DIESE ARBEIT WURDE VORGELEGT AM INSTITUT FÜR GETRIEBETECHNIK
MASCHINENDYNAMIK UND ROBOTIK

Berechnung und Bewertung diverser Druckpfade der Multidirektionalen additiven Fertigung

Arbeit vorgelegt von
Jens Temminghoff, Alexander Scheele, Janis Schmitt
am 9. September 2019

Prüfer: Univ.-Prof. Dr.-Ing. Dr. h. c. Burkhard Corves
Betreuer: M.Sc. Markus Schmitz

Eidestattliche Erklärung

Hiermit erklären wir, Alexander Scheele, Janis Schmitt und Jens Temminghoff, an Eides statt, dass wir die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt haben.

Aachen, den 9. September 2019

Vorwort

Diverse Entwicklungen im Fachbereich der additiven Fertigung gehen derzeit über den reinen schichtweisen Aufbau hinaus oder erweitern diesen durch Manipulatoren mit mehr als drei Freiheitsgraden. Für spezielle Anwendungen ist eine vollständige Manipulation der Druckplattform unter einem feststehenden Druckkopf notwendig. Für die Umsetzung dieser Anwendungen kann eine Vielzahl unterschiedlicher Manipulatoren zur Bewegung der Druckplattform verwendet werden.
Um zunächst einen allgemeinen Überblick über die Thematik zu verschaffen, beginnt die Arbeit mit einer Recherche zu den Verfahren der multidirektionalen additiven Fertigung (MAF) mit einem breiten Blick auf verschiedene Manipulatoren, zur Erstellung einer Übersicht mit verschiedenen Robotertypen. Außerdem werden die relevanten Roboterkinematiken recherchiert und miteinander verglichen.
Anschließend wird ein Parametersatz zur Beschreibung von Druckpfaden der MAF entwickelt und beschrieben. Die für die Parameter notwendigen mathematischen Grundlagen beziehungsweise Berechnungsmethoden werden dementsprechend erläutert.
Im Umfang der Projektarbeit wurden drei Skripte mit dem Programm MATLAB erstellt. Mit dem Hauptprogramm wird der entwickelte Parametersatz berechnet. Ein weiteres Programm erzeugt einen beispielhaften Druckpfad, dessen Parameter im Folgenden mit dem Hauptprogramm berechnet werden. Mit dem Trajektorienplaner wird letztlich auf die Zeitabhängigkeit der Druckpfade Rücksicht genommen.
Neben der Beschreibung der Programme werden auch Beispiele aufgezeigt, an denen man sehen kann wie die Programme von der Druckpfad Erstellung über die Parameterberechnung bis zur Trajektorie arbeiten.
Anhand dieser Ergebnisse können Vergleiche über die Notwendigkeit mehrerer Freiheitsgrade der Roboter und die Notwendigkeit einer multidirektionalen Fertigung gezogen werden.
Abschließend findet eine Analyse und Bewertung diverser Beispiel-Druckpfade anhand des entwickelten Parametersatzes statt.

Inhaltsverzeichnis

1 Literaturrecherche

Die multidirektionale additive Fertigung (MAF) ist ein Zweig der additiven Fertigung bei dem es um den schichtweisen Aufbau von Bauteilen geht. In diesem Fall geht es um die Erweiterung der bereits verbreiteten unidirektionalen additiven Fertigung mittels FDM-3D-Druckern wie sie in der Industrie, aber auch in privaten Haushalten, anzutreffen sind.
Die Erweiterung bezieht sich auf die Freiheitsgerade des Prozesses. Herkömmliche 3D-Drucker basieren oft auf einem 3 degrees of freedom (dof) Linearachsen- oder Deltasystem wodurch keine Orientierungsänderung der Nozzle möglich ist. Wobei dies in der MAF durch das Einführen von mehreren Orientierungsachsen möglich gemacht werden soll. Besonders hierbei ist, dass nicht der Extruder (Nozzle) bewegt/rotiert werden soll, sondern die Druckplattform. Idee ist es, diese an einen Roboter mit passender Kinematik zu befestigen und diesen so unter dem Extruder zu bewegen, dass das Bauteil aufgebaut oder durch Andrucken von Material ergänzt werden kann.
Manipulatoren werden bereits seit vielen Jahren in der Industrie eingesetzt und wurden für ihre jeweiligen Einsatzgebiete angepasst und optimiert. Aus diesem Grund sind bis zu diesem Zeitpunkt viele verschiedene Roboterkinematiken entwickelt worden, wobei hier auf die Bedeutsamsten eingegangen werden soll. Für diese Anwendung relevante Eigenschaften der Manipulatoren begrenzen sich auf die Gelenkwinkelbegrenzungen, die maximale Geschwindigkeit und Beschleunigung der Gelenke und der Bewegungsbereich der Roboter.
Da es auch um die Orientierung im Prozess geht, werden im Folgenden nur Manipulatoren mit fünf bis sechs Achsen betrachtet. Unterschieden werden hier zwischen mehreren Typen. Als erste Beispiele für die weit verbreiteten Roboter mit 6 rotatorischen Achsen (RRRRRR), wovon einer auch für spätere Simulationen benutzt wird, dient der Kuka kr6 900 und der sich durch ein etwas anderes kinematisches Layout auszeichnende UR-5 von Universal Robots.

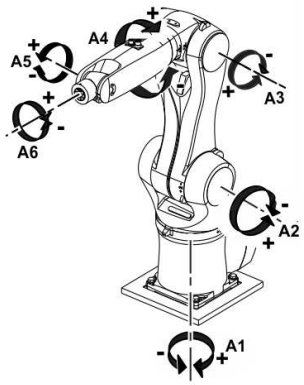

Abb. 1.1: Achsen KUKA[1]

Die 6 Achsen ermöglichen vollkommen freie Positionierung und Orientierung im Raum wie es in der MAF benötigt wird. Eine ähnliche Konfiguration, allerdings als Roboter mit nur 5 rotatorischen Achsen (RRRRR), stellt der LR Mate 200iD/4SH von Fanuc dar. Dieser Roboter ist somit bei der räumlichen Orientierung seines Endeffektors beschränkt.

Tab. 1.1: UR3e von Universal Robots[2]

Achse	Arbeitsreichweite	Maximale Geschwindigkeit
A1	$\pm 360°$	$\pm 180°/Sek.$
A2	$\pm 360°$	$\pm 180°/Sek.$
A3	$\pm 360°$	$\pm 180°/Sek.$
A4	$\pm 360°$	$\pm 180°/Sek.$
A5	$\pm 360°$	$\pm 180°/Sek.$
A6	∞	$\pm 180°/Sek.$

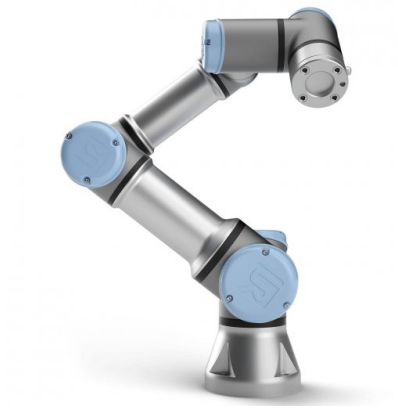

Abb. 1.2: UR3 kinematisches Layout[2]

Tab. 1.2: Kuka KR 6 R900 sixx[1]

Achse	Arbeitsreichweite	Geschwindigkeit bei Nenntraglast
A 1	$\pm 170°$	$360°/Sek.$
A 2	$+45°$ bis $-190°$	$300°/Sek.$
A 3	$+156°$ bis $-120°$	$360°/Sek.$
A 4	$\pm 185°$	$381°/Sek.$
A 5	$\pm 120°$	$388°/Sek.$
A 6	$\pm 350°$	$615°/Sek.$

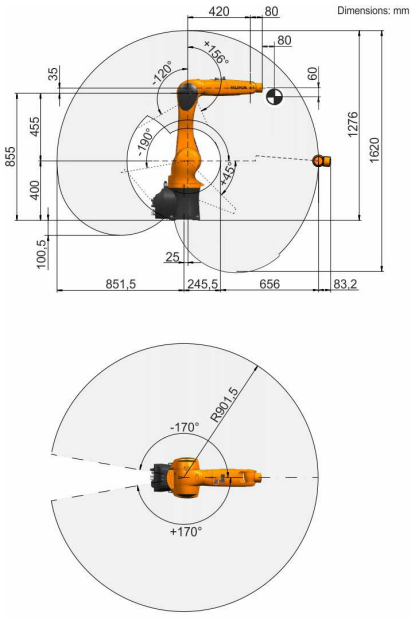

Abb. 1.3: KUKA Arbeitsreichweite[1]

Des Weiteren existieren Robotertypen, die nicht ausschließlich Manipulatoren mit rotatorischen Achsen sondern sich anderen Konfigurationen bedienen. Auf diese wird im Weiteren eingegangen.
Besonderes Augenmerk fällt hier auf die Verbindung von 3 Linearachsen und 3 rotatorischen Achsen für die Orientierung und der Delta-Konfiguration. Für die (LLLRRR) wird ein Manipulator angeführt, wie er im Heidelberger Druck verwendet wird. Für das Deltasystem wird im Folgenden der Fanuc M-3iA/6A als Beispiel betrachtet.

Tab. 1.3: Fanuc Robot M-3iA[3]

Achse	Arbeitsreichweite	Geschwindigkeit bei Nenntraglast
A 1 / A 2 / A 3	Durchmesser 1350 mm, Höhe 500 mm	
A 4	720°	$2000°/Sek.$
A 5	300°	$2000°/Sek.$
A 6	720°	$2000°/Sek.$

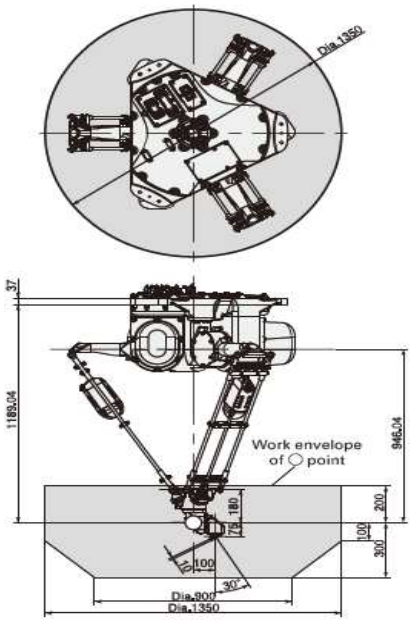

Abb. 1.4: Fanuc Robot M-3iA[3]

2 Mathematische Grundlagen

2.1 Orientierung

Zur Beschreibung von Koordinatensystemen und Transformationsmatrizen ist die richtige Beschreibung der Orientierung des betrachteten Systems von grundlegender Wichtigkeit. Die hier aufgeführten Methoden sind die Repräsentation der Orientierung als Matrix $\mathbf{R} \in \mathbb{R}^{3\times3}$ nach Euler sowie die Repräsentation der Orientierung mittels Einheitsquaternionen, die vor allem, aufgrund ihrer Effizienz, bei der Trajektorienplanung und der Berechnung der inversen Kinematik benutzt werden.

2.1.1 Rotationsmatrix nach Euler und Eulerwinkel

Eine Rotationsmatrix $\mathbf{R}$ ist eine reelle, orthogonale Matrix mit der $|\mathbf{R}|$ =1. Sie wird benötigt, um Vektoren oder ein gesamtes Koordinatensystem bei feststehendem Koordinatenursprung im euklidischen Raum um einen Winkel σ zu drehen. Die fett geschriebene Notation $\mathbf{R}$ weist auf einen Tensor höherer Ordnung hin, wie beispielsweise einen Vektor oder eine Matrix wohingegen eine normale Notation auf eine skalare Größe hinweist.

2.1.1.1 Elementare Rotationsmatrizen

Die folgenden Matrizen beschreiben jeweils eine elementare Rotation um eine der feststehenden Achsen. Multipliziert man beispielsweise $\mathbf{R}_z(\gamma)$ mit der Matrix eines Koordinatensystems oder einem Vektor, so drehen sie sich mit dem Winkel γ um die Z-Achse wie in Abbildung (2.1) veranschaulicht.

- Drehung um die X-Achse

$$\mathbf{R}_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & cos(\alpha) & -sin(\alpha) \\ 0 & sin(\alpha) & cos(\alpha) \end{pmatrix} \tag{2.1}$$

- Drehung um die Y-Achse

$$\mathbf{R}_y(\beta) = \begin{pmatrix} cos(\beta) & 0 & sin(\beta) \\ 0 & 1 & 0 \\ -sin(\beta) & 0 & cos(\beta) \end{pmatrix} \tag{2.2}$$

- Drehung um die Z-Achse

$$\mathbf{R}_z(\gamma) = \begin{pmatrix} cos(\gamma) & -sin(\gamma) & 0 \\ sin(\gamma) & cos(\gamma) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.3}$$

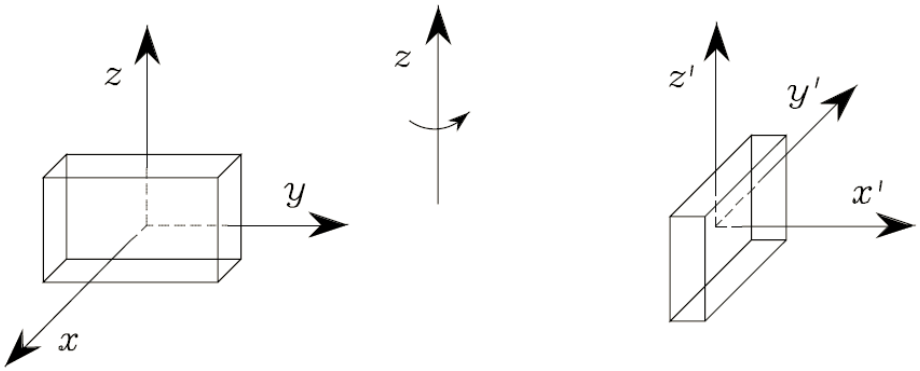

Abb. 2.1: 90° Drehung um die Z-Achse.[4]

2.1.1.2 Zusammensetzung einer Rotation (ZY'X")

Es gibt verschiedene Konventionen bei der Zusammensetzung von Rotationsmatrizen. Hier wurde die ZY'X"-Konvention, auch Roll-Nick-Gier Konvention genannt, verwendet. Die Eulerschen Winkel können eine beliebige Orientierung eines Koordinatensystems im euklidischen Raum beschreiben, bei der es sich nicht zwingend um eine elementare Drehung handeln muss. Zur Erzeugung einer Drehung um die Eulerschen Winkel $\mathbf{\Phi} = (\varphi\ \vartheta\ \psi)$ werden die folgenden Rotationsmatrizen benötigt.

