Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Andreas Dubs

Entwicklung einer industriellen Robotersteuerung mit Trajektorienplanung

Fakultät Technik und Informatik
Department Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Andreas Dubs

Entwicklung einer industriellen Robotersteuerung mit Trajektorienplanung

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang Technische Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr.Ing. Andreas Meisel
Zweitgutachter : Prof. Dr. rer. nat. Wolfgang Fohl

Abgegeben am 25. Mai 2011

Andreas Dubs

Thema der Bachelorarbeit

Entwicklung einer industriellen Robotersteuerung mit Trajektorienplanung

Stichworte

Trajekotrie, Interpolation, Bézierkurve, Überschleife, Koordinatentransformation, Konstruktion der Körper mit Dreiecken, Kollisionsprüfung, bewegungsorientierte Roboterprogrammierung

Kurzzusammenfassung

Der Gegenstand dieser Bachelorarbeit ist der Entwurf und die Implementation einer Roboterarmsteuerung mit der Möglichkeit die Trajektorien zu planen. Mit einer Bedienoberfläche kann der Roboterarm durch die Eingabe von Trajektorien und deren Parametern angesteuert werden. Die eingegebenen Trajektorien werden interpoliert und auf Kollisionen geprüft. Nach der Rückwärtstransformation werden die Punkte an den Roboterarm weiter geleitet.

Andreas Dubs

Title of the paper

Development of an industrial robot control system with trajectory planning

Keywords

Trajectory, Interpolation, Bézier curve, coordinate transformation, construction of the body with triangles, collision detection, motion-oriented robot programming

Abstract

This subject of this bachelor thesis is the design and the implementation of a robot control system with the ability to plan trajectories. The robot arm can be controlled by inputting trajectories and their parameters into an user interface. The trajectories will be interpolated and checked for collisions. After reverse transformation the points will be transferred to the robot arm.

Danksagung

An erster Stelle möchte ich meiner Familie danken, besonders meiner Mutter, die mich über mein ganzes Studim hinweg unterstüzt hat und natürlich meiner Schwester.

Einen ganz besonderen Dank möchte ich an meinen betreuenden Professor Dr. Andreas Meisel richten, der mir dieses interessante Thema zur Verfügung stellte und während unserer Gespräche immer wieder Ideen und Vorschläge äußerte.

Weiterhin möchte ich mich bei dem Robot Vision Team, besonders bei Felix Kolbe für seine Unterstützung und Carsten Fries für die gute Laune bedanken.

Zu guter Letzt, jedoch mit größtem persönlichem Anliegen, möchte ich meinen Freunden danken. Ich danke euch, dass ihr mich während meiner Studienzeit immer wieder ermuntert habt.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einführung

Die Robotersteuerung enthält alle Komponenten und Funktionen, die zum Betrieb, zur Bedienung, zur Programmierung und zur Überwachung eines Roboters erforderlich sind. Die Robotersteuerung übernimmt die verschiedenartigsten Aufgaben:

- Steuerung der Verfahrbewegungen des Roboters
- Beeinflussung der Prozesskomponenten im System
- Beeinflussung der Roboterkomponenten
- Aufnahme und Auswertung von Signalen
- Aufnahme und Verarbeitung von Prozessinformationen zur Beeinflussung des Prozesses
- Synchronisation der Roboterbewegungen mit externen Ereignissen
- Fehlererkennung und -diagnose am Roboter und am Prozess
- Sichere Robotersteuerung für Kollisionserkennung und -vermeidung
- Unterstützung des Bedieners

Um all diese Aufgaben ausführen zu können, benötigt die Robotersteuerung neben einer leistungsfähigen Rechnereinheit Schnittstellen zum Roboter, zur Peripherie, zum Prozess und zum Bediener.

1.1 Struktur einer Robotersteuerung

Die Abbildung 1.1 zeigt die typische Struktur einer Robotersteuerung.

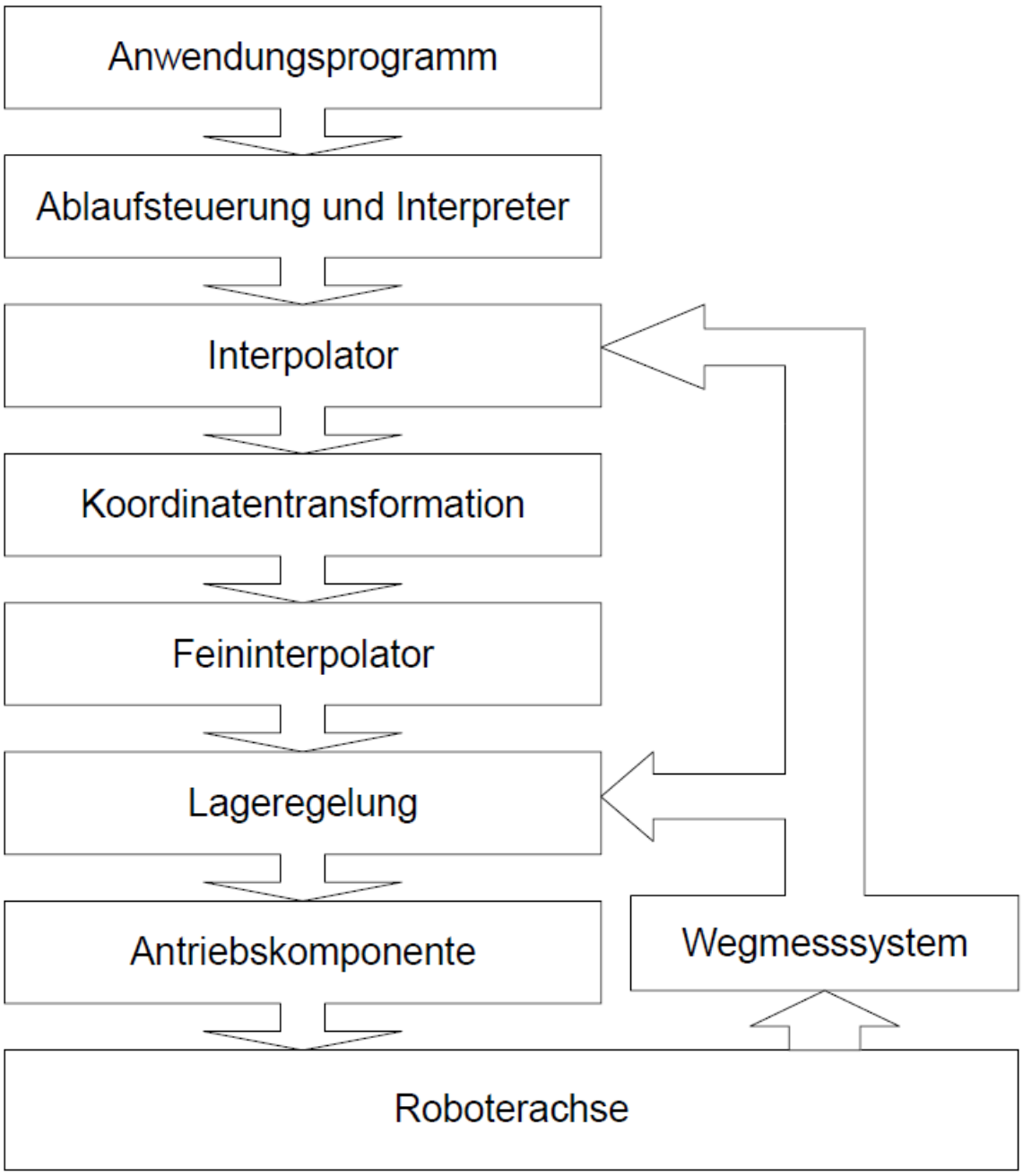

Abbildung 1.1: Struktur einer Robotersteuerung

1.1.1 Anwendungsprogramm

Das Anwendungsprogramm stellt die Bedienung dar. Dieses unfasst die Eingabe von Arbeitsparametern, Programmstart/-stopp, Bewegungsabläufe laden/speichern sowie die Betriebsartwahl. Das Anwendungsprogramm beinhaltet Programmierkomponente der Robotersteuerung für die Erstellung, Wartung und Verwaltung von Bewegungsabläufen. Bewegungsabläufe oder ausgewählte Stellungen des Roboters beziehungsweise Effektors können eingegeben oder getestet werden. Das Anwendungsprogramm enthält Anweisungen für

- Bewegungen des Roboters
- Ansteuerung des Effektors (z.B. Greifer)
- Sensordatenverarbeitung und -auswertung
- Arithmetik
- Aktionssteuerung
- evtl. Zusatzaufgaben

Ein Teil des Anwendungsprogramms sind die Ablaufsteuerung und der Interpreter, welche die Anweisungen der Bewegungsabläufe lesen und dekodieren sowie die entsprechende Ausführungsroutinen aufrufen.

1.1.2 Interpolator und Koordinatentransformation

Der Interpolator ist für die Ansteuerung der jeweils an der Bewegung beteiligten Handhabungseinrichtungen zuständig. Für Bewegungen zwischen zwei Zielstellungen werden anhand der programmierten Vorgaben entsprechende Zwischenstellungen berechnet (interpoliert). Interpolationsarten sind z.B. Punktsteuerung, Vielpunktsteuerung oder Bahnsteuerung.

Es erfolgt die Umrechnung von Weltkoordinaten in Roboterkoordinaten und umgekehrt.

Die Feininterpolation, die Lageregelung und die Antriebskomponenten gehören zur roboterinternen Steuerung. Die Lageregelung versucht, die Motorposition auf der vom Programm gelieferten Sollposition festzuhalten. Von jeder Antriebskomponente wird die Ist-Position überwacht und ausgeregelt. Die errechneten Werte der Achsenwinkel und Achsenwege werden in Motorstrom, Motorspannungen oder Motorinkremente umgesetzt.

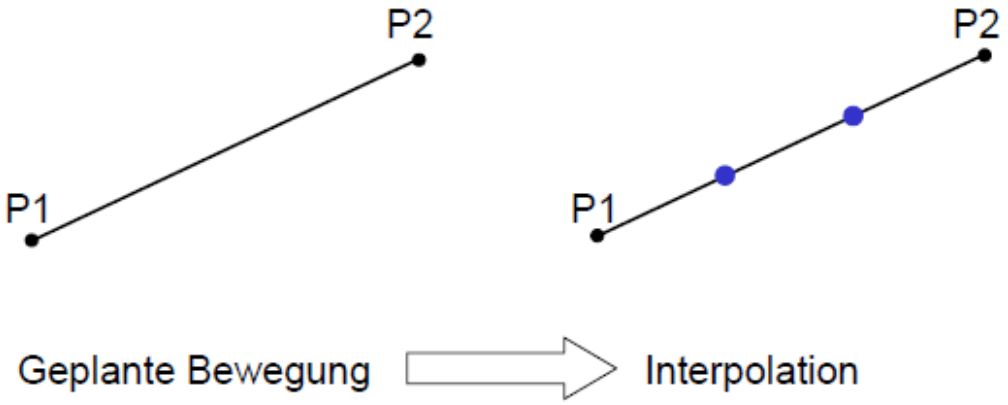

Abbildung 1.2: Interpolation

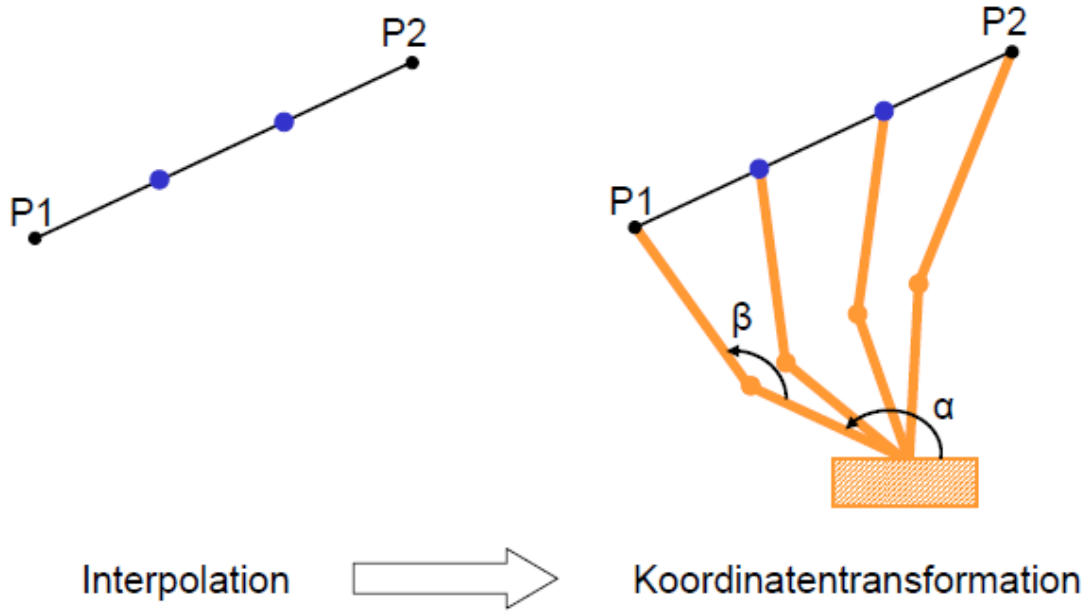

Abbildung 1.3: Koordinatentransformation

1.2 Bewegungssteuerung

Die Bewegungsbefehle eines Roboters enthalten Angaben über eine Roboterstellung und detaillierte Angaben zu der Bewegung, mit der die betreffende Roboterstellung angefahren werden soll.

