{7.4}$$

nicht ändert. Die Abbildung kann dabei lediglich die Länge des Eigenvektors ändern (siehe [BS96]). Die Eigenvektoren dienen als Basis eines neuen Koordinatensystems, in das die Vertices des Objekts transformiert werden. In diesem Koordinatensystem werden die Eckpunkte einer Axis Aligned Bounding Box für das Objekt bestimmt. Diese Eckpunkte werden schließlich in das ursprüngliche Koordinatensystem transformiert, in dem diese jetzt die Eckpunkte der Oriented Bounding Box repräsentieren (siehe [Got00]).

7.4. Die Datenstruktur Octree

Octrees sind hierarchische Suchdatenstrukturen, um Objekte im dreidimensionalen Raum zu unterteilen und stellen eine Erweiterung der Quadtrees dar, welche die gleiche Aufgabe für zweidimensionale Objekte erfüllen (siehe [FvDF+94]). In *RoSi3D* wird jedes Objekt einer Szene durch einen Octree repräsentiert, daher muss dementsprechend für jedes Objekt ein eigener Octree erstellt und mit den Geometriedaten des Objekts gefüllt werden. Ein Octree besteht aus einem Wurzelknoten, des Weiteren kann er Knoten mit bis zu acht Kindern und Knoten ohne Kinder, sogenannten Blättern, enthalten. In der Realisierung der Projektgruppe können in jedem Knoten Geometriedaten gespeichert werden, zusätzlich besitzt jeder Knoten eine Oriented Bounding Box (siehe Kapitel 7.3.2), welche alle in dem Knoten gespeicherten Geometriedaten sowie die Geometriedaten seiner Nachfolgeknoten umschließt. Ein Beispiel für einen Octree ist in Abbildung 7.5 gegeben.

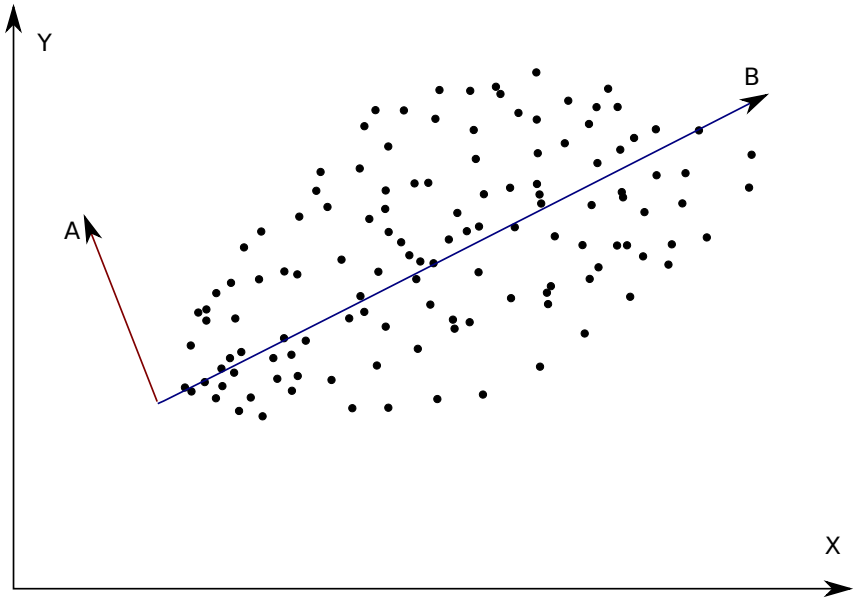


Abbildung 7.3.: Erzeugung der Achsen anhand der Streuung der Vertices

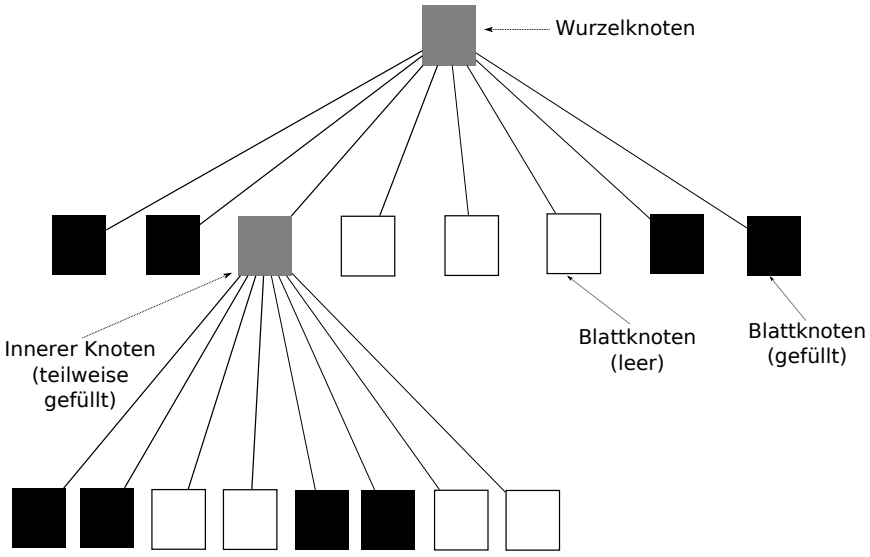


Abbildung 7.4.: Ein Octree-Beispiel

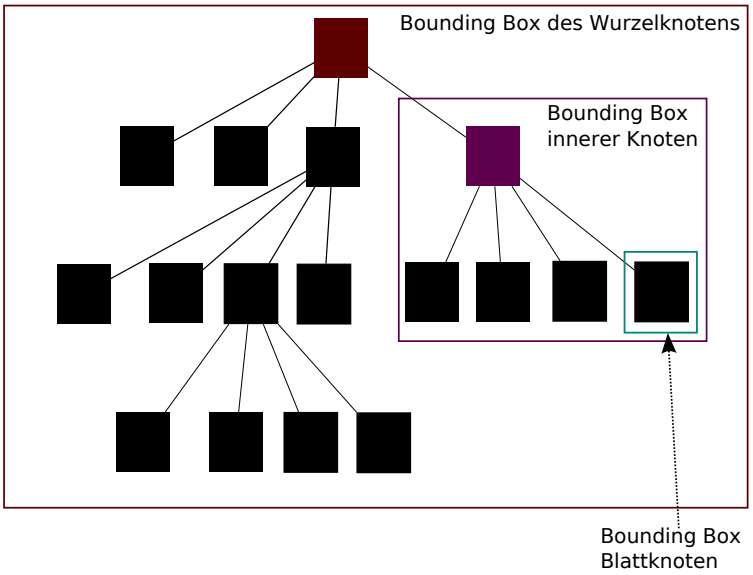


Abbildung 7.5.: Ein Quadtree mit Beispielen für Bounding Boxes

7.4.1. Erzeugen von Octrees in RoSi3D

Beim Erzeugen eines Octrees für ein Objekt werden die Geometriedaten in den Octree eingefügt. Dabei gilt für alle Knoten des Octrees, dass die Anzahl der Geometriedaten, die in einem einzelnen Knoten gespeichert werden können, durch einen festgelegten Schwellwert begrenzt ist. So wird bei der Erzeugung als erstes einmalig ein Wurzelknoten mit der entsprechenden Oriented Bounding Box erstellt. Danach wird die Anzahl Geometriedaten des Objekts mit dem festgelegten Schwellwert verglichen. Wird der Schwellwert für den Wurzelknoten nicht überschritten, werden die Geometriedaten dem Knoten zugeordnet. Ansonsten wird die Oriented Bounding Box des Wurzelknotens mittels Ebenen, die an den Außenseiten der Oriented Bounding Box ausgerichtet sind und durch den Schwerpunkt der Bounding Box verlaufen in acht gleichgroße Teilbereiche unterteilt (siehe Abbildung 7.6). Anschließend werden für den Wurzelknoten bis zu acht Kindknoten erzeugt, jeder Kindknoten ist dabei einem Teilbreich der unterteilten Box des Wurzelknotens zugeordnet und bekommt die im Teilbereich befindlichen Geometriedaten zugewiesen. Befinden sich in einem Teilbereich keine Geometriedaten, wird auch kein Kindknoten für diesen Teilbereich erstellt. Die Größe und Ausrichtung der Oriented Bounding Box jedes Kindknotens wird schließlich anhand der zugewiesenen Geometriedaten aus den Teilbereichen ermittelt. Dabei ist der Sonderfall zu beachten, dass sich die Punkte eines Dreiecks der Geometriedaten in verschiedenen Teilbereichen befinden. In diesem Fall wird das Dreieck dem Elterknoten zugeordnet. Nachdem die Kindknoten angelegt wurden, wird auch für diese nach dem hier beschriebenen Verfahren überprüft, ob

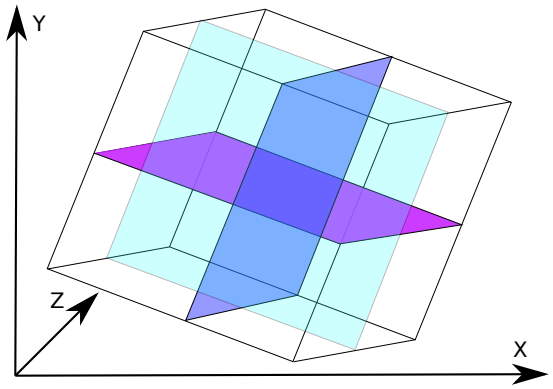


Abbildung 7.6.: Die Oriented Bounding Box eines Knotens wird mittels Ebenen in acht Teilbereiche unterteilt

auf Grund der Anzahl der Geometriedaten weitere Kindknoten angelegt werden müssen. Das Verfahren wird solange angewendet, bis alle Geometriedaten des Objekts auf Knoten des Octrees verteilt sind. Die dabei entstandene Datenstruktur ist in Abbildung 7.5 dargestellt. Zur Veranschaulichung wird der Octree durch einen Quadtree ersetzt.

7.4.2. Schnittest zwischen Objekten einer Szene

Zwei durch Octrees repräsentierte Szenenobjekte werden auf Kollision überprüft, indem zuerst die Oriented Bounding Boxes der beiden Wurzelknoten auf Kollision getestet werden. Zu diesem Zweck wird der bereits im Grundlagenkapitel erläuterte Quader-Quader Schnittest eingesetzt (siehe Kapitel 2.3.6). Existiert kein Schnitt zwischen den beiden Boxen, gibt es auch keine Kollision zwischen den beiden Objekten und der Kollisionstest kann an dieser Stelle abgebrochen werden. Liegt ein Schnitt vor, werden alle Dreiecke, die in den beiden Knoten enthalten sind für jeden Octree separat gespeichert. Danach werden alle Oriented Bounding Boxes der Kindknoten des einen Octrees mit allen Oriented Bounding Boxes der Kindknoten des anderen Octrees nach dem gleichen Verfahren auf Kollison überprüft und gegebenenfalls weitere noch auf Kollision zu testende Dreiecke für jeden Octree abgespeichert. Schließlich wird das Verfahren für alle Kindknoten fortgeführt, deren Oriented Bounding Boxes eine Kollision aufwiesen. Auf diese Weise werden die beiden Octrees traversiert, bis jeweils ein Blatt in beiden Bäumen erreicht wird. Nachdem für jeden Octree alle Dreiecke herausgefiltert wurden, die potentiell eine Kollision verursachen, werden all jene Dreiecke des einen Octrees mit allen Dreiecken des anderen Octrees auf Kollision überprüft. Zu diesem Zweck kommt der ebenfalls im Grundlagenkapitel erläuterte Dreieck-Dreieck-Schnittest zum Einsatz (siehe Kapitel 2.3.5). Fällt auch dieser Test für alle Dreieckspaare negativ hinsichtlich einer Kollision aus, sind die beiden Objekte frei von Kollisionen. Abbildung 7.7 soll dabei als Beispiel für die Baumtraversierung dienen. Auch in diesem Beispiel werden die Octrees durch

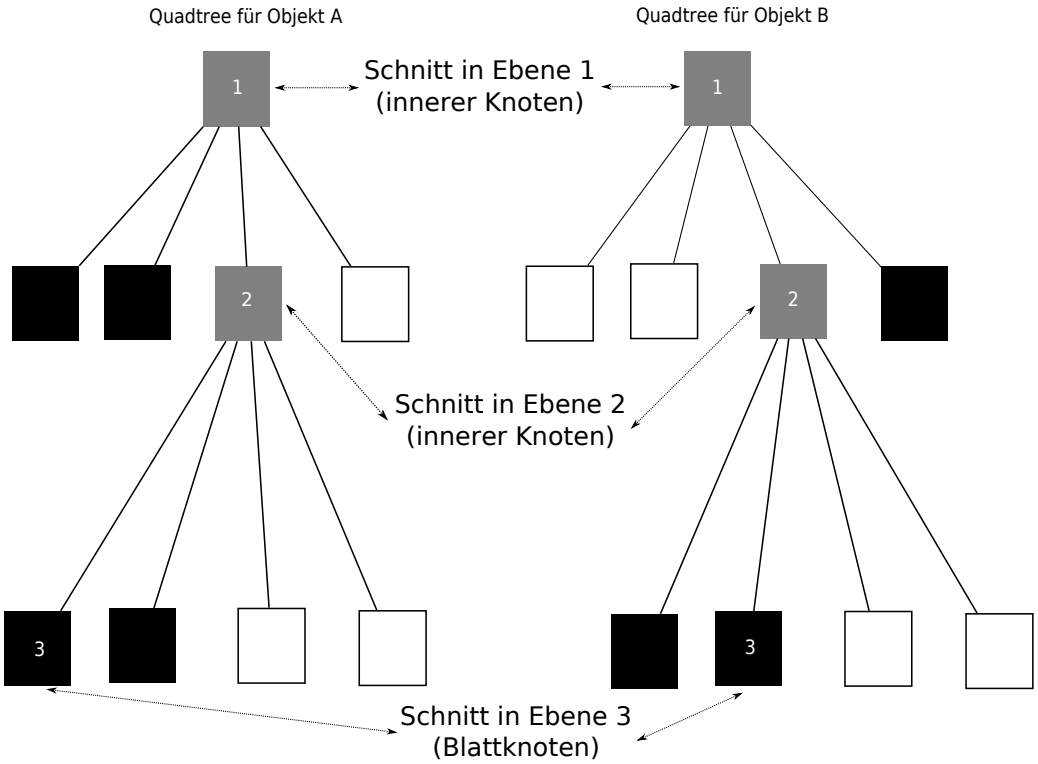


Abbildung 7.7.: Kollisionserkennung mit Quadtrees

Quadtrees repräsentiert.

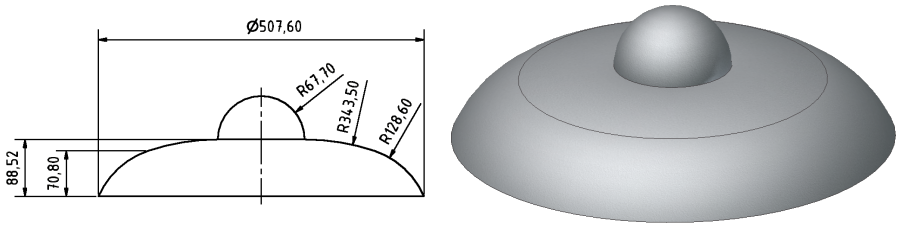
8. Evaluierung

Dieses Kapitel präsentiert und erläutert die Ergebnisse der Evaluation der Software *Rosi3D*. Dabei wurden sowohl der reale Arbeitsprozess in einer Roboterzelle, auf Basis der Berechnungen von *Rosi3D*, als auch einzelne Zwischenschritte bei der Berechnung untersucht, die von der Projektgruppe als essentiell für die Qualität des Endergebnisses angesehen wurden. Zu den betrachteten Teilverfahren gehören die Netzoptimierung und Segmentierung der Werkstückgeometrie, deren Ergebnisse in den Kapiteln 8.1.1 und 8.1.2 vorgestellt und bewertet werden, sowie die Pfadberechnung in Kapitel 8.1.3. Im darauf folgenden Kapitel 8.1.4 wird die Laufzeit der Berechnungen von *RoSi3D* analysiert. In Kapitel 8.2.1 folgt die Beschreibung des Aufbaus für die durchgeführten Versuche in der Roboterzelle, deren Ergebnisse in Kapitel 8.2.2 erläutert werden. Sowohl bei der Softwareevaluierung, als auch im realen Versuch wurden regelmäßige geometrische Körper, wie beispielsweise Würfel oder Torus, sowie ein von der Projektgruppe definierter Demonstrator verwendet. Eine technische Zeichnung und ein entsprechendes 3D-Modell des Demonstrators sind in Abbildung 8.1 dargestellt.