- Drehung um die Z-Achse

$$\mathbf{R}_z(\varphi) = \begin{pmatrix} cos(\varphi) & -sin(\varphi) & 0 \\ sin(\varphi) & cos(\varphi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.4}$$

- Drehung um die Y-Achse

$$\mathbf{R}_y(\vartheta) = \begin{pmatrix} cos(\vartheta) & 0 & sin(\vartheta) \\ 0 & 1 & 0 \\ -sin(\vartheta) & 0 & cos(\vartheta) \end{pmatrix} \tag{2.5}$$

- Drehung um die X-Achse

$$\mathbf{R}_x(\psi) = \begin{pmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2.6}$$

Die Matrix der zusammengesetzten Drehung erhält man durch Matrixmultiplikation aus den Matrizen der elementaren Drehungen. Das Ergebnis der Drehung ist abhängig von der Reihenfolge der Multiplikation. Im Fall der ZY'X"-Rotation wird zuerst um die Z-Achse gedreht. Anschließend um die durch die Verschiebung neu entstandene Y'-Achse und letztendlich um die ebenfalls neu erzeugte X"-Achse.

$$\mathbf{R}_{zy'x''} = \mathbf{R}_z(\varphi)\mathbf{R}_{y'}(\vartheta)\mathbf{R}_{x''}(\psi) = \begin{pmatrix} c_\varphi c_\vartheta & c_\varphi s_\vartheta s_\psi - s_\varphi c_\psi & c_\varphi s_\vartheta c_\psi + s_\varphi s_\psi \\ s_\varphi c_\vartheta & s_\varphi s_\vartheta s_\psi + c_\varphi c_\psi & s_\varphi s_\vartheta c_\psi - c_\varphi s_\psi \\ -s_\vartheta & c_\vartheta s_\psi & c_\vartheta c_\psi \end{pmatrix} \tag{2.7}$$

Um aus einer Rotationsmatrix

$$\mathbf{R} = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} \tag{2.8}$$

die Eulerschen Winkel zu bestimmen werden die Gleichungen aus (2.9) und (2.10) benötigt. Hierbei muss jedoch aufgrund der Uneindeutigkeit der Eulerwinkel eine Fallunterscheidung vorgenommen werden.

Lösung I $\quad \vartheta \, \epsilon \, (-\frac{\pi}{2}, \frac{\pi}{2})$

$$\begin{aligned} \varphi &= Atan2(r_{21}, r_{11}) \\ \vartheta &= Atan2(-r_{31}, \sqrt{r_{32}^2 + r_{33}^2}) \\ \psi &= Atan2(r_{32}, r_{33}) \end{aligned} \tag{2.9}$$

Lösung II $\quad \vartheta \, \epsilon \, (-\frac{\pi}{2}, \frac{3\pi}{2})$

$$\begin{aligned} \varphi &= Atan2(-r_{21}, -r_{11}) \\ \vartheta &= Atan2(-r_{31}, -\sqrt{r_{32}^2 + r_{33}^2}) \\ \psi &= Atan2(-r_{32}, -r_{33}) \end{aligned} \tag{2.10}$$

Dieser Algorithmus unterscheidet sich je nach benutzter Konvention. Neben der Uneindeutigkeit leidet Orientierungsrepräsentation nach Euler auch unter dem Begriff *Gimballock*, der besagt, dass die Repräsentation einen Freiheitsgrad bei $\vartheta = 0$ oder $\vartheta = \pi$ verliert, wodurch nicht jede Orientierung im Raum wiedergegeben werden kann.

2.1.2 Quaternionen

Als performante Alternative zu den Rotationmatrizen nach Euler bieten sich die Quaternionen an. Besonders ist die Berechnung effizienter, da die kompaktere Form mit den vier Elementen $\eta\ \epsilon_x\ \epsilon_y\ \epsilon_z$ im Vergleich zu den neun Elementen einer Rotationsmatrix weniger Rechenoperationen benötigt. Die vier Elemente bestehen aus dem reellen Skalarteil des Quaternions η und dem imaginären Vektorteil $\boldsymbol{\epsilon} = [\epsilon_x\ \epsilon_y\ \epsilon_z]^T$.

Für die Beschreibung der Orientierung werden Einheitsquaternionen benutzt mit der Eigenschaft:

$$\eta^2 + \epsilon_x^2 + \epsilon_y^2 + \epsilon_z^2 = 1 \tag{2.11}$$

Quaternionen haben keinen Gimballock und sind immer eindeutig, was vorteilhaft gegenüber den Eulerwinkeln ist. Da die Elemente der Quaternionen allerdings nicht intuitiv sind, kann zur besseren Beschreibung eines Koordinatensystems aus einem Quaternion mit Formel (2.12) eine Rotationsmatrix berechnet werden,

$$\mathbf{R}(\eta, \boldsymbol{\epsilon}) = \begin{pmatrix} 2(\eta^2 + \epsilon_x^2) - 1 & 2(\epsilon_x\epsilon_y - \eta\epsilon_z) & 2(\epsilon_x\epsilon_z - \eta\epsilon_y) \\ 2(\epsilon_x\epsilon_y + \eta\epsilon_z) & 2(\eta^2 + \epsilon_y^2) - 1 & 2(\epsilon_y\epsilon_z - \eta\epsilon_x) \\ 2(\epsilon_x\epsilon_z + \eta\epsilon_y) & 2(\epsilon_y\epsilon_z + \eta\epsilon_x) & 2(\eta^2 + \epsilon_z^2) - 1 \end{pmatrix} \tag{2.12}$$

die äquivalent zu den Rotationsmatrizen nach Euler benutzt werden kann. Für die inverse Orientierung kann aus jeder Rotationsmatrix

$$\mathbf{R} = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} \tag{2.13}$$

mit dem folgenden Algorithmus das dazugehörige Einheitsquaternion $Q = (\eta\ \boldsymbol{\epsilon})$ berechnet werden.

$$\eta = \frac{1}{4}\sqrt{(r_{11} + r_{22} + r_{33} + 1)^2 + (r_{32} - r_{23})^2 + (r_{13} - r_{31})^2 + (r_{21} - r_{12})^2} \tag{2.14}$$

$$\boldsymbol{\epsilon} = \frac{1}{4}\begin{pmatrix} \sqrt{(r_{11} - r_{22} - r_{33} + 1)^2 + (r_{32} - r_{23})^2 + (r_{21} + r_{12})^2 + (r_{31} - r_{13})^2} \\ \sqrt{(r_{22} - r_{11} - r_{33} + 1)^2 + (r_{32} + r_{23})^2 + (r_{21} + r_{12})^2 + (r_{13} - r_{31})^2} \\ \sqrt{(r_{33} - r_{11} - r_{22} + 1)^2 + (r_{32} + r_{23})^2 + (r_{21} - r_{12})^2 + (r_{31} + r_{13})^2} \end{pmatrix} \tag{2.15}$$

Da die Vorzeichen der Einträge von $\boldsymbol{\epsilon}$ noch nicht definiert sind, können diese mittels der Signum-Funktion

$$sgn(x) = \begin{cases} 1, & x \in (0, \infty) \\ 0, & x = 0 \\ -1, & x \in (-\infty, 0) \end{cases} \tag{2.16}$$

ermittelt werden. So ergeben sich die Vorzeichen von ϵ_x, ϵ_y und ϵ_z zu $sgn(r_{32} - r_{23})$, $sgn(r_{13} - r_{31})$ und $sgn(r_{21} - r_{12})$.
Das Quaternion aus $\mathbf{R}^{-1} = \mathbf{R}^T$ wird Q^{-1} genannt, wobei

$$Q^{-1} = \{\eta, -\boldsymbol{\epsilon}\} \tag{2.17}$$

Wie bei den Rotationsmatrizen können aufeinander folgende Rotationen durch Multiplikation miteinander verkettet werden. Dafür benötigt man die Quaternionenmultiplikation wie in (2.18).

$$Q_1 * Q_2 = \{\eta_1\eta_2 - \boldsymbol{\epsilon}_1^T\boldsymbol{\epsilon}_2, \eta_1\boldsymbol{\epsilon}_2 + \eta_2\boldsymbol{\epsilon}_1 + \boldsymbol{\epsilon}_1 \times \boldsymbol{\epsilon}_2\} \tag{2.18}$$

Zusammenfassend sind Quaternionen eine elegante Methode zur Beschreibung der räumlichen Orientierung, welche die Nachteile der Orientierungsrepräsentation nach Euler umgehen.

2.2 Koordinatentransformation

Da die Orientierung keinen Einfluss auf den Ort des Koordinatenursprungs hat, werden die Rotationsmatrizen **R** durch einen Ortsvektor

$$\mathbf{p} = \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} \tag{2.19}$$

erweitert, woraus

$$\mathbf{T} = \begin{pmatrix} \mathbf{R} & \mathbf{p} \\ \mathbf{0}^T & 1 \end{pmatrix} = \begin{pmatrix} r_{11} & r_{12} & r_{13} & p_x \\ r_{21} & r_{22} & r_{23} & p_y \\ r_{31} & r_{32} & r_{33} & p_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.20}$$

folgt.
T wird in den folgenden Kapiteln als *Transformationsmatrix*, *Pose* oder *Frame* bezeichnet.
Da eine standardmäßige Berechnung der Matrixinverse von **T** keinen Sinn als Transformation im mehrdimensionalen Raum birgt, muss die Inverse einer Transformationsmatrix als

$$\mathbf{T}^{-1} = \begin{pmatrix} \mathbf{R} & \mathbf{p} \\ \mathbf{0}^T & 1 \end{pmatrix}^{-1} = \begin{pmatrix} \mathbf{R}^T & -\mathbf{R}^T\mathbf{p} \\ \mathbf{0}^T & 1 \end{pmatrix} \tag{2.21}$$

berechnet werden.
Es gilt $\mathbf{T}\mathbf{T}^{-1} = \mathbf{I}^{4\times4}$.
Diese Eigenschaft wird in späteren Kapiteln nützlich sein, um nötige Transformationen rückgängig zu machen, oder um eine Trajektorie auf eine, für einen Roboter abfahrbare, Trajektorie herunterzubrechen.

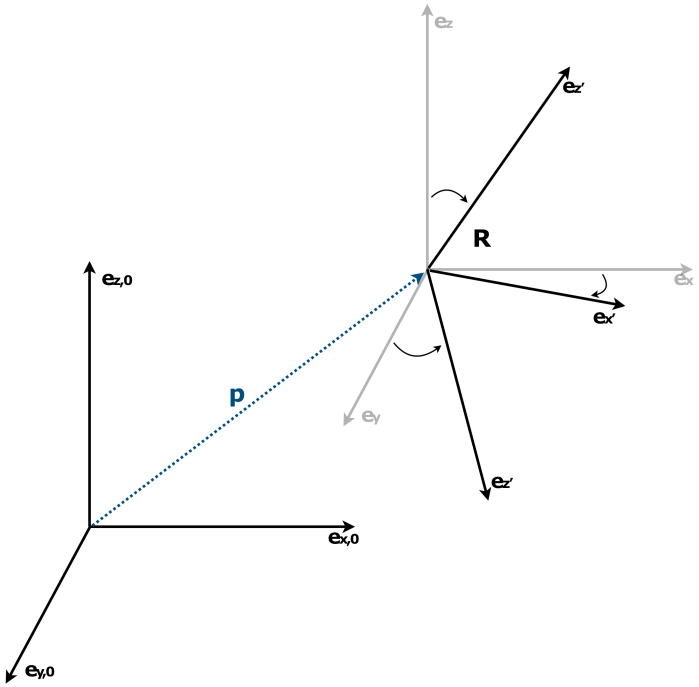

Abb. 2.2: Koordinatentransformation mit Translation **p** und Rotation **R**

2.3 Parameterkurven

Zur Beschreibung von Kurven im dreidimensionalen Raum können mathematische Parameterkurven verwendet werden, die nicht wie üblich von zwei, sondern nur noch von einer Variable abhängig sind. Diese Variable steigt monoton über die Länge der Kurve. Eine parametrisierte Kurve ist dann wie folgt aufgebaut:

$$\mathbf{p}(t) = \begin{pmatrix} x(t) \\ y(t) \\ z(t) \end{pmatrix} \tag{2.22}$$

Die Funktionen $x(t)$, $y(t)$ und $z(t)$ sind von der Laufkoordinate t abhängig und können als kartesische Koordinaten der überliegenden Kurve benutzt werden.
Die Tangente eines Punktes auf der Parameterkurve ergibt

$$\mathbf{t} = \frac{\dot{\mathbf{r}}}{|\dot{\mathbf{r}}|} \tag{2.23}$$

Die Normale eines Punktes auf der Parameterkurve ergibt

$$\mathbf{n} = \frac{(\dot{\mathbf{r}} \times \ddot{\mathbf{r}}) \times \dot{\mathbf{r}}}{|(\dot{\mathbf{r}} \times \ddot{\mathbf{r}}) \times \dot{\mathbf{r}}|} = \frac{\ddot{\mathbf{r}}}{|\ddot{\mathbf{r}}|} \tag{2.24}$$

Die Binormale eines Punktes auf der Parameterkurve ergibt

$$\mathbf{b} = \frac{\dot{\mathbf{r}} \times \ddot{\mathbf{r}}}{|\dot{\mathbf{r}} \times \ddot{\mathbf{r}}|} = \mathbf{t} \times \mathbf{n} \tag{2.25}$$

Hieraus ergibt sich das von der Kurve $\mathbf{r}(t)$ abhängige, begleitende Koordinatensystem

$$\mathbf{R}(\mathbf{r}(t)) = (\mathbf{t}\ \mathbf{b}\ \mathbf{n}) \tag{2.26}$$

Zur Berechnung der Krümmung der Kurve wird die folgende Gleichung benötigt

$$\kappa = \frac{|\dot{\mathbf{r}} \times \ddot{\mathbf{r}}|}{|\dot{\mathbf{r}}|^3} \tag{2.27}$$

Als Beispiel wird die betrachtete, eine Helix darstellende, Parameterkurve

$$\mathbf{k}(t) = \begin{pmatrix} cos(\omega t) \\ sin(\omega t) \\ h(t) \end{pmatrix} \tag{2.28}$$

in Abbildung (2.3), mit dem konstruierten Koordinatensystem $\mathbf{R}(\mathbf{k}(t_{betrachtet})$ angeführt.