- Geschwindigkeit / Ausführungszeit
- Überschleifzone
- verwendetes Werkzeug
- Koordinatensystem auf das sich die Koordinaten beziehen
- Bahnkorrektur

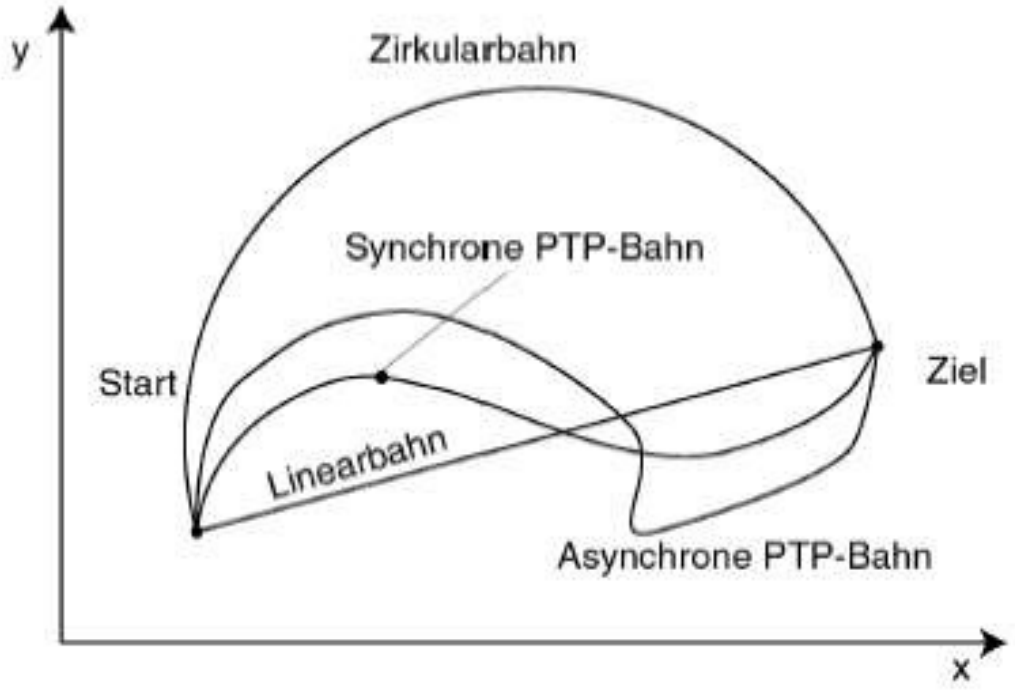

Abbildung 1.4: Bahnbeispiele für verschiedene Steuerungsarten

1.2.1 Continuos Path

Für die Bahnsteuerung (Continuos Path) des TCP (Tool Center Point) werden in kurzen Abständen Interpolationspunkte entlang der gewünschten Bahn erzeugt. Bestimmend für die

Genauigkeit der Bahnbewegung ist der Interpolationstakt, der insbesondere bei stark gekrümmten Konturen, hohen Geschwindigkeiten und hoher Bahngenauigkeit kurz sein sollte. Zusätzlich wird die Mechanik des Roboters durch weiche Bewegungen entlastet.

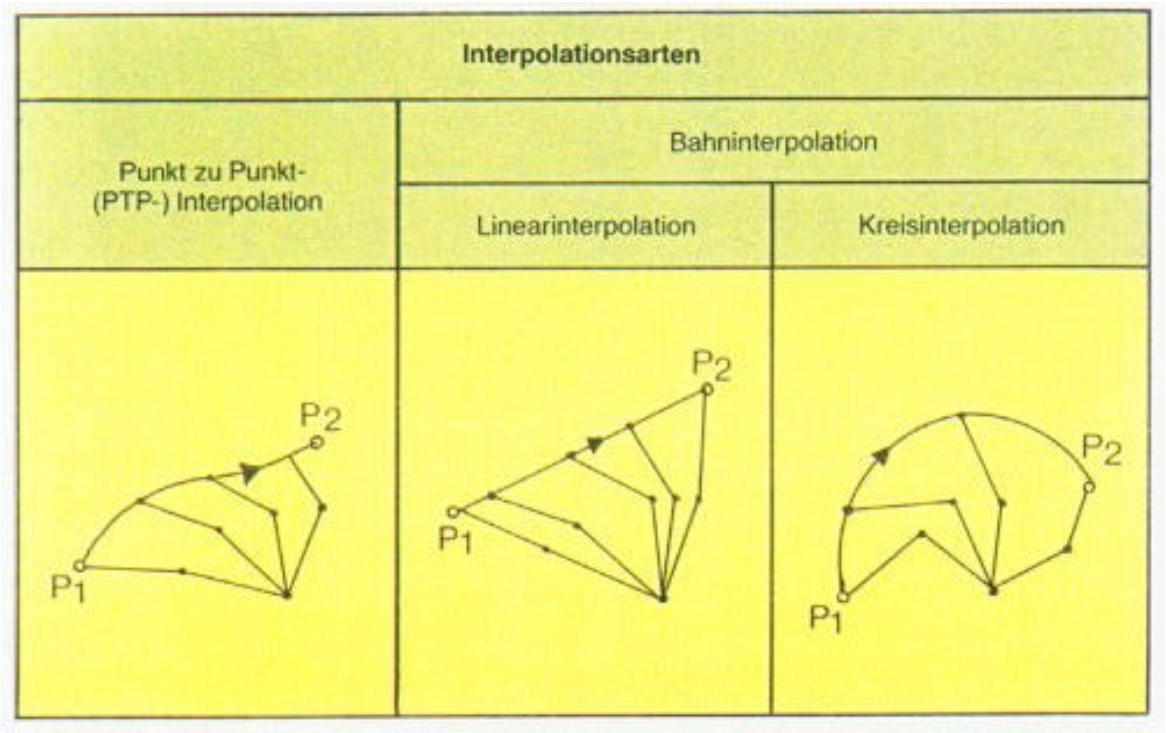

Abbildung 1.5: Interpolationsarten

Für jeden Interpolationspunkt wird die Rückwärtstransformation durchgeführt um Gelenkpositionen zu ermitteln. Die Interpolation findet unter Benutzung mehrerer Interpolationsarten je nach benötigter Bahnkurve statt. Die wichtigsten Interpolationsarten sind:

- Linearinterpolation
- Kreisinterpolation
- Splineinterpolation
- Bézierinterpolation
- Sonderinterpolationen, z.B. Spirale

Sollen Zwischenpunkte exakt durchfahren und weiche Bahnkonturen erzeugt werden, werden Splines verwendet. Durch die Wahl geeigneter Zwischenpunkte lassen sich beliebig gekrümmte Bahnkurven erzeugen.

1.2.2 Überschleifen

Der wichtige Punkt der Bewegungssteuerung sind die Überschleifen, bei denen die einzelne Positionen nur annäherungsweise angefahren werden und der Roboter an diesen Positionen nicht abgebremst wird. Die Positionen dienen also oft nur als Stützpunkte für die Bewegungsplanung oder zum Umfahren von Hindernissen. Mittels Überschleiffaktor kann der Programmierer wählen, wie exakt der Roboter die Position annähern soll.

Da der Roboter die Raumpositionen möglichst ohne Veränderung der Geschwindigkeit durchfährt, reduziert sich der Maschinenverschleiß, der Energieverbrauch und die Ausführungszeit des Bewegungsprogramms erheblich. Die Überschleifen sind zwischen verschiedenen Interpolationen und Bahnen möglich.

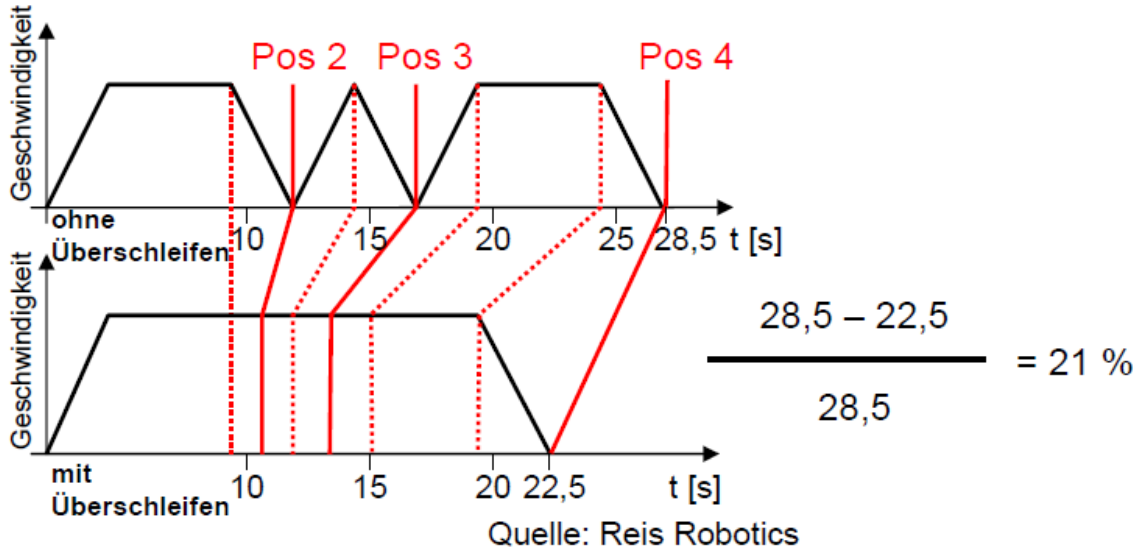

Abbildung 1.6: Zeitersparnis durch Überschleifen

Das Überschleifen kann auf der Gelenkwinkelebene oder auf der Continuos Path Ebene erfolgen.

1.2.3 Koordinatensysteme

Die Berechnungen können sich auf verscheidne Koordinatensysteme stützen. Für die direkte Ansteuerung eines Roboterarmes werden Gelenkkoordinaten gebraucht, die aber oft nicht vorliegen und sich erst durch Koordinatentransformation aus raumbezogenen Koordinaten berechnen lassen. Die raumbezogenen Koordinaten können sich je nach Bezugspunkt unterscheiden:

- Weltkoordinaten - der Ursprung liegt auf der Grundachse, es ist das Hauptkoordinatensystem, auf das sich alle anderen beziehen.
- Werkzeugkoordinaten - der Urpsrung liegt am TCP und die Ausrichtung des Systems ist mit der Ausrichtung des Werkzeugs verbunden.
- Werkstückkoordinaten - das System ist mit dem Werkstück verbunden

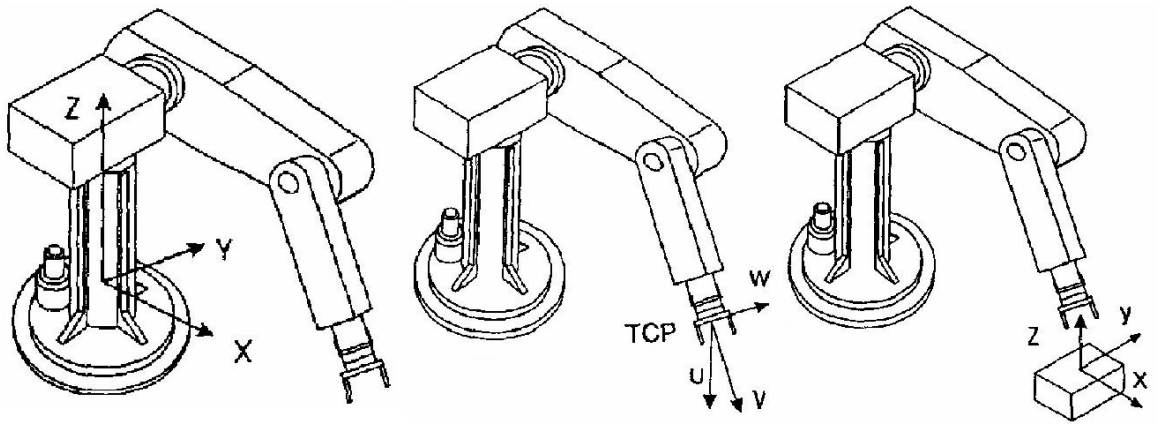

Abbildung 1.7: Koordinatensystemen nach Bezugspunkt

Außer kartesischen Koordinaten können auch Zylinder- oder Polarkoordinaten bunutzt werden. Diese sind ineinander übertragbar und je nach Zweck ist die eine oder andere einfacher zu handhaben.

Die Rückwärtstransformation von Weltkoordinaten zu Gelenkkoordinaten verursacht Mehrdeutigkeiten und Singularitäten, welche zu Problemen führen können. In diesen Fällen müssen spezielle Angaben in die Steuerungssoftware aufgenommen werden, damit Eindeutigkeit erreicht wird. Abbildung 1.8 verdeutlicht dies.

1.2.4 Geschwindigkeitsprofile

Es gibt zwei Grundprofile für die Geschwindigkeitsansteuerung, die in Abbilding 1.9 dargestellt wird.

Das Rampenprofil, bei dem die Geschwindigkeit linear wird, d.h. mit konstanter Beschleunigung verändert wird bis der gewünschte Wert erreicht ist. Das ergibt den Nachteil, dass sich die Beschleunigung und damit die Kraft sprunghaft ändert.