8.1. Evaluierung der Software RoSi3D

8.1.1. Netzoptimierung

Zur Beurteilung der Qualität der Netzoptimierung wurden die Netzeigenschaften verschiedener Werkstückgeometrien in Form von Freiformflächen nach der Triangulierung



(a) Technische Zeichnung (b) 3D-Modell

Abbildung 8.1.: Technische Zeichnung und 3D-Modell des Demonstrators

	OCC				Umvernetzt			
3D-Modell	**Dreiecke**	**Max.**	**∅**	**Var.**	**Dreiecke**	**Max.**	**∅**	**Var.**
Demonstrator	1342	50	5,316	3,718	36508	10	5,998	1,473
Würfel	12	3	2,5	0,25	1400	8	5,983	0,567
Torus	1900	8	5,746	1,06	269952	10	6	2,97

Tabelle 8.1.: Knotengrad nach der Triangulierung durch OpenCASCADE und nach der Umvernetzung

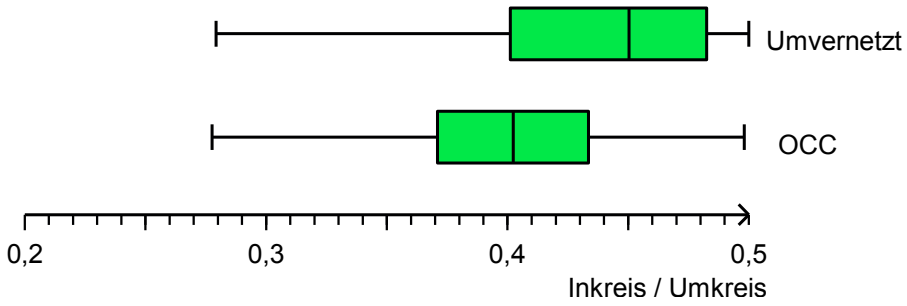


Abbildung 8.2.: Box-Whisker-Diagramm des Verhältnisses von Inkreis zu Umkreis bei der Triangulierung durch OCC und nach der Umvernetzung

durch OpenCASCADE (OCC), sowie nach der durchgeführten Umvernetzung (siehe Kapitel 2.2.2) untersucht. Die betrachteten Eigenschaften sind der Knotengrad und das Verhältnis von Inkreis zu Umkreis der Dreiecke der aus der Werkstückgeometrie erstellten Netze. Tabelle 8.1 gibt eine Übersicht zur Untersuchung bezüglich des Knotengrades. Angegeben sind die Gesamtanzahl aller Dreiecke des untersuchten Netzes, der maximale und durchschnittliche Knotengrad, sowie die Varianz der Messwerte vom durschnittlichen Knotengrad. Die Verhältnisse von Inkreis zu Umkreis der Dreiecke eines von OpenCASCADE triangulierten Netzes und des selben Netzes nach dem Umvernetzungsvorgang sind in Abbildung 8.2 in einem Box-Whisker-Diagramm dargestellt. Dieses Verhältnis gibt Auskunft über das Auftreten spitzer Winkel in einem Dreieck. Im optimalen Fall, einem gleichseitigen Dreieck mit Innenwinkeln von je 60°, läge das Verhältnis von Inkreis zu Umkreis bei dem Wert 0,5.

Bei einfachen Geometrien mit ebenen Flächen sind die Ergebnisse der Triangulierung durch OpenCASCADE für eine weitere Verwendung in Bahn- und Bewegungsplanung geeignet. Bei komplexeren Geometrien, die auch gekrümmte Flächen enthalten, empfiehlt sich der Einsatz des Umvernetzungsalgorithmuses, da sich somit wie aus Tabelle 8.1 ersichtlich, hohe maximale Knotengrade reduzieren lassen. Ein weiterer positiver Effekt ist die Tatsache, dass der durchschnittliche Knotengrad nach der Umvernetzung stets dem optimalen Grad sechs (siehe Kapitel 2.2.2) stark angenähert ist (siehe Tabelle

Winkeldifferenz	Anzahl Patches	Laufzeit in Sekunden
1	675	8,0
2	414	5,3
3	141	3,8
4	141	3,8
5	84	2,5
6	89	2,4
7	75	2,3
8	19	1,8
9 - 84	5	1,4 - 1,5
85 und mehr	1	0,9

Tabelle 8.2.: Segmentierung des Demonstrators

Winkeldifferenz	Anzahl Patches	Laufzeit in Sekunden
1 - 89	6	1,9
90	4	1,9
91 und mehr	1	1,9

Tabelle 8.3.: Segmentierung eines Würfels

8.1). Des Weiteren wird die Häufigkeit sehr spitzer Innenwinkel in den Dreicken, welche zu numerischen Ungenauigkeiten führen könnten, reduziert (siehe Abbildung 8.2). Generell ist anzumerken, dass für ein gutes Ergebnis der Umvernetzung strukturerhaltende Kanten manuell definiert und geeignete Parameter vom Benutzenden gewählt werden müssen, welches eine gute Kenntnis des Netzeditors (siehe Kapitel C.6.4) voraussetzt.

8.1.2. Segmentierung

Um das zur Segmentierung des Werkstücks verwendete Bereichswachstumsverfahren (inklusive des Post-Processing Schritts zum Entfernen von Patch-Löchern) zu evaluieren, wurden die durch das Umvernetzen mit einer mittleren Kantenlänge von $10mm$ entstandenen Dreiecksnetze zweier Testobjekte – einem Würfel und dem Demonstrator (siehe Abbildung 8.1) – mehrmals segmentiert. Dabei wurde der Parameter der zulässigen Winkeldifferenz für adjazente Dreiecksfacetten zu Beginn auf 1° festgesetzt und mit jeder weiteren Segmentierung um 1° erhöht. Nach jeder Segmentierung wurde die Anzahl der Patches, sowie die Laufzeit des Algorithmus notiert, um den Einfluss des Parameters Winkeldifferenz auf Qualität und Geschwindigkeit der Segmentierung festzustellen. Die Ergebnisse sind in den Tabellen 8.2 und 8.3 zusammengefasst.

Es hat sich gezeigt, dass für kleine Werte für die Winkeldifferenz bei Objekten, die gekrümmte Oberflächen aufweisen, wie beispielsweise dem Demonstrator, eine Segmen-

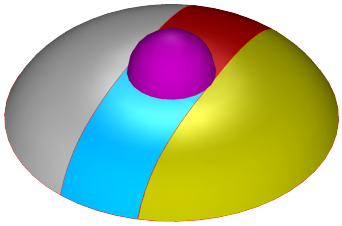


Abbildung 8.3.: Automatische Segmentierung des Demonstrators

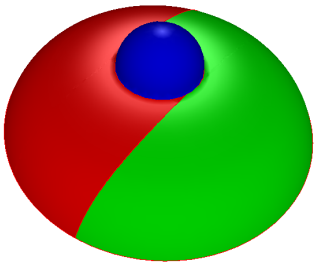


Abbildung 8.4.: Manuelle Segmentierung des Demonstrators

tierung in weitaus mehr Patches als gewünscht, stattfindet (siehe Tabelle 8.2), während Objekte, die nur aus ebenen Flächen bestehen auch für eine Winkeldifferenz von 1° zufriedenstellend segmentiert werden (siehe Tabelle 8.3). Da die Normalenvektoren von Dreiecken, die eine ebene Fläche bilden, alle identisch sind, war dies zu erwarten. Ab einem bestimmten Wert für die Winkeldifferenz (bei Demonstrator 1 ab 85° und bei Demonstrator 2 ab 90°) werden alle Dreiecke dem gleichen Patch zugeordnet.

Bezüglich der Laufzeit lässt sich feststellen, dass im Falle des Demonstrators mit zunehmender Winkeldifferenz und somit mit abnehmender Anzahl der Patches auch die Laufzeit geringer wird. Bei dem Würfel hingegen ist die Laufzeit konstant. Der Grund hierfür dürfte sein, dass im Falle des Würfels im Gegensatz zu dem Demonstrator keine Patch Löcher gefunden werden und – wie im Kapitel Laufzeitanalyse (siehe Kapitel 8.1.4) festgestellt wird – der Post-Processing Schritt des Patch Löcher Entfernens den Grossteil der für die Segmentierung benötigten Laufzeit ausmacht.

Ein weiterer Schritt zur Evaluierung des Ergebnisses der Segmentierung war der Vergleich der automatisch erzeugten Segmentierung mit der manuell von einem Teilnehmer der Projektgruppe erstellten Segmentierung im Hinblick auf die weitere Verarbeitung im Rahmen der Pfadberechnung. Für den Würfel war die Segmentierung identisch (jede Seite des Würfels wurde ein Patch), für den Demonstrator hingegen war die automatisch erstellte Segmentierung etwas schlechter als die manuelle: fünf Patches im Falle der automatischen Segmentierung (siehe Abbildung 8.3) und drei Patches bei der manuellen Segmentierung (siehe Abbildung 8.4).

8.1.3. Pfadberechnung

Ein Ansatz zur Beurteilung der berechneten Pfade war die Bestimmung des prozentualen Anteils der Interpatchverbindungen an der Gesamtbahn. Außerdem wurde die implementierte TSP Heuristik mit der Lösung durch Ausprobieren aller Möglichkeiten verglichen.

Anteil der Interpatchverbindungen an der Gesamtbahn

Für den Demonstrator und einen Würfel wurden jeweils die Summe der Länge der Interpatchverbindungen, sowie die Länge der Gesamtbahn berechnet und anschliessend bestimmt, wie gross der prozentuale Anteil der Interpatchverbindungen an der Gesamtbahn ist. Die Ergebnisse sind in Tabelle 8.4 zusammengefasst. Obwohl die Segmentierung des Würfels nur zu einem Patch mehr führt als die Segmentierung des Demonstrators, ist der prozentuale Anteil der Interpatchverbindungen an der Gesamtbahn mehr als doppelt so gross. Folglich ist der prozentuale Anteil der Interpatchverbindungen an der Gesamtbahn stark von der Geometrie des Werkstücks abhängig. Dies gilt für beide für die Interpatchverbindungen zur Verfügung stehenden Strategien. Eine weitere aus Tabelle 8.4 ablesbare Beobachtung ist die Tatsache, dass sowohl für den Demonstrator als auch für den Würfel, die Bounding Box-Methode zu kürzeren Interpatchverbindungen führt als die Hemisphären-Methode.

TSP

Zum Testen des TSPs wurden synthetische Daten generiert, um beliebig viele Patches zu simulieren. Eine garantiert optimale Rundreise kann aufgrund der eingesetzten Heuristiken nicht gefunden werden, jedoch ist eine gefundene Rundreise nachweislich nie teurer als das Doppelte der Optimalen (siehe Kapitel 5.2.2). Bei bis zu fünf Patches wird durch Berechnen aller Lösungen die Optimale bestimmt. Bei mehr als fünf Patches werden die Heuristiken eingesetzt. Die Berechnung einer Rundreise für 50 Patches bzw. 100 Knoten erfolgt somit bereits in wenigen Sekunden, bei 1000 Knoten in ca. 20 Minuten. Jedoch werden 45 Sekunden benötigt, um eine minimale Rundreise für fünf Patches bzw. zehn Knoten zu berechnen, da hier die optimale Lösung bestimmt wird. Zum Vergleich: Für acht Patches läge der Zeitaufwand für das Bestimmen der optimalen Lösung bei ca. 20 Jahren. Die Tabellen 8.5 und 8.6 stellen die Laufzeit für das Ausprobieren aller Lösungen der Laufzeit der eingesetzten Heuristiken gegenüber, um die Zeitersparnis zu verdeutlichen.

(a) Demonstrator mit Hemisphären-Methode

Anzahl Patches	5
Werkstückbahn (in mm)	10884
Interpatchverbindungsbahn (in mm)	1382
Gesamtbahn (in mm)	12267
Anteil der Interpatchverb. in %	11

(b) Würfel mit Hemisphären-Methode

Anzahl Patches	6
Werkstückbahn (in mm)	11681
Interpatchverbindungsbahn (in mm)	3770
Gesamtbahn (in mm)	15451
Anteil der Interpatchverb. in %	24

(c) Demonstrator mit Bounding Box-Methode

Anzahl Patches	5
Werkstückbahn (in mm)	10884
Interpatchverbindungsbahn (in mm)	568
Gesamtbahn (in mm)	11452
Anteil der Interpatchverb. in %	5

(d) Würfel mit Bounding Box-Methode

Anzahl Patches	6
Werkstückbahn (in mm)	11681
Interpatchverbindungsbahn (in mm)	1832
Gesamtbahn (in mm)	13513
Anteil der Interpatchverb. in %	14

Tabelle 8.4.: Ergebnisse der Pfadberechnung

Anzahl Patches	**Laufzeit (in Sekunden)**
3	2
4	9
5	45
6	7280

Tabelle 8.5.: Laufzeiten (in Sekunden) für synthetische Daten bei optimaler Lösung

Anzahl Patches	Laufzeit (in Sekunden)
3	1
4	1
5	2
6	5
7	7
50	78
500	1263

Tabelle 8.6.: Laufzeiten (in Sekunden) für synthetische Daten bei Nutzung der Heuristiken

Anzahl Dreiecke	Segmentierung	Bahnberechnung	Bewegungsberechnung
3072	1,06	2,64	164,21
12288	3,86	9,26	168,37
49200	14,98	34,23	249,11

Tabelle 8.7.: Laufzeiten (in Sekunden) für den Würfel

8.1.4. Laufzeitanalyse

Um die Laufzeit der Kernkomponenten von *Rosi3D* in Abhängigkeit von der Auflösung des Dreiecksnetzes zu bestimmen, wurden der Würfel und der Demonstrator jeweils drei Mal umvernetzt. Für die mittlere Kantenlänge wurden dabei die Parameterwerte $5mm$, $10mm$ und $20mm$ verwendet, wodurch sich für den Würfel Dreiecksnetze mit 3072, 12288 und 49200 Facetten und für den Demonstrator Dreiecksnetze mit 1302, 5010 und 20326 Facetten ergaben. Für jedes der sechs resultierenden Dreiecksnetze wurde die Bahn- und Bewegungsberechnung durchgeführt und die Laufzeit der Module Segmentierung, Bahnberechnung und Bewegungsplanung gemessen. Des Weiteren wurde die Laufzeit der Teilkomponenten dieser Module gemessen, für die Segmentierung war dies das Bereichswachstum und der Algorithmus zum Entfernen von Patch-Löchern, für die Bahnberechnung die Erzeugung von Zick-Zack-Bahnen, sowie der implementierte TSP-Algorithmus. Die Bewegungsplanung wurde sowohl mit aktivierter Kollisionsdetektion als auch ohne durchgeführt, um den Einfluss selbiger auf die Laufzeit festzustellen.

Die Tabellen 8.7 und 8.8 zeigen die Laufzeiten der Module Segmentierung, Bahnberechnung und Bewegungsplanung. Es fällt auf, dass der Faktor um den die Laufzeit der Segmentierung ansteigt ähnlich gross ist wie der Faktor um den die Anzahl der Dreiecke ansteigt. Eine solche Korrelation lässt sich für den Würfel auch bei der Bahnberechnung feststellen. Für den Demonstrator hingegen scheint die Anzahl der Dreiecke keinen signifikanten Einfluss auf die Laufzeit der Bahnberechnung zu haben. Da der Demonstrator in fünf Patches segmentiert wird und somit anstelle der TSP Heuristiken, die optimale

Anzahl Dreiecke	Segmentierung	Bahnberechnung	Bewegungsberechnung
1302	0,73	85,65	114,12
5010	2,56	87,78	119,46
20326	10,09	87,53	129,54

Tabelle 8.8.: Laufzeiten (in Sekunden) für den Demonstrator

Bereichswachstum	Patch Löcher entfernen	Gesamt
0,96	1,60	2,56

Tabelle 8.9.: Laufzeit (in Sekunden) der Komponenten der Segmentierung

Lösung durch Ausprobieren aller Möglichkeiten bestimmt wird, liegt die Vermutung nahe, dass ein Grossteil der Laufzeit der Bahnberechnung auf die Bestimmung der Lösung des TSP entfällt, da die Anzahl der Dreiecke darauf keinen Einfluss hat. Die Laufzeit der Bewegungsberechnung steigt für beide Testobjekte relativ langsam in Relation zum Anstieg der Anzahl der Dreiecke.