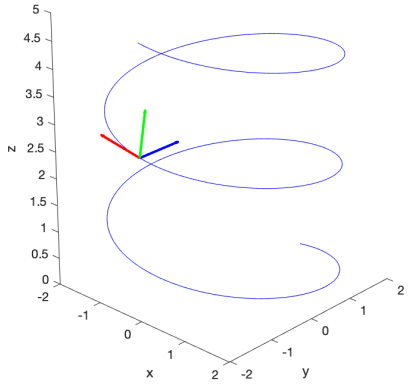

Abb. 2.3: Parameterkurve mit begleitendem Koordinatensystem zum Zeitpunkt $t_{betrachtet}$.

2.4 Schnittpunkt von Ebene und Gerade

Eine durch den Punkt $\mathbf{p} = (p_x\ p_y\ p_z)^T$ und dem auf dem Punkt $\mathbf{p}$ stehenden Vektor $\mathbf{n} = (n_x\ n_y\ n_z)$ durch die Gleichung

$$\mathbf{E} : \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} \left[\mathbf{x} - \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} \right] = 0 \tag{2.29}$$

aufgespannte Ebene **E**,wie man in Abbildung (2.4) sieht, kann mit einer Geraden

$$\mathbf{g} : \mathbf{x}_g = \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} + \lambda \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} \tag{2.30}$$

geschnitten werden. Setzt man die Geradengleichung $\mathbf{x}_g$ in die Ebenengleichung **E** ein, so erhält man das lineare Gleichungssystem

$$\begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} \left[\mathbf{x}_g - \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} \right] = 0 \tag{2.31}$$

Löst man dieses Gleichungssystem nach Variable λ auf, so erhält man nach erneutem Einsetzen in die Geradengleichung den Schnittpunkt $\mathbf{p}_{schnitt}$.

$$\mathbf{p}_{schnitt} = \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} + \lambda_{ermittelt} \begin{pmatrix} n_x \\ n_y \\ n_z \end{pmatrix} \tag{2.32}$$

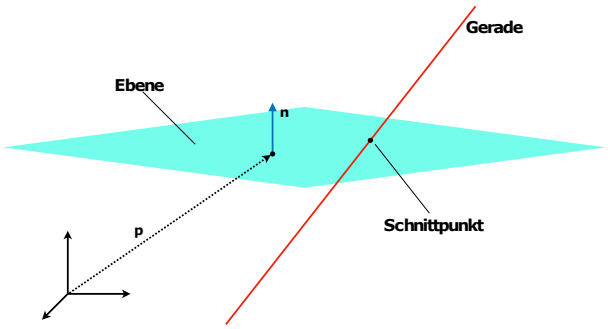

Abb. 2.4: Schnittpunkt der Ebene **E** und Gerade **g** .

3 Methodik

3.1 Beispielpfaderzeugung

Da die Analyse von gesamten Pfaden in der Fertigung komplex und undurchsichtig werden kann, werden zur besseren Darstellung und Verständlichkeit einige Beispielpfade erzeugt. Diese Beispiele sind verschiedene, auf Oberflächen projizierte, Parameterkurven und die normal auf der Oberfläche stehenden Vektoren $\mathbf{n}_i$ in den jeweiligen Punkten des Pfades $\mathbf{p}_i$. Die betrachteten Oberflächen werden, wie auch die in den Grundlagen beschriebenen Kurven, mit

$$\mathbf{O}(u,v) = \begin{pmatrix} x(u,v) \\ y(u,v) \\ z(u,v) \end{pmatrix} \tag{3.1}$$

parametrisiert, wobei die Oberfläche $\mathbf{O}$ von zwei Variablen abhängig ist, welche bei der Projektion von einer 2D Parameterkurve $\mathbf{r}(t) = (x(t)\ y(t))^T$ ersetzt werden,

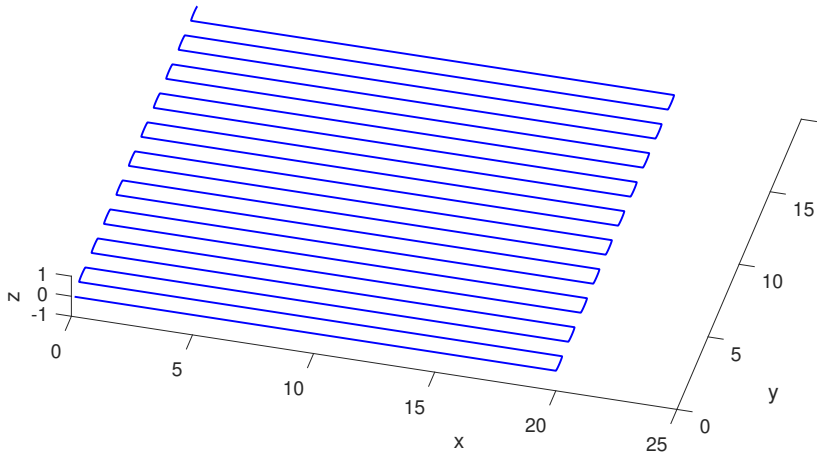

Abb. 3.1: 2D Parameterkurve $\mathbf{r}(t) = (x(t)\ y(t))^T$

wodurch der Pfad auf der Oberfläche als

$$\mathbf{O}_{Pfad}(\mathbf{r}(t)) = \begin{pmatrix} x(\mathbf{r}(t)) \\ y(\mathbf{r}(t)) \\ z(\mathbf{r}(t)) \end{pmatrix} \tag{3.2}$$

folgt.

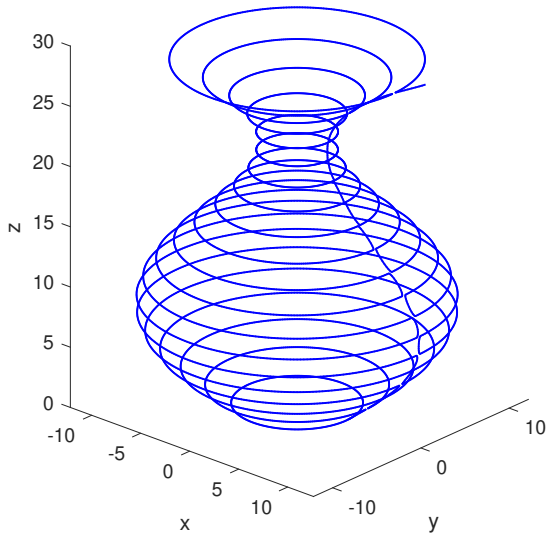

Abb. 3.2: Auf Oberfläche projizierter Pfad $\mathbf{O}_{Pfad}(\mathbf{r}(t))$

Die Normalen $\mathbf{n}_i$ werden mittels der partiellen Ableitungen nach u und v berechnet. Der gesuchte Vektor ergibt sich dann mittels des Kreuzprodukts zu

$$\mathbf{n}_i = \frac{\partial \mathbf{O}(u_i, v_i)}{\partial u} \times \frac{\partial \mathbf{O}(u_i, v_i)}{\partial v} \tag{3.3}$$

3.2 Vorgehen zur Pfadanalyse

Zur Analyse der gegebenen Pfade, die beim Input durch eine Auflistung des Vektors $\mathbf{v}_{Pfad,i} = (p_{x,i}, p_{y,i}, p_{z,i}, n_{x,i}, n_{y,i}, n_{z,i})^T$ mit $i = 1, 2, \ldots, n$ beschrieben werden, stellt sich die Frage zu welchen relevanten Eigenschaften der Pfad analysiert werden soll.

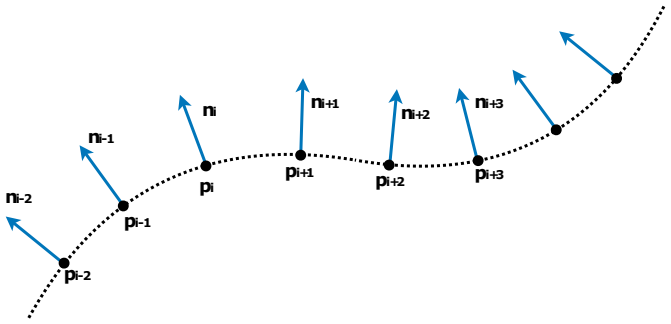

Abb. 3.3: Veranschaulichung der Pfadpunkte $\mathbf{p}_i$ und den darauf stehenden Normalenvektoren $\mathbf{n}_i$

Da es in der Robotik oft darauf ankommt, innerhalb des eingesetzten Manipulator bestimmten Einschränkungen zu arbeiten, wie Joint- oder Geschwindigkeitlimits, kann ein nicht abfahrbarer Pfad oder eine mit der Zeitkomponente versehene Trajektorie zur Selbstkollision, Kollision mit dem Bauteil, schlechtem dynamischen Verhalten, Präzisionsproblemen, oder zu Vibrationen beim Abfahren des Pfades führen. Angenommen der Manipulator führt einen Druckauftrag durch, dessen Pfad im Optimalfall mit einer konstanten Geschwindigkeit durchfahren werden soll, gibt es vor allem bei plötzlichen Richtungs- und Orientierungswechseln Probleme aufgrund der unstetigen TCP-Geschwindigkeit. Beim Umfahren einer scharfen Ecke kann es passieren, dass die Orientierung des Endeffektors plötzlich um einen sehr großen Betrag springt, was physikalisch nicht ausführbar ist. Einige dieser Probleme müssen direkt bei der Pfad-/Trajektorienplanung beachtet werden und können im Nachhinein nicht durch geeignete Interpolation gelöst werden. Problematisch sind auch zu große Beschleunigungen, zum Beispiel beim Fahren durch Abschnitte eines Pfades mit zu großen Krümmungen. Zusammenfassend ist zunächst die Orientierung des Koordinatensystems des Pfades, sowie eine Untersuchung der Beschleunigung im Programm realisiert worden. Zunächst wird mit Hilfe der Normalenvektoren $\mathbf{n}_i = (n_{x,i}\ n_{y,i}\ n_{z,i})^T$ auf den Pfadpunkten ein rechtshändiges Koordinatensystem konstruiert. Die folgende Transformationsmatrix, die man durch Verkupplung der Rotatiationsmatrix $\mathbf{R}$ und dem Ortsvektor $\mathbf{p}$ erhält, ist später für die Trajektorienerzeugung und die Berechnung der inversen Kinematik wichtig. Die Erweiterung der Beschreibung der Orientierung des Normalenvektors zu einer vollständigen Rotationsmatrix gewährleistet später vollkommene Orientierungstreue, die

für spezielle Aufgaben des Roboters wichtig sein können. Andererseits kann die Orientierung um den Normalenvektor auch unwichtig sein und offen gelassen werden. In einem additiven Druckvorgang mit einem Extruder, spielt es keine Rolle wie die Rotation um den Normalenvektor auf der Oberfläche des Bauteils definiert ist. Beim Farbbedrucken von Oberflächen spielt die Orientierung allerdings eine Rolle und muss voll definiert werden. Das benötigte Koordinatensystem ist wie in den Grundlagen beschrieben aus den Vektoren $\mathbf{e}_x$, $\mathbf{e}_y$, und $\mathbf{e}_z$ aufgebaut, wobei definiert wird, dass

$$\mathbf{e}_z = \mathbf{n} \tag{3.4}$$

da die Oberflächennormale des Pfades $\mathbf{e}_z = \mathbf{n}$ bereits gegeben ist. Zusätzlich soll $\mathbf{e}_x$ immer tangential zur Kurve ausgerichtet sein.

$$\mathbf{e}_x = \mathbf{t} \tag{3.5}$$

womit $\mathbf{e}_y$ aufgrund der Orthonormalität der Basis ist $\mathbf{e}_y$ als

$$\mathbf{e}_y = \mathbf{t} \times \mathbf{n} = \mathbf{e}_x \times \mathbf{e}_z \tag{3.6}$$

zu definieren.
Der Pfad kann als diskrete Parameterkurve betrachtet werden, wodurch die Tangente wie in den Grundlagen (2.23) berechnet werden kann. Da die Kurve diskretisiert ist und es keine Zeitabhängigkeit gibt, vereinfacht sich der Differenzenquotient

$$\dot{\mathbf{r}} = \frac{\mathbf{r}(t+h) - \mathbf{r}(t)}{h} \tag{3.7}$$

wegen den Einheitsschritten zwischen den Pfadprimitiven mit $h = 1$ zu

$$\dot{\mathbf{r}} = (\mathbf{p}_{i+1} - \mathbf{p}_i) \tag{3.8}$$

mit $\mathbf{r}(i) = \mathbf{p}_i$, wodurch sich der Vektor $\mathbf{e}_x$ durch Normieren von $\mathbf{t}$ zu

$$\mathbf{e}_x = \frac{\dot{\mathbf{r}}}{|\dot{\mathbf{r}}|} = \frac{\mathbf{t}}{|\mathbf{t}|} \tag{3.9}$$

berechnen lässt. Schließlich folgt mit $\mathbf{e}_y$ wie in Gleichung (3.6) der letzte benötigte Vektor, um die Rotationsmatrix

$$\mathbf{R}_{Pfad} = (\mathbf{e}_x\ \mathbf{e}_y\ \mathbf{e}_z) \tag{3.10}$$

aufstellen zu können.