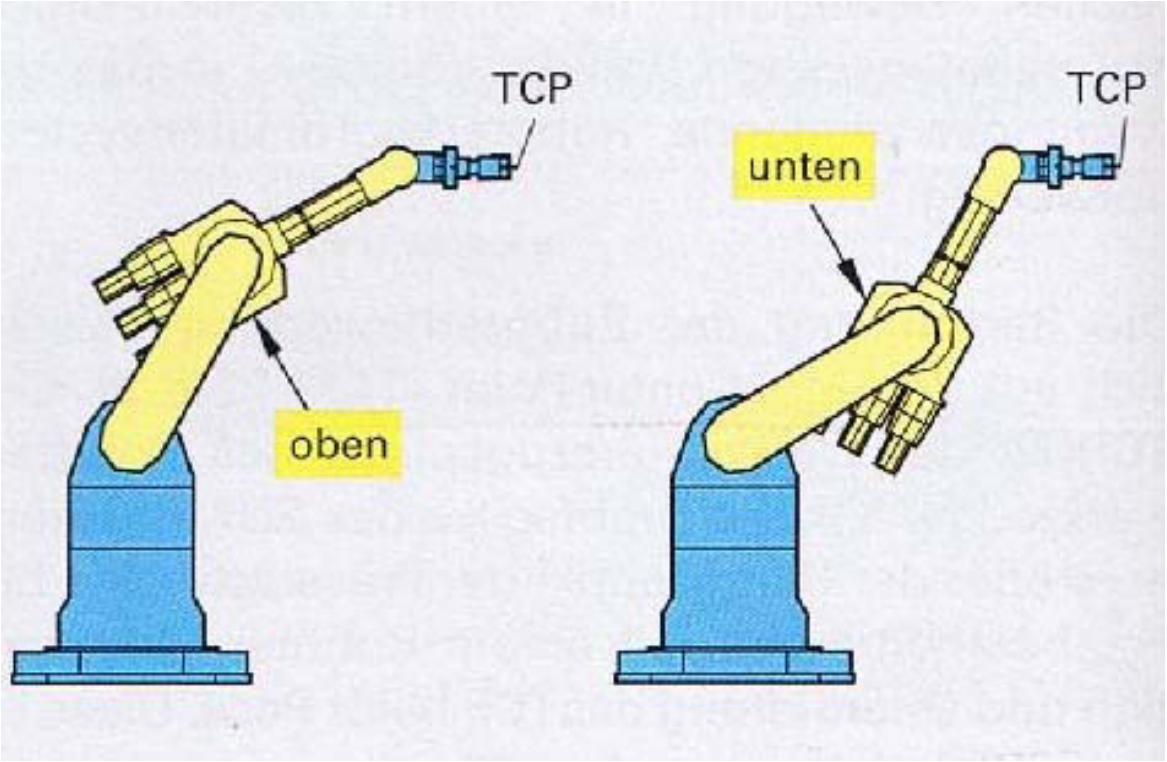

Abbildung 1.8: Mehrdeutigkeit

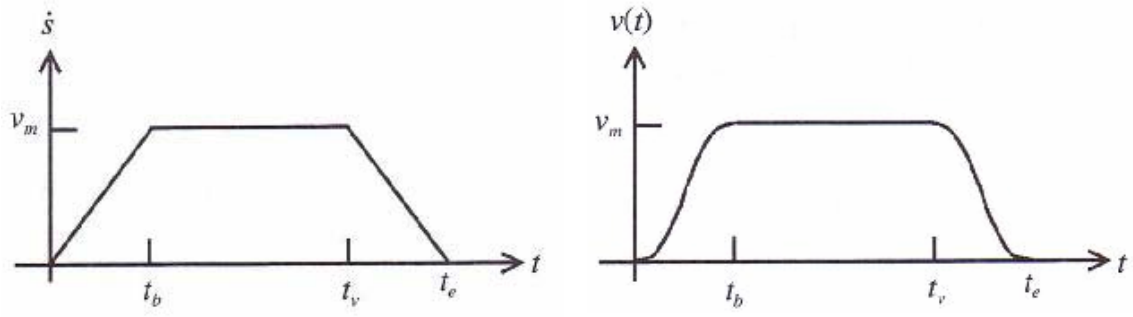

Abbildung 1.9: Geschwindigkeit bei Rampen- und Sinuidenprofil

Das Sinuidenprofil dagegen beschreibt die Beschleinigung mit einer $\sin^2$ Funktion, wobei die Beschleinigung bis zum Maximalwert weich gesteigert wird und dann weich wieder auf Null abfällt. Dabei ändert sich die Kraft stetig, weich und die Bewegung wird ruckfrei und flüssig.

1.3 Sichere Robotersteuerung

Der Hauptaspekt einer sicheren Robotersteuerung sind Kollisionserkennung und -vermeidung durch verschidene Sicherheitsfunktionalitäten:

- achsenspezifische und kartesische Schutzbereiche
- sicher überwachte Geschwindigkeiten
- Stillstandsüberwachung
- Bremsrampenüberwachung

Sie bilden eine Grundlage, sind allein jedoch nicht ausreichend. Durch geeignete Sensoren lassen sich dynamische Körper einrichten. Auch ein Umweltmodell ist erforderlich, damit der Roboter sich sicher zwischen Hindernissen bewegen kann. Für solche Kollisionsbereiche werden geometrische Grundkörper angelegt, z. B. Fläche, Quader, Zylinder. Durch die Verbindung mehrerer Grundkörper lassen sich komplexere 3D-Modelle erzeugen.

Alle Objekte des Roboterumfeldes können im Umweltmodell abgebildet werden, dabei auch der Roboter selbst. Es wird zwischen statischen und dynamischen Objekten unterschieden. Statische Objekte können sich selbst nicht bewegen und sind damit leicht zu handhaben. Das Umweltmodell kann die dynamischen Objekte einbeziehen. Hierzu kann ein Laserscanner das Umfeld des Roboters abtasten.

Es sollte ständig der Abstand zwischen dem Roboter und den Objekten geprüft werden. Befindet sich der Roboter in unmittelbarer Nähe zu einem Objekt, wird die Verfahrgeschwindigkeit in Abhängigkeit des Abstandes reduziert. Um eine Kollision zu vermieden, muss der Roboter automatisch vor dem Hindernis stehen bleiben oder Bahnkorrektur vornehmen und ausweichen. Dabei ist es gleich, ob es sich um statische oder dynamische Objekte handelt.

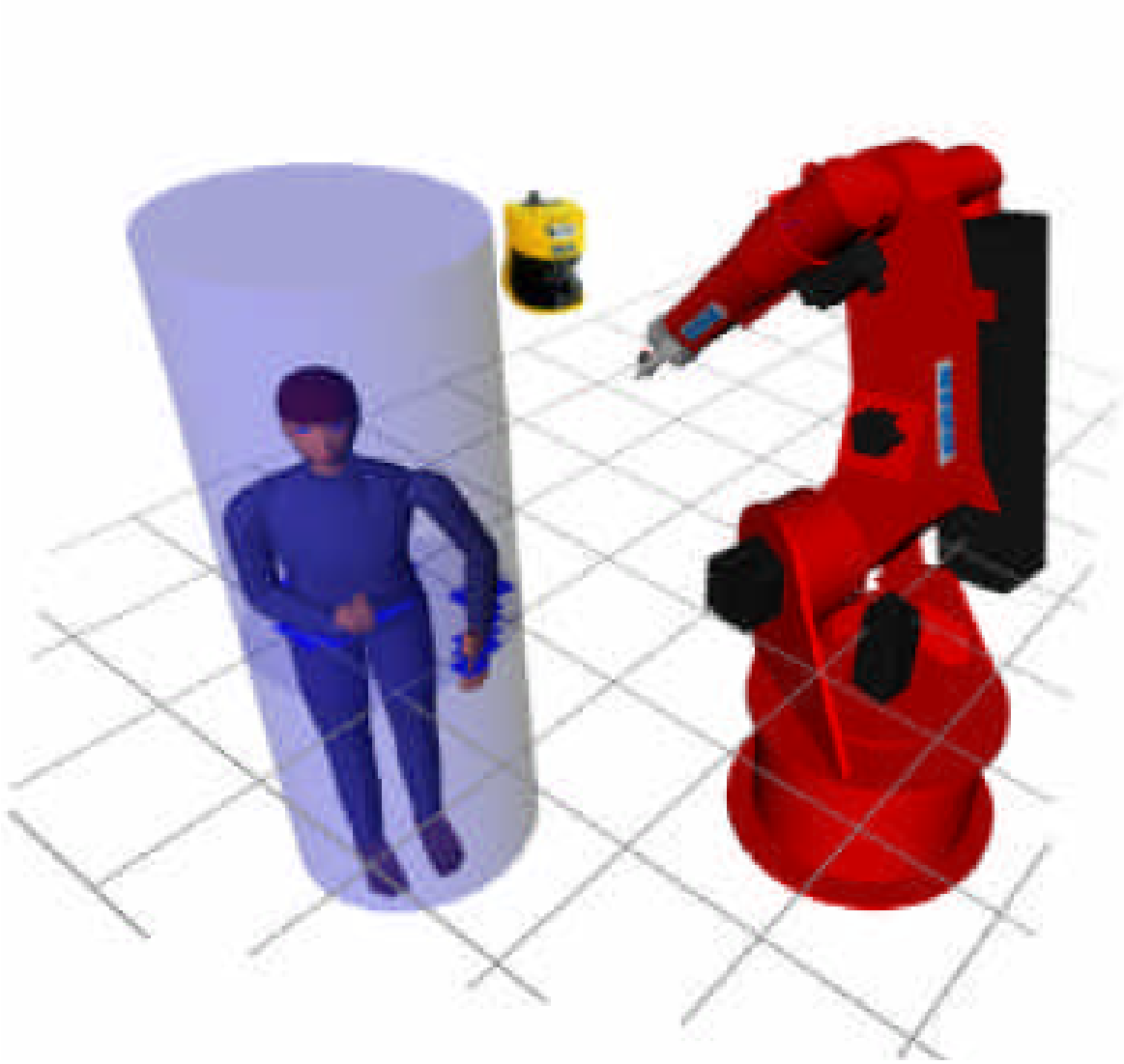

Abbildung 1.10: Dynamischer Körper

1.4 Programmierverfahren

Die Programmierverfahren von Industrierobotern teilen sich in zwei Gruppen:

- Direkte Verfahren - Online Programmierung
- Indirekte Verfahren - Offline Programmierung

1.4.1 Direkte Verfahren

Zu den direkten Verfahren gehören:

- Teach In
- Play-Back - Verfahren
- Master-Slave - Programmierung
- Sensorunterstützte Programmierverfahren

Allgemeine Vorteile solche Programmierarten sind:

- Anschauliche Programmierung
- Geringe Programmierkenntnisse erforderlich
- Kollisionen, Ungenauigkeiten werden direkt sichtbar
- Testen der Programme am realen System
- Umgebung und Objekt müssen nicht vermessen werden
- Erfahrungen des Anwenders fliessen in Programmierung ein

Der Nachteil hingegen ist, dass das Robotersystem dafür erforderlich ist und damit während der Programmierung nicht in den Arbeitsprozess involviert ist.

Teach In beinhaltet manuelles Anfahren der gewünschten Raumpunkte durch Eingabe in ein Bedienfeld und Abspeichern der Bewegungsinformationen durch Bestätigen einer Funktionstaste. Es können zusätzliche Bewegungsanweisungen, wie Geschwindigkeit und Beschleunigung, eingegeben werden.

Die Programmierung bei dem Play-Back - Verfahren erfolgt durch manuelles Führen des Roboters entlang der gewünschten Raumkurve. Dabei werden die Punkte in bestimmten Zeitabständen übernommen.

Die Master-Slave - Programmierung unterscheidet sich von dem Play-Back - Verfahren nur durch ein kleines Modell des Roboters (Master), das anstatt des Roboters selbst bewegt wird. Der Roboter (Slave) folgt und zeichnet die Punkte auf.

1.4.2 Indirekte Verfahren

Bei der Offline - Programmierung erfolgt die Erstellung der Programme aus gesonderten Rechneranlagen. Die textuelle Programmierung findet durch das Schreiben eines Programms in einer von maschinennahen bis zu höherer Roboter-Programmiersprache statt.

CAD-unterstützte Programmierverfahren nutzen geometrische Modelle der beteiligten Komponenten. Es werden Funktionen zur Verfügung gestellt, die eine Festlegung von anzufahrenden Positionen und Verfahrwegen ermöglichen. Die Visualisierung der Roboterbewegungen ermöglicht die Simulation.

Bei expliziten (bewegungsorientierten) Programmierverfahren werden die Ausführungsparameter vom Programmierer vorgegeben und im Gegensatz zu impliziten (aufgabenorientierten) Programmierverfahren werden die Weginformationen vom Programmiersystem unter Verwendung eines Umweltmodells selbsttätig abgeleitet. Einprogrammiert werden lediglich die Beschreibungen von Handhabungsregeln.