Ein weiterer Schritt war die Bestimmung des jeweiligen Anteils der Teilkomponenten an der Gesamtlaufzeit der einzelnen Module. So zeigen die Tabellen 8.9 und 8.10 exemplarisch, wie sich die Laufzeiten zusammensetzen für den Demonstrator, welcher mit einer mittleren Kantenlänge von $10mm$ umvernetzt wurde. Aus Tabelle 8.9 geht hervor, dass der Post-Processing Schritt des Patch Löcher Entfernens mehr Zeit in Anspruch nimmt als die eigentliche Segmentierung. Tabelle 8.10 macht deutlich, dass der TSP-Algorithmus den Großteil der Laufzeit der Bahnberechnung ausmacht, die im vorigen Absatz aufgestellte Vermutung hat sich also bestätigt.

Tabelle 8.11 zeigt, dass die Laufzeit der Bewegungsplanung mit aktivierter Kollisionsdetektion signifikant grösser ist als bei deaktivierter Kollisionsdetektion.

Bahnberechnung pro Patch	TSP Algorithmus	Gesamt
0,55	87,23	87,78

Tabelle 8.10.: Laufzeit (in Sekunden) der Komponenten der Bahnberechnung für den Demonstrator

Bewegungsberechnung ohne Kollisionsdetektion	mit Kollisionsdetektion
4,64	119,46

Tabelle 8.11.: Laufzeit (in Sekunden) der Bewegungsberechnung mit und ohne Kollisionsdetektion

(a) (b)

Abbildung 8.5.: Roboterzelle

8.2. Versuche in einer realen Roboterzelle

8.2.1. Versuchsaufbau

Zur praktischen Überprüfung der Leistung von *RoSi3D* konnte die Projektgruppe eine vorhandene Roboterzelle am Institut für Roboterforschung der Technischen Universität Dortmund nutzen. Diese besteht aus einem Roboter des Typs ABB IRB 4400/60 und einem zweiten Roboter des Typs ABB IRB 2400/16 (siehe Kapitel 4.3). Die Basen der Roboter sind ca. 2,77 Meter voneinander entfernt und zueinander um ca. 45° geneigt. Als Halterung für den Demonstrator wird ein Tisch verwendet, der zwischen den Robotern platziert ist. Einen Überblick über die Anordnung der genannten Komponenten innerhalb der Zelle gibt Abbildung 8.5.

Am Roboter des Typs ABB IRB 4400/60 wurde ein Sprühwerkzeug der Firma Walther Pilot [Wal08], Modell PILOT WA 600, angebracht, zur Durchführung des Arbeitsvorganges Sprühen. Zur Trocknung wurde der zweite Roboter mit einem Heißluftgebläse der Firma Steinel [Ste08], Modell HL 1502 S, ausgerüstet (siehe Abbildung 8.6).

(a) Spritzpistole (b) Heißlufttrockner

Abbildung 8.6.: Werkzeuge

(a) Gussform (b) Demonstrator aus Gips

Abbildung 8.7.: Demonstrator für den Realversuch

Für den Materialauftrag wurde aus arbeitsschutztechnischen Gründen Lebensmittelfarbe auf Wasserbasis verwendet. Zur Herstellung des Demonstrators wurde eine Gussform erstellt, um diesen aus Gips gießen zu können (siehe Abbildung 8.7). Die Materialeigenschaften des verwendeten Gips waren ausreichend, um das verwendete Farb-Wasser-Gemisch aufzunehmen, ohne das Ergebnis durch ein Verlaufen der Flüssigkeit auf der Oberfläche zu stark zu verfälschen.

Vor Durchführung des Arbeitsprozesses wurde ein Teilbereich der Oberfläche des Demonstrators als Bearbeitungsfläche definiert. Diese entspricht einem Achtel der Gesamtoberfläche des Demonstrators. Für ein Modell dieser Teilfläche (siehe Abbildung 8.8) wurde mit *RoSi3D* ein RAPID-Programm berechnet, mit dem Ziel, die gesamte Oberfläche des definierten Teilbereiches mit Farbe abzudecken.

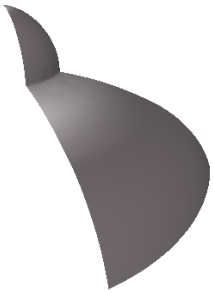


Abbildung 8.8.: Modell des zu besprühenden Bereichs des Demonstrators

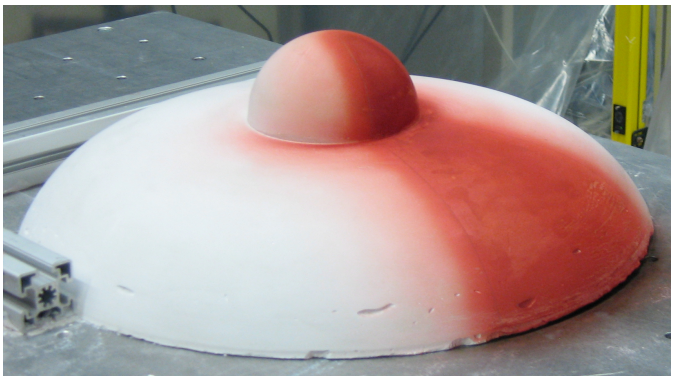


Abbildung 8.9.: Ergebnis des Farbauftrags im Versuch

8.2.2. Versuchsergebnisse und Bewertung

Das durch die Software *RoSi3D* erstellte RAPID-Programm konnte in *RobotStudio* geladen werden und ein Test auf Kollisionen ergab, wie schon zuvor in *RoSi3D*, dass ein Abfahren der Bahn ohne Kollisionen möglich ist. Das Ziel des Farbauftrags auf der gesamten definierten Teilfläche des Demonstrators wurde erreicht (siehe Abbildung 8.9). Der Farbauftrag außerhalb der gewünschten Oberflächenregion in unterschiedlichen Bereichen ergibt sich aus zwei wesentlichen Faktoren. Zum einen kommt es an den Rändern der Fläche zu einem Auftrag auf den angrenzenden Oberflächenbereichen. Dieser ergibt sich aus dem berechneten Pfadabstand zu den Außenkanten der Bearbeitungsfläche (siehe Kapitel 5.2) und ist beabsichtigt, um zu sichern, dass innerhalb der Bearbeitungsfläche die Oberfläche vollständig mit Farbe bedeckt ist. Der Farbauftrag um die obere Halbkugel des Demonstrators ist eine Folge der Kombination der Oberflächenbeschaffenheit des Demonstrators, des Farb-Wasser-Gemisches und des produzierten Luftstroms des Werkzeuges. Das Sprühmittel kann an der Kante der Halbkugel nicht vollständig aufgenommen werden und wird durch den Luftdruck des Sprühstrahls in Richtung dieser Kante auf der Oberfläche verteilt.

9. Fazit

In diesem Kapitel folgt eine kurze Zusammenfassung des Projekts, sowie eine kritische Betrachtung des erzielten Resultats. Abschließend werden in dem Unterkapitel Ausblick (siehe Kapitel 9.2) einige Erweiterungsmöglichkeiten der entwickelten Software aufgeführt.

9.1. Zusammenfassung und Bewertung

Zu Beginn des Projekts wurden im Rahmen des Pflichtenhefts (siehe Anhang D) die Anforderungen an die zu entwickelnde Software festgelegt. Dabei stellten sich bereits die Szenenbeschreibung, die Bahn- und Bewegungsplanung, sowie die Kollisionsdetektion als Kernkomponenten heraus. Die Visualisierung der Szene und der Roboterbewegung galt als optional, wurde jedoch ein wichtiger Bestandteil der graphischen Benutzeroberfläche. Der Schwerpunkt der Entwicklung lag in der Bahnplanung und der kollisionsfreien Bewegungsplanung, da diese zwei Punkte essentiell für einen erfolgreichen Sprühvorgang sind. Dementsprechend wurde die meiste Zeit in die Verwirklichung dieser Ziele investiert. Dennoch besteht in diesen Teilbereichen noch Spielraum für zusätzliche Realisierungen, um das Erzielte einerseits weiter zu optimieren und andererseits zu erweitern (siehe Kapitel 9.2).

Da die Projektgruppenteilnehmer für viele relevante Teile des Projekts keine praktischen Erfahrungen besaßen, insbesondere bezüglich der Programmierung von Robotern, wurden mehrere Entwürfe der Bahn- und Bewegungsplanung, sowie einer passenden Segmentierung des Werkstücks entworfen und wieder verworfen. So wurden zum Beispiel in der Bewegungsplanung mehrere Ansätze zur Suche von kreisbogenförmigen Segmenten ausprobiert bis sich das in Kapitel 6.1.2 beschriebene Verfahren als praktikabelste Lösung erwies. Bei der Bahnplanung wurde der ursprüngliche Ansatz der Segmentierung mittels des Meanshift-Verfahrens wieder verworfen, da sich das Bereichswachstumsverfahren als zweckmäßiger in Bezug auf die Anzahl der generierten Patches herausstellte.

Nach der Implementierungs- und Debugphase wurde das erzeugte RAPID-Programm letzlich auf die Roboter übertragen und der Sprühvorgang durchgeführt. Das Ergebnis des Sprühvorgangs und insbesondere die erzielte Farbüberdeckung des Werkstücks dienten der Evaluierung (siehe Kapitel 8) von *RoSi3D*.

In Bezug auf die oben genannten Kernkomponenten, sowie dem eigentlichen Sprühvorgang wurden alle im Pflichtenheft unter dem Punkt Musskriterien (siehe Kapitel D.3) definierten Zielvorgaben der Projektgruppe erreicht. Die für den praktischen Einsatz relevanten Funktionen von *RoSi3D*, wie zum Beispiel die Auswahl und Positionierung der Roboter, das Einladen eines Werkstücks im IGES, STEP oder PLY Format, die Wahl des Arbeitsprozesses, sowie die Bahn- und Bewegungsplanung stehen dem Nutzer in der graphischen Oberfläche klar strukturiert zur Verfügung. Ferner ist ein Laden einer gesamten Szene, sowie dessen Persistierung in vollem Umfang möglich. Die graphische Benutzeroberfläche wurde übersichtlich und intuitiv entworfen. Für die speziellen Gegebenheiten des Produkts, wie beispielsweise der Notwendigkeit des Umvernetzens des Werkstückmodells ist es hingegen notwendig, das Handbuch zu konsultieren.

Ein kooperativer Bewegungsprozess in sequentieller Form wurde ermöglicht. Ein Parallelbetrieb galt als optional und wurde nicht realisiert. Eine flächendeckende Besprühung und Trocknung des Werkstücks durch die Roboter wurde erreicht. Dabei wird auf eine eventuelle Kollision geachtet.

Ein unpassendes Werkstück, d.h. ein zu großes, eins mit einer solchen Einbuchtung in der Oberfläche, das ein Besprühen unter Berücksichtigung des einzuhaltenden tool stand-offs nicht möglich ist oder eines deren Oberfläche auf Grund der Limitierung des Arbeitsraums der Roboter nicht komplett besprüht werden kann, sollte begründet verworfen werden. Diese Funktion ist in *RoSi3D* nicht gegeben, da es an Zeit mangelte und der Fertigstellung der primären Funktionen Vorrang gewährt wurde. Jedoch wurde der Fall, dass die Roboter nicht alle Punkte des Werkstücks anfahren können, insofern behandelt, als dass die Geometrie des Werkstückmodells um die nicht erreichbaren Teilflächen reduziert wird. Ob es sinnvoll ist nur eine Teilfläche zu besprühen, sei hierbei dem Benutzenden überlassen. Ferner sei angemerkt, dass die nicht besprühte, folglich trockene Fläche unter Umständen durch den zweiten Roboter zusätzlich getrocknet wird.

9.2. Ausblick

Im Rahmen der Projektgruppe 523 wurde ein lauffähiges System entworfen und entwickelt. Die vorliegende Arbeit bietet die Möglichkeit fortgeführt zu werden. Im Folgenden sollen Erweiterungsmöglichkeiten aufgezeigt werden.

Eine erste Erweiterungsmöglichkeit von *Rosi3D* wäre es, die bereits erwähnte Funktion des begründeten Verwurfs eines Werkstücks hinzuzufügen. Außerdem könnten weitere Datenformate für den Import von Geometriedaten unterstützt werden. Ebenso wäre eine Erweiterung der unterstützten Robotertypen erstrebenswert, da in *RoSi3D* bisher nur drei verschiedene Roboter (siehe Kapitel 4.3) unterstützt werden.

Die Segmentierung liesse sich verbessern, indem das Meanshift-Verfahren wie in Kapitel

5.1.1 beschrieben mit dem Bereichswachstumsverfahren (siehe Kapitel 5.1.2) kombiniert wird. Außerdem könnten weitere Segmentierungsstrategien implementiert werden.

Die Pfadberechnung für einzelne Patches (siehe Kapitel 5.2.1) könnte ebenfalls um weitere Strategien erweitert werden, so könnte die verwendete Strategie in Abhängigkeit von der Form des Patches gewählt werden (zum Beispiel Zirkulärbahnen für kreisförmige Patches). Eine weitere Optimierung der Pfade könnte erreicht werden, indem benachbarte Patches in den Planungsprozess mit einbezogen würden. So könnte der Farbauftrag gleichmäßiger erfolgen, indem der Pfadabstand zu den Außenkanten eines Patches unter Berücksichtigung der Winkel angrenzender Patches gewählt wird. Bei Patches am äußeren Rand der Werkstückgeometrie könnten sogenannte Overspray-Points verwendet werden, die außerhalb der eigentlichen Geometrie liegen und zu einem gleichmäßigeren Farbauftrag an Kanten des Werkstücks beitragen sollen.

Bezüglich der Interpatchverbindungen liesse sich der TSP-Algorithmus um weitere Heuristiken erweitern und die verwendete Strategie für die Interpatch-Verbindungen (Hemisphäre oder BoundingBox) könnte automatisch gewählt werden. Die momentane Implementierung setzt voraus, dass der Benutzende sich vor der Berechnung für eine der Strategien entscheidet, die dann für jede Interpatchverbindung angewandt wird. Eine automatisierte Lösung, die alle Strategien bezüglich der Erreichbarkeit durch die Roboter testet und eine Minimierung der Pfadlänge der Interpatch-Verbindung zum Ziel hat, wäre von Vorteil. Dies wäre realisierbar, indem pro Interpatch-Verbindung von einem Patch A zu einem anderen Patch B alle Verbindungsarten berechnet werden, die für den Roboter eine gültige Bewegungsabfolge ergeben und von diesem erreichbar sind. Aus dieser Auswahl möglicher Interpatch-Verbindungen wird dann die mit der kürzesten Wegstrecke übernommen.

Des Weiteren werden die Bahnen der beiden Roboter unabhängig voneinander geplant und der zweiteilige Arbeitsprozess des Besprühens und Trocknens sequentiell realisiert. Eine Bahn- und Bewegungsplanung, die einen zeitlichen Parallel-Betrieb beider Roboter ermöglicht, wäre wünschenswert.

Ein weiterer Ansatzpunkt für Verbesserungen ist der Sprühvorgang selbst. Der Farbauftrag im Sprühkegel wurde in der Projektgruppe als konstant angenommen. Die Modellierung als Normalverteilung wäre hier realistischer. Ein realistischeres Modell des Sprühprozesses hätte zur Folge, dass die Pfade auf dem Werkstück genauer geplant werden könnten. Der Sprühvorgang könnte außerdem in die Simulation integriert werden. Dies hätte zur Folge, dass bei einem hinreichend fein aufgelösten Dreiecksnetz die Möglichkeit bestünde, an den Dreiecksknoten die Dicke des Farbauftrags zu ermitteln.

Ein weiterer Punkt, den es zu verbessern gilt, wäre die Gesamtlaufzeit des Programms. Diese könnte enorm verbessert werden, indem gezielt Multithreading eingesetzt würde, um so die weit verbreiteten Mehrkern-Prozessoren zu nutzen.

A. Werkzeuge

Nachfolgend werden die von der Projektgruppe verwendeten Werkzeuge zur Implementierung kurz erläutert.

A.1. Microsoft Visual Studio 2005

Für die Programmierung wurde Microsoft Visual Studio 2005 verwendet. Visual Studio bietet die üblichen Features, die von einer modernen Entwicklungsumgebung erwartet werden: Klassenbrowser, Syntaxhighlighting, einfache Nutzung des Compilers und ein Debugger. Des Weiteren bietet die Programmierumgebung die direkte Einbindung von Qt an.