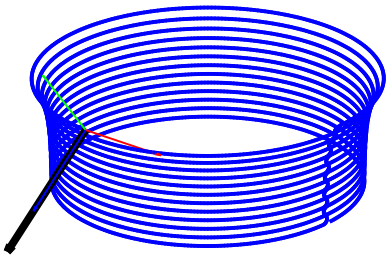

Abb. 3.4: Beispielpfad mit $\mathbf{R}_{Pfad}$ und Z-Achse von $\mathbf{R}_{Tool}$ (schwarz)

Diese Matrix folgt dem Pfad und kann benutzt werden, um die Orientierung des Pfades vollständig zu beschreiben - so wie es bei vielen inverse Kinematik Algorithmen benötigt wird. Da das Koordinatensystem jetzt mit $\mathbf{e}_z$ immer normal auf der unterliegenden Fläche steht, durch die der Pfad erzeugt wurde, muss eine weitere Rotationsmatrix $\mathbf{R}_o$ eingeführt werden, um eine andere von der Normalen abweichende Orientierung auf der Oberfläche einstellen zu können. $\mathbf{R}_o$ wird aufgrund der einfachen Handhabung durch die ZY'X"-Konvention definiert, die es ermöglicht, dass das auf dem Pfad stehende Koordinatensystem willkürlich mit den Eulerwinkeln $\mathbf{\Phi} = (\phi\ \vartheta\ \psi)^T$ zu manipulieren. Die Manipulation wird durch die Verkettung der beiden Rotation

$$\mathbf{R}_{Tool} = \mathbf{R}_{Pfad}\mathbf{R}_o(\mathbf{\Phi}) \tag{3.11}$$

ermöglicht, wobei $\mathbf{R}_{Tool}$ das relevante und weiterhin betrachtete Koordinatensystem darstellt. Man beachte, dass

$$\mathbf{R}_{Tool} = \mathbf{R}_{Pfad},\ bei\ \mathbf{\Phi} = (0\ 0\ 0)^T \tag{3.12}$$

auch für Anwendungen, bei denen eine definierte Orientierung um die Oberflächennormale $\mathbf{n}$ beziehungsweise der Z-Achse des Tool-Koordinatensystems $\mathbf{R}_{Tool}$, nicht nötig zu sein

scheint, muss diese doch dem Pfad bzw. der Trajektorie übergeben werden. Zusammen mit dem Ort kann dem Trajektorienplaner dann eine Liste mit den zusammengesetzten Transformationsmatrizen $\mathbf{T}_i$ übergeben werden. Zur Analyse der Krümmung kann diese mit der in den Grundlagen beschriebenen Formel (2.27) berechnet werden. Die hierfür benötigte zweite Ableitung des Pfades

$$\ddot{\mathbf{r}} = \frac{\mathbf{r}(t+h) - 2 \cdot \mathbf{r}(t) + \mathbf{r}(t-h)}{h^2} \tag{3.13}$$

vereinfacht sich aus dem selben Grund der Vereinfachung der ersten Ableitung.

$$\ddot{\mathbf{r}} = \mathbf{r}_{i+1} - 2 \cdot \mathbf{r}_i + \mathbf{r}_{i-1} \tag{3.14}$$

Zur Berechnung der projizierten Krümmung wird wieder das Vorgehen aus den Grundlagen herangezogen. Die durch den Punkt $\mathbf{p}_i$ und die Z-Achse der Tool-Rotationsmatrix aufgespannte Ebene projizierten Punkte sind $\mathbf{p}_{projiziert,i+1}$ und $\mathbf{p}_{projiziert,i-1}$. Sie werden durch ihren eigenen $\mathbf{e}_z$ Vektoren mittels Schnittpunktermittlung auf die Ebene geworfen.
Auffällig ist, dass die reale Krümmung der Kurve je nach Orientierung des Toolframes von der projizierten Krümmung abweicht, wobei $\kappa_{projiziert} \in [0, \kappa_{Pfad}]$ ist - womit sich Kapitel 4 auseinandersetzt.

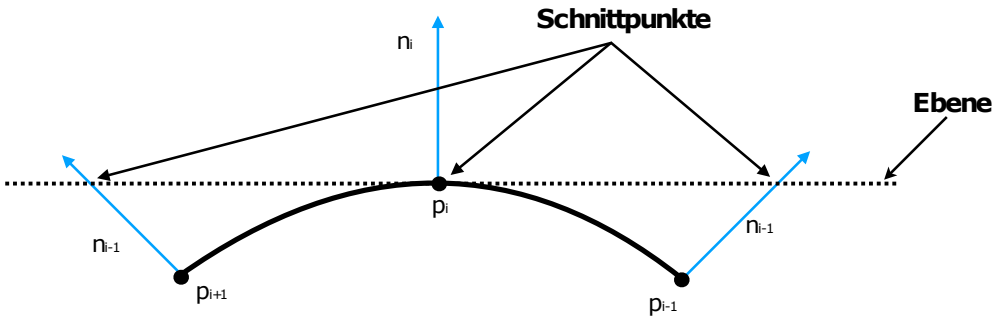

Abb. 3.5: Projektion der Punkte $\mathbf{p}_{i+1}$, $\mathbf{p}_i$ und $\mathbf{p}_{i+1}$ auf die Ebene.

Die Berechnung der Orientierung kann in zwei Themenbereiche geteilt werden. Zum Einen die Analyse der Euler Winkel des Pfades $\mathbf{\Phi}_{Pfad}$ und zum Anderen die Berechnung der Jointwinkel der Achsen in der Spherical Wrist des Manipulators $\mathbf{q}_o = (q_4\ q_5\ q_6)^T$.
Zur Berechnung von $\mathbf{\Phi}_{Pfad}$ müssen die Eulerwinkel mittels der, in den Grundlagen gegebenen inverse Orientierungsrechnung, aus der die Tool Rotationsmatrix $\mathbf{R}_{Tool}$ mit einer der angegebenen Varianten Formel (2.9) und (2.10) berechnet werden.
Die berechneten Winkel $\mathbf{\Phi}_{Pfad} = (\varphi\ \theta\ \psi)$ spiegeln dann zu jeder Zeit die genaue Orientierung der optimalen Druckausrichtung für den ausgewählten Prozess, in Relation zur Objektbasis des bedruckten/gedruckten Körpers wieder, bieten aber nur mittelbar Auskunft über die Orientierung des Roboter-Endeffektors.
Bevor man $\mathbf{q}_o$ aus $\mathbf{R}_{Tool}$ berechnen kann, muss man sich klar werden, was während des Prozesses genau passiert. Wie anfangs bereits beschriebenen, ist es Ziel, einen an den Endeffektor fixierten Körper mittels FDM zu bedrucken, wobei der Pfad den Druckpfad, den der Extruder abfährt, beschreiben soll. Die optimale Druckorientierungsmatrix $\mathbf{R}_{Tool}$ beschreibt dann bei fortschreitender Laufvariable den Punkt, auf der die Nozzle zu diesem Zeitpunkt, mit der aus der Matrix extrahierten Orientierung stehen soll, um einen stetigen Materialauftrag zu ermöglichen.
Um von Nozzle-Koordinatensystem $\mathbf{T}_{Nozzle}$ auf den Roboterflansch beziehungsweise das benötigte Endeffektor Koordinatensystem $\mathbf{T}_6^0$ schließen zu können, welches das Objekt zu jeder Zeit in der richtigen Orientierung unter den Extruder führt, muss eine Rücktransformation durchgeführt werden. Da hier nur die Orientierung betrachtet werden soll, reicht es aus, mit Rotationsmatrizen zu rechnen.
Bei gegebener, konstanter Nozzleorientierung $\mathbf{R}_{Nozzle}$ und variabler Pfadorientierung $\mathbf{R}_{Tool}(t)$ kann mittels Rücktransformation die Endeffektororientierung

$$\mathbf{R}_6^0(t) = \mathbf{R}_{Nozzle}\mathbf{R}_{Tool}^T(t) \tag{3.15}$$

berechnet werden. Nach der Berechnung von $\mathbf{R}_6^0(t)$ kann mittels dem klassischen analytischen inversen Orientierungsalgorithmus für Manipulatoren mit Spherical Wrist, welcher die inverse Orientierung der ZY'Z"-Konvention nach Euler beschreibt, auf die Winkel $\mathbf{q}_o = (q_4\ q_5\ q_6)^T$ geschlossen werden.

3.3 Trajektorienplaner

Dieses Kapitel befasst sich mit der Erzeugung einer Trajektorie aus einem Pfad, sprich dem Hinzufügen der Zeitkomponente. Ein Pfad, als Liste von Posen, welche der Manipulator nacheinander abfahren soll, wird im folgenden Algorithmus als .CSV Datei mit den zwölf relevanten Einträgen einer Transformationsmatrix abgespeichert.
Nach dem Auslesen der Elemente (folgendes Format: $p_x, p_y, p_z, x_1, x_2, x_3, y_1, y_2, y_3, z_1, z_2, z_3$) und Konvertierung in die ursprüngliche Transformationsmatrix

$$\mathbf{T} = \begin{pmatrix} \mathbf{R} & \mathbf{p} \\ \mathbf{0}^T & 1 \end{pmatrix} = \begin{pmatrix} x_1 & y_1 & z_1 & p_x \\ x_2 & y_2 & z_2 & p_y \\ x_3 & y_3 & z_3 & p_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3.16}$$

wird diese in die Rotationsmatrix **R** und den Ortsvektor **p** zerlegt. Relevante Größen der Trajektorie sind in diesem Algorithmus (für den vereinfachten Fall) eine konstante, gegebene Geschwindigkeit $|\mathbf{v}_{soll}|$, mit der der Pfad durchfahren werden soll, wobei darauf geachtet werden muss, dass eine bestimmte Winkelgeschwindigkeit $|\boldsymbol{\omega}_{max}|$ nicht überschritten werden darf. Diese Einschränkung sind von Roboter zu Roboter unterschiedlich und können nach Belieben angepasst werden. Wichtig ist auch, dass bei zu großen Abständen der Posen, also wenn das Δt zwischen den Posen zu groß wird, eine Interpolation durchgeführt werden muss, damit die Trajektorie flüssig vom Manipulator ausgeführt werden kann. Für die Zeitdiskretisierung wurde hier ein Δt von 15ms als Maximalwert gewählt. Zur Berechnung der Zeit, die der Roboter zum Abfahren der Distanz zwischen zwei Posen benötigt, kann mittels der vergebenen Geschwindigkeit $|\mathbf{v}_{soll}|$ und der euklidischen Distanz zwischen den Posen

$$l = |\mathbf{p}_{i+1} - \mathbf{p}_i| \tag{3.17}$$

mit der Formel

$$\Delta t = \frac{l}{|\mathbf{v}_{soll}|} \tag{3.18}$$

die Zeit Δt zwischen beiden Transformationsmatrizen berechnet werden. Um zu überprüfen, ob die maximal zulässige Winkelgeschwindigkeit $|\boldsymbol{\omega}_{max}|$ überschritten wird, müssen zunächst die Quaternionen Q_i und Q_{i+1} aus den zugehörigen Rotationsmatrizen $\mathbf{R}_i$ und $\mathbf{R}_{i+1}$, die wiederum aus den Posen $\mathbf{T}_i$ und $\mathbf{T}_{i+1}$ des Pfades zum Progressionspunkt i aus

der Liste der Matrizen extrahiert werden. Das geschieht mit dem inversen Orientierungsproblem wie in Kapitel 2.1.2 der Quaternionen beschrieben.
Zur Berechnung der Winkelgeschwindigkeit wird folgende Formel herangezogen:

$$\dot{Q} = \frac{1}{2} Q_\omega Q \tag{3.19}$$

wobei $\dot{Q}$ im zeitdiskreten Fall mit der Formel

$$\dot{Q} = \frac{Q_{i+1} - Qi}{\Delta t} \tag{3.20}$$

berechnet werden kann.
Nachdem man nach Q_ω umstellt, erhält man Gleichung (3.19)

$$Q_\omega = 2\dot{Q}Q^{-1} \tag{3.21}$$

Wenn man den Vektorteil $\boldsymbol{\epsilon}$ aus Q_ω extrahiert, entspricht dies genau der Winkelgeschwindigkeit zwischen den beiden Posen mit dem Zeitschritt Δt. Falls der Betrag der Winkelgeschwindigkeit $|\boldsymbol{\omega}|$ den Betrag der maximalen Winkelgeschwindigkeit $|\boldsymbol{\omega}_{max}|$ überschreitet, muss Δt neu berechnet werden.
Da die Winkelgeschwindigkeit zwischen den beiden Posen nichts Anderes beschreibt als die Änderung des Winkels zwischen den beiden Frames, muss besagter Winkel berechnet werden. Dies kann auf verschiedene Möglichkeiten geschehen.