Vorteile einer Offline - Programmierung auf einen Blick:

- Programmierung in der Arbeitsvorbereitung als Teil der Planung
- Unterstützung des Programmierers durch rechnerbasierte Hilfsmittel
- Testen der Programme durch Simulation

Aber auch Nachteile sind sehr bedeutend:

- Rechnermodell von Robotersystem und Umgebung erforderlich
- Hohe Kosten durch Programmiersysteme und CAD-Unterstützung
- Aufwändige Einarbeitung in die oft komplexen Programmiersysteme

2 Interpolation

Die Interpolation der vorgegebenen Trajektorien findet in kartesichen Koordinaten statt, die sich leicht aus vorgegebenen Zylinderkoordinaten gewinnen lassen. Dabei zeigt die X-Achse nach vorne, die Y-Achse nach oben und die Z-Achse in die jeweilige Richtung durch Benutzung der Rechte-Hand-Regel.

$$x = r * \cos\phi$$

$$x = h$$

$$x = r * \sin\phi$$

und umgekehrt

$$r = \sqrt{x^2 + z^2}$$

$$h = y$$

$$\phi = \arctan\left(\frac{z}{x}\right)$$

Je nach ausgewählter Trajektorie wird eine passende Bearbeitung vorgenommen. In der Praxis werden die Spline-Kurven oft stückweise durch kubische Bézierkurven ersetzt, weil sie leichter zu berechnen sind und einen ähnlichen Verlauf aufweisen. Daher werden zu linearen und kreisförmigen Trajektorien quadratische und kubische Bézierkurven implementiert.

Zu einem interpolierten Punkt gehören außer Koordinaten auch andere Parameter:

- Neigungswinkel (benötig für Inverse Kinematik)
- Geschwindigkeit, mit der der Punkt angefahren werden soll
- Zeit, die für das Befahren eines Abschnitts gebraucht wird

Die Trajektorieninterpolationen finden zwischen Ein- und Austrittspunkten der Überschleifen statt und stimmen mit dem Anfangs- bzw. Endpunkt der Trajektorie nur im Fall nicht vorhandener Überschleife überein.

2.1 Linear

Mit der inversen Kinematik kann nur der Punkt berechnet werden, der nah zu dem vorherigen liegt. Deswegen sollten auch bei linearen Bewegungen mehrere Punkte dazwischen berechnet werden. Dadurch werden auch größere Abweichungen von der Bahn ausgeschlossen.

Bei linearen Trajektorien ist die Interpolation am einfachsten. Es sind zwei Punkte und die Anzahl zu findender Interpolationspunkte dazwischen vorgegeben. Bei vorhandener Überschleife wird der Ein- bzw. Austrittspunkt berechnet. Dabei wird das Verhältnis der gesamten Länge zu dem Überschleifradius ausgerechnet und ergibt zusammen mit der Koordinatendifferenz zwischen Anfangs- und Endpunkt einen Überschleifpunkt. Die Überschleifen verkürzen so die Trajektorie und ergeben deren neue Anfangs- und Endpunkte.

Durch abziehen der Koordinaten des Anfangspunktes $P_1 = (x_1, y_1, z_1)$ von dem Endpunkt $P_2 = (x_2, y_2, z_2)$ und teilen durch Anzahl n der Abschnitte (Anzahl der Interpolationspunkte + 1) wird die Differenz zwischen Koordinaten zweier Punkte eines Abschnitts errechnet.

$$dx = (x_2 - x_1)/n$$

$$dy = (y_2 - y_1)/n$$

$$dz = (z_2 - z_1)/n$$

Mit dem Neigungswinkel des Greifers am TCP wird auf deie gleiche Weise verfahren.

$$d\phi = (\phi_2 - \phi_1)/n$$

In einer Schleife wird durch Addition der Differenz zu dem letzten Punkt ein neuer Punkt und ein Neigungswinkel erzeugt. Um die für die Bewegung benötigte Zeit zu finden, wird der Abstand zwischen dem errechneten und dem letzten Punkt durch die vorgegebene Geschwindigkeit geteilt.

2.2 Kreis

Ein Bogen der kreisförmigen Trajektorie wird eindeutig durch drei Punkte beschrieben: Anfangspunkt $P_1 = (x_1, y_1, z_1)$, Endpunkt $P_2 = (x_2, y_2, z_2)$ und ein beliebiger Punkt $P_3 = (x_3, y_3, z_3)$, der auf dem Kreis liegt. Der Punkt P_3 sollte aber auf dem Bogen der Trajektorie ausgewählt werden, um zu zeigen in welche Richtung die Trajektorie verlaufen soll, da sonst von P_1 zu P_2 zwei Wege führen könnten.

2.2.1 Mittelpunkt finden

Um den Kreisbogen interpolieren zu können muss erst der Kreismittelpunkt $P_0 = (x_0, y_0, z_0)$ gefunden werden. Jeder Kreispunkt hat den gleichen Radius R vom Mittelpunkt P_0.

$$(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2 = R^2$$

Nach dem Ausklammern wird der Term $x_0^2 + y_0^2 + z_0^2 - R^2 = A$ eingesetzt.

$$2xx_0 + 2yy_0 + 2zz_0 - A = x^2 + y^2 + z^2$$

Dabei werden alle drei vorhandene Punkte P_1, P_2 und P_3 nacheinander in die Gleichung eingefügt.

$$2x_1x_0 + 2y_1y_0 + 2z_1z_0 - A = x_1^2 + y_1^2 + z_1^2 \tag{2.1}$$
$$2x_2x_0 + 2y_2y_0 + 2z_2z_0 - A = x_2^2 + y_2^2 + z_2^2 \tag{2.2}$$
$$2x_3x_0 + 2y_3y_0 + 2z_3z_0 - A = x_3^2 + y_3^2 + z_3^2 \tag{2.3}$$

Durch Subtrahieren (3.1)-(3.2), (3.1)-(3.3) und (3.2)-(3.3) ergeben sich drei neue Gleichungen ohne Term A.

$$x_0(x_1 - x_2) + y_0(y_1 - y_2) + z_0(z_1 - z_2) = \frac{(x_1^2 + y_1^2 + z_1^2) - (x_2^2 + y_2^2 + z_2^2)}{2}$$
$$x_0(x_1 - x_3) + y_0(y_1 - y_3) + z_0(z_1 - z_3) = \frac{(x_1^2 + y_1^2 + z_1^2) - (x_3^2 + y_3^2 + z_3^2)}{2}$$
$$x_0(x_2 - x_3) + y_0(y_2 - y_3) + z_0(z_2 - z_3) = \frac{(x_2^2 + y_2^2 + z_2^2) - (x_3^2 + y_3^2 + z_3^2)}{2}$$

Die Lösung dieses linearen Gleichungssystems ergibt die Koordinaten des Kreismittelpunktes. Auch der Radius R ist mit einem beliebigen der drei vorgegebenen Punkten berechenbar.

2.2.2 2-Punkt Methode

Um weitere Berechnungen zu vereinfachen wird der Kreismittelpunkt P_0 in den Ursprung gesetzt und später zu den berechneten Punkten wieder addiert. Der gesuchte Punkt $P = (x, y, z)$ hat den Radius R.

$$x^2 + y^2 + z^2 = R^2$$

Der Winkel α zwischen den Ortsvektoren $\vec{P}_1$ und $\vec{P}_2$ ermöglicht die Interpolation in dem er durch die benötigte Anzahl n von Abschnitten geteilt wird.

$$\alpha = \arccos\left(\frac{x_1x_2 + y_1y_2 + z_1z_2}{\sqrt{x_1^2 + y_1^2 + z_1^2}\sqrt{x_2^2 + y_2^2 + z_2^2}}\right)$$

$$\beta = \alpha/n$$

Der Interpolationswinkel β zwischen den Ortsvektoren $\vec{P}$ und $\vec{P}_1$ ergibt die zweite Voraussetzung, die für den Punkt P erfüllt werden muss.

$$\frac{x_1x + y_1y + z_1z}{\sqrt{x_1^2 + y_1^2 + z_1^2}\sqrt{x^2 + y^2 + z^2}} = \cos\beta$$

Da die beiden Punkte auf dem Kreis liegen und den gleichen Radius R von dem Kreismittelpunkt besitzen, kann die Gleichung umgeformt werden.

$$x_1x + y_1y + z_1z = R^2\cos\beta$$

Es gibt unendlich viele Punkte, die mit dem selben Interpolationswinkel β und Radius R um den Anfangspunkt P_1 herum im Kreis angeordnet sind. Um diese Vielzahl an Punkten auf zwei zu reduzieren wird ein Normalenvektor der Ebene des Kreises mit Benutzung des Kreuzproduktes der Anfangs- und Endpunktortsvektoren definiert.

$$\vec{n} = \vec{P}_1 \times \vec{P}_2 = \begin{pmatrix} y_1z_2 - y_2z_1 \\ x_2z_1 - x_1z_2 \\ x_1y_2 - x_2y_1 \end{pmatrix} = \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix}$$

Daraus wird die Koordinatengleichung der Ebene, die durch den Ursprung geht, aufgestellt.

$$n_x x + n_y y + n_z z = 0$$

Durch diese drei Gleichungen mit drei Unbekannten, von denen eine quadratisch ist, werden zwei Punkte ermittelt. Damit festgestellt werden kann, welcher der beiden Punkte verwendet werden soll, ist eine Begrenzung für die kreisförmige Trajektorie auf kleiner als 180° notwendig. Der Punkt mit dem kleinsten Abstand zu dem Endpunkt P_2 ist dann der Gesuchte.

2.2.3 Vektoriele Lösung

Eine einfachere Lösung zum Finden eines Punktes ist es, einen Vektor $\vec{v}$ in der Kreisebene aufzuspannen, der auf dem Kreis liegt (also Radius R hat) und das Skalarprodukt mit dem Vektor $\vec{u}$ aus dem Mittelpunkt zum Anfangspunkt 0 beträgt (also mit Winkel 90°). Daraus kann jeder Punkt durch den Interpolationswinkel β zu Vektor $\vec{u}$ und Vektor $\vec{v}$ beschrieben werden.

$$\vec{u} = \vec{P_1} - \vec{P_0}$$

$$\vec{P} = \vec{P_0} + \vec{u}\cos\beta + \vec{v}\sin\beta \tag{2.4}$$

Um Vektor $\vec{v}$ zu finden wird das Kreuzprodukt von dem Normalenvektor $\vec{n}$ der Ebene und Vektor $\vec{u}$ benötigt.

$$\vec{n} \times \vec{u} = \begin{pmatrix} n_y u_z - n_z u_y \\ n_z u_x - n_x u_z \\ n_x u_y - n_y u_x \end{pmatrix} = \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}$$

Das Kreuzprodukt soll noch auf den Radius R des Kreises normiert werden.

$$R_v = v_x^2 + v_y^2 + v_z^2$$

$$\vec{v} = \frac{R}{R_v} \begin{pmatrix} v_x \\ v_y \\ v_z \end{pmatrix}$$

Jetzt kann jeder Punkt P in Abhängigkeit von dem Interpolationswinkel β zu Anfangsvektor $\vec{u}$ gefunden werden.

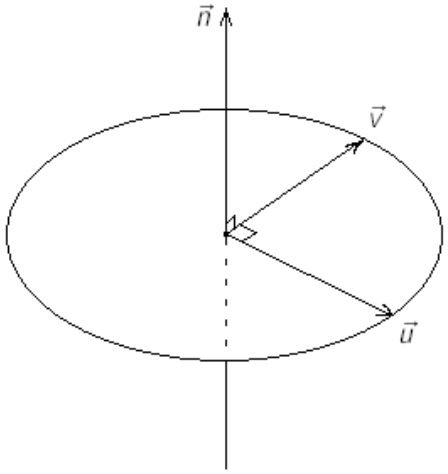

Abbildung 2.1: Aufbau eines Koordinatensystems

Es besteht aber immer noch die Problematik zu entscheiden, ob die Interpolation im oder gegen den Uhrzeigersinn stattfinden soll, also muss das Vorzeichen des Interpolationswinkels β ermittelt werden. Bei der 2-Punkt Methode war es die Entscheidung, welcher der beiden errechneten Punkte näher an dem Endpunkt P_2 liegt, was aber die Begrenzung auf 180° mit sich zog. Um dieses Problem zu lösen, wird der Winkel δ errechnet. Dafür wird in die Gleichung (2.4) der dritter vorgegebene Punkt P_3 eingesetzt, der auf dem Bogen der Trajektorie liegen sollte, wie Oben schon erwähnt wurde.

$$\vec{P_3} = \vec{P_0} + \vec{u}\cos\delta + \vec{v}\sin\delta$$

Es reicht den Winkel δ nur für eine Koordinate zu berechnen.

$$x_3 = x_0 + u_1\cos\delta + v_1\sin\delta$$

$$\delta = \arcsin\left(\frac{x_3 - x_0}{\sqrt{u_1^2 + v_1^2}}\right) - \arctan\left(\frac{u_1}{v_1}\right)$$

Wenn $0 < \delta < \alpha$, dann ist eine positive Drehrichtung vorhanden, andernfalls eine negative Drehrichtung.

Die Berechnung des Neigungswinkels und der Zeit erfolgt wie bei der linearen Interpolation. Für die Suche der Überschleifpunkte wird der Winkel γ berechnet, der sich aus dem Radius R des Trajektorienkreises und Überschleifengröße s ergibt. Dafür wird ein gleichschenkliges Dreieck aufgebaut (beide Radien stellen die Schenkel dar).

$$\gamma = 2 \arcsin\left(\frac{s}{2R}\right)$$

Durch die Interpolation mit dem Winkel γ lassen sich die Überschleifpunkte schnell finden.