A.2. Qt

Qt ist ein GUI Toolkit für C++ und wurde für verschiedene gängige Plattformen implementiert. Bei dem Projekt wurde Qt in der Version 4.3.1 verwendet. Qt ist objektorientiert und bietet dem Entwickler die Möglichkeit, Anwendungen mit einem Benutzerinterface zu entwickeln. Zur einfacheren Gestaltung der GUI stand der Qt-Designer in der Version 4.3.1 zur Verfügung (siehe [Qt08]).

A.3. OpenGL

OpenGL 2.0 ist eine plattform- und programmiersprachenunabhängige API zur Entwicklung von 3D-Computergrafik. Die API erlaubt die Darstellung komplexer 3D-Szenen in Echtzeit. Bei dem Projekt wurde OpenGL 2.0 verwendet (siehe [Ope08b]).

A.4. OpenMesh

OpenMesh ist eine Bibliothek für polygonale Netze. Zum Speichern der Netze wird die Half-Edge Datenstruktur (siehe 2.2.1) verwendet, welche sich je nach Anwendungsfall beliebig durch weitere Eigenschaften erweitern lassen. Des Weiteren bietet OpenMesh ein IO-System zum Laden und Speichern gängiger Dateiformate für Netze (siehe [Ope08c]).

A.5. OpenCASCADE

OpenCASCADE bietet die Möglichkeit CAD-Anwendungen zu implementieren und stellt alle wichtigen Bestandteile für die Erzeugung, Verwaltung, Modellierung und Darstellung von 3D-Objekten zur Verfügung. Im Rahmen der Projektgruppe wird OpenCASCADE für den Im- und Export von IGES, STEP und BRep benutzt (siehe [Ope08a]).

A.6. RobotStudio

RobotStudio ist eine Simulations- und Offline-Programmiersoftware von ABB. Es ermöglicht die Erstellung von Roboterprogrammen am PC. Roboterprogramme können auf diese Weise erstellt und getestet werden. Die von der Projektgruppe verwendete *RobotStudio* Version ist 5.09.

A.7. Newmat

Newmat ist eine Mathematikbibliothek für das Rechnen mit Matrizen. Im Rahmen der Implementierung der inversen Kinematik, war die Verwendung von Newmat wichtig um die Rotationsmatrizen zu berechnen (siehe [New08]).

A.8. Boost

Boost ist eine freie C++-Bibliothek, welche aus einer Vielzahl von portablen Unterbibliotheken besteht. Die Unterbibliotheken dienen unterschiedlichsten Aufgaben von Algorithmen auf Graphen über Metaprogrammierung bis hin zur Speicherverwaltung (siehe [Boo08]).

A.9. libply

Libply ist eine C++-Bibliothek zum Parsen des PLY Dateiformats (siehe [Lib08]).

A.10. SVN

Bei der Entwicklung mit einer größeren Gruppe kann es schnell zu Problemen mit der Konsistenz der Dateien kommen. Um dies zu vermeiden wurde SVN benutzt. Auftauchende Konflikte können bearbeitet werden und es besteht die Möglichkeit, ältere Versionen wiederherzustellen (siehe [SVN08]).

A.11. LaTeX

LaTeX ist ein Softwarepacket welches die Benutzung des Textsatzsystem TeX mit Hilfe von Makros vereinfacht. Sämtliche Dokumente wie Seminarband und Endbericht wurden mit LaTeX erstellt. LaTeX eignet sich gut für das Verfassen wissenschaftlicher Dokumente. Es bietet viele hilfreiche Pakete und die Einteilung in separate Teile, die einzeln bearbeitet werden können, stellt kein Problem dar (siehe [Lat08]).

A.12. Blender

Zum Testen der Software wurde das freie OpenSource Programm Blender verwendet um CAD-Modelle verschiedener Werkstücke zu erzeugen (siehe [Ble08]).

B. Klassendokumentation

Zur Darstellung der Problemstellung wurde die gesamte Struktur in einzelne Pakete unterteilt, die wiederum mehrere Klassen beinhalten. Nachfolgend werden die wichtigsten Klassen vorgstellt, die für die Realisierung der Anforderungen und angestrebten minimalen Ziele des Projekts implementiert wurden.

Abbildung B.1 zeigt die Beziehungen zwischen den einzelnen Paketen auf. Eine zentrale Rolle bei der Entwicklung spielten die Pakete Paths, Motionplanning, Collision, Kinematics, SceneModel, und MVC, die im folgenden näher beschrieben werden.

Das Paket **Paths** (siehe Abbildung B.7) enthält Klassen für die Generierung eines Gesamtpfades auf dem Werkstück. Mit der Klasse **IntersectAlgo** erfolgt zuerst die Segmentierung des Werkstück-Dreiecknetzes in kleine Patches, worauf die Teilbahnen berechnet werden. Die Bahnen auf den einzelnen Patches, die das ganze Netz abdecken, werden von der Klasse **Path** basierend auf dem Netz selbst, dem Wirkungsradius, der den Pfadabstand bestimmt und dem Werkzeugabstand, der den Abstand des Pfades zur Oberfläche bestimmt. Anschließend werden die einzelnen Bahnen miteinander verbunden, die sogenannte Interpatch-Verbindung, was durch die Klasse **TSP** realisiert wird. Dabei wird ein Verbindungspfad nach dem TSP Algorithmus berechnet, der die einzelnen Patches, auf Basis eines mininmalen Spannbaums miteinander verbindet.

Im Packet **MotionPlanning** (siehe Abbildung B.4) werden die eigentlichen Programme der Roboter ermittelt (siehe Kapitel 6). Die Bahnen, die im Rahmen der Bahnplanung (siehe Kapitel 5) berechnet wurden, werden dabei durch lineare und kreisbogenförmige Segmente approximiert und für die resultierenden Punkte TCPs (siehe Kapitel 4.2.1) festgelegt. Darin repräsentiert die Klasse **Movement** eine spezifische Bahn auf einem Werkstück. Prozeßspezifische Eigenschaften werden durch Spezialisierungen von **MotionPlanner** definiert. Prozeßspezifische Eigenschaften umfassen u.a.: Das Hinzufügen von Synchronisations-Instruktionen beim kooperierenden Prozeß und das De- und Aktivieren der Werkezuge. Die Klasse **SingleMotionPlanner** stellt eine Spezialisierung der Klasse **MotionPlanner** dar und dient der Ermittlung einer Bahn für einen Arbeitsprozeß, bei dem lediglich ein einzelner Roboter verwendet wird. Dementsprechend stellt die Klasse **DualMotionPlanner** ebenfalls eine Spezialisierung der Klasse **MotionPlanner** dar und wird zur Berechnung von Bahnen im Rahmen eines Arbeitsprozesses verwendet, bei dem zwei Roboter kooperierend zum Einsatz kommen.

Die Kollisionsüberprüfung (siehe Abbildung B.2) wird im Paket **Collision** durchgeführt.

Zu diesem Zweck wird für ein Objekt, welches wir auf Kollision testen, eine Bounding Box mit der Klasse **OrientedBoundingBox** berechnet. Diese Box wird danach mit Hilfe der Klassen **OctTree** und **OctTreeNode** geschachtelt, dabei werden die passenden Facetten in jedes Achtel einsortiert, indem für jedes Achtel ein OctTreeNode-Kind erzeugt wird mit den entsprechenden Vertex- und Facetten-Indizes, die dort hinein sollen. Facetten, welche nicht in eines der Kinder einsortiert werden können, da sie in keine der Achtelung hinein passen, bleiben direkt im Knoten gespeichert. Anschließend wird die Kollisionsüberprüfung von der Klasse **Collidable** gehandhabt, die die Schnitttests der Bounding Boxen durchführt.

Im Paket **Kinematics** (siehe Abbildung B.3) wird in der Klasse **KinematicChain** die im Abschnitt 4.4.3 beschriebene Berechnung der Inversen Kinematik implementiert. Die Klasse liefert für einen gegebenen TCP die entsprechenden Gelenkwinkeln des Roboters, die sich beim erreichen dieser Position ergeben haben. Wie bereits erwähnt (siehe Kapitel 4), stehen für den Roboter mehrere Konfigurationen zur Auswahl eine Position zu erreichen. Diese werden in der Klasse **Configuration** in Abhängigkeit von der nächsten anzufahrenden Position festgelegt.

Das Paket **SzeneModel** (siehe Abbildung B.5) werden die, für die Visulaisierung der Szene (siehe Kapitel 7.2) benötigten 3D-Objekte, wie z.B. die Roboter, das Werkstück, das Werkzeug und den Tisch geometrisch beschrieben. Dabei beschreibt die Klasse **World** erstmal eine leere Szene, in der erst Objekte der Klassen **IRB4400_60_m2000_rev1**, **IRB2400_16_m2000_rev3**, **Table**, **Workpart** und **Tool** eingefügt werden, so dass sie der realen Roboterzelle entsprechen.

Jede interaktive Anwendung, wie es auch *RoSi3D* ist, besteht aus Eingaben, einem Datenmodell und einer visuellen Darstellung der Daten. Wird ein bestimmter Datensatz geändert, so soll die graphische Darstellung unmittelbar aktualisiert werden. Zu diesem Zweck wurde das Model-View-Controller Konzept *(kurz: MVC)* entwickelt. Das Model beschreibt hierbei die Repärsentation eines Objektes mit seinen Attributen und ist von der Anzeige, der View, gekappselt. Es wird ledeglich für die Datenverarbeitung verwendet. Die View dient der visuellen Darstellung eines Models. Sie setzt sich aus Anzeigeobjekten, wie z.B. Menüs und Dialogboxen, also der gesammten GUI zusammen. Der Controller überwacht die Eingabe des Nutzers und leitet diese an das Model weiter und sendet anschließend ein Signal an die View sich dementsprechend zu aktualisieren. Um dies zu realiseren steht das Paket **MVC** (siehe Abbildung B.6) zur Verfügung. Es enthält zwei verschiedene Obejkttypen – Obersver und Notifikation –. Wobei ein Observer ein Objekt auf Änderungen hin überwacht, welches mittels einer Notifikation seine Änderung dem Observer meldet.