1. Möglichkeit:

$$\Delta Q = Q_{i+1} Q_i^{-1} \tag{3.22}$$

mit

$$\theta = 2arcos(\eta) \tag{3.23}$$

oder

$$\theta = 2Atan2(|\boldsymbol{\epsilon}|, \eta) \tag{3.24}$$

als, vor allem im Code, stabilere Variante.

2. Möglichkeit:

$$\Delta\mathbf{R} = \mathbf{R}_{i+1}\mathbf{R}_i^T \tag{3.25}$$

wobei $\Delta\mathbf{R}$ dann auch in ein, mit dem Algorithmus aus (2.14) und (2.15) berechneten, Quaternion ΔQ konvertiert wird, um den Winkel θ wie in (3.23) und (3.24) zu berechnen. Damit ergibt sich Δt_{neu} zu

$$\Delta t_{neu} = \frac{\theta}{|\boldsymbol{\omega}_{max}|} \tag{3.26}$$

Als letzter Schritt muss überprüft werden, ob Δt bei den gewählten Parametern $|\mathbf{v}_{soll}|$ und $|\boldsymbol{\omega}_{max}|$ genügend klein ist, um ein flüssiges Abfahren der Trajektorie zu gewährleisten. Falls Δt größer als Δt_{max} ist, müssen Ortsvektor und Orientierung zwischen $\mathbf{T}_i$ und $\mathbf{T}_{i+1}$ interpoliert werden. Im Folgenden werden die Interpolationsalgorithmen angeführt, die im Programm benutzt werden. Zur Interpolation des Ortes genügt eine lineare Interpolation zwischen den beiden Punkten mit

$$\mathbf{p}_{int} = \mathbf{p}_i + s(\mathbf{p}_{i+1} - \mathbf{p}_i) \tag{3.27}$$

wobei s eine Zeitskalierung beschreibt, die in diesem Algorithmus vereinfacht, und zum Einhalten der konstanten Geschwindigkeit zwischen den Punkten einfach linear von 0 auf 1 steigt. Dies kann zu Unstetigkeiten der Geschwindigkeit führen und noch weiter optimiert werden, ist für die benötigten Zwecke allerdings vollkommen ausreichend.

$$s(T,t) = \frac{t}{T} \tag{3.28}$$

mit $t \in [0,T]$ als Variable der Zeit und T als Zeit, in der s von 0 auf 1 steigen soll.
Für die Interpolation der Orientierung wird ein so genannter Spherical Linear Interpolation (SLERP) Algorithmus benutzt, der auf Quaternionen basiert:

$$Q_{int} = slerp(Q_1, Q_2, s) \tag{3.29}$$

Er dreht eine Orientierung, ausgehend von Q_1 als Startquaternion, über den kürzesten Weg in die Endorientierung, repräsentiert durch das Quaternion Q_2. Es gibt verschiedene Möglichkeiten der Quaternioneninterpolation, benutzt wird hier aber folgender Algorithmus: Zunächst muss wieder die Winkeldifferenz der beiden Quaternionen

$$cos\left(\frac{\theta}{2}\right) = \langle Q_1, Q_2 \rangle \tag{3.30}$$

woraus

$$\frac{\theta}{2} = acos(\langle Q_1, Q_2 \rangle) \tag{3.31}$$

folgt, berechnet werden, wobei der Operator

$$\langle Q_1, Q_2 \rangle = \sum_i Q_{1i} \cdot Q_{2i} \tag{3.32}$$

das Skalarprodukt der beiden Quaternionen darstellt. Des Weiteren werden für die Interpolation der Quaternionen die Werte r_1 und r_2 und der Sinus der Winkeldifferenz benötigt, den man aus dem Additionstheorem

$$cos^2 + sin^2 = 1 \tag{3.33}$$

herleiten kann. Durch Umstellen der Gleichung (3.31) erhält man

$$sin(\frac{\theta}{2}) = \sqrt{1 - cos(\frac{\theta}{2})^2} \tag{3.34}$$

Die eigentliche Interpolation über die Zeitskalierung s, wie sie auch in (3.27) benutzt wurde, folgt dann über die beiden Werte

$$r_1 = \frac{sin((1-s) \cdot \frac{\theta}{2})}{sin(\frac{\theta}{2})} \tag{3.35}$$

und

$$r_2 = \frac{sin(s \cdot \frac{\theta}{2})}{sin(\frac{\theta}{2})} \tag{3.36}$$

wobei r_1 von 1 auf 0 sinkt und r_2 von 0 auf 1 steigt, da $s \in [0, 1]$. Daraus folgt das interpolierte Quaternion

$$Q_{int} = (\eta_{int}, \boldsymbol{\epsilon}_{int}) \tag{3.37}$$

mit

$$\eta_{int} = \eta_1 \cdot r_1 + \eta_2 \cdot r_2 \tag{3.38}$$

$$\boldsymbol{\epsilon}_{int} = \begin{pmatrix} \epsilon_{x1} \cdot r_1 + \epsilon_{x2} \cdot r_2 \\ \epsilon_{y1} \cdot r_1 + \epsilon_{y2} \cdot r_2 \\ \epsilon_{z1} \cdot r_1 + \epsilon_{z2} \cdot r_2 \end{pmatrix} \tag{3.39}$$

4 Analyse

4.1 Inverse Kinematik Algorithmus

Dieses Kapitel befasst sich mit der inversen Kinematik eines RRRRRR Roboters mit Spherical Wrist. Da der Algorithmus nicht grundlegendes Thema dieser Arbeit ist, wird er nur grob dargelegt, um eine Einsicht in die Thematik zu gewährleisten.

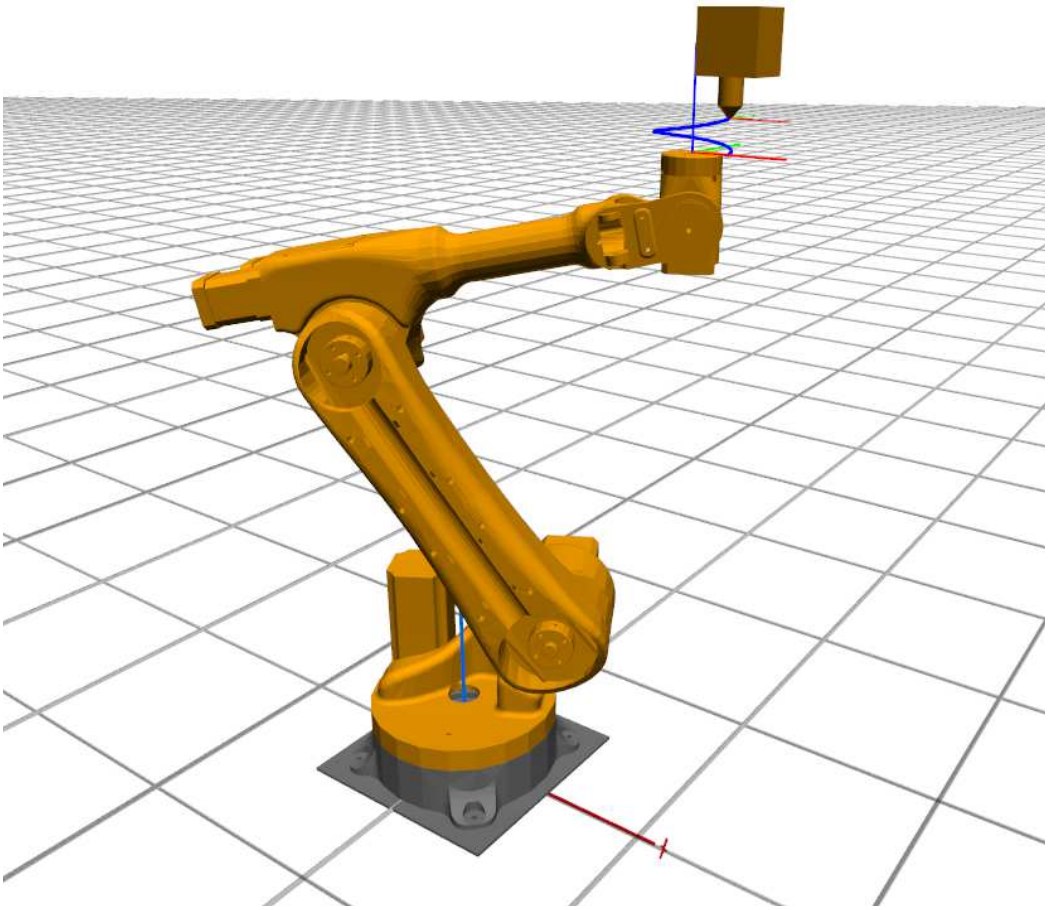

Abb. 4.1: RRRRRR Manipulator mit Spherical Wirst bei einem simulierten Druckprozess.

4.1.1 Forward Kinematik

Die Forward Kinematik eines Roboters beschreibt die Transformation vom Welt-Koordinatensystem bis zum TCP-Koordinatensystem beziehungsweise dem Roboterflansch. Für diese Transformation sind mehrere Transformationsmatrizen nötig, die von den Joint-Winkeln abhängen. Im zweiten Kapitel werden diese näher erläutert.
Für die Kinematik eines Roboters befolgt man für gewöhnlich die Denavit-Hartenberg-Konvention zur Aufstellung dieser Transformationsmatrizen. Eine nach dieser Konvention abhängige Transformationsmatrix ist dann von vier Denavit-Hartenberg-Parametern α, ϑ, a und d abhängig, die von Roboter zu Roboter variieren und von deren Aufbau abhängig sind. Wobei ϑ den Winkel des Gelenkes beschreibt. Pro Gelenk (hier sechs) gibt es eine Transformationsmatrix.

$$\mathbf{T}_i^{i-1} = \begin{pmatrix} c_{\theta_i} & -s_{\theta_i}c_{\alpha_i} & s_{\theta_i}s_{\alpha_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i}c_{\alpha_i} & -c_{\theta_i}s_{\alpha_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{4.1}$$

Die Verkettung beziehungsweise Multiplikation dieser Transformationen zu

$$\mathbf{T}_6^0(\mathbf{q}) = \mathbf{T}_1^0(q_1)\mathbf{T}_2^1(q_2)\mathbf{T}_3^2(q_3)\mathbf{T}_4^3(q_4)\mathbf{T}_5^4(q_5)\mathbf{T}_6^5(q_6) \tag{4.2}$$

birgt dann die gewünschte Transformation vom Weltkoordinatensystem zum Roboterflansch bei der Joint Konfiguration $\mathbf{q}$, wobei $\mathbf{q} = (q_1\ q_2\ \dots\ q_6)^T$ die Jointwinkel sind. Für die inverse Differentialkinematik, speziell zum Aufstellen der geometrischen Jacobimatrix, werden auch die Koordinatensysteme der einzelnen Gelenke benötigt, die sich aus den Zwischenschritten der Formel (4.2) ergeben

$$\mathbf{T}_1^0(\mathbf{q}) = \mathbf{T}_1^0(q_1) \tag{4.3}$$

$$\mathbf{T}_2^0(\mathbf{q}) = \mathbf{T}_1^0(q_1)\mathbf{T}_2^1(q_2) \tag{4.4}$$

$$\mathbf{T}_3^0(\mathbf{q}) = \mathbf{T}_1^0(q_1)\mathbf{T}_2^1(q_2)\mathbf{T}_3^2(q_3) \tag{4.5}$$

$$\mathbf{T}_4^0(\mathbf{q}) = \mathbf{T}_1^0(q_1)\mathbf{T}_2^1(q_2)\mathbf{T}_3^2(q_3)\mathbf{T}_4^3(q_4) \tag{4.6}$$

$$\mathbf{T}_5^0(\mathbf{q}) = \mathbf{T}_1^0(q_1)\mathbf{T}_2^1(q_2)\mathbf{T}_3^2(q_3)\mathbf{T}_4^3(q_4)\mathbf{T}_5^4(q_5) \tag{4.7}$$

4.1.2 Inverse Differential Kinematik

Die inverse differential Kinematik wird oft benutzt und ist vor allem gut geeignet um Trajektorien abzufahren. Bei diesem Algorithmus wird die gewünschte und durch die Trajektorie vorgegebene Toolframe-Geschwindigkeit $\mathbf{v}_E$ mittels einer Jacobimatrix $\mathbf{J}_G$, die speziell für die jeweilige Roboterkonfiguration $\mathbf{q}$ berechnet wird, in Joint-Geschwindigkeiten übersetzt.