2.3 Bézierkurve

Der Vorteil von Bézierkurven ist, dass wenn alle Kontrollpunkte auf einer Geraden liegen, eine lineare Trajektorie zustande kommt. Außerdem erscheint es in dem Anfangspunkt P_1 und Endpunkt P_2 so, als ob die Kurve jeweils zu dem erstem oder aus dem letzten Kontrollpunkt kommt. Die Bézierkurve ist aber auch auf zwei Kurven gleichen Grades teilbar, wobei die neuen Kontrollpunkte sich aus vorherigen ergeben.

2.3.1 Quadratische

Quadratische Bézierkurven haben einen Kontrollpunkt, mit dem der Trajektorienverlauf verändert werden kann. Je grösser der Abstand des Kontrollpunkts P_1 von der Geraden durch die Anfags- und Endpunkte P_0 und P_2, desto größer weicht der Verlauf von einer linearen Trajektorie ab.

Die quadratische Funktion lässt sich einfach durch den Parameter t interpolieren. Zwar sind die Abstände zwischen den Punkten nicht gleich, was unter Umständen zu Problemen führen könnte, der Vorteil ist aber, dass meist bei den Krümungen die Punkte näher beieinander liegen und so höhere Genauigkeiten resultieren.

$$C(t) = \sum_{i=0}^{2} \binom{2}{i} t^i (1-t)^{2-i} P_i = (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2, t \in [0,1]$$

Da die Interpolation durch den Parameter t stattfindet und t von 0 bis 1 läuft, wird die Differenz dt des Parameters zwischen zwei Punkten durch die Abhängigkeit von der Anzahl n berechnet.

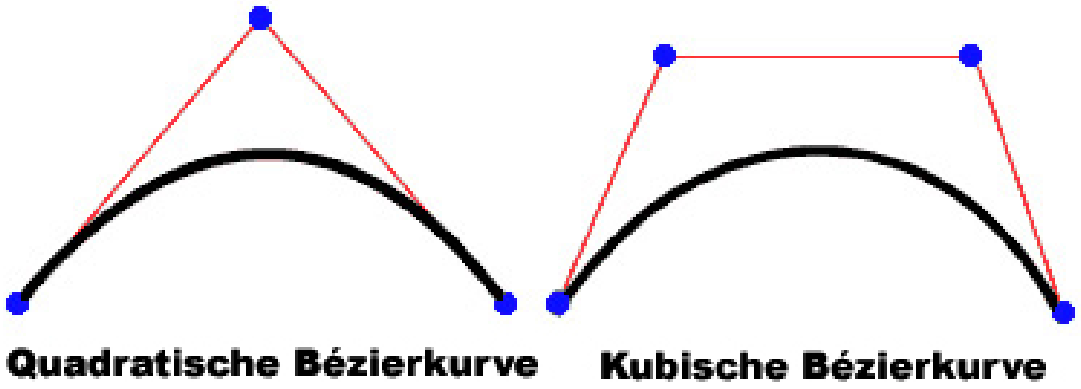

Abbildung 2.2: Bézierkurven

$$dt = \frac{1}{n}$$

In einer Schleife wird diese Differenz zu t des letzten Durchganges addiert und in die Formel eingesetzt, um den nächsten Punkt zu finden. Jede Koordinate eines Punktes wird einzeln mit dieser Formel berechnet.

$$x(t) = (1-t)^2 x_0 + 2t(1-t)x_1 + t^2 x_2$$

2.3.2 Kubische

Bei der kubischen Bézierkurve kommt ein zusätzlicher Kontrollpunkt hinzu, welcher die Funktion um ein Grad erweitert.

$$C(t) = \sum_{i=0}^{3} \binom{3}{i} t^i (1-t)^{3-i} P_i = (1-t)^3 P_0 + 3t(1-t)^2 P_1 + 3t^2(1-t)P_2 + t^3 P_3,\ t \in [0,1]$$

Das Berechnungsprinzip der Punkte ist genau wie bei der quadratischen Bézierkurve. Nach Bedarf können Bézierkurven höheren Grades implementiert werden, da die Umsetzung sich kaum unterscheidet. Lediglich die Funktion wird um einen weiteren Kontrollpunkt erweitert.

Beim Berechnen des Neigungswinkels können bei Bézierkurven Probleme auftretten, da die Interpolationspunkte nicht den gleichen Abstand haben und damit keine gleichmäßige Neigungswinkeländerung erfolgt. Besonders bei starker Krümmung können nahe liegende Punkte eine starke Neigungswinkeländerung hervor rufen. Auf die von der Geschwindigkeit abhängende Zeit wirkt sich das Problem nicht aus, weil die komplette Trajektorie mit gleicher Geschwindigkeit verfahren wird und sich durch kürzere Abstände zu vorherigem Punkt auch kleinere Zeiten ergeben.

Die Suche von Überschleifpunkten ist aber schwieriger und kann mit dem Newton-Verfahren gelöst werden. Auch für die Interpolation kann es sinnvoll sein, wenn die Punkte in gleichen Abständen oder im gleichen Zeitraster berechnet werden sollen.

3 Überschleife

Die Überschleife wird benutzt, um den Übergang von einer Trajektorie zu einer anderen möglichst glatt darstellen zu können. Dies vermeidet abrupte Richtungsänderungen der Bewegung des Roboterarmes. Um dies zu erreichen hat jeder Punkt, der das Ende einer und den Anfang der nächsten Trajektorie darstellt, einen Überschleiffaktor (Radius), wodurch eine Kugel entsteht.

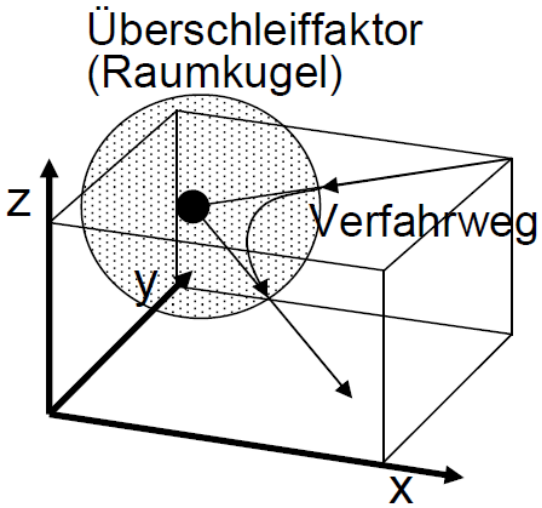

Abbildung 3.1: Überschleifkugel

Wichtig sind die Punkte, die mit Ein- oder Austretten der Kugels durch eine Trajektorie entstehen. Oft ist es schwierig einen solchen Punkt passend zu der vorhandenen Trajektorie zu finden. Wenn bei einer Kreistrajektorie der Winkel zwischen dem Mittelpunkt und einem Ein- oder Austrittspunkt berechnet werden kann und eine Überlschleifeinterpolation ermöglicht, so ist bei der Bézierkurve eine genaue Berechnung solcher Punkte äußerst schwierig, besonders wenn es sich um Bézierkurven höheren Grades handelt. Schon bei einer quadratischen Bézierkurve ensteht ein Polynom 4 Grades und bei kubischer 6 Grades. Unter Benutzung von Näherungsverfahren lässt sich der Punkt finden.

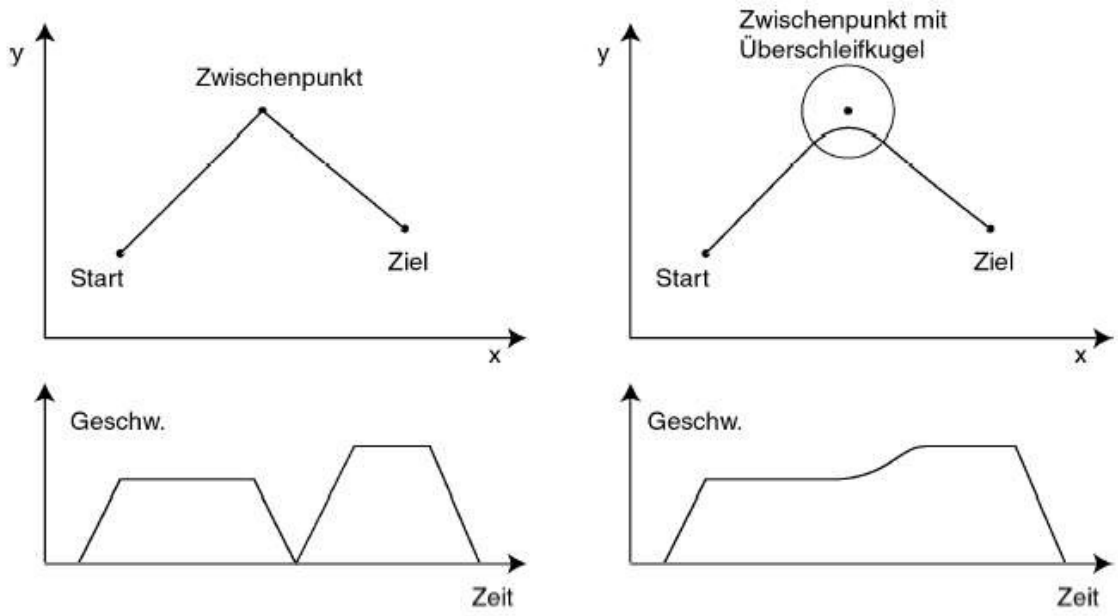

Abbildung 3.2: Beispiel einer Überschleife

Es sind bei einer Überschleife drei Punkte vorhanden: Mittelpunkt, Eintrittspunkt, Austrittspunkt. Implementiert wurden zwei Verfahren, die eine Überschleife interpolieren und eigene Vor- und Nachteile haben.

3.1 Kreisförmig

Die kreisförmige Überschleife benutz die Erkentnisse von der Interpolation eines Kreises. Der Unterschied besteht in den Vorgaben. Wenn bei der Interpolation drei Punkte eines Kreises gegeben waren, so sind bei einer Überschleife nur zwei Punkte dafür benötigten Kreises bekannt: Ein- und Austrittspunkte P_1 und P_2. Der Mittelpunkt P_m der Überschleifkugel kann, muss aber nicht, auf dem für die Überschleife benötigten Kreis liegen. Trotzdem hilft P_m dabei, den Mittelpunkt P_0 zu finden.

Alle drei Punkte liegen in einer Ebene, die die Überschleifkugel schneidet und auf der der gesuchte Kreis liegt. In der selben Ebene muss also auch der Mittelpunkt P_0 liegen. Bekannt ist auch der Übereschleifradius R_u der Übreschleifkugel, der sich notfalls aus dem Abstand zwischen P_1 bzw. P_2 und P_m ergibt. Die Geraden P_0P_1 und P_0P_2 sind die Tangenten des gesuchten Kreises. Die Tangenten der Überschleifkugel in der Schnittebene stellen die Radien des gesuchten Kreises dar und bilden im Schnittpunkt den Mittelpunkt P_0. Die Abbildung 3.3 auf Seite 32 verdeutlicht die Zusammenhänge dieser Konstruktion.

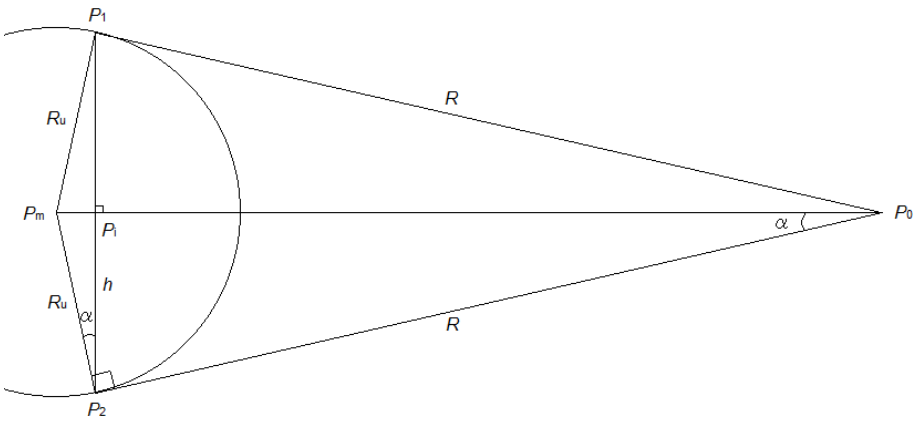

Abbildung 3.3: kreisförmige Überschleife

Radius R lässt sich schnell aus dem Winkel α, R_u und h berechnen.

$$h = \frac{1}{2}\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

$$\alpha = \arccos\left(\frac{h}{R_u}\right)$$

$$R = \frac{h}{\sin\alpha}$$

Auch der innere Punkt P_i ist für die Berechnung des Mittelpunktes P_0 von Bedeutung.

$$\vec{P_i} = \frac{\vec{P_2} - \vec{P_1}}{2} + \vec{P_1} = \frac{\vec{P_2} + \vec{P_1}}{2}$$

$$\vec{P_0} = \frac{\vec{P_i} - \vec{P_m}}{\sin^2\alpha} + \vec{P_m}$$

Die Interpolation der Übreschleife kann jetzt wie in 2.2.3 erfolgen. Für die Richtungsbestimmung kann der Punkt P_m verwendet werden. Für die Anzahl der Abschnitte wird der Kreisbogen zwischen P_1 und P_2 berechnet und durch die für die Überschleife in den Einstellungen vorgegebene Abschnittlänge geteilt. Dies verhindert den Verlust der Genauigkeit bei höheren Überschleifradien.