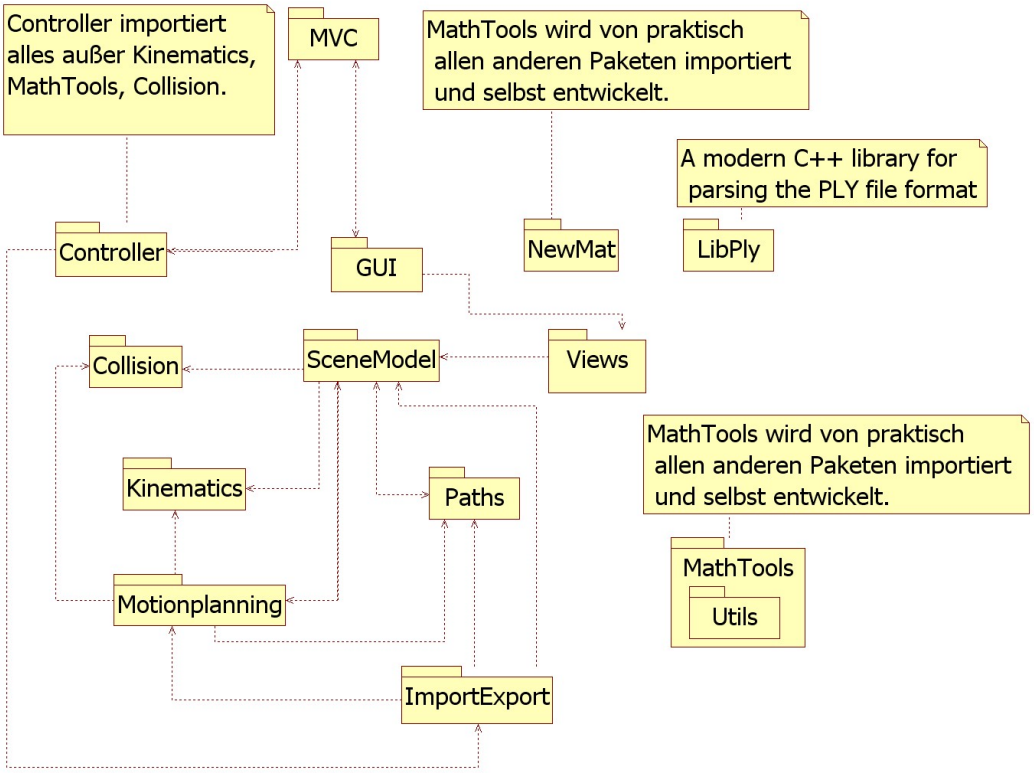


Abbildung B.1.: Paketdiagramm

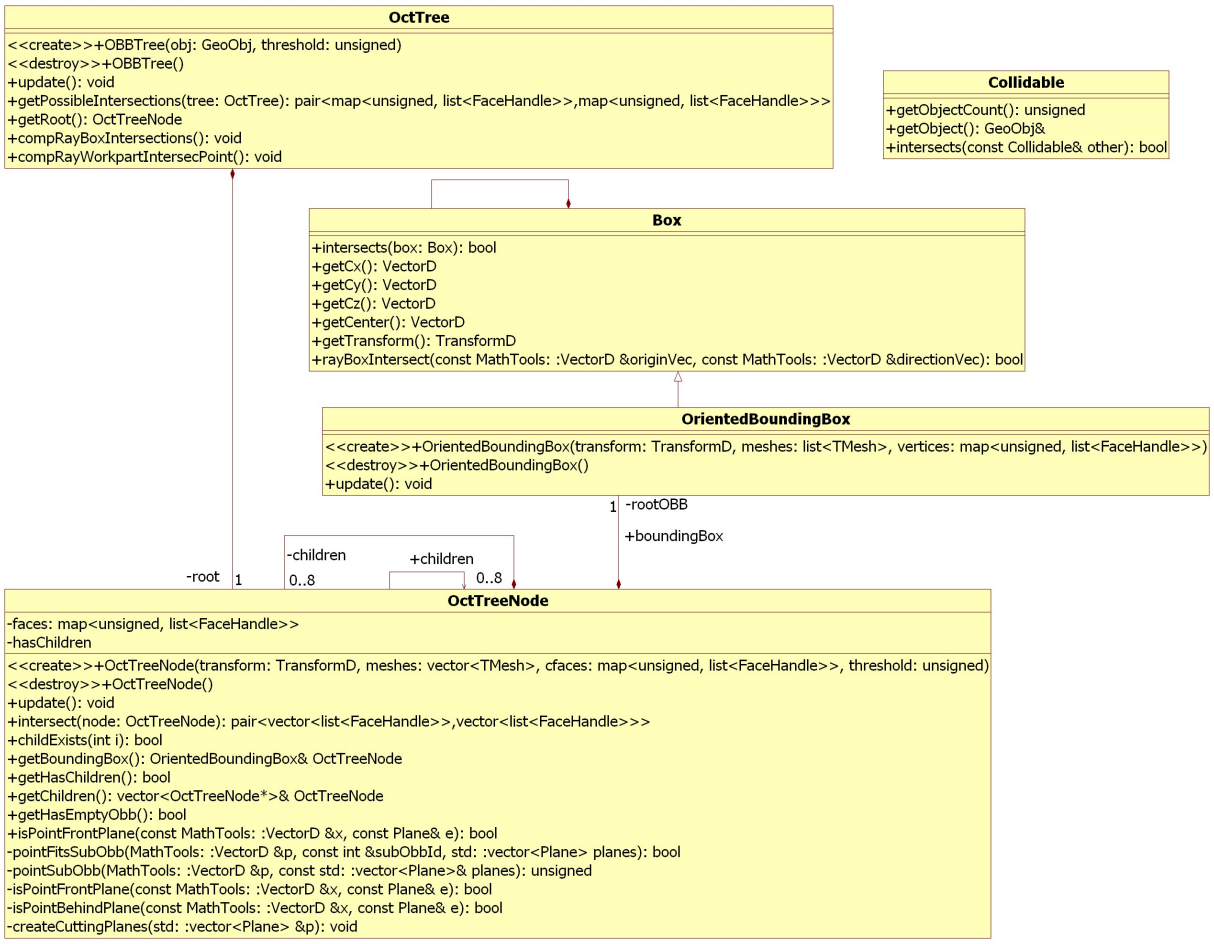


Abbildung B.2.: Klassendiagramm Collision

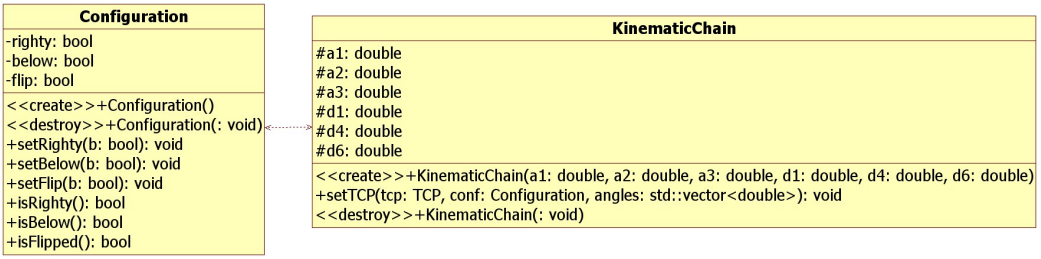


Abbildung B.3.: Klassendiagramm Kinematics

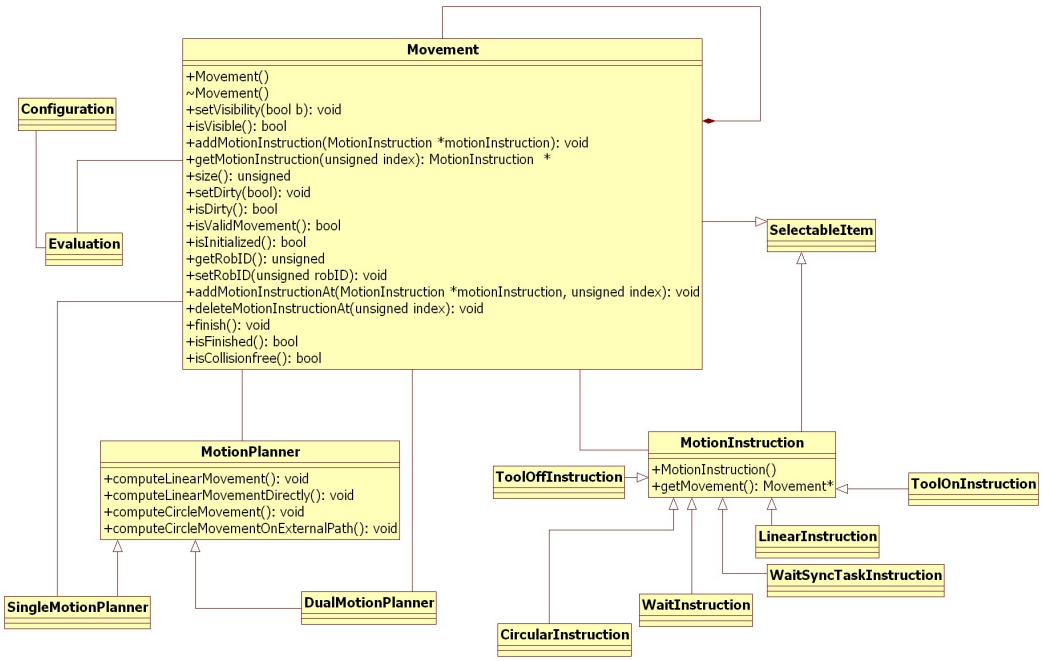


Abbildung B.4.: Klassendiagramm MotionPlanning

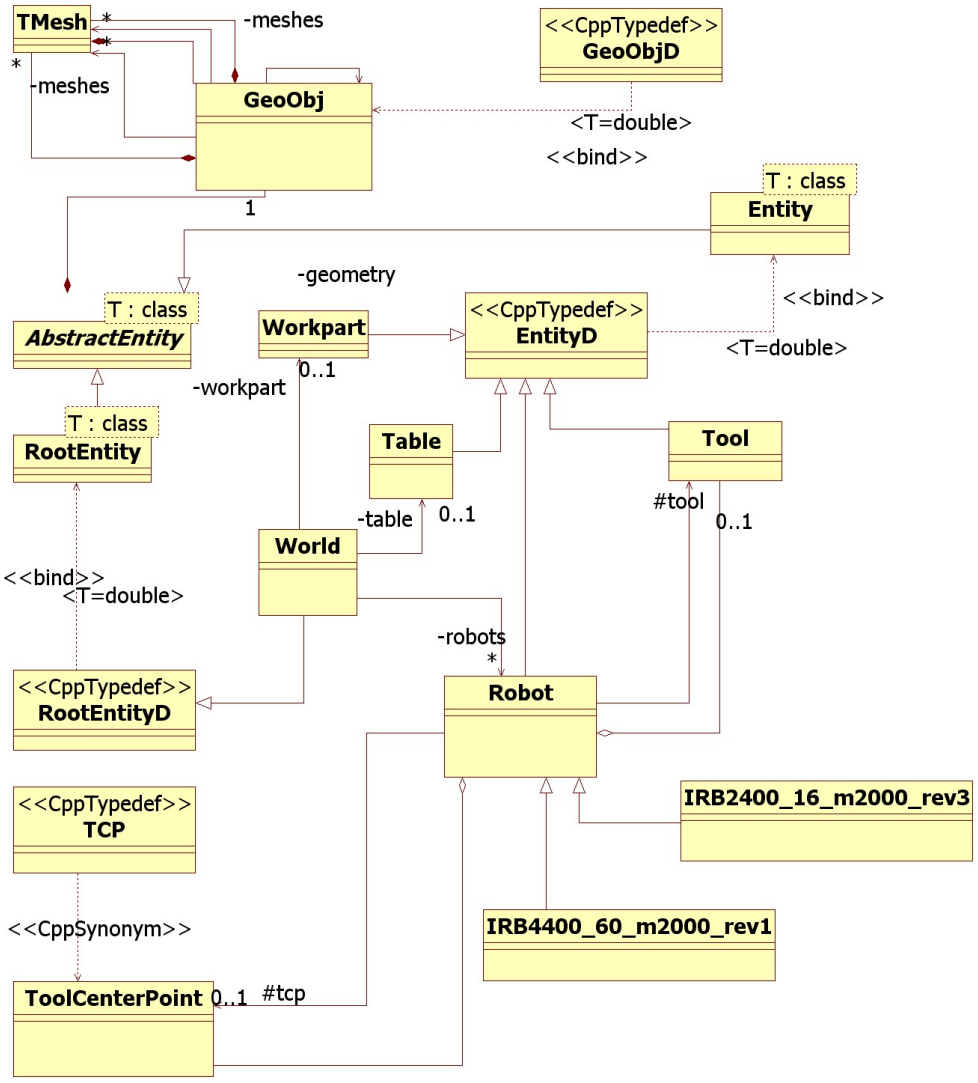


Abbildung B.5.: Klassendiagramm SzeneModel

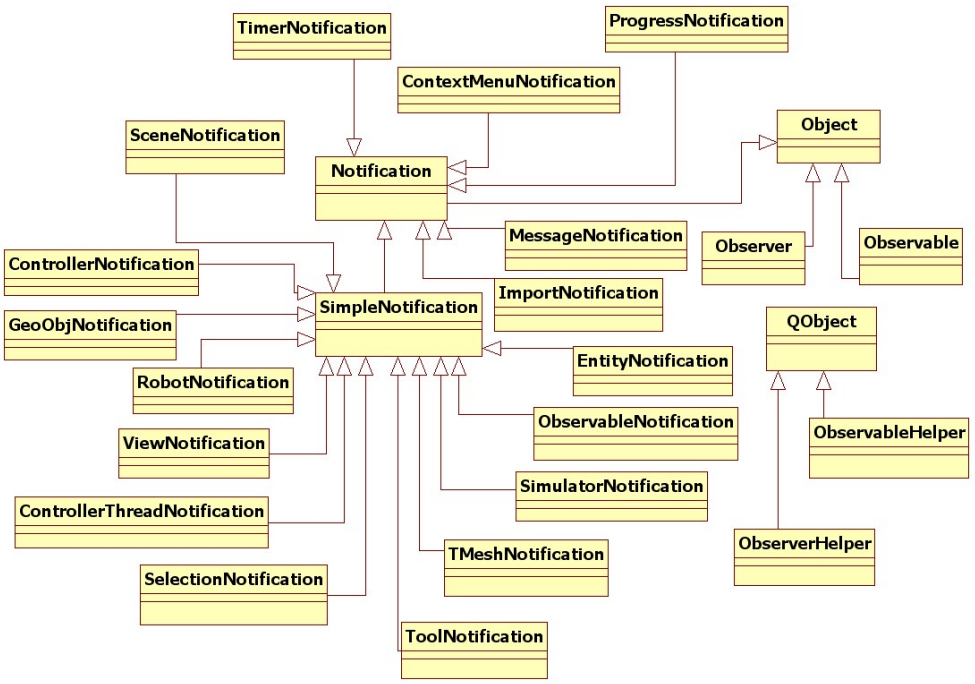


Abbildung B.6.: Klassendiagramm MVC

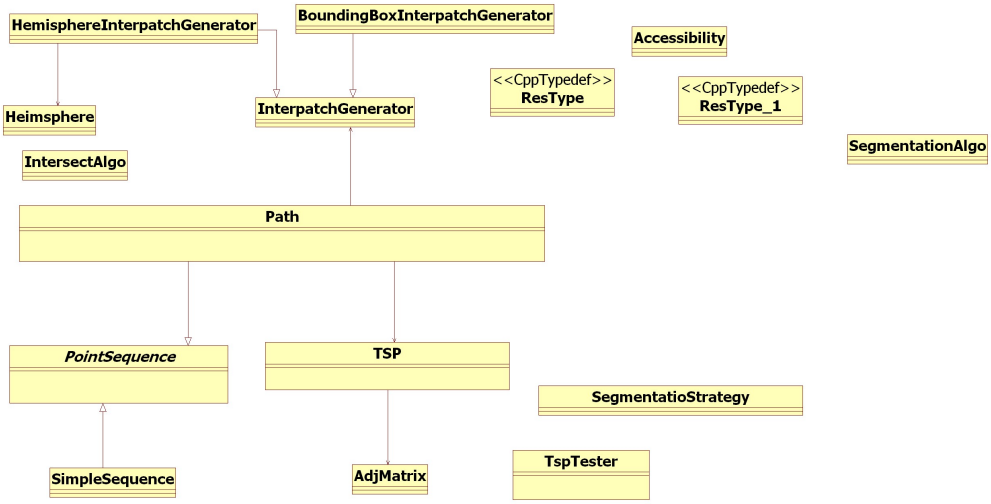


Abbildung B.7.: Klassendiagramm Paths

C. Handbuch

C.1. Vorwort

Im Kontext fortschreitender Automatisierung werden zunehmend Roboter in industriellen Anwendungsbereichen eingesetzt. Verbreitete Anwendungsbereiche sind das Lackieren von Automobilkarosserien und die Bearbeitung von Werkstücken in unterschiedlichen Fertigungsprozessen. Ein wesentlicher Aspekt innerhalb einer effizienten Bearbeitung durch Roboter stellt die möglichst automatische Ermittlung von optimierten Fertigungsbahnen dar. Dieses Prinzip der Bahnplanung betrachtet neben der eigentlichen Bewegungsplanung der Roboterarme sowohl die Geometrie des zu fertigenden Werkstücks als auch Nebenbedingungen, die sich aus dem Bearbeitungsprozess ergeben.

C.2. Was ist RoSi3D

RoSi3D ist eine Software zur Berechnung von Fertigungsbahnen für die Oberflächenbearbeitung unter Verwendung von bis zu zwei Sechs-Achs-Industrierobotern mit Zentralhand, die sequentiell arbeiten. Die Berechnung der Roboterbahnen basiert dabei auf der Geometrie des zu bearbeitenden Werkstücks und den Eigenschaften der verwendeten Werkzeuge.

C.3. Programmablauf

Bevor die Roboterbahnen von der Software berechnet werden können, muss der Benutzende eine sogenannte Szene definieren. Diese beschreibt sowohl die Roboterzelle, in welcher der Bearbeitungsprozess durchgeführt werden soll, als auch das zu bearbeitende Werkstück. Der Benutzende hat dabei die Möglichkeit eine vollständig neue Szene zu erstellen oder eine bereits zuvor erzeugte zu importieren. In dieser Szene können dann durch die Angabe von Werkzeugparametern die Arbeitsprozesse definiert werden. Sind auf diese Weise alle benötigten Informationen vorhanden, kann die Software vollautomatisch die Bewegungsbahnen für die Roboter ermitteln. Im Anschluss daran hat der Benutzer die Möglichkeit aus dem Berechnungsergebnis ein RAPID-

Programm zu erstellen. Dieses kann zur Weiterverarbeitung in der Simulations- und Offline-Programmiersoftware *RobotStudio* von ABB oder zur Übertragung auf die realen Roboter exportiert werden.

C.4. Installation

Die nachfolgenden Hard- und Softwarevoraussetzungen müssen vor der Installation der Software erfüllt sein, um *RoSi3D* mit allen Funktionen uneingeschränkt nutzen zu können.

C.4.1. Hardwarevoraussetzungen

- Prozessor mit 3.0 GHz Rechenleistung oder besser.
- 2GB Arbeitsspeicher.
- Grafikkarte mit Hardeware-Beschleunigung von OpenGl 2.X.
- 100 GB Festplattenspeicher.
- Ein oder zwei Sechs-Achs-Industrieroboter mit Zentralhand vom Typ ABB 2440, 4400 oder 6620, für den realen Betrieb.

C.4.2. Softwarevoraussetzungen

- Windows XP mit installiertem Service Pack 2.
- ABB *RobotStudio*, zur Überprüfung und Übertragung der generierten RAPID-Programme an die Roboter.
- OpenGL 2.X
- Microsoft .NET Framework 2.0
- QT Version 4.3.1
- Adobe Reader 7.0 oder höher, zur Betrachtung des Benutzerhandbuches.

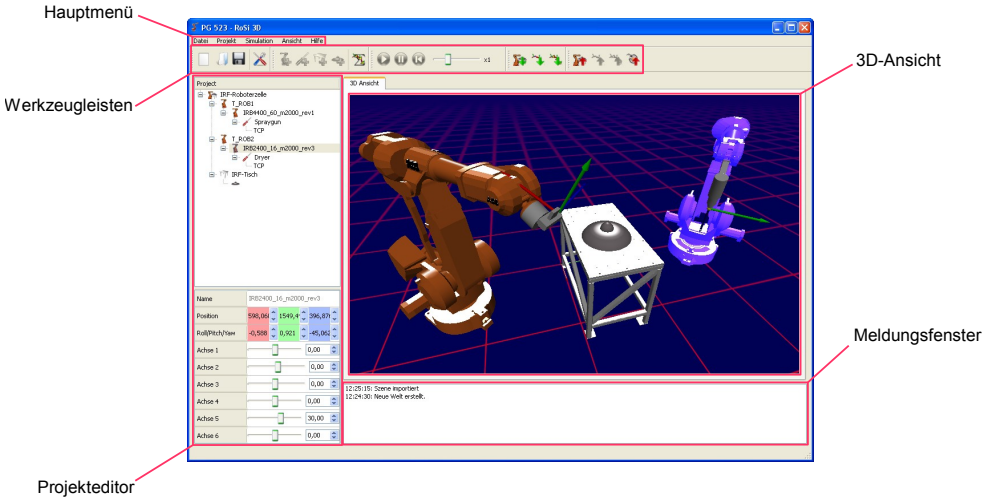


Abbildung C.1.: Die Benutzeroberfläche von RoSi3D

C.5. Schnelleinstieg

Dieses Kapitel bietet eine Übersicht über alle wesentlichen Schritte die notwendig sind um, ausgehend von den CAD-Daten der Roboterzelle, des Werkstücks und der Parameter des Arbeitsprozesses, Roboterbahnen zu berechnen und zur weiteren Verwendung zu exportieren. Detaillierte Informationen über Funktionen, die in diesem Kapitel nicht genauer beschrieben sind, befinden sich in Kapitel C.6.

C.5.1. Die Benutzeroberfläche

Die Benutzeroberfläche von *RoSi3D* ist in vier wesentliche Bereiche gegliedert: Das *Hauptmenü* und die *Werkzeugleisten*, den *Projekteditor*, die *3D-Ansicht* und das *Meldungsfenster* (siehe Abbildung C.1).

Im *Hauptmenü* und den *Werkzeugleisten* sind alle wichtigen Funktionen von *RoSi3D* zu finden. Gegliedert sind diese in die Bereiche *Datei*, *Projekt*, *Simulation*, *Ansicht* und *Hilfe*. Im Bereich *Datei* können Projekte neu angelegt, gespeichert und geladen werden, sowie grundlegende Einstellungen vorgenommen werden. In der Steuerungsgruppe *Projekt* sind die Funktionen zum Laden, Importieren und Exportieren einzelner Szenenelementen und die Funktionen zur Pfadberechnung zusammengefasst. Im Bereich *Simulation* befinden sich die Steuerungselemente zur Simulation der Roboterbewegungen in der *3D-Ansicht*.