$$\dot{\mathbf{q}} = \mathbf{J}_G(\mathbf{q})^{-1}\mathbf{v}_E \tag{4.8}$$

Diese Joint-Geschwindigkeiten können dann im zeitdiskreten Fall mit Hilfe der Eulerschen Integration

$$\mathbf{q}(t_{k+1}) = \mathbf{q}(t_k) + \dot{\mathbf{q}}\Delta t \tag{4.9}$$

zu den Jointwinkeln integriert werden, was dazu führt, dass der Roboter die gewünschte Trajektorie abfährt. Zur Berechnung der Joint Geschwindigkeiten muss zunächst die geometrische Jacobimatrix

$$\mathbf{J}_G = \begin{pmatrix} \mathbf{J}_P \\ \mathbf{J}_O \end{pmatrix} \tag{4.10}$$

aufgestellt werden, wobei die Einträge

$$\mathbf{J}_{Pi} = \mathbf{e}_{z,i-1} \times (\mathbf{p}_E - \mathbf{p}_{i-1}) \tag{4.11}$$

aus der Teilmatrix $\mathbf{J}_P$ die translatorische Geschwindigkeit berechnen und die Einträge

$$\mathbf{J}_{Oi} = \mathbf{e}_{z,i-1} \tag{4.12}$$

aus der Teilmatrix $\mathbf{J}_O$ die Winkelgeschwindigkeit $\boldsymbol{\omega}$ berechnen. Mit Vektor

$$\mathbf{e}_{z,i} = \begin{pmatrix} z_1 \\ z_2 \\ z_3 \end{pmatrix} \tag{4.13}$$

und Positionsvektor

$$\mathbf{p}_i = \begin{pmatrix} p_x \\ p_y \\ p_z \end{pmatrix} \tag{4.14}$$

aus der Transformation $\mathbf{T}_i^0$.
Dies resultiert in der Matrix

$$\mathbf{J}_G(\mathbf{q}) = \begin{pmatrix} \mathbf{J}_{P1} & \dots & \mathbf{J}_{P6} \\ \mathbf{J}_{O1} & \dots & \mathbf{J}_{O6} \end{pmatrix} \tag{4.15}$$

Der Vektor

$$\mathbf{v}_E = (\dot{p_x}\ \dot{p_y}\ \dot{p_z}\ \omega_x\ \omega_y\ \omega_z)^T \tag{4.16}$$

besteht zum Einen aus der gewünschten linearen TCP-Geschwindigkeit $\dot{\mathbf{p}}$ und zum Anderen aus der gewünschten Winkelgeschwindigkeit des Toolframes $\boldsymbol{\omega}$. Diese werden beim Abfahren der Trajektorie wie folgt berechnet: Zunächst wird die Transformationsmatrix $\mathbf{T}_{t_k}$, welche die gewünschte Pose des Endeffektors zum Zeitpunkt t_k ist, sowie die Pose im nächsten Schritt $\mathbf{T}_{t_{k+1}}$ aus der Trajektorienliste gezogen. Diese werden in Rotation und Translation unterteilt, wodurch sich $\mathbf{p}_{t_k}$ und $\mathbf{p}_{t_{k+1}}$, sowie $\mathbf{R}_{t_k}$ und $\mathbf{R}_{t_{k+1}}$ ergeben. Die gewünschte translatorische Geschwindigkeit zum diskreten Zeitpunkt t_k ergibt sich aus

$$\dot{\mathbf{p}} = \frac{\mathbf{p}_{t_{k+1}} - \mathbf{p}_{t_k}}{\Delta t} \tag{4.17}$$

mit Δt als diskretem Zeitintervall, wie es zuvor im Trajektorienplaner berechnet wurde.
Genauso wie auch im Trajektorienplaner werden nun die Rotationsmatrizen in Quaternionen überführt, um die gewünschte Winkelgeschwindigkeit zu berechnen.

$$\dot{Q} = \frac{Q_{t_{k+1}} - Q_{t_k}}{\Delta t} \tag{4.18}$$

und

$$Q_\omega = 2\dot{Q}Q_{t_k}^{-1} \tag{4.19}$$

Zusammen mit $\dot{\mathbf{p}}$ ergibt der Vektorteil $\boldsymbol{\epsilon}$ aus Q_ω den gewünschten Geschwindigkeitsvektor $\mathbf{v}_E$, und

$$\dot{\mathbf{q}} = \mathbf{J}_G(\mathbf{q})^{-1}\mathbf{v}_E \tag{4.20}$$

Bei der Euler Integration der Winkelgeschwindigkeiten der Joints kommt es zu einer Akkumulation von kleinen Fehlern, deshalb divergiert die berechnete Lösung von der gewünschten Pose bei längeren Trajektorien. Aus diesem Grund muss ein Controller in den Algorithmus eingebaut werden, der die Abweichung möglichst gering hält. Hier bietet sich als simplere Variante ein Proportional (P) Regler an und als performante Variante ein PID-Regler, der den Fehler beim Abfahren der Trajektorie gegen null laufen lassen. Da man die Implementierung des P-Reglers aus der des PID-Reglers ableiten kann und der PID-Regler ein besseres dynamisches Verhalten aufweist, wird im Folgenden nur auf den PID-Regler eingegangen.
Zur Implementierung braucht man zunächst den Abweichungsfehler der gewünschten Pose zu der reellen Endeffektorpose zum diskreten Zeitpunkt t_k. Der Fehler $\mathbf{e}$ kann mit Gleichung (4.21) berechnet werden.

$$\mathbf{e} = \mathbf{X}_d - \mathbf{X}_E = \begin{pmatrix} \mathbf{e}_p \\ \mathbf{e}_o \end{pmatrix} \tag{4.21}$$

Zur Berechnung des Fehlers braucht man die Endeffektorposition $\mathbf{p}_E$ sowie die gewünschte Position $\mathbf{p}_d$, wobei $\mathbf{p}_E$ aus der Transformationsmatrix des Roboterflansches extrahiert wird und $\mathbf{p}_d$ die Position der Trajektorie zum Zeitpunkt t_k ist. Mit der Gleichung

$$\mathbf{e}_p = \mathbf{p}_d - \mathbf{p}_E \tag{4.22}$$

lässt sich anschließend die translatorische Fehlerabweichung beim Versuch, der Pose der Trajektorie zu folgen, zum Zeitpunkt t_k des Roboters berechnen. Den Fehler in der Orientierung $\mathbf{e}_o$ erhält man durch die Verrechnung von der gewünschten Orientierung, repräsentiert durch $\mathbf{R}_d$ und der tatsächlichen Orientierung des Endeffektors $\mathbf{R}_E$.

$$\Delta\mathbf{R} = \mathbf{R}_E^T\mathbf{R}_d \tag{4.23}$$

Hier $\Delta\mathbf{R}$ ist der Orientierungsunterschied beider Frames. Um den Orientierungsfehler aus

dieser Rotationsmatrix zu erhalten, kann die Eigenschaft

$$Q_1 Q_2^{-1} = Q_0 = (1, (0\ 0\ 0)^T),\ wenn\ Q_1 = Q_2 \tag{4.24}$$

der Quaternionen genutzt werden. Extrahiert man das Quaternion mit dem Vorgehen aus (2.13) und (2.14) aus der Rotationsmatrix $\Delta\mathbf{R}$, spiegelt der Vektorteil $\boldsymbol{\epsilon}$ den Orientierungsfehler $\mathbf{e}_o$ wieder

$$\mathbf{e}_o = \mathbf{R}_E \boldsymbol{\epsilon} \tag{4.25}$$

Die Ableitung der Fehlerabweichung $\dot{\mathbf{e}}$ kann mithilfe der geometrischen Jacobimatrix $\mathbf{J}_G(\mathbf{q})$ und der Formel

$$\dot{\mathbf{e}} = \mathbf{v}_E - \mathbf{J}_G(\mathbf{q})\dot{\mathbf{q}} \tag{4.26}$$

berechnet werden, wodurch kein numerisches Differenzieren für das D Glied notwendig ist.
Der PID Regler ist wie folgt aufgebaut

$$\mathbf{u}(t) = K_D \left[\mathbf{e}(t) + \frac{1}{T_i} \int_0^t \mathbf{e}(\tau) d\tau + T_d \frac{d\mathbf{e}(t)}{dt} \right] \tag{4.27}$$

mit den Faktoren K_D , T_i und T_d zur Einstellung des dynamischen Verhaltens des Reglers. Die benötigte numerische Integration kann wie bei der Integration der Winkel in (4.9) im zeitlich diskreten Fall mittels Euler Integration

$$\mathbf{e}_{integriert}(t_{k+1}) = \mathbf{e}_{integriert}(t_k) + \mathbf{e}(t_k)\Delta t \tag{4.28}$$

gelöst werden, die hier eine ausreichende Genauigkeit bietet. Alternativ könnte man für eine genauere Integration allerdings auch die Trapezregel verwenden. Mit dem Fehler $\mathbf{e}(t_k)$, der Fehlerfortpflanzung $\dot{\mathbf{e}}$ und dem integrierten Fehler $\mathbf{e}_{integriert}(t_k)$, sowie den Faktoren K_D, T_i und T_d ergibt sich der Kontrollterm dann zu

$$\mathbf{u}(t_k) = K_D[\mathbf{e}(t_k) + \frac{1}{T_i} \cdot \mathbf{e}_{integriert}(t_k) + T_D \mathbf{e}(t_k)] \tag{4.29}$$

die einfach , wie hier gezeigt,

$$\dot{\mathbf{q}}(t_{k+1}) = \mathbf{J}_G(\mathbf{q}(t_k))(\mathbf{v}_E(t_k) + \mathbf{u}(t_k)) \tag{4.30}$$

mit in den inverse differential Kinematik Algorithmus integriert werden muss. Wie in Abbildung 4.1 zu sehen ist, gewährleistet der PID-Regler ein sehr gutes Tracking der Trajektorie.

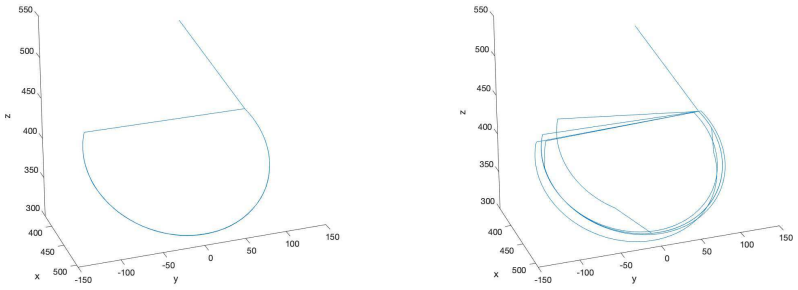

Abb. 4.2: Vergleich einer abgefahrenen Trajektorie mit PID-Regelung (links) mit $K_D = 50, T_i = 150$ und $T_d = 0.0001$ und ohne PID-Reglung (rechts).

4.2 Analyse und Interpretation der Ergebnisse

Nun stellt sich die Frage, wie die im Hauptteil hergeleiteten Algorithmen eingesetzt und die Ergebnisse interpretiert werden können. Zunächst muss unterschieden werden, ob die Orientierung an der Nozzle vollständig definiert sein muss, oder ob die Orientierung um die Z-Achse des Toolframes, als letzter Freiheitsgrad des Systems, freigelassen werden kann. Dies führt wie im nächstem Absatz beschrieben, zu erheblichen Unterschieden in den Anforderungen an den Roboter. Abgesehen davon ist die Analyse des Verhältnisses

$$\xi = \frac{\kappa_{projiziert}}{\kappa_{Pfad}} \tag{4.31}$$

interessant, da sich auch hier deutliche Unterschiede für den Roboter in der Fertigung ergeben. Unabhängig davon, ob die Orientierung vollständig definiert ist oder nicht.

4.2.1 Analyse der Freiheitsgrade

Im Zuge der Aufstellung der Tool-Koordinatensysteme bei der Pfaderzeugung für den Roboter, können hier mit Hilfe der in den Grundlagen erläuterten inversen Orientierung (2.9)

und (2.10) die Eulerwinkel der Orientierung berechnet und analysiert werden. Wenn man die Ergebnisse der inversen Orientierung nach Euler über einen Beispielpfad, bei dem die Orientierung im Druckpunkt voll definiert ist,

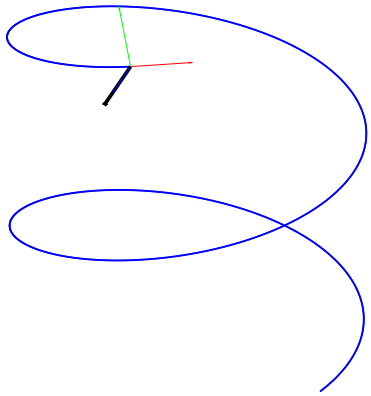

Abb. 4.3: Pfad simuliert aufschweißen einer Schnecke auf einen Zylinder, wobei $\mathbf{e}_z$ aus $\mathbf{T}_{Tool}$ (schwarzer Vektor) immer normal auf der unterliegenden Fläche steht.

in ein Diagramm plottet, wird deutlich, dass bei bestimmten Pfaden einer der Eulerwinkel $\mathbf{\Phi} = (\varphi\ \theta\ \psi)^T$ monoton über den Pfad ansteigt.

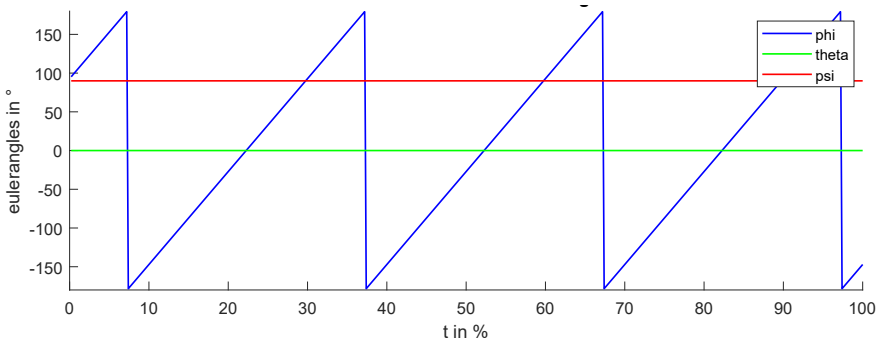

Abb. 4.4: Eulerwinkel φ des Toolframes $\mathbf{T}_{Tool}$ steigt monoton an.