Die kreisförmige Übereschleife hat einen Nachteil. Wenn die Punkte P_1 und P_2 Diameter der Überschleifkugel bilden, werden die Tangenten parallel zueinander und können nicht den

Mittelpunkt P_0 erzeugen. Der Ansatz mit der quadratischen Bézierkurve hat dieses Problem nicht, da dieser Spezialfall die Bézierkurve zu einer Geraden entarten lässt.

Die Trajektorien können verschiedene Geschwindigkeitsangaben haben und so muss bei der Übreschleifinterpolation auch die Geschwindigkeit angepasst werden. Der Neigungswinkel dagegen bleibt in einer Überschleife konstant in der ganzen Überschleifkugel.

3.2 Quadratische Bézierkurve

Die Berechnung der Überschleife mit der quadratischen Bézierkurve erfolgt wie in 2.3.1 beschrieben, nur mit den Punkten P_1 und P_2 als Anfangs- und Endpunkte und dem Kontrollpunkt P_m. Leider ist es nicht möglich ohne weiteres die Länge einer Bézierkurve zu berechnen, was wiederum bei höheren Überschleifradien bei einer festen Anzahl an Interpolationspunkten zu größeren Ungenaugkeiten führt. Die Länge L lässt sich schätzungweise mit dem Winkel α zwischen den Vektoren $\vec{P_1} - \vec{P_m}$ und $\vec{P_2} - \vec{P_m}$ und dem Überschleifradius R_u ermitteln.

$$L = R_u \left(1 + \frac{\alpha}{\pi}\right)$$

Bei 180° stellt die Überschleifkurve eine Gerade dar. Die Länge L ist nichst anderes als der Diameter der Übreschleifkugel mit $L = 2R_u$. Bei 90° ähnelt der Verlauf mit einem Viertel des Kreises mit Bogenlänge $R_u\frac{\pi}{2} \approx 1.57R_u$ im Vegleich zu $l = 1.5R_u$ aus der Schätzung.

Wenn es bei der kreisförmigen Interpolation zu einer gleichmäßigen Anpassung kommt, so tritt bei der Bézierkurve das Problem bei starken Krümmungen und großen Geschwindigkeitsunterschieden der anliegenden Trajektorien ein.

4 Sichere Bereiche

Es können Ebenen und Körper durch Dreiecke (begrenzte Ebenen) erstellt werden. Durch das Erweitern der Funktionalität können auch Grundkörper, wie Kugel, Zylinder oder Kegel konstruiert werden. Darüber hinaus ist noch ein Bereich eingebaut, der erreicht werden kann, wenn also ein Punkt sich außerhalb dieses Bereiches befindet, so kann er auch nicht erreicht werden.

4.1 Ebene

Der Boden, di eDecke und die Wände können über Ebenen representiert werden. Durch drei Punkte mit Ortsvektoren $\vec{a}$, $\vec{b}$, $\vec{c}$ kann eine Ebene beschrieben werden und in die Hessesche Normalform gebracht.

$$\vec{n} = \left(\vec{b} - \vec{a}\right) \times (\vec{c} - \vec{a})$$

$$\vec{n}_0 = \frac{\vec{n}}{|\vec{n}|}$$

Der Abstand s vom Punkt P zu Ebene kann ermittelt werden. Dabei ist das Vorzeichen des Abstandes ausschlaggebend. Est es negativ, so befindet sich der Punkt P in dem selben Halbraum wie der Ursprung (Montierpunkt des Roboterarmes auf der Roboterplatte), ist es dagegen positiv, so ist eine Kollision in Aussicht.

$$s = \vec{p} \cdot \vec{n}_0 - \vec{a} \cdot \vec{n}_0 = (\vec{p} - \vec{a}) \cdot \vec{n}_0$$

Bei vorgegebenen Abständen zu den Ebenen kann auch die Geschwindigkeit reduziert werden, um Ungenauigkeiten auszuschließen und Benutzer aufmerksam zu machen.

Mit den Ebenen lassen sich auch andere Körper konstruiren, wobei die Schnittgeraden der Ebenen die Kanten darstellen. Das Schwierige dabei ist es festzuhalten, welche Ebenen beim Schnitt als Kanten des Körpers gelten und welche vernachlässigt werden können.

4.2 Körper durch Dreiecke

Ein Körper kann auch durch Dreiecke konstruiert werden, bei denen die Verbindungen von den Eckpunkten die Kanten representieren (falls Dreiecke nicht in der selben Ebene sind) und die Ebene begrenzen. Wenn also zwei Dreiecke zwei gleiche Punkte besitzen, die sie beschreiben, so besitzt der Körper eine Kante, die durch diese Punkte beschrieben wird.

Um zu bestimmen, welche Seite des Dreiecks innen und welche außen liegt, wird ein Normalenvektor so aufgebaut, dass er vom Körper ausgeht, also die Richtung vom Körper weg zeigt. Bei der Eingabe der Punkte sollte also die Rechte-Hand-Regel benutzt werden, um die Punkte in richtiger Reihenfolge einzugeben. Für die Berechnung wird erst der Normaleinheitsvektor $\vec{n}_0$ bestimmt.

$$\vec{u} = \vec{b} - \vec{a}$$

$$\vec{v} = \vec{c} - \vec{a}$$

$$\vec{n} = \vec{u} \times \vec{v}$$

$$\vec{n}_0 = \frac{\vec{n}}{|\vec{n}|} = \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix}$$

Wenn sich ein Punkt innerhalb eines Körpers befindet, so muss ein Projektionsvektor mit dem Innenpunkt als Anfang und senkrecht auf die Projektionsebene treffend (Endpunkt gehört zu Projektionsebene) die gleiche Richtung haben wie der Normaleinheitsvektor $\vec{n}_0$. Der Punkt $P = (p_x, p_y, p_z)$ mit Ortsvektor $\vec{p}$ stellt TCP dar. Seine Projektion auf die Ebene mit dem Normaleneinheitsvektor $\vec{n}_0$ ergibt den Ortsvektor $\vec{q}$ und kann durch die Gerade (Punkt-Richtungs-Form) dargestellt werden.

$$\vec{q} = \vec{p} + \lambda \vec{n}_0$$

Weil der Punkt Q in der Ebene liegt, so erfüllt er die Gleichung:

$$\vec{n}_0 \left(\vec{q} - \vec{a}\right) = 0$$

Nach dem Einsetzen von $\vec{q}$ lässt sich λ berechnen.

$$\vec{n}_0 (\vec{p} + \lambda \vec{n}_0 - \vec{a}) = 0$$

$$\vec{n}_0 (\vec{p} - \vec{a}) + \lambda \vec{n}_0 \vec{n}_0 = 0$$

$$\lambda = \frac{(\vec{a} - \vec{p}) \cdot \vec{n}_0}{|\vec{n}_0|^2}$$

Wenn also λ negativ ist, so befindet sich der Punkt P auf der Innnenseite. Diese Eigenschaft muss für alle Körperebenen gelten, damit der Punkt P sich innerhalb des Körpers befindet. Bei $\lambda = 0$ liegt der Punkt P auf der Ebene und damit auf der Oberfläche des Körpers.

Bei komplexen Körpern, die eine gekrümmte Form aufweisen, können die Ebenen mit der Normale in die gegengesetze Richtung auftretten als eigenlich erwünscht. Also scheint der Punkt auf der Außenseite zu liegen, obwohl er sich im körper befindet. Um dies zu vermeiden, können nur die Ebenen berücksichtigt werden, bei denen die Projektion mit dem Ortsvektor $\vec{q}$ in dem Dreieck liegt.

$$\vec{q} = \begin{pmatrix} p_x + \lambda n_x \\ p_y + \lambda n_y \\ p_z + \lambda n_z \end{pmatrix}$$

Die Projektion $\vec{q}$ kann durch die linear unabhängigen Vektoren $\vec{u}$ und $\vec{v}$ dargestellt werden, die als Basis genommen werden.

$$\vec{q} = s\vec{u} + t\vec{v}$$

Nach dem Einsetzen der Werte für $\vec{q}$, $\vec{u}$ und $\vec{v}$ ensteht ein überbestimmtes lineares Gleichungssystem. Nachdem s und t gefunden wurden, erfolgt die Bewertung. Der Punkt Q befindet sich innerhalb des Dreiecks oder auf dessen Rand, wenn gilt: $s \geq 0, t \leq 1$ und $s + t \leq 1$. Wenn dass der Fall ist, so kann der Punkt P auf das Dreieck abc projiziert werden.

Es können aber immer noch keine Körper geprüft werden, deren Krümung kleiner als 90° ist, also keine Körper mit „U-Form“. Es ist ratsam solche Körper in Teilkörpern darzustellen.

5 Aufbau

Um die Wartung zu erleichtern, wird das Programm auf zwei unabhängige Module *UI* und *Control Thread* des Roboterarms aufgeteilt. Beide Module kommunizieren miteinander durch das Signal-Slot Konzept von der Entwicklungsumgebung Qt (Abbildung 5.1). Es ist dadurch möglich zum späterem Zeitpunkt einzelne Module zu erweitern/ersetzen oder mit neuen zu verbinden.

5.1 User Interface

Das User Interface wurde mit Auslagerung der Roboterarmsteuerung in *Control Thread* und Veränderung der Funktionalitäten an den Vorgänger angelehnt. So wurde der Tab *Inverse Kinematik* komplett rausgenommen, da die Berechnung in *Control Thread* statt findet. Es wurden die Tabs *Helppoints*, *Bodies* und *Settings* hinzugefügt, um die Funktionalität der Trajektorienplanung zu erweitern.

Tab Control

Auf der linken Seite können direkte Werte der Module des Roboterarms eingegeben und mit *Exec Position* ausgeführt werden. Mit *Save Position* lassen sich diese Eingaben in die Trajektorie übernehmen.

Auf der rechten Seite wird die aktuelle Positon des Tool Center Points angezeigt und kann mit *Save current Position* für Trajektorie übernommen. Außerdem befinden sich hier die Buttons für die Roboterarmstuerung, wie Restart, Halt, Stop. *Start Motion* stößt die Ausführung der Trajektorien, falls zu diesem Zeitpunkt bei *Control Thread* nicht abgearbeitete Punkte vorhanden sind. Auch der Infobereich ist hilfreich beim Auswerten auftrettender Probleme.