Der *Projekteditor* listet alle Elemente der aktuellen Szene auf. Einzelne Einträge können hier selektiert und die zugehörigen Eigenschaften editiert werden. Über Kontextmenüs besteht die Möglichkeit des direkten Zugriffs auf zentrale Funktionen zu dem selektierten Element.

In der *3D-Ansicht* wird die aktuelle Szene visualisiert. Alternativ zum *Projekteditor* können auch hier einzelne Elemente der Szene selektiert werden und mit Hilfe von Kontextmenüs Einstellungen vorgenommen werden.

Das *Meldungsfenster* protokolliert alle wichtigen Ereignisse und bietet Rückmeldungen sowie zusätzliche Informationen.

C.5.2. Erstellen einer Szene

Um die Berechnung der Roboterbahnen durchführen zu können müssen in der Szene mindestens ein Roboter, ein an diesen Roboter angebrachtes Werkzeug und ein Werkstück vorhanden sein.

Ein Roboter kann über den Befehl *Projekt→Hinzufügen→Roboter* oder direkt über den entsprechenden Eintrag in der *Werkzeugleiste* hinzugefügt werden. Im daraufhin erscheinenden Dialog kann der Robotertyp ausgewählt und über die Schaltfläche *Ok* in die Szene integriert werden.

Um ein Werkzeug hinzuzufügen muss zunächst ein Roboter ausgewählt werden, welchem das Werkzeug zugeordnet wird. Ein Roboter lässt sicht mit der linken Maustaste im *Projekteditor* oder direkt in der *3D-Ansicht* auswählen. Das Hinzufügen eines Werkzeugs geschieht dann über den Befehl *Projekt→Hinzufügen→ Werkzeug* oder ebenfalls über die *Werkzeugleiste*. Alternativ kann auch das Kontextmenü mit einem Rechtsklick auf den Roboter im *Projekteditor* oder der *3D-Ansicht* genutzt werden. Anschließend öffnet sich der *Werkzeugdialog*. In diesem muss die CAD-Datei des Werkzeugs ausgewählt werden. Zusätzlich lassen sich Position und Rotation des Werkzeugs, sowie des Tool-Center-Points (TCP) verändern und es kann ein Signal zur Ansteuerung des Werkzeugs definiert werden (eine ausführlichere Beschreibung des *Werkzeugdialogs* erfolgt in Kapitel C.6.6). Die Schaltfläche *OK* fügt das Werkzeug der Szene hinzu.

Für das Werkstück muss lediglich eine CAD-Datei ausgewählt werden. Dies geschieht über den Befehl *Projekt→Hinzufügen→ Werkzeug* oder direkt über die *Werkzeugleiste*.

Sind alle Elemente der Szene hinzugefügt können diese frei im Arbeitsbereich positioniert werden. Um ein Szenenelement zu Positionieren muss dieses zunächst im *Projekteditor* oder der *3D-Ansicht* ausgewählt werden. Im Projekteditor kann dann die Position und Rotation des Elements editiert werden. Änderungen werden direkt übernommen und in der *3D-Ansicht* angezeigt.

C.5.3. Bearbeitung des Werkstücks

Um die Qualität und Geschwindigkeit der Bahnplanung zu optimieren besteht die Möglichkeit das 3D-Modell des Werkstücks zu bearbeiten. Dies geschieht im *Netzeditor* der über das Kontextmenü des Werkstücks im *Projekteditor* oder in der *3D-Ansicht* geöffnet wird. Im *Netzeditor* kann das gesamte Werkstück skaliert und seine Position und Rotation bezüglich des Ursprungs des Werkstückkoordinatensystems verändern werden. Änderungen in den dafür vorgesehenen Eingabefeldern im linken Bereich des *Netzeditors* werden erst nach einer Bestätigung über die entsprechende Schaltfläche, unterhalb der Eingabefelder, übernommen. Falls das Werkstück aus mehr als einem Netz besteht, ist es möglich die Einzelnetze zu vereinen. Zusätzlich können Einzelnetze entfernt werden, die nicht mit in die Berechnung aufgenommen werden sollen. Ist die Bearbeitung des Werkstücks abgeschlossen müssen die Änderungen über *Datei→Schließen und übernehmen* in das aktuelle Projekt übernommen werden.

C.5.4. Pfadberechnung

Sind Roboter, Werkzeug, und Werkstück in der Szene vorhanden kann die Bewegungsbahn für den Roboter über den Befehl *Projekt→Bewegungen berechnen* ermittelt werden. Im erscheinenden *Einstellungsdialog* (siehe Kapitel C.6.5) können noch die Parameter für die Berechnung festgelegt werden, bevor durch die Schaltfläche *OK* die Berechnung gestartet wird. Sobald die Pfadberechnung abgeschlossen ist werden die berechneten Pfade in der *3D-Ansicht* angezeigt. Für Details zur Ein- und Ausblendung der Pfade und der Werkstücksegmentierung siehe Kapitel C.6.3.

C.5.5. Export zu RobotStudio

Nach der Berechnung der Roboterbahnen können diese mit dem Befehl *Projekt→Export→RAPID exportieren* als RAPID-Programm exportiert werden. Um die erstellte Szene ebenfalls in *RobotStudio* nutzen zu können kann diese als XML-Datei für *RobotStudio* über den Befehl *Projekt→Export→RobotStudio Export* exportiert werden. Mit dem *RobotStudio* Plug-in von *RoSi3D* kann diese XML-Datei in *RobotStudio* geladen werden.

C.6. Bedienung des Programms im Detail

Nach der Einführung der wichtigsten Elemente und Funktionen von *RoSi3D* im vorherigen Kapitel, folgt nun eine detaillierte Auflistung und Beschreibung der einzelnen Programmkomponenten und ihrer Funktionen. Genauer betrachtet werden das Menü

und die Werkezugleisten, die 3D-Ansicht, der Projekteditor, der Netzeditor, der Einstellungsdialog und der Werkzeugdialog.

C.6.1. Das Menü und die Werkzeugleisten

Über die Menüleiste des Hauptfensters sind alle wichtigen Funktionen von *RoSi3D* zugänglich, gegliedert in die folgenden Menüs und Menüpunkte:

Datei

Neu
Erstellt ein neues Projekt, mit einer leeren Szene.

Projekt laden
Lädt ein zuvor gespeichertes Projekt.

Projekt speichern
Speichert das aktuelle Projekt.

Einstellungen
Öffnet den Einstellungsdialog zum Festlegen der Projekteinstellungen. Für Details zur Verwendung des *Einstellungsdialogs* siehe Kapitel C.6.5.

Beenden
Beendet *RoSi3D*

Projekt

Hinzufügen
Über das Untermenü Hinzufügen können die Szenenelemente Roboter, Werkzeug, Werkstück und Halterung der Szene hinzugefügt werden. Um ein Werkzeug hinzuzufügen muss zuvor in der *3D-Ansicht* oder dem *Projekteditor* ein Roboter ausgewählt werden.

Import
Über das Untermenü Import können eine zuvor erstellte Szene oder Bewegungspfade geladen werden, die mit der entsprechenden Export-Funktion von *RoSi3D* exportiert wurden.

Export
Über das Untermenü Export können zum einen die aktuelle Szene und die bereits berechneten Pfade als XML-Datei exportiert werden, um diese zu einem späteren Zeitpunkt wiederverwenden zu können. Zum anderen können die Szene, zur Verwendung in *RobotStudio*, und die aktuell berechnete Bewegung als RAPID Programm exportiert werden.

Bewegungen berechnen
Öffnet den Einstellungsdialog (siehe Kapitel C.6.5). Nach Bestätigung der Einstellungen über die Schaltfläche *Ok* startet die Berechnung der Roboterbewegungen aller aktiven Tasks.

Simulation

Starten
Startet die Simulation der Roboterbewegung in der *3D-Ansicht*.

Anhalten
Stoppt die Simulation der Roboterbewegung in der *3D-Ansicht*. Die Simulation kann zu einem späteren Zeitpunkt, mittels des Befehls *Starten*, fortgesetzt werden oder über den Befehl *Zurücksetzen* wieder auf den Startpunkt gesetzt werden.

Zurücksetzen
Setzt die Simulation der Roboterbewegung in der *3D-Ansicht* auf den Startpunkt zurück.

Ansicht
Im Untermenü Werkzeugleisten können die einzelnen Werkzeugleisten eingeblendet bzw. ausgeblendet werden.

Hilfe
Im Hilfemenü kann das Benutzerhandbuch von *RoSi3D* im Adobe-PDF-Format geöffnet werden.

C.6.2. Die 3D-Ansicht

In der *3D-Ansicht* wird die aktuelle Szene visualisiert. Alternativ zum *Projektfenster* können auch hier einzelne Elemente der Szene selektiert werden und, mit Hilfe von Kontextmenüs, Einstellungen vorgenommen werden. In Tabelle C.1 sind die Maus- und Tastaturbefehle zur Steuerung der *3D-Ansicht* aufgelistet.

Eingabe	Wirkung
Linke Maustaste	Selektieren von Szenenelementen
Rechte Maustaste	Öffnet das zum Szenenelement gehörige Kontextmenü
Strg + Linke Maustaste	Ansicht drehen
Strg + Rechte Maustaste	Ansicht zoomen
Mausrad	Ansicht zoomen
Alt + W + Linke Maustaste	Drahtgittermodel-Ansicht des Szenenelements
Alt + F + Linke Maustaste	Nicht geglättete Flächendarstellung des Szenenelements
Alt + S + Linke Maustaste	Geglättete Flächendarstellung des Szenenelements
Alt + N + Linke Maustaste	Anzeige der Flächennormalen des Szenenelements

Tabelle C.1.: Vollständige Liste der Maus- und Tastaturbefehle der *3D-Ansicht*.

C.6.3. Der Projekteditor

Der *Projekteditor* listet alle Elemente der aktuellen Szene in einer Baumdarstellung auf (siehe Abbildung C.1). Der Elementtyp ist dabei durch die Picktogramme aus Tabelle C.2 gekennzeichnet. Alle Szenenelemente können mit der linken Maustaste ausgewählt werden, woraufhin im unteren Teil des *Projekteditor*s zusätzliche Steuerungsmöglichkeiten für dieses Element zur Verfügung stehen. Alle Szenenelemente lassen sich mit dieser Steuerung verschieben und drehen. Ist im *Projekteditor* ein Roboter ausgewählt bietet die Steuerung zusätzlich die Möglichkeit, die Achsen des Roboters manuell einzustellen. Alle Änderungen werden direkt übernommen und in der *3D-Ansicht* dargestellt.

Des Weiteren kann mit der rechten Maustaste für jedes Szenenelement ein individuelles Kontextmenü aufgerufen werden. Es folgt eine Auflistung aller Funktionen, die über diese Kontextmenüs abrufbar sind, gegliedert nach selektiertem Szenenelement.

Alle Szenenelemente

Sichtbarkeit an/aus
Zeigt das Szenenelement in der *3D-Ansicht* an oder blendet dieses aus, falls es bereits angezeigt wird.

Normalen an/aus
Zeigt die Flächennormalen des 3D-Modells des Szenenelements in der *3D-Ansicht* an oder blendet diese aus, falls sie bereits angezeigt werden.

Obbs an/aus
Zeigt die Oriented-Bounding-Boxes des 3D-Modells des Szenenelements in der *3D-Ansicht* an, die für die Kollisionserkennung verwendet werden, oder blendet diese aus, falls sie bereits angezeigt werden.

Icon	Szenenelementtyp
	Szene
	Roboter
	Werkzeug
	Werkstück
	Halterung
	Pfad
	Bewegung
	Lineare Bewegungsinstruktion
	Zirkuläre Bewegungsinstruktion
	Nicht durchführbare Lineare Bewegungsinstruktion
	Nicht durchführbare Zirkuläre Bewegungsinstruktion
	Synchronisationsinstruktion
	Instruktion zum An- und Abschalten der Werkzeuge

Tabelle C.2.: Picktogramme des Projekteditors

Entfernen
Entfernt das Element aus der Szene.

Task

Pfad laden
Lädt einen Pfad der zu einem früheren Zeitpunkt mit *Rosi3D* erstellt wurde und mittels der Funktion Pfade exportieren als XML-Datei gespeichert wurde. Ausserdem können hier externe Pfade geladen werden.

Bewegung laden
Lädt eine Bewegung die zu einem früheren Zeitpunkt mit *Rosi3D* erstellt wurde und mittels der Funktion Bewegung exportieren als XML-Datei gespeichert wurde.

Segmentierung anzeigen
Zeigt die bei der Pfadberechnung verwendete Segmentierung des Werkstücks für diesen Task, das bedeutet für die Werkzeugparameter dieses Tasks, an.

Roboter

Achsen an/aus
Zeigt die Achskoordinatensysteme des Roboters in der *3D-Ansicht* an.

In Grundstellung bringen
Die fünfte Achse des Roboters wird auf 30° gesetzt, die übrigen Achsen auf den Wert null.

Werkzeug

Geometrie bearbeiten
Öffnet den *Netzeditor* zur Bearbeitung der Geometrie des Werkzeuges. Die Funktion kann genutzt werden um Probleme in der Darstellung der *3D-Ansicht* zu beheben oder die Netzauflösung, zur Beschleunigung der Kollisionsdetektion, zu verringern.

Werkstück

Geometrie bearbeiten
Öffnet den *Netzeditor* für dieses Werkstück.

Eingabe	Wirkung
Linke Maustaste	Fläche selektieren
Linke Maustaste + Alt + W	Drahtgittermodel-Ansicht
Linke Maustaste + Alt + F	Flächendarstellung (flach)
Linke Maustaste + Alt + S	Flächendarstellung (geglättet)
Linke Maustaste + Alt + N	Flächennormalen anzeigen

Tabelle C.3.: Liste der Maus- und Tastaturbefehle des *Netzeditors*.

Orginalnetz anzeigen
Blendet die Segmentierung des Werkstücks in der *3D-Ansicht* aus.

Halterung

Geometrie bearbeiten
Öffnet den *Netzeditor* zur Bearbeitung der Geometrie der Werkstückhalterung. Die Funktion kann genutzt werden um Probleme in der Darstellung der *3D-Ansicht* zu beheben oder die Netzauflösung, zur Beschleunigung der Kollisionsdetektion, zu verringern.

Bewegung

RAPID speichern
Exportiert die Bewegungsinstruktionen als RAPID-Programm.

C.6.4. Der Netzeditor

RoSi3D verwendet bei der Pfadberechnung Dreiecksnetze zur Geometriebeschreibung der Werkstücke. Die Güte der Bahnplanung ist von der Qualität dieser Dreiecksnetze abhängig. Der *Netzeditor* bietet die Möglichkeit Einfluss auf die Triangulierung der Werkstück-CAD-Daten zu nehmen, um die Qualität und Geschwindigkeit der Bahnplanung zu optimieren.

Ähnlich wie die Benutzeroberfläche des Hauptfensters gliedert sich der *Netzeditor* in die Bereiche *Menü* und *Werkzeugleisten*, die *3D-Ansicht*, das *Infofenster* und die *Positions- und Skalierungssteuerung* (siehe Abbildung C.2).

Die **3D-Ansicht** zeigt eine dreidimensionale Ansicht des Werkstücks, die sich mit Hilfe der Maus drehen, verschieben und zoomen lässt. Eine genaue Auflistung der Maus- und Tastaturbefehle des Netzeditors ist in Tabelle C.3 zu finden.