Das bedeutet, dass je nach eingesetztem Roboter einer der Jointwinkel ebenfalls monoton steigen muss, was bei einigen Modellen nicht möglich ist. Am Beispiel des Kuka kr6 kann man dies zeigen, indem man den Pfad auf den Roboterflansch beziehungsweise die Druckplattform rücktransformiert. Dieser Schritt kann mit der Gleichung

$$\mathbf{R}_{Flansch} = \mathbf{R}_{Nozzle}\mathbf{R}_{Pfad}^{T} \tag{4.32}$$

realisiert werden. Wird der inverse differential Kinematik-, oder der analytische inverse Orientierungsalgorithmus für Roboter mit Spherical Wrist, beziehungsweise die inverse Orientierung nach Euler für die ZY'Z"-Konvention angewendet, kann man in Abbildung (4.4) erkennen, dass in diesem Fall der Jointwinkel q_6 monoton steigen müsste, um die Trajektorie wie gewünscht abfahren zu können.

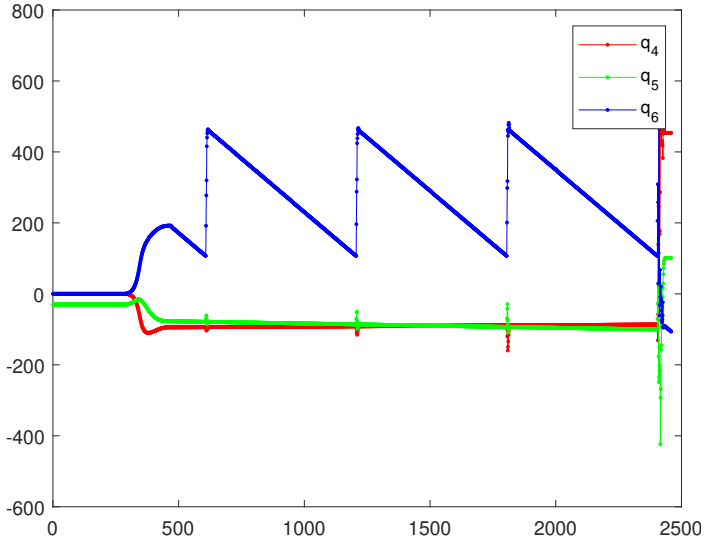

Abb. 4.5: Jointwinkel der Spherical Wrist $\mathbf{q}_o = (q_4\ q_5\ q_6)^T$, wobei q_6 monoton sinkt.

Das ist für den Kuka kr6 nicht möglich, da seine Gelenkwinkelbegrenzung für Joint 6 in $q_6 \in [-350°, 350°]$ liegen. Dies ist auch für eine aufbauende Fertigung, wie in dem analogen Beispiel aus Abbildung (4.5), der Fall.
Hier steht die Z-Achse nicht normal, sondern um $90°$ zur Normalen verdreht auf der zu druckenden Fläche des Objektes. Die Ausrichtung der Z-Achse gewährleistet, dass die für additiven Fertigungsprozesse optimale Orientierung stetig eingehalten werden kann und keine Overhangs, beziehungsweise Supportstrukturen für die Fertigung des Modells nötig sind. Dadurch wird Material und Zeit eingespart. Die Verletzungen der Winkelbegrenzungen sind aber bei Fertigungen mit dieser Methode nicht zwingend zu erwarten. Bleiben die Winkel $\mathbf{q} = (q_1\ q_2\ \dots\ q_6)^T$ in der durch die technischen Daten des Roboters vorgegebenen konvexen Hülle, sind die vollständig orientierungsbestimmten Pfade gut ausführbar.
Ist andererseits die Orientierung um die Z-Achse nicht vorgegeben, wie dies für einen 3D-Druck Prozess der Fall ist, kann man beobachten, dass durch das Offenlassen dieses Freiheitsgrads bei sich aufbauenden Prozessen der Effekt zu beobachten ist, dass der Eulerwinkel nicht mehr monoton ansteigt, wie es im ersten Fall bei einer festgelegten Orientierung auf der Oberfläche war.

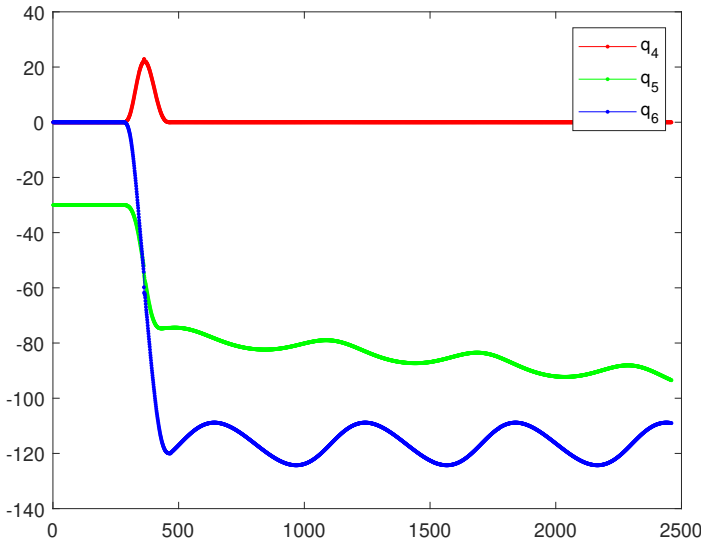

Abb. 4.6: Prozess ohne definierter Orientierung um die Z-Achse der Nozzle. Die Jointwinkel der Spherical Wrist $\mathbf{q}_o = (q_4\ q_5\ q_6)^T$ bleiben im ausführbaren Bereich.

Der Eulerwinkel bleibt somit in einem schmalen Band, was für viele Roboter sehr gut ausführbar ist, da es keine Konflikte mit den diktierten Jointbegrenzungen gibt. Beim Bedrucken von Oberflächen, bei denen die Z-Achse des Tool-Koordinatensystems normal auf der Oberfläche steht, gilt dies allerdings nicht, da es hier ansonsten zur Kollision zwischen Nozzle oder Fertigungswerkzeug und dem Körper kommen würde. Solange der Pfad keine Begrenzungen des Roboters verletzt, wird es keine Probleme bei der Ausführung geben.

4.2.2 Analyse der projizierten Krümmung

Berechnet man die Krümmung κ_{Pfad}, definiert als der Kehrwert des Radius des an den Pfad im Punkt $\mathbf{p}_i$ angeschmiegten Kreises $\kappa_{Pfad,i} = \frac{1}{r_i}$, kann man erkennen, dass diese proportional zu der TCP-Beschleunigung $\ddot{\mathbf{q}}$ ist. Da zu große oder gar unendliche Beschleunigungen manchmal nicht auszuschließen sind, sie jedoch meist vermieden werden sollten, gibt die Krümmung κ_{Pfad} bereits Aufschlüsse über den Pfad und die darauf folgende Trajektorie. Zieht man jetzt noch die projizierte Krümmung $\kappa_{projiziert}$ hinzu und berechnet das Verhältnis

$$\xi = \frac{\kappa_{projiziert}}{\kappa_{Pfad}} \tag{4.33}$$

kann die Dynamik noch weiter entschlüsselt werden.
Wie bereits beschrieben, ist $\kappa_{projiziert} \in [0, \kappa_{Pfad}]$ was bedeutet, dass $\xi \in [0, 1]$ sein muss.

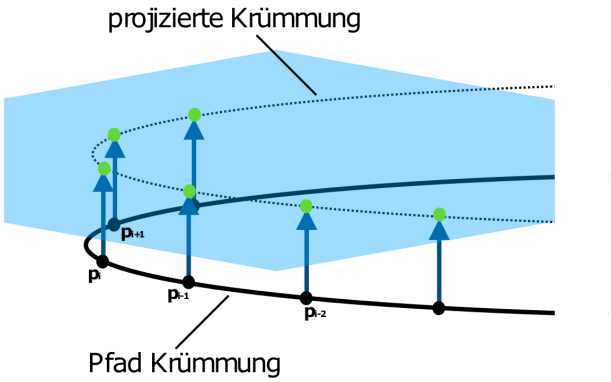

Abb. 4.7: Krümmungsprojektion mit $\xi = 1$.

Auffällig ist, dass die Winkelgeschwindigkeit $\boldsymbol{\omega}$ in Pfadsegmenten mit $\xi = 1$ gegen null geht, sofern man den Freiheitsgrad der Nozzle um die Z-Achse frei lässt, wobei $\boldsymbol{\omega}$ bei selbigem Pfadsegment mit $\xi = 0$ maximal wird.

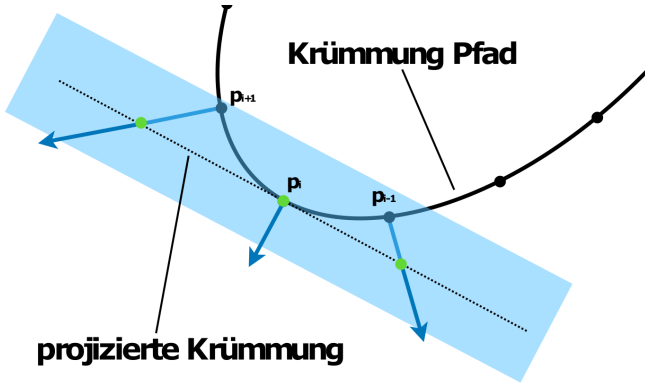

Abb. 4.8: Krümmungsprojektion mit $\xi = 0$.

Angenommen der Pfad beschreibt eine scharfe Ecke wie in Abbildung (4.9) zu sehen ist und der Pfad soll mit der konstanten, translatorischen Geschwindigkeit $|\mathbf{v}_{soll}|$ durchfahren werden, um einen möglichst konstanten Materialauftrag der Nozzle zu gewährleisten, dann heißt das im Fall $\xi = 0$, dass es bei dem Umfahren der Ecke eine Polstelle der TCP-Winkelgeschwindigkeit gibt, wohingegen die translatorische Geschwindigkeit unstetig ist.

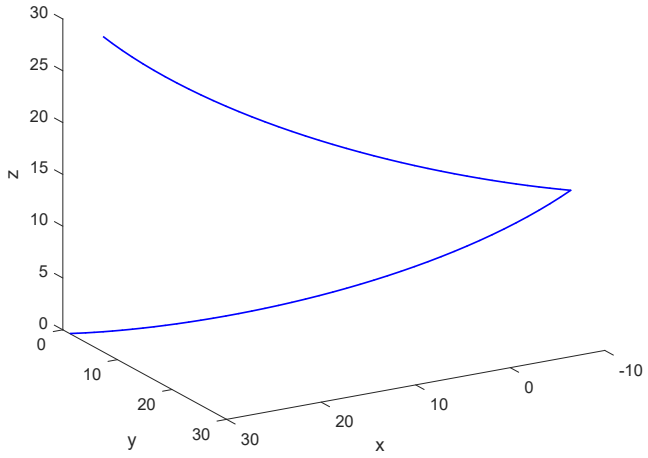

Abb. 4.9: Eckiger Pfad auf einer gekrümmten Oberfläche.

Eine unstetige Geschwindigkeit ist zwar nicht wünschenswert und führt zu einem Ruck beim Abfahren, der für den Manipulator jedoch realisierbar ist, wohingegen unendliche Geschwindigkeiten nie realisierbar sind, was den Pfad in diesem Fall nicht abfahrbar macht.

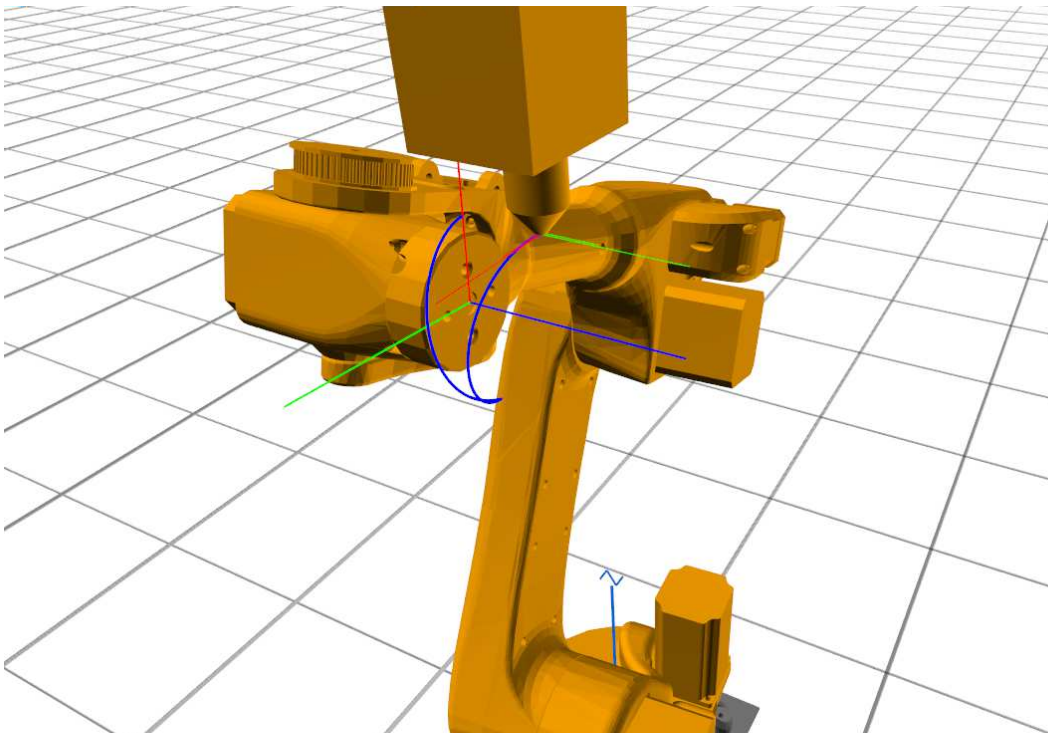

Abb. 4.10: Manipulator simuliert das Abfahren der Trajektorie mit $\xi = 1$.