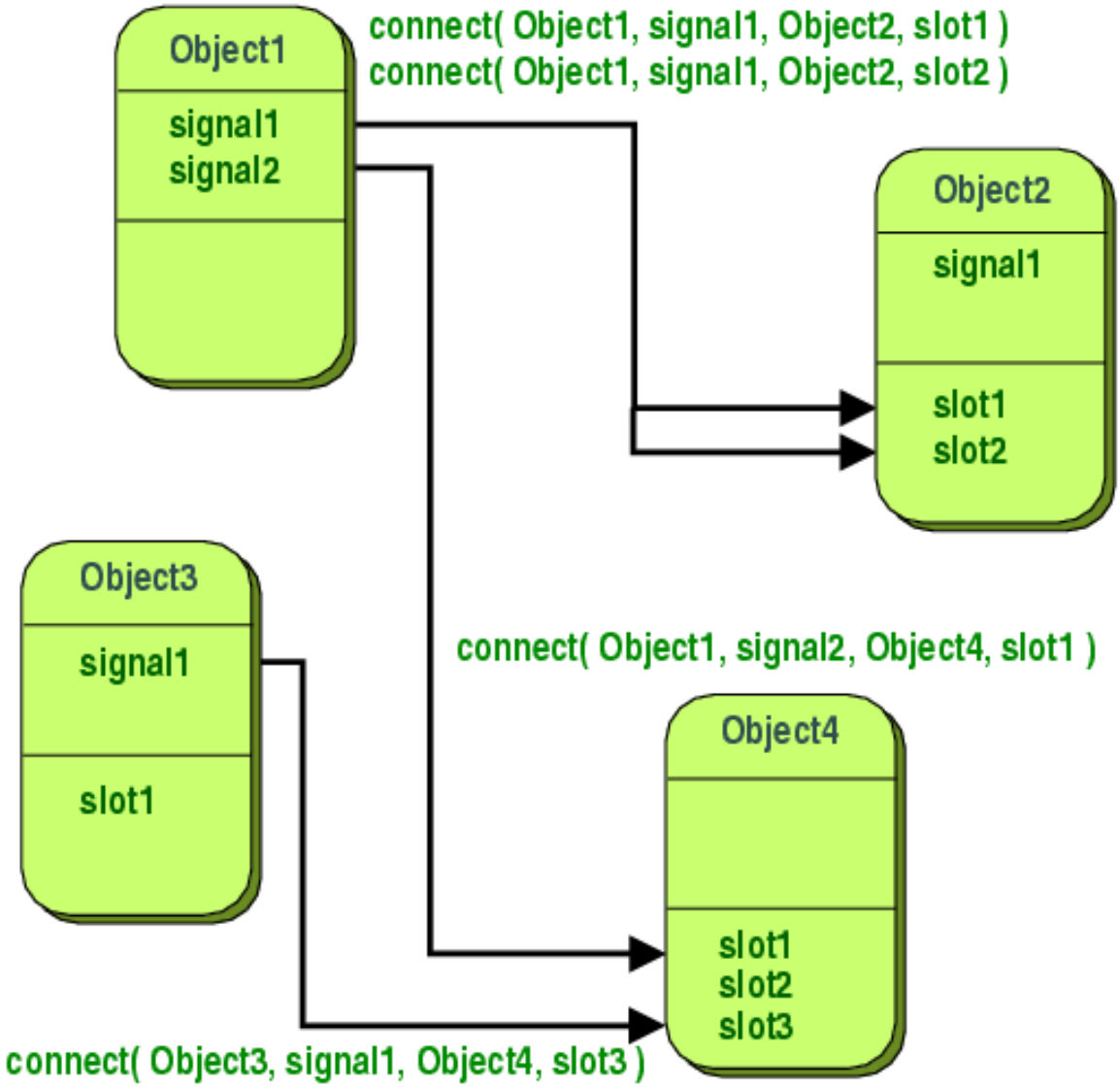

Abbildung 5.1: Signal-Slot Konzept von Qt

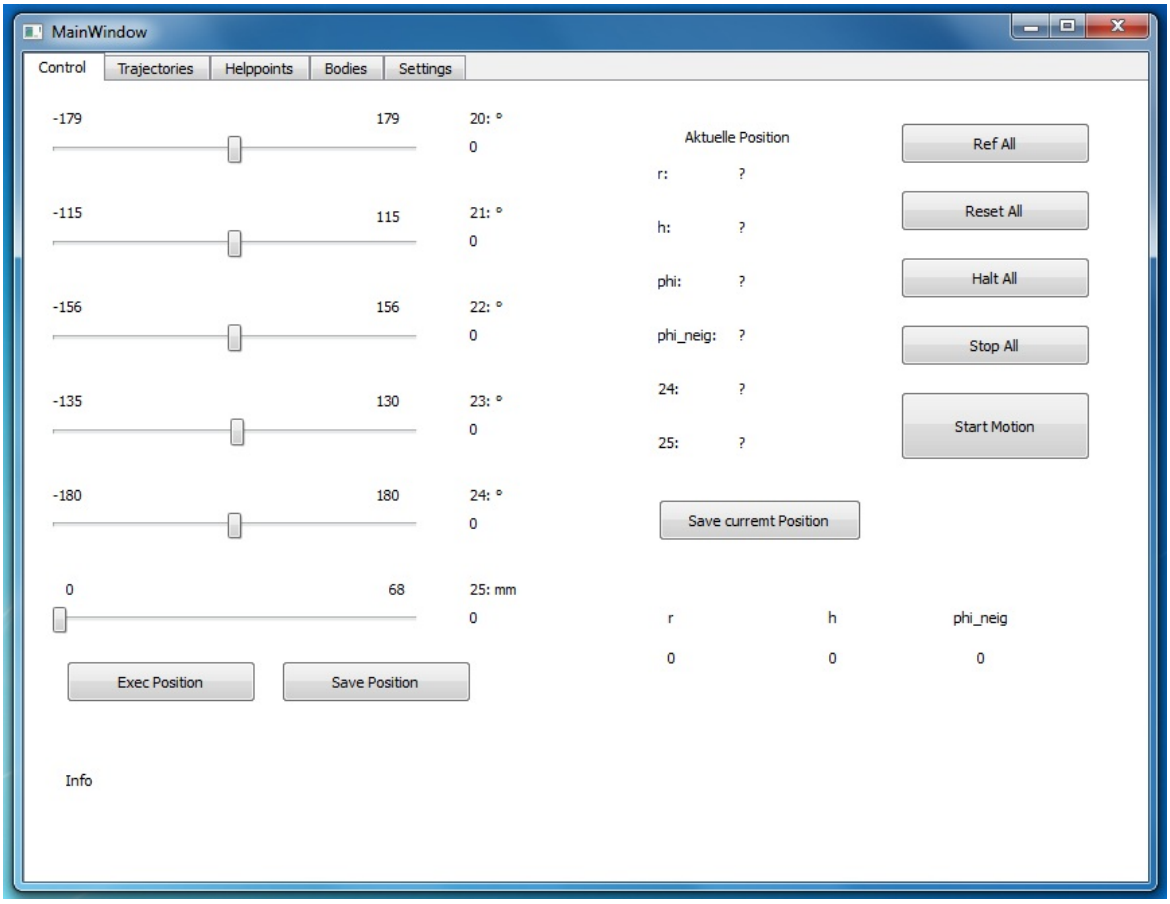

Abbildung 5.2: Tab Control

Tab Trajectories

Dieser Tab ist wohl der Wichtigste bei dem Erstellen von Trajektorien. Hier werden die benötigten Informationen zu einer Trajektorie in die Tabelle eingegeben. Auch die Funktion, um die Tabellen abzuspeichern bzw. zu laden wird zusätlich zu anderen Verwaltungsmöglichkeiten, wie Löschen einer einzelnen Trajektorie oder einer kompletten Tabelle, geboten. Der Button *Send Trajectory* sorgt für die Übermittlung eingegebener Informationen zu *Control Thread*. Auch der Pointer auf festegelgte Körper wird zuvor übermittelt, damit *Control Thread* diese Körper in die Berechnungen mit einbeziehen kann.

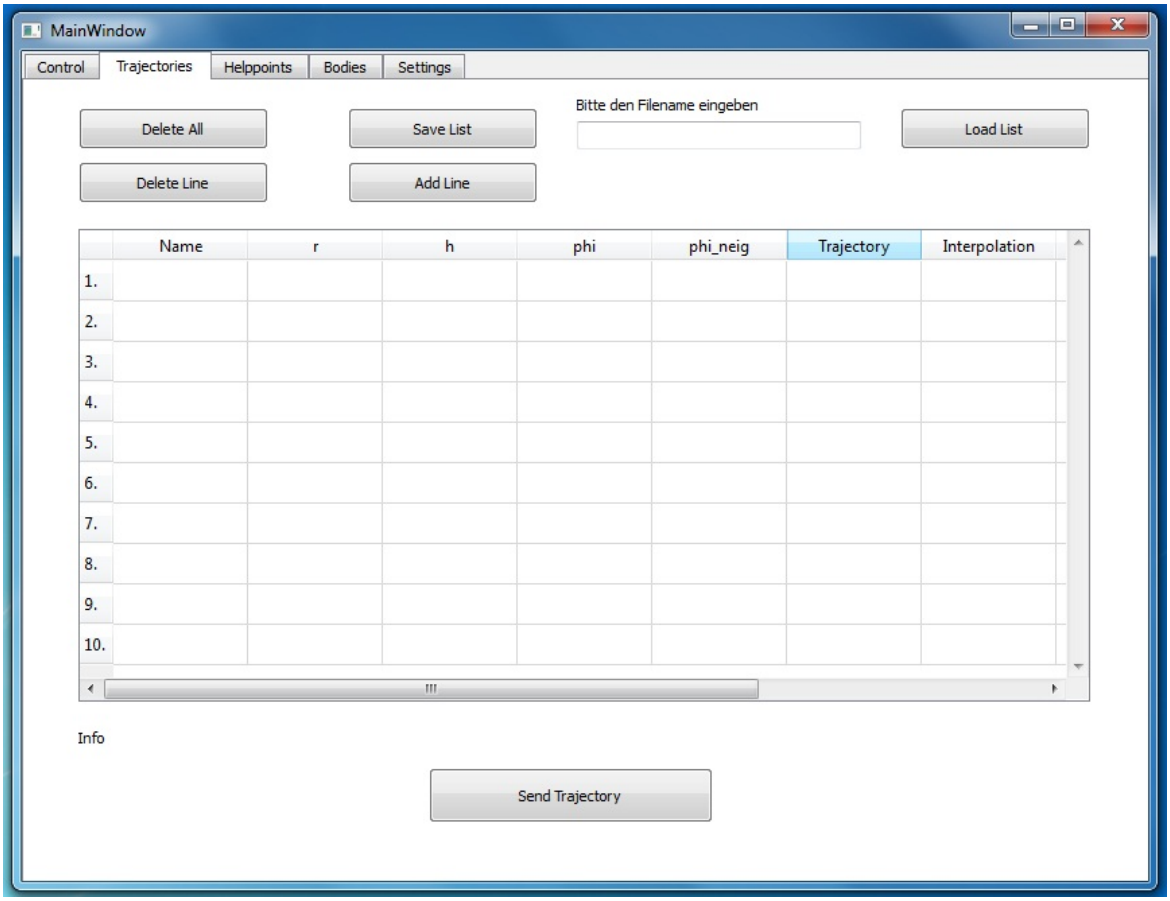

Abbildung 5.3: Tab Trajectories

Zu den Trajektorieninformationen außer den Punktkoordinaten und dem Neigungswinkel gehören auch solche Informationen, wie Trajektorienart, Anzahl benötigter Interpolationspunkte, Referenz auf gleichnamige Hilfspunkte, falls solche benötigt werden, Geschwindigkeitsangabe und Überschleifradius. Interpolation, Geschwindigkeit und Überschleifradius können

auch global in *Settings* eingestellt werden. Diese werden übernommen, falls zu der Trajektorie keine Angaben gemacht worden sind.

Tab Helppoints

Wenn eine Trajektorie bestimmte Hilfspunkte braucht, z.B. für einen Kreis oder eine Bézierkurve, werden diese hier eingetragen. Es muss darauf geachtet werden, dass die Namen der Hilfspunkte klar definiert und in der Trajektorien-Tabelle für die Benutzung jeweils eingefügt sind.

Tab Bodies

In diesem Tab werden die Ebenen und die Körper definiert. Diese beziehen sich auf die Weltkoordinaten. So ist es möglich Roboter selbst und andere Objekte in der Umgebung durch die Dreiecke zu modellieren. Bei der Angabe von den Dreiecken müssen die Punkte in richtiger Reihenfolge eingegeben werden, damit die Überprüfung korrekt arbeitet. Dazu wird die Rechte-Hand-Regel benutzt. Von dem ersten eingegebenen Punkt ausgehend, zeigt der Daumen auf den zweiten Punkt und der Zeigefinger auf den dritten. Der Mittelfinger stellt dann die Normale, die von dem konstruierten Körper ausgehen muss, dar.

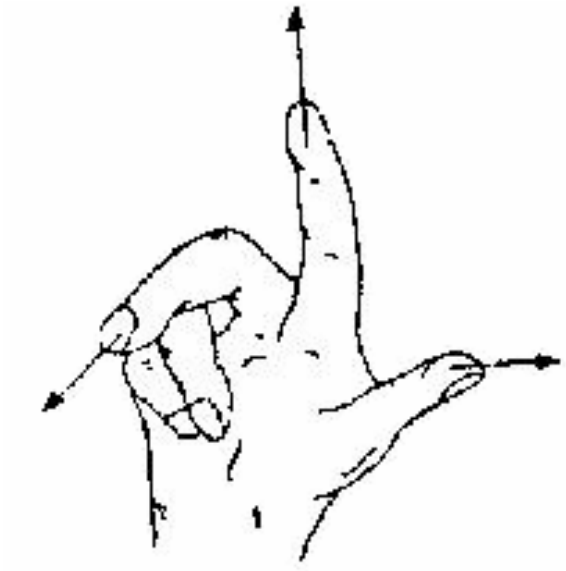

Abbildung 5.4: Rechte-Hand-Regel

Tab Settings

Wenn in einer Trajektorien-Tabelle die Werte wie Geschwindigkeit oder Überschleife nicht vorgegeben sind, kommen die globalen Einstellungen ins Spiel. Diese werden auf der linken Seite des Tabs eingestellt. Auch die Einstellungen, wie Überschleifgenauigkeit, Geschwindigkeitsreduzierungen und Abstände, werden auf der rechten Seite verwaltet. Bei Bedarf können andere hilfreiche Einstellungen einbezogen werden.

5.2 Control Thread

Der Control Thread stellt das Herzstück des Programms dar und ist auf zwei von einander unabhängige Teile zerstückelt. Der erste Teil beinhaltet die direkt auf den Roboterarm auswirkenden Grundfunktionen, wie Referenzieren, Anhalten und Zurücksetzen der Module sowie das Ausführen einer Bewegung durch die Eingabe direkter Modulparameter oder das Starten der Ausführung der durch die Trajekotienberechnung erzeugten Punkte. Dazu zählen die Methoden *refAllHandler*, *stopHandler*, *haltAllHandler*, *resetAllHandler*, *onemotionHandler* und *startHandler*.

Der zweite Teil beschäftigt sich mit der Übernahme der eingehenden Trajektorien und deren Bewertung, Interpolation und Prüfung der Ergebnisse nach den Kollisionen mit abschließender inverser Kinematik. Dabei werden die Punkte erst in eine temporäre Liste abgelegt und bei Fehlern verworfen, wobei eine entsprechende Fehlermeldung an die Benutzeroberfläche übermittelt wird. Bei der erfolgreichen Kollisionsprüfung und inverser Kinematik werden die Punkte in eine Warteschlange abgelegt, die aus der Benutzeroberfläche ausführbar ist.

Die Methode, mit der die Trajektorien berechnet werden, wird mit *addHandler* aufgerufen. In dieser Methode findet die Suche nach Ein- und Austrittspunkten für Überschleifen und die Interpolation statt. Auch werden die Methoden *uberschleifebezier* bzw. *uberschleifekreis* für die Überschleifinterpolation, *checkSaveareas* für Kollisionsprüfung und *newton_verfahren* für inverse Kinematik sowie *checkResults* zur Prüfung resultierender Gelenkewinkel aufgerufen.