Die Kernfunktionen des Netzeditors sind über das **Menü** und die **Werkzeugleisten** zu

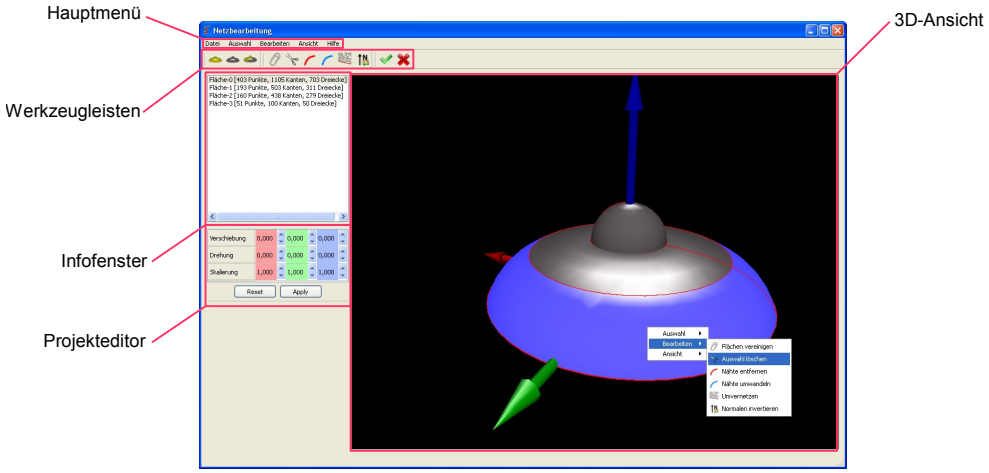


Abbildung C.2.: Der Netzeditor von RoSi3D

erreichen. Zusätzlich sind die Funktionen aus den Bereichen Auswahl, Bearbeiten und Ansicht auch über ein Kontextmenü in der *3D-Ansicht* zugänglich. Jede Funktion wird im Nachfolgenden genauer beschrieben.

Datei

Schließen und übernehmen
Schließt den Netzeditor und übernimmt die am Werkstück vorgenommenen Änderungen in das aktuelle Projekt.

Abbrechen
Schließt den Netzeditor. Alle am Werkstück vorgenommenen Änderungen werden verworfen.

Speichern
Speichert das Werkstück im aktuellen Bearbeitungszustand.

Auswahl

Alles auswählen
Wählt das komplette Werkstück aus. Die Auswahl wird in der 3D-Ansicht blau hervorgehoben.

Auswahl aufheben
Die aktuelle Auswahl am Werkstückmodell wird aufgehoben.

Auswahl umkehren
Wählt alle zur Zeit nicht ausgewählten Bereiche des Werkzeugmodells aus. Die Auswahl wird in der 3D-Ansicht blau hervorgehoben.

Bearbeiten

Flächen vereinigen
Alle aktuell ausgewählten Teilflächen werden zu einer gemeinsamen Auswahl vereint.

Auswahl löschen
Alle aktuell ausgewählten Teilflächen werden aus dem 3D-Modell entfernt.

Nähte entfernen
Die beim Vereinigen der Teilflächen verbliebenen Kanten werden hiermit entfernt und das Modell des Werkstücks zu einem Gesamtnetz umgewandelt.

Nähte umwandeln
Verbliebene Nähte werden hiermit in charakteristische Kanten umgewandelt.

Umvernetzen
Die aktuell ausgewählten Teilflächen werden hiermit neu vernetzt. Das heißt, es kann ein feiner oder gröber aufgelöstes 3D-Modell des Werkstücks erzeugt werden.

Normalen invertieren
Alle Flächennormalen des Werkzeugmodells werden invertiert.

Ansicht

Alles anzeigen
Das vollständige 3D-Modell wird angezeigt.

Auswahl einblenden
Ausgewählte Teilflächen, die derzeit nicht angezeigt werden, sind in der 3D-Ansicht wieder sichtbar.

Auswahl ausblenden
Alle aktuell ausgewählten Teilflächen werden in der 3D-Ansicht ausgeblendet.

Flächendarstellung (geglättet)
Zeigt das 3D-Modell des Werkstücks in der 3D-Ansicht in einer geglätteten Flächendarstellung an.

Flächendarstellung (flach)
Zeigt das 3D-Modell des Werkstücks in der 3D-Ansicht in einer flachen Flächendarstellung an.

Drahtgittermodel
Zeigt das 3D-Modell des Werkstücks in der 3D-Ansicht als Drahtgittermodell an.

Hilfe
Über das Menü *Hilfe* können die Tastaturbefehle für den Netzeditor angezeigt werden.

Zusätzlich kann das gesamte Werkstück mit Hilfe der **Positions- und Skalierungssteuerung** im Raum verschoben, gedreht und seine Größe verändert werden. Sämtliche Werte lassen sich durch die Schaltflächen neben den Eingabefeldern oder durch direkte Eingabe neuer Werte über die Tastatur ändern. Dabei werden Änderungen direkt in der *3D-Ansicht* angezeigt. Übernommen werden diese Änderungen allerdings erst durch die Betätigung der Schaltfläche *Anwenden*. Mit der Schaltfläche *Zurücksetzen* werden die ursprünglichen Werte wieder hergestellt, sofern diese noch nicht übernommen wurden.

C.6.5. Der Einstellungsdialog

Im *Einstellungsdialog* (siehe Abbildung C.3) können die Berechnungsparameter der Roboterbahnen für jeden Task individuell eingestellt werden. In der obersten Auswahl kann der Task gewählt werden, für den die Parameter festgelegt werden sollen. Über den Auswahlkasten *Berechnungen durchführen* kann der ausgewählte Task deaktiviert werden, so dass keine Berechnungen durchgeführt werden. Im oberen Bereich *Bahnplanung* können die Werkzeugparameter *Sprühradius*, *Überdeckungsdistanz* und *Werkzuegabstand* festgelegt werden, die den Abstand von Bahnzügen untereinander und den Abstand des Pfades zur Werkstückoberfläche regeln. Die *Winkeltoleranz* steuert die Segmentierung des Werkstücks als Vorverarbeitungsschritt bei der Pfadberechnug.

Im Abschnitt *Bewegungsplanung* werden Art und Approximation der Roboterbahnen sowie die Geschwindigkeit der Roboter beim Abfahren der Bahn gesteuert. *Lineare Abweichung* und *Kreisbahn Abweichung* regeln die Approximation durch je einen Grenzwert. Die Approximation der Pfade durch Kreisbogensegmenten ist optional und kann durch das Kontrollfeld *Kreisbahnen* aktiviert oder deaktiviert werden. Die Geschwindigkeit der Roboter kann in Millimeter pro Sekunde schrittweise im Bereich von 10 mm/s bis 7000

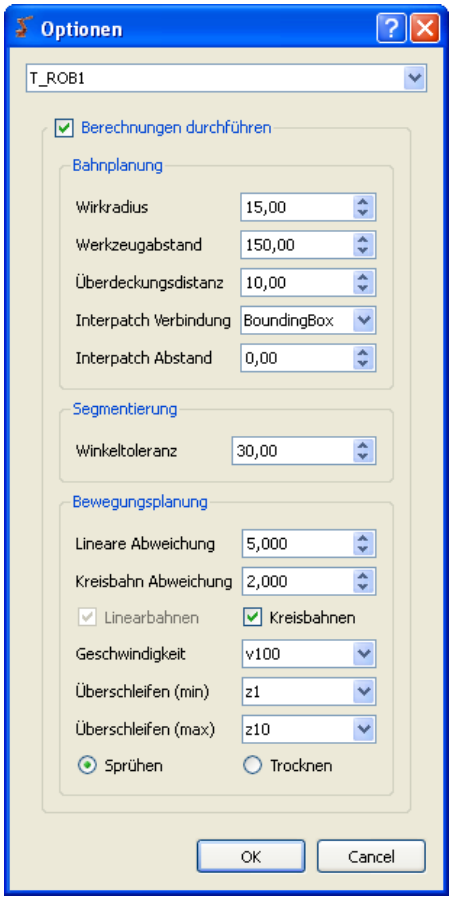


Abbildung C.3.: Der Einstellungsdialog von RoSi3D

mm/s für die gesamte Bewegung des Roboters eingestellt werden. Die Spezifikation des Arbeitsprozesses als *Sprühen* oder *Trocknen* regelt die Reihenfolge der Arbeitsprozesse, so dass der Roboter dem der Task Sprühen zugeordnet ist als erster seine Arbeit aufnimmt. Ebenfalls führt die Auswahl des *Sprühen* Prozesses dazu, dass das Robotersignal zur Ansteuerung des Werkzeugs beim Erreichen des Werkstücks gesetzt wird, welches im Falle des *Trocknens* nicht geschieht.

C.6.6. Der Werkzeugdialog

Im *Werkzeugdialog* (siehe Abbildung C.4) können die Parameter des Werkzeugs, welches der Szene hinzugefügt werden soll, definiert werden. Im obersten Eingabefeld muss zunächst der Verzeichnispfad der CAD-Datei angegeben werden. Die Eingabefelder zur Rotation und Position dienen dazu die Lage des CAD-Modells in der Szene anzupassen bzw. die Position des Tool-Center-Points (TCP) zu korrigieren. Die Transformatio-

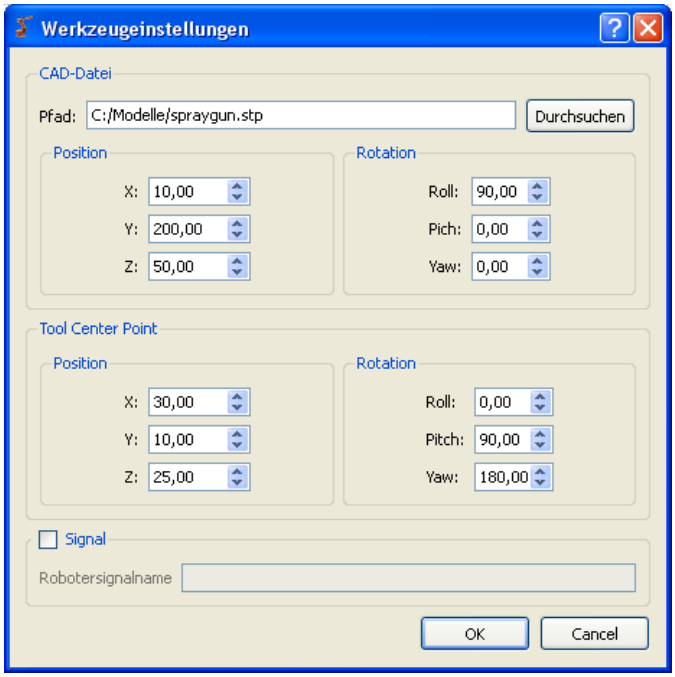


Abbildung C.4.: Der Werkzeugdialog von RoSi3D

nen werden relativ zum Koordinatensystem der letzten Roboterachse angegeben. Beides kann auch zu einem späteren Zeitpunkt im *Projekteditor* bearbeitet werden (siehe Kapitel C.6.3). Im Eingabefeld *Robotersignale* kann ein Signalname zur Ansteuerung des Werkzeugs angegeben werden. Dadurch wird von der Robotersteuerung ein Signal an das Werkzeug geschickt, um das Werkzeug einzuschalten, wenn der Roboter den ersten Pfadpunkt am Werkstück erreicht. Ebenso wird am Ende der Roboterbahn das Werkzeug mit Hilfe dieses Signals deaktiviert.

D. Pflichtenheft

D.1. Vorwort

Dieser Abschnitt enthält das Pflichtenheft der Projektgruppe 523. Bei dem Pflichtenheft handelt es sich um ein Dokument, das typischerweise zu Beginn eines Softwareprojektes vom Entwicklungsteam erstellt wird, um die Anforderungen an die zu entwickelnde Software festzulegen. Das Ziel des Pflichtenhefts ist es, aus den Anforderungen des Kunden, die im sogenannten Lastenheft festgelegt sind, eine möglichst vollständige, konsistente und eindeutige Produktdefinition zu erstellen, die dann die Grundlage für das zu erstellende Produkt bildet. Eine wichtige Eigenschaft des Pflichtenhefts ist, dass es beschreibt *was* realisiert werden soll, aber nicht *wie* dieses geschehen soll – Fragen der Implementierung werden also zunächst nicht berücksichtigt. Die essentiellen Fragen, die das Pflichtenheft hingegen klären soll, sind die Folgenden:

- *Was* muss *wofür* produziert werden?
- Für *was* wird das Endprodukt eingesetzt?
- *Was* muss das Endprodukt demnach beinhalten?
- *Was* müssen die Funktionen des Endprodukts bewerkstelligen?
- *Was* muss gespeichert werden?
- *Womit* wird entwickelt?

Diese kurze Beschreibung zum Pflichtenheft, sowie die verwendete Gliederung basieren auf [Bal01].

D.2. Zielbestimmungen

Es wird ein Softwaresystem entwickelt, welches automatisch Fertigungsbahnen ermittelt, so dass zwei sechs-achsige Industrieroboter kooperierend ein Werkstück bearbeiten können, ohne dabei miteinander oder mit dem Werkstück zu kollidieren. Die so berechne-

te Bewegung dient als Eingabe für das Programm *RobotStudio*, welches zur Simulation der Bewegung entlang der berechneten Bahn verwendet wird. Anschliessend wird die Fertigungsbahn in Form eines RAPID-Programms auf die Roboter übertragen.

D.3. Musskriterien

Folgende Leistungen sind für den erfolgreichen Einsatz des Produkts unabdingbar und müssen bis zum Ende des Projekts realisiert worden sein.

D.3.1. Durchführen des kooperativen Prozesses

Nach dem Planen der Bahn und der Bewegung müssen die Roboter in der Lage sein das zu bearbeitende Werkstück zu besprühen und anschließend zu trocknen. Es ist nicht erforderlich, dass beide Roboter im Parallelbetrieb agieren. Die Besprühung muss einseitig und flächendeckend erfolgen. Eine kollisionsfreie Bewegung ist dabei zwingend erforderlich.

D.3.2. Bahnplanung

Die Bahnplanung erfolgt auf Dreiecksnetzen, daher wird das in Form einer Freiformfläche vorliegende Werkstück trianguliert. Das entstandene Dreiecksnetz wird anhand der Werkstückgeometrie in Segmente eingeteilt, auf denen anschliessend die Bahnen geplant werden.

D.3.3. Kollisionsfreie Bewegungsplanung

Um eine effiziente Kollisionsdetektion zu realisieren, werden die auf Kollision zu testenden Objekte hierarchich unterteilt und die Einzelteile mit quaderförmigen Hüllkörpern umgeben. Außerdem müssen diverse Schnittests durchführbar sein. Die Bewegungsplanung an sich erfolgt durch selbst entworfene Algorithmen.

D.3.4. Erstellen des Rapid-Programms

Anhand der berechneten Fertigungsbahn wird ein RAPID-Programm erzeugt und nach erfolgreicher Simulation durch *RobotStudio* auf die Roboter übertragen.

D.3.5. Eigenschaften des Demonstrators

Die Grösse und Form des Demonstrators sei durch den Arbeitsbereich der Roboter eingeschränkt. Ausserdem soll der Demonstrator keine Löcher oder Vertiefungen aufweisen. Als Eingabeformat müssen mindestens die Formate STEP und IGES unterstützt werden.

D.3.6. Unterstützte Roboter

Folgende Roboter müssen unterstützt werden: ABB IRB 2400/16, ABB IRB 4400/60 und ABB IRB 6620.

D.4. Wunschkriterien

Diese Kriterien sind für die ordnungsgemäße Ausführung des Produkts nicht zwingend erforderlich und müssen daher für den erfolgreichen Abschluss des Projekts nicht realisiert werden. Sie können zusätzlich dem Produkt hinzugefügt werden.

- Bahn- und Bewegungsplanung für zwei Roboter im parallelen Betrieb
- Unterstützung beliebig geformter Werkstücke
- Beidseitige Besprühung
- Besprühung ausgesuchter Teilfächen
- Visualisierung der Szene inklusive der Bahn und der Bewegung

D.5. Abgrenzungskriterien

Diese Kriterien sollen bewusst nicht erreicht werden.

- Bahn- und Bewegungsplanung für mehr als zwei Roboter
- Unterstützung anderer Roboter als Sechs-Achs-Industrieroboter mit Zentralhand
- Berechnung und Simulation in Echtzeit

D.6. Produkteinsatz

D.6.1. Anwendungsbereich

Das fertige Produkt dient zur Bearbeitung von Werkstücken in unterschiedlichen Fertigungsprozessen mit Hilfe von Sechs-Achs-Industrierobotern mittels einer automatischen Ermittlung von optimierten Fertigungsbahnen.

D.6.2. Zielgruppe

Das zu entwickelnde Software-System soll zunächst von Mitarbeitern des Instituts für Roboterforschung der Technischen Universität Dortmund verwendet werden. Generell soll es allen Inhabern von Sechs-Achs-Industrieroboter ermöglicht werden dieses Produkt nutzen zu können. Die benötigte Qualifikation beschränkt sich auf durchschnittliche PC-Kenntnisse und auf die Bereitstellung der spezifizierten Datenformate.