Auf dieses Kriterium muss in der Trajektorienplanung besonders geachtet werden und es müssen Maßnahmen folgen, um diesen nicht wünschenswerten Zustand zu umgehen.

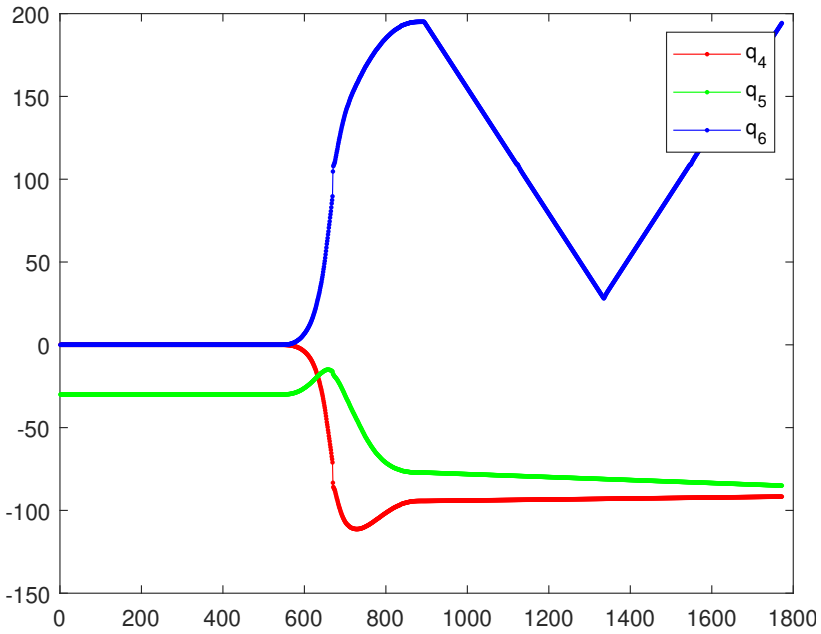

Abb. 4.11: Winkel der Spherical Wrist $\mathbf{q}_o = (q_4\ q_5\ q_6)^T$ des Pfades aus Abbildung (4.9) mit eingestellter Orientierung, so dass $\xi = 1$.

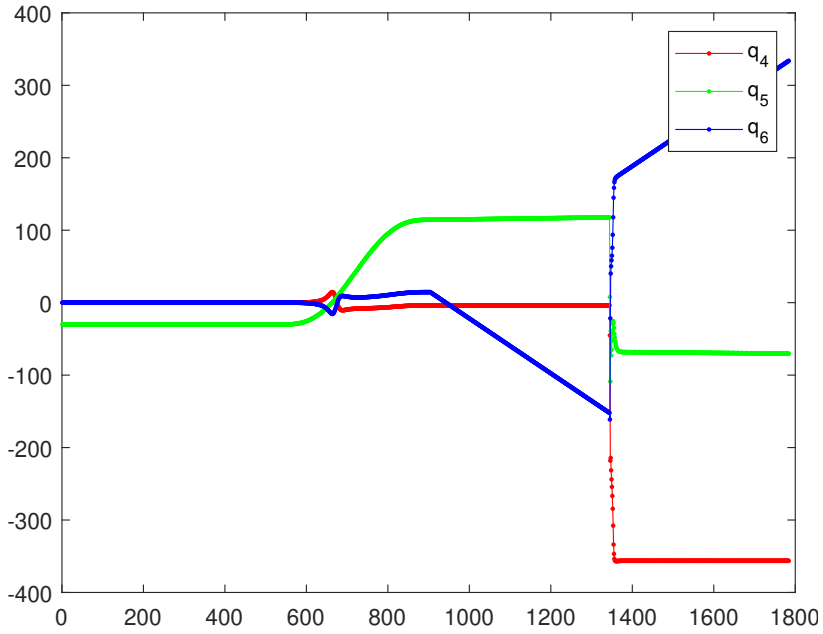

Abb. 4.12: Winkel der Spherical Wrist $\mathbf{q}_o = (q_4\ q_5\ q_6)^T$ des Pfades aus Abbildung (4.9) mit eingestellter Orientierung, so dass $\xi = 0$.

Es wird deutlich, dass es bei Konfiguration $\xi = 0$ einen deutlichen Sprung der Gelenkwinkel gibt, was auf eine unendliche Winkelgeschwindigkeit $\boldsymbol{\omega}$ hindeutet. In Abbildung (4.11) hingegen sind keine Anzeichen solcher Probleme erkennbar, lediglich ein Knick in q_6 ist auffällig, was auf eine unstetige Jointgeschwindigkeit $\dot{q}_6$ hindeutet. Dies ist allerdings nicht weiter beachtenswert.
Als Lösung zu problematischen Pfadorientierung nahe $\xi = 0$ können die im Trajektorienplaner vorgeschlagenen Methoden aufgegriffen werden. Vor allem die, die zu große Winkelgeschwindigkeiten der Trajektorie unterdrückt.
Angewandt auf den Pfad mit der Orientierung, aus der Faktor $\xi = 0$ folgt, ist in Abbildung (4.13) deutlich erkennbar, dass die Winkelgeschwindigkeitsbegrenzung zum Zeitpunkt der Unstetigkeiten im vorherigen Beispiel greift, und die Probleme behebt.

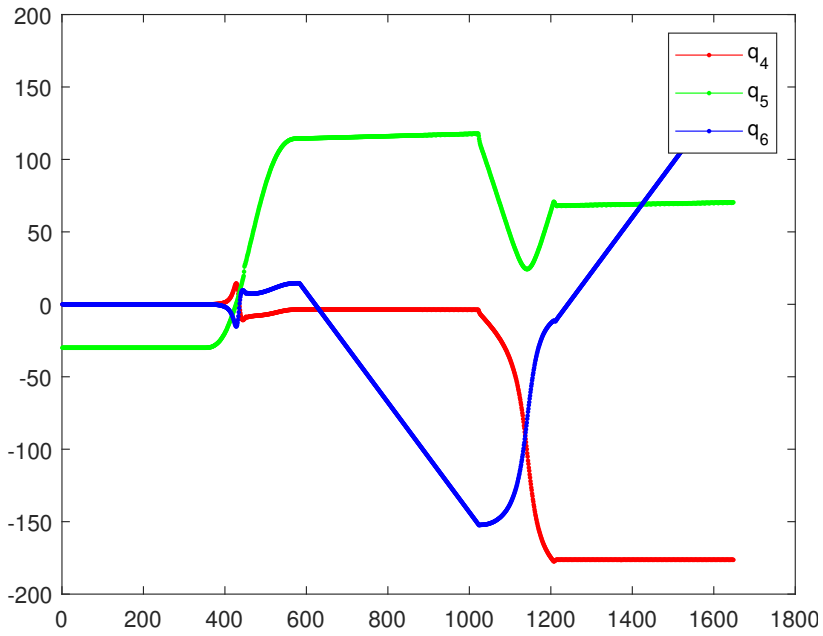

Abb. 4.13: Winkel der Spherical Wrist $\mathbf{q}_o = (q_4\ q_5\ q_6)^T$ des Pfades aus Abbildung (4.9) mit eingestellter Orientierung, so dass $\xi = 0$ mit eingeschaltetem Controller in der Trajektorienplanung.

5 Zusammenfassung und Ausblick

Zusammenfassend und abschließend wurde eine weitführende Analyse von Pfaden in der Multidirektionalen additiven Fertigung vorgestellt und erörtert, die zweckerfüllende Ergebnisse liefert. Diese beschriebenen Methoden könnten zukünftig in Trajektorierenplaner integriert werden, da das Planen der Pfade und vor allem der Trajektorie auschlaggebend für einen stabilen Prozess zu sein scheint.
Rückblickend auf die einführende Vorstellung verschiedener Robotertypen kann gesagt werden dass es Irrelevant ist welcher Roboter für die Aufgabe gewählt wird, solange die vom Hersteller definierten Jointlimits für die gewählte Aufgabe ausreichend sind. Es sollte allerdings ein Manipulator mit sechs oder mehr Achsen gewählt werden, da ansonsten eine Erfüllung der Aufgabe nicht gewährleistet werden kann, und das Ziel einer vollen multidirektionalität des Prozesses verfehlt wird. Eine Ausnahme der benutzbaren Roboter stellt allerdings der vom IGMR entwickelte Paragrip dar, der aufgrund seines besonderen kinematischen Layouts besonders für die manipulation von Objekten, aber weniger für das anbringen von Bauteilen an einer Druckplattform und das bedrucken dieser Bauteile geeignet ist. Ein besonderes Augenmerk liegt allerdings auf dem vorgestellten Robotertypen der wegen seiner großen Verbreitung und Flexibilität besonders geeignet sein zu scheint.
Weiterführend wurden im Rahmen dieser Projektarbeit allerdings auch weitere Verfahren der Analyse des dynamischen Verhalten im Druckprozess eines RRRRRR Manipulators erforscht, die sich von der vorgestellten Methodik unterschieden.
Das vielversprechendste, alternative Verfahren war mit starkem Bezug auf die inverse Kinematik des Roboters, und die Anpassung dieser zur Erfüllung und der Optimierung der gewünschten Aufgaben, indem man keine Rücktransformation von der Nozzle durchgeführt hat, sondern die Kinematik des Roboters mit dem Pfad durch eine Verkettung mit der Trajektorie erweitert.

$$\mathbf{T}_E^0(\mathbf{q}) = \mathbf{T}_1^0(q_1)\mathbf{T}_2^1(q_2)\mathbf{T}_3^2(q_3)\mathbf{T}_4^3(q_4)\mathbf{T}_5^4(q_5)\mathbf{T}_6^5(q_6)\mathbf{T}_{Trajektorie}^6(t) \tag{5.1}$$

Der inverse Kinematik-Controller muss dann zur Ausführung der Fertigung den Manipulator so ausrichten, dass $\mathbf{T}_E^0$ auf $\mathbf{T}_{Nozzle}$ liegt.

Dies barg den Vorteil, dass man die Orientierung im Arbeitspunkt mit Hilfe eines virtuellen Gelenks offenlassen konnte, was dem Controller die Gelegenheit bietet, diese Redundanz für Optimierungsfunktionen nutzbar zu machen.
Eine derartige Erweiterung sähe so aus:

$$\mathbf{T}_E^0(\mathbf{q}) = \mathbf{T}_1^0(q_1)\mathbf{T}_2^1(q_2)\mathbf{T}_3^2(q_3)\mathbf{T}_4^3(q_4)\mathbf{T}_5^4(q_5)\mathbf{T}_6^5(q_6)\mathbf{T}_{Trajektorie}^6(t)\mathbf{T}_{virtuell}^{Trajektorie}(q_{7,virtuell}) \tag{5.2}$$

mit der außerdem benötigten Erweiterung der geometrischen Jacobimatrix:

$$\mathbf{J}_G(\mathbf{q}) = \begin{pmatrix} \mathbf{J}_{P1} & \cdots & \mathbf{J}_{P6} & \mathbf{J}_{P7,virtuell} \\ \mathbf{J}_{O1} & \cdots & \mathbf{J}_{O6} & \mathbf{J}_{O7,virtuell} \end{pmatrix} \tag{5.3}$$

wobei das virtuelle siebte Gelenk in der Kinematik keinen translatorischen Einfluss hat, sondern lediglich die Orientierung um die Z-Achse von $\mathbf{T}_E^0$ beeinflusst.
Ignoriert man das virtuelle Gelenk im weiteren Verlauf, so wird man feststellen können, dass sich $\mathbf{T}_E^0$ zwar in das Ziel bewegt und beide Z-Achsen Kollinear aufeinander liegen, die Orientierung um die Z-Achse allerdings undefiniert bleibt. Dies kann bei der Fertigung, vor allem mit den angesprochenen Optimierungsmechanismen durch die Redundanz, vorteilhaft sein.
Da dieses Vorgehen den Prozess allerdings zu einer geschlossenen Kette macht und in ersten Implementierungsversuchen Instabilitäten und Probleme hervorbrachte, wurde im Rahmen dieser Projektarbeit auf eine Weiterverfolgung dieses Ansatzes verzichtet, wobei dieser Gedankengang als Thematik einer anderen Arbeit vorgeschlagen werden kann.

Anhang

[1] Spezifikation Kr Agilus Sixx
URL: *www.kuka.com > media > imported > spez_ kr_ agilus_ sixx_ de*

[2] e-Series von Universal Robots.
URL: *www.universal-robots.com/media/1802621/de-e_ series_ brochure_ german.pdf*

[3] Genkotsu-Robot 3 High Speed Picking and Assembly Robot Fanuc Robot M-3iA
URL: *www.fanucamerica.com > m-3ia-series_ 168*

[4] Siciliano, B. (2009). *Robotics: Modelling, Planning and Control*

[5] Karpfinger, C. (2017). *Höhere Mahematik in Rezepten: Begriffe, Sätze und zahlreiche Beispiele in kurzen Lerneinheiten*

[6] Niggemann, H. (2016). *Heidelberger Druck - Entwicklung des kinematischen Systems einer Digitaldruckmaschine zur Dekoration beliebig gekrümmter Oberflächen*