Die Übergabe der zu berechnenden Trajektorien erfolgt einzeln. Dies ermöglicht es, auf die Fehlermeldungen rechtzeitig zu reagieren und die fehlerhaften Trajektorien zu korrigieren und zu ersetzen. Die konstruierten Körper können fortlaufend aktualiesiert werden. Die gleichzeitiege Übergabe von Trajektorien sollte vermieden werden, weil es zu Inkonsistenzen kommen kann.

Die Überprüfung der interpolierten Punkte auf die möglichen Kollisionen wird zu jedem berechneten Punkt durchgeführt und anschließend eine Koordinatentransformation vorgenommen. Dies berücksichtigt nicht die dynamischen Objekte und die durch die Bewegung des Roboters enstandenen Veränderungen der Koordinaten von den statischen Objekten. Nur die zu dem Roboterarm statisch verhaltenden Objekte können in so einem frühen Stadium geprüft werden.

5.3 Schnittstelle zu Roboterarm

Die Schnittstelle stellt Funktionen zur Verfügung, mit denen sich die Motoren des Roboterarms ansteuern und die Informationen auslesen lassen. Die Funktionen sind auf mehrere Gruppen je nach Art aufgeteilt.

- Status operations
- Module status callbacks
- Module operations
- Module movement operations

5.3.1 Status operations

initialize

Die Methode initialisiert die CAN-Bus-Kommunikationsschnittstelle und erkennt die verfügbaren Amtec Protokoll Module.

Die Parameter sind:

- pDevice - Der CAN-Bus-Gerät
- pDefVelPercent - Der Standardwert für die Zielgeschwindigkeit.
- pDefAccelPercent - Der Standardwert für das Zielbeschleunigung.
- pUpdateInterval - Der Zustands-Update-Intervall in [s].

Aus Sicherheitsgründen sind *pDefVelPercent* und *pDefAccelPercent* auf 0,5 (= 50 Prozent) begrenzt.

getModules

Diese Methode gibt die Liste der erkannten Module zurück. Über diese Liste kann auf die einzelnen Module zugegriffen und die Konfigurationen ausgelesen werden.

getModuleConfig

Diese Methode gibt die Modul-Konfiguration eines Moduls züruck. Eine Analogie zu *getModules* mit dem Unterschied, dass nur die Werte für ein Modul zurückgegeben werden.

Die Parameter sind:

- pModuleID - ID des Moduls
- pConfig - Pointer auf die Konfiguration, an welcher Stelle abgespeichert werden soll

5.3.2 Module status callbacks

addModuleStatusCallback

Diese Methode registriert eine Callback-Methode, die bei einer Statusänderung aufgerufen wird.

Die Parameter sind:

- pCallback - Der Callback-Handler
- pModuleID - ID des Moduls

Wenn *pModuleID* auf 0 gesetzt ist, wird die Callback-Methode bei der Statusänderung aller Module aufgerufen.

removeModuleStatusCallback

Diese Methode entfernt den in dem Parameter eingegebenen Callback-Handler aus der Liste.

Der Callback-Handler ist eine abstrakte Klasse, die eine Methode *moduleStatusUpdate* beinhaltet. Diese muss implementiert werden. Bei einem Aufruf des Callback-Handler werden die Informationen mitgegeben, welche für die Bearbeitung der Statusänderung ausschlaggebend sind.

- pModuleID - ID des Moduls, welches die Statusänderung hervorgerufen hat
- pActPos - Aktuelle Position dieses Moduls
- pActVel - Aktuelle Geschwindigkeit dieses Moduls
- pActCur - Aktuelle Beschleunigung dieses Moduls
- pStatus - Aktueller Status dieses Moduls

Der Callback-Handler soll anstelle der Methode *getModules* bzw. *getModuleConfig* benutzt werden. So wird Pooling auf Statusbits oder bestimmte Werte vermieden. Außerdem kann mit den Callback-Handlern eine komplexe Ansteuerung der Module und sogar ein Real-Time System realisiert werden.

5.3.3 Module operations

reset, ref, halt, softStop

Diese vier Methodenaufrufe benötigen jeweils eine ID des Moduls, das angesteuert werden soll. Wenn die ID = 0 ist, werden alle Module gemeinsam angesteuert. Die Methode *ref* (referenzieren) sollte nur für den Greifer benutzt werden, weil es sonst den bei Drehmodulen zu unerwarteten Bewegungen kommen kann. Bei der Methode *softStop* wird das Modul ausgebremst. Die Methode *halt* sorgt für das sofortige Anhalten unter der Benutzung von den Magnetbremsen. Für einer weitere Bewegung nach *halt* sollen erst die Statusbits mit der Methode *reset* zurückgesetzt werden.

setTargetVelocity, setTargetAcceleration

Diese beiden Methoden setzen die Soll-Werte für die Geschwindigkeit und die Beschleinigung für das eingegebene Modul. Diese Werte werden in dem Zusammenhang mit RAMP_MODE benutzt.

5.3.4 Module movement operations

execMotion

Die eigentliche Bewegung wird mit dieser Methode ausgeführt.

Die Parameter sind:

- pModuleID - ID des Moduls
- pType - Art der Bewegung
- pValue - Der Wert in den von der Art abhängenden Einheiten
- pTime - Zeit in ms, wird nur in STEP_MODE gebraucht

setSyncMotionMode

Wenn die Module gleichzeitig starten sollen, kann mit dieser Methode die synchronisierte Bewegung ein- oder ausgeschaltet werden.

execSyncMotion

Falls eine synchronisierte Bewegung gesetzt worden war, werden die Module durch die Methode *execMotion* nicht gleich losfahren, sondern auf die Methode *execSyncMotion* warten. Diese stößt alle ausstehenden Bewegungen gemeinsam an und hebt die Synchronisation auf.

Weitere Informationen zu der Schnittstelle können in der Dokumentation zu MLRobotic nachgeschaut werden. Leider ist die Dokumentation nicht vollständig und lässt einige Fragen offen.

6 Ausblick

6.1 Inerpolationsverfahren

Bei den Bézierkurven werden die Kontrollpunkte nicht angefahren. Wenn eine weiche Bahn benötigt wird, die auch durch die vorgegebenen Punkte verläuft, muss die Trajektorienbearbeitung um die Splineinterpolation erweitert werden. Zwar können die Bézierkurven verwendet werden, aber eine manuelle Suche nach den geeigneten Kontrollpunken kann viel Zeit in Anspruch nehmen.

Eine Möglichkeit ist die Verwendung der kubischen c^2-Spline, die zwei Stützstellen benötigt und nicht zu überschwingen neigt. Der Rechenaufwand bei der kubischen c^2-Spline ist enorm. Als Alternative können B-Splines dienen. Diese werden auch oft bei numerischen Verfahren eingesetzt.

Durch die Koordinatentransformationen von den Zylinderkoordinaten zu den kartesischen Koordinaten und umgekehrt entstehen die Informationsverluste. So geht bei der Umrechnung von den kartesichen Koordinaten zu den Zylinderkoordinaten der Winkel ϕ bei Radius $r = 0$ verloren. Um dies zu vermeiden, soll immer $r \neq 0$ sein, also muss eine minimale Abweichung eingebaut werden.

Bei einem Übergang durch die Position mit einem sehr kleinen Radius r kann es zu sehr schnellen Veränderungen des Winkels ϕ und dadurch zu großen nicht ausführbaren Geschwindigkeiten kommen. Bei der Ansteuerung der Module muss dies berücksichtigt werden.

6.2 Größere Überschleifen

Bei den im Verhältnis zu der Trajekotrienlänge großen Überschleifen und der Kreis-, bzw. Bézierkurveninterpolation können zu starke Richtungsänderungen entstehen, weil die Überschleifinterpolation auf den senkrechten Ein- und Austritt der Trajektorie ausgelegt ist. Um dies zu beheben kann anstatt der quadratischen die kubische Bézierkurve verwendet werden.

In den Ein- und Austrittspunkten können die Tangenten angelegt werden, auf denen die Kontrollpunkte ausgewählt werden sollten. Um die Kontrollpunkte auszuwählen, könnte der zweite Schnittpunkt jeder Tangente mit der Überschleifkugel genommen werden. Dieser Ansatz könnte hilfreich sein, um eine optimale Lösung zu finden.

6.3 Kollisionsprüfung

6.3.1 Grundkörper

Die Körper, deren Oberflächen sich schlecht durch die Dreiecke beschreiben lassen, können über die Grundkörper dargestellt werden. Besonders bei den Körpern mit runden Oberflächen wie Kugel oder Zylinder ist es von Vorteil solche Konstrukte bei der Körpererstellung einzubringen.

Für eine Kugel reicht es aus, den Mittelpunkt und den Radius zu kennen. Ein Zylinder und ein Kegel brauchen mehr Informationen, um klar definiert zu werden. Diese Körper können in einer einheitlichen Form dargestellt werden. Dafür wird der Normalenvektor und zwei Mittelpunkte benötigt. Die beiden Mittelpunkte definieren eindeutig zwei parallele Ebenen. Zu jedem Mittelpunkt gehört ein Radius, um konusförmige Körper darstellen zu können.

Mit dem aus den Dreiecken konstruierten Körper und solchen Grundkörpern können komplexe Abbildungen des Umfeldes dargestellt und in das Umweltmodell eingefügt werden.

6.3.2 Koordinatenänderungen bei Bewegung

Wenn die Roboter bewegt wird, verändern sich die Koordinaten der Körper in dem Weltkoordinatensystem des Roboterarms. Bei der Ansteuerung des Armes können die Trajektorien nur im Stillstand der Plattform mit den sich geänderten Körperkoordinaten berechnet werden und die Armbewegung kann ohne größeren Aufwand ausgeführt werden. Mit der gleichzeitigen Bewegung der Plattform und des Roboterarms muss die Trajektorienberechnung die plattformbedingten Änderungen berücksichtigen und umsetzen können.

Die Zeitsynchronisation beider Bewegungen kann durch ein Zeitraster erfolgen. Dabei werden die Interpolationspunkte nicht in bestimmten Raumabständen, sondern Zeitschritten berechnet. Dies ist bei den Bézierkurven nicht ohne Probleme lösbar. Auch die Trajektorie kann nicht komplett erzeugt werden und muss mit der Zeit nachberechnet werden.

Bei jedem Zeitschritt muss das Umweltmodell aktualisiert werden. Für die zeitgleiche Bewegungen der Plattform und des Roboterarms muss das Umweltmodell sogar für den nächsten Zeitschritt vorhanden sein. Wenn sich die Plattform im nächsten Zeitschritt bewegen soll, wird das Umweltmodell für diesen berechnet und der Roboterarm kann schon mit dem aktualisierten Umweltmodell arbeiten, um zum Schluss des Zeitschritts die richtige Position anzunehmen.

Zum anderen soll die Konsistenz des Umweltmodells gewährleistet werden.

6.4 Aufgabenorientierte Programmierung

Das Programm kann nach Bedarf erweitert werden. Das *User Interface* und der *Control Thread* können bearbeitet und verändert werden, solange die Schnittstelle zwischen diesen Komponenten erhalten bleibt. Auch die Schnittstelle selbst kann mit dem Signal-Slot Konzept von Qt angepasst werden.

Die Trajektorienplanung stützt sich auf das bewegungsorientierte Programmierverfahren, bei dem der Programmierer die Trajektorien vorgibt, bei den möglichen Kollisionen eingreift und die Änderungen vornimmt, um diese zu vermeiden. Eine Umstellung der Steuerung auf das aufgabenorientierte Programmierverfahren ermöglicht dem Programm die Erstellung von Trajektorien mit einer automatischen Kollisionsvermeidung und der Auswahl geeigneter Bahnkurven für optimale Bewegungsergebnisse. Ein umfangreiches Umweltmodell ist hierfür notwendig.

Die aufgabenorientierte Programmierung hift bei der Zusammenarbeit der Plattform mit dem Roboterarm. Dabei agieren die beiden Komponenten vereint, um eine gemeinsame Aufgabe zu bewältigen. Künftig können aufgabenorientierte Programmierverfahren auf vielfältige Weise eingesetzt werden.

Literaturverzeichnis

[Brünner] BRÜNNER, Arndt: *Kreis durch drei Punkte*. – URL http://www.arndt-bruenner.de/mathe/scripts/kreis3p.htm

[Dresselhaus 2006] DRESSELHAUS, Manfred: Sichere Steuerungstechnik für die Mensch-Roboter Kooperation REIS ROBOTICS (Veranst.), 2006

[Dubcova 2010] DUBCOVA, Julia: *Steuerung eines 5-DOF-Handhabungsroboters in Arbeitsraumkoordinaten*, Hochschule für Angewandte Wissenschaften Hamburg, Bachelorarbeit, 2010

[Kolbe 2011] KOLBE, Felix: *Inbetriebnahme und Programmierung der Scitos G5-Roboterplattform*. 2011. – Ausarbeitung

[Linnemann 2010] LINNEMANN: Robotertechnik Beuth Hochschule für Technik Berlin (Veranst.), 2010

[MetraLabs GmbH 2011a] MetraLabs GmbH (Veranst.): *MLRobotic Documentation*. 1.7.1. 2011

[MetraLabs GmbH 2011b] MetraLabs GmbH (Veranst.): *Software Guide - MLRobotic*. Version 1.19.0. 2011

[REIS ROBOTICS] REIS ROBOTICS (Veranst.): *Roboterstuerung*

[Ziegler] ZIEGLER, Wolfgang: Handhabungs- und Montagetechnik Fachhochschule Düsseldorf (Veranst.)

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 25. Mai 2011	
Ort, Datum	Unterschrift