D.7. Produktumgebung

D.7.1. Software

Folgende Software-Umgebung muss auf den Arbeitsplätzen existent sein, um eine volle Funktion des Produktes gewährleisten zu können:

- Betriebssystem: Windows XP SP2
- RobotStudio
- OpenGL 2.x
- .NET Framework 2.0

D.7.2. Hardware

- Zwei sechs-achsige Industrie-Roboter mit Zentralhand folgender Typen: ABB IRB 2440/16, 4400/60, 6620
- Computer mit ausreichender Rechenleistung: Prozessor mit 2.6GHz, Quad-Core und mindestens 2GB RAM

- Grafikkarte mit Hardwarebeschleunigung für OpenGL 2.x und Shader Model 4
- 500 GB Festplattenspeicher

D.7.3. Schnittstellen

- 100 MBit Ethernet-Karte

D.8. Produktfunktionen

In diesem Abschnitt werden die Funktionsbausteine des Projektes aus Benutzersicht aufgeführt. Funktionen sind mit **/F Funktionsnummer /** eindeutig bezeichnet. Ein nachgestelltes W kennzeichnet ein Wunschkriterium.

/F010/ : Werkstücksmodell aus IGES oder STEP Format laden.

Mit dieser Funktion kann das gewünschte Werkstück aus einer IGES oder STEP Datei importiert und geladen werden.

/F110/ : Werkzeug auswählen.

Hiermit kann der Benutzende sein gewünschtes Werkzeug auswählen. Die Werkstückgeometrie muss ebenfalls im IGES oder STEP Format gespeichert sein. Er hat die Auswahl zwischen einem Sprühwerkzeug und einem Heißlufttrockner.

/F120/ : Robotertyp auswählen.

Mit dieser Funktion wird einer der unterstützten Roboter ausgewählt – ebenfalls im IGES oder STEP Format.

/F130W/ : Arbeitsprozess spezifizieren.

Der Benutzende soll die Möglichkeit besitzen den Arbeitsprozess zu spezifizieren: Beispielsweise Sprühen, Trocknen oder eine Kombination aus beidem.

/F140/ : Roboterposition auswählen.

Hiermit wird die Roboterposition im Weltkoordinatensystem angegeben, so dass die Roboterposition nicht von der Software abhängt.

/F210/ : Rückmeldung über Machbarkeit.

Wenn ein Werkstück nicht bearbeitet werden kann, soll dies dem Benutzenden mitgeteilt

werden.

/F220W/ : Begründeter Verwurf eines Werkstücks. Wenn ein Werkstück nicht bearbeitet werden kann, so soll dem Benutzenden eine Begründung gegeben werden, weshalb das Werkstück verworfen wurde.

/F310/ : Bahn planen.

Mit dieser Funktion soll eine mögliche Bahn berechnet werden. Eine Berechnung der Bewegung erfolgt durch diese Funktion noch nicht.

/F320/ : Bahn speichern.

Wenn eine Bahn erfolgreich berechnet wurde, so besteht die Möglichkeit mit dieser Funktion auf ein nicht flüchtiges Speichermedium zu speichern.

/F330/ : Bahn laden.

Der Benutzende soll eine zuvor gespeicherte Bahn in sein Projekt laden können. Zu beachten ist, dass zwingend das gleiche Werkstück und die gleichen Positionierungen von Werkstück und Robotern notwendig sind.

/F340W/ : Visualisierung von Bahn und Szene.

Die berechnete Bahn und die gesamte Szene kann mit dieser Funktion als 3D-Ansicht betrachtet werden.

/F410/ : Bewegungen planen.

Zu einer berechneten oder geladenen Bahn wird die dazugehörige kollisionsfreie Bewegung der Roboter bestimmt.

/F510/ : Projekt speichern .

Hiermit kann die Szene gespeichert werden, diese beinhaltet das Werkstück, die ausgewählten Roboter und Werkzeuge inklusive ihrer Position, sowie die Pose der Roboter. Ausserdem kann die Projektdatei eine berechnete Bahn und eine berechnete Bewegung enthalten.

/F520/ : Projekt laden.

Hiermit kann die Projektdatei geladen werden.

/F610/ : RAPID-Programm simulieren.

Diese Funktion erlaubt das Simulieren der Bewegung der Roboter in *RobotStudio*.

/F620/ : RAPID-Programm speichern.

Der Benutzende hat die Möglichkeit, das RAPID-Programm zu speichern.

/F630/ : RAPID-Programm auf Roboter übertragen.

Nach erfolgreicher Simulation kann das RAPID-Programm auf die Roboter übertragen und auf Wunsch ausgeführt werden.

D.9. Produktdaten

In diesem Abschnitt werden die Daten aufgeführt, die aus Benutzersicht gespeichert werden.

/D010/ : Die Szene

/D020/ : Die berechneten Bahnen

/D030/ : Das RAPID-Programm

D.10. Produktleistungen

An dieser Stelle werden die zeit- und umfangsbezogenen Leistungen genannt, die das Produkt erbringen soll.

/L010/ : Berechnung einer kollisionsfreien Bewegung für Sechs-Achs-Industrieroboter, falls möglich, in angemessener Zeit, abhängig von der Komplexität des zu bearbeitenden Werkstücks und der zur Verfügung gestellten Hardware, jedoch nicht in Echtzeit.

D.11. Benutzeroberfläche

Es wird eine graphische Benutzeroberfläche zur Verfügung gestellt. Diese sollte übersichtlich und intuitiv sein. Sie wird mittels Qt erzeugt und soll menü-orientiert sein. Des Weiteren wird es eine 2D-Darstellung und eine 3D-Ansicht geben.

	sehr sehr wichtig	wichtig	weniger wichtig	unwichtig
Robustheit		X		
Zuverlässigkeit		X		
Korrektheit	X			
Benutzerfreundlichkeit		X		
Effizienz			X	
Portierbarkeit				X
Kompatibilität				X

Tabelle D.1.: Qualitätszielbestimmungen

D.12. Qualitätszielbestimmungen

Tabelle D.1 ist zu entnehmen, wie stark die unterschiedlichen Qualitätsanforderungen an die Software in der Projektgruppe 523 gewichtet wurden.

D.13. Globale Testszenarien

An dieser Stelle werden Testszenarien zum Test der Produktfunktionen (siehe Kapitel D.8) festgelegt.

/T010/ : Einlesen von Werkstücken, die konvex sind und keine Löcher enthalten.

/T020/ : Einlesen von Werkstücken, die konvex sind und Löcher enthalten.

/T030/ : Einlesen von Werkstücken, die konkav sind und Löcher enthalten.

/T040/ : Einlesen eines ungeeigneten Werkstücks.

/T050/ : Bahn- und Bewegungsplanung für die oben genannten Werkstücke durchführen.

D.14. Entwicklungsumgebung

In diesem Kapitel wird ein Überblick gegeben, welche Soft-, Hard- und Orgware für die Entwicklung eingesetzt werden soll.

D.14.1. Software

- Windows XP SP2
- Visual Studio 2005
- OpenCASCADE
- OpenGL 2.x
- .NET Framework 2.0
- Qt
- *RobotStudio*
- MikTeX
- TeXnicCenter
- Notepad++
- WinSCP
- Mozilla Firefox
- Putty
- FileZilla
- Subversion
- OpenOffice 2.3
- FTP Client
- LaTeX

D.14.2. Hardware

- Vom Lehrstuhl 7 der Technischen Universität Dortmund zur Verfügung gestellte PCs
- Sechs-Achs-Industrieroboter mit Zentralhand folgender Typen: ABB IRB 2400/16, 4400/60 und 6620.

- Sprühpistole für den Farbauftrag
- Heißlufttrockner

D.14.3. Orgware

- Subversion
- DoxyGen
- Internetanbindung

D.15. Schlussbemerkung

Da das Pflichtenheft vor dem Beginn der Implementierungsphase erstellt wird, werden folglich nicht alle Punkt des Pflichtenhefts in der Form, wie sie im Pflichtenheft beschrieben sind, umgesetzt.

Auch ergeben sich während der Implementierung teilweise andere Anforderungen bezüglich der Entwicklungsumgebung, weshalb beispielsweise in der Liste der zu verwendenden Software einige der tatsächlich verwendeteten Werkzeuge (siehe Anhang A) fehlen, wohingegen andere Werkzeuge, deren Verwendung eingeplant war, gar nicht zum Einsatz kamen.

Literaturverzeichnis

[3D 89] 3D Systems: *Stereolithography Interface Specification*, October 1989.

[ABB07] ABB AG: *RAPID Referenzhandbuch*, 2007.

[ABB08a] http://www.abb.de, 2008. [Online am 07.08.2008].

[ABB08b] http://www.abb.de/product/seitp327/bfa033f50ad87bd1c12570c9003fe979.aspx, 2008. [Online am 07.08.2008].

[ABB08c] http://www.abb.de/product/seitp327/112776cc678bb61ac1257353003f493e.aspx, 2008. [Online am 07.08.2008].

[ACG+04] Atkar, P. N., D. C. Conner, A. Greenfield, H. Choset und A. A. Rizzi: *Uniform Coverage of Simple Surfaces Embedded in R^3 for Auto-Body Painting.* In: *in Proc. Workshop on Algorithmic Foundations of Robotics, Utrecht/Zeist, The*, Seiten 383–398, 2004.

[ADE08] http://www.engr.colostate.edu/~dga/adept.html, 2008. [Online am 31.10.2008].

[AKM+06] Attene, M., S. Katz, M. Mortara, G. Patane, M. Spagnuolo und A. Tal: *Mesh Segmentation - A Comparative Study.* In: *International Conference On Shape Modeling And Applications*, Seiten 14–25, 2006.

[AM91] Abramowski, S. und H. Müller: *Geometrisches Modellieren.* BI-Wissenschaftsverlag, 1991.

[Ame81] American National Standards Institute: *ANS Y14.26M-1981*, 1981.

[Bal01] Balzert, H.: *Lehrbuch der Software-Technik: Software-Entwicklung.* Lehrbücher der Informatik. Spektrum Akademischer Verlag, Heidelberg, 2nd ed. Auflage, 2001.

[Bau72] Baumgart, B. G.: *Winged edge polyhedron representation.* Technischer Bericht, Stanford University, Stanford, CA, USA, 1972.

[BL07] Bi, Z. M. und S. Y. T. Lang: *Automated robotic programming for products*

with changes. International Journal of Production Research, 45:2105–2118, 2007.

[Ble08] http://www.blender.org, 2008. [Online am 04.09.2008].

[Boo08] http://www.boost.org, 2008. [Online am 04.09.2008].

[BS96] Bronstein, I. N. und K. A. Semendjajew: *Teubner-Taschenbuch der Mathematik*. B. G. Teubner Stuttgart, Leipzig, 2996.

[CKS98] Campagna, S., L. P. Kobbelt und H. P. Seidel: *Directed Edges — A Scalable Representation for Triangle Meshes*. Journal of Graphics Tools: JGT, 3(4):1–12, 1998.

[CXM+04] Chen, H., N. Xi, S. K. Masood, Y. Chen und J. Dahl: *Development of Automated Chopper Gun Trajectory Planning for Spray Forming*. Industrial Robot: An International Journal, 31:297–307, 2004.

[CXSC05a] Chen, H., N. Xi, W. Sheng und Y. Chen: *General Framework of Optimal Tool Trajectory Planning for Free-Form Surfaces*. Journal of Manufactirung Science and Engineering, 127:49–59, 2005.

[CXSC05b] Chen, H., N. Xi, W. Sheng und Y. Chen: *Tool Path Planning for Compound Surfaces in Spray Forming Processes*. Automotion Science and Engineering, 2:240–249, 2005.

[DH55] Denavit, J. und R. S. Hartenberg: *A Kinematic Notation for Lower-Pair Mechanisms Based on Matrices*. J. Appl. Mechanics, June 1955, 22:215–221, 1955.

[DH73] Duda, R. O. und P. E. Hart: *Pattern Classification and Scene Analysis*. John Wiley and Sons Inc, 1973.

[FvDF+94] Foley, J. D., A. van Dam, S. K. Feiner, J. F. Hughes und R. L. Phillips: *Grundlagen der Computergraphik*. Addison-Wesley, 1994.

[GLM96] Gottschalk, S., M. C. Lin und D. Manocha: *OBBTree: A Hierarchical Structure for Rapid Interference Detection*. Computer Graphics, 30(Annual Conference Series):171–180, 1996.

[Got00] Gottschalk, S.: *Collision Queries using Oriented Bounding Boxes*. Doktorarbeit, University of North Carolina at Chapel Hill, 2000.

[IGE80] IGES: *Initial Graphics Exchange Specification – Formerly ANS US PRO/IPO-100-1996*. U.S. Product Data Association, 1980.

[ISO94] ISO 10303: *Industrial automation systems and integration – Product data representation and exchange – Part 1 - 1785.* ISO, 1994.

[ISO06] ISO 14496-16:2006: *Information technology – Coding of audio-visual objects – Part 16: Animation Framework eXtension (AFX).* ISO, 2006.

[ISO08] ISO 19775-1:2008: *Information technology – Computer graphics and image processing – Extensible 3D (X3D)– Part 1: Architecture and base components.* ISO, 2008.

[KBS00] Kobbelt, L. P., T. Bareuther und H. P. Seidel: *Multiresolution Shape Deformations for Meshes with Dynamic Vertex Connectivity.* EUROGRAPHICS 2000, 19(3):C–249–C–259, 2000.

[Lat08] http://www.latex-project.org/ftp.html, 2008. [Online am 13.10.2008].

[Len02] Lengyel, E.: *Mathematics for 3D Game Programming and Computer Graphics.* Charles River Media, 2002.

[Lib08] http://www.cs.kuleuven.ac.be/~ares/libply, 2008. [Online am 04.09.2008].

[Möl97] Möller, T.: *A Fast Triangle-Triangle Intersection Test.* Journal of Graphics Tools, 2(2):25–30, 1997.

[MvR96] Murray, J. D. und W. van Ryper: *Encyclopedia of Graphics File Formats (2nd edition).* O'Reilly, Sebastopol, CA, USA, Mai 1996.

[New08] http://www.robertnz.net/nm_intro.htm, 2008. [Online am 04.09.2008].

[OCPZ04] Oh, J. S., Y. H. Choi, J. B. Park und Y. F. Zheng: *Complete Coverage Navigation of Cleaning Robots using Triangular-Cell-Based Map.* IEEE Transactions on Industrial Electronics, 51(3):718–726, 2004.

[Ope08a] http://www.opencascade.org, 2008. [Online am 04.09.2008].

[Ope08b] http://www.opengl.org, 2008. [Online am 04.09.2008].

[Ope08c] http://www-i8.informatik.rwth-aachen.de/OpenMesh/html/download.html, 2008. [Online am 04.09.2008].

[PKA03] Page, D. L., A. Koschan und M. Abidi: *Perception-based 3D Triangle Mesh Segmentation Using Fast Marching Watersheds.* In: *International Conference On Computer Vision And Pattern Recognition*, Seiten 27–32, 2003.

[Qt08] http://trolltech.com/products/qt, 2008. [Online am 04.09.2008].

[Rei05] Reiter, S.: *Laufzeit-effizientes Raytracing.* Diplomarbeit, Johannes Kepler Universität Linz, 2005.

[Rob08] http://www.maschinenmarkt.vogel.de/themenkanaele/automatisierung/robotik/articles/94352/, 2008. [Online am 28.08.2008].

[Sac07] Sachs, M.: *Wahrscheinlichkeitsrechnung und Statistik für Ingenieurstudenten an Fachhochschulen.* Fachbuchverlag Leipzig im Carl Hanser Verlag, 2007.

[Ste08] http://www.steinel.de, 2008. [Online am 07.10.2008].

[SVN08] http://tortoisesvn.net/downloads, 2008. [Online am 13.10.2008].

[SWG+03] Sander, P. V., Z. J. Wood, S. J. Gortler, J. Snyder und H. Hoppe: *Multi-Chart Geometry Images.* In: *Symposium On Geometry Processing*, Seiten 146–155, 2003.

[VDI08] http://www.vdi.de/, 2008. [Online am 31.10.2008].

[Wal08] http://www.walther-pilot.de, 2008. [Online am 07.10.2008].

[Web02] Weber, W.: *Industrieroboter - Methoden der Steuerung und Regelung.* Fachbuchverlag Leipzig im Carl-Hanser-Verlag, 2002.

[YLL+05] Yamauchi, H., S. Lee, Y Lee, Y. Ohtake, A. Belyaev und H. P. Seidel: *Feature Sensitive Mesh Segmentation With Mean Shift.* In: *International Conference On Shape Modeling And Applications*, Seiten 236–243, 2005.

[Yos90] Yoshikawa, T.: *Foundations of Robotics: Analysis and Control.* MIT Press, 1990.