



Hochschule für Angewandte Wissenschaften Hamburg  
*Hamburg University of Applied Sciences*

# **Bachelorarbeit**

Julius Schultz

## **Trajektorienplanung für kooperative Industrieroboter**

*Fakultät Technik und Informatik  
Department Maschinenbau und Produktion*

*Faculty of Engineering and Computer Science  
Department of Mechanical Engineering and  
Production Management*

**Julius Schultz**  
**Trajektorienplanung für kooperative**  
**Industrieroboter**

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Maschinenbau, Entwicklung und Konstruktion  
am Department Maschinenbau und Produktion  
der Fakultät Technik und Informatik  
der Hochschule für Angewandte Wissenschaften Hamburg

Erstprüfer/in: Herr Prof. Dr.-Ing. Frischgesell

Zweitprüfer/in: Frau Dipl.-Ing. Bohnert

Abgabedatum: 31.08.2016

## **Zusammenfassung**

**Julius Schultz**

### **Thema der Bachelorthesis**

Trajektorienplanung für kooperative Industrieroboter

### **Stichworte**

Online Trajektorienplanung, Trajektorienadaption, Bahnsteuerung, Modellierung Sechachs-Industrieroboter, Echtzeitsimulation, direkte/indirekte Kinematik, KUKA Agilus KR sixx, Matlab/Simulink, SimMechanics 2. Gen, Stateflow-Zustandsautomat, Hokuyo Laserscanner, Abstandsüberwachung

### **Kurzzusammenfassung**

Im Rahmen dieser Bachelor-Arbeit wird ein 6-Achs Industrieroboter in Matlab/Simulink modelliert und mittels CAD Daten visualisiert. Das Modell wertet über einen Laserscanner Umgebungsdaten aus und nimmt direkt Einfluss auf die Echtzeitsimulation. So können Sicherheitskonzepte, wie zum Beispiel eine Geschwindigkeitsreduktion bei Zonenverletzung, simuliert und für die Anwendung geprüft werden.

**Julius Schultz**

### **Title of the paper**

Trajectory planning for cooperative industrial robots

### **Keywords**

Online trajectory planning, trajectory adaptation, modelling of a six axis industrial robot, real-time simulation, direct/inverse kinematics, KUKA Agilus KR six, Matlab/Simulink, SimMechanics 2. Gen, Stateflow, Hokuyo laserscanner, distance-control

### **Abstract**

In the framework of this Bachelor-thesis a six axis industrial robot is simulated in Matlab/Simulink and visualised with CAD data. The model receives and evaluates distance-data from a laser scanner and directly takes impact on the real-time simulation. The presented model enables to simulate and investigate security-concepts, for instance velocity reduction as a consequence of distance measurements.

# Inhalt

<b>1</b>	<b>Einleitung .....</b>	<b>0</b>
1.1	Motivation .....	0
1.2	Aufgabenstellung.....	1
1.3	Aufbau der Arbeit.....	2
<b>2</b>	<b>Theoretische Grundlagen .....</b>	<b>3</b>
2.1	Homogene Matrizen .....	3
2.1.1	Bestimmung der Orientierungsmatrix .....	4
2.2	Kinematische Struktur .....	4
2.2.1	Denervit-Hartenberg Transformation .....	4
<b>3</b>	<b>Modellierung .....</b>	<b>6</b>
3.1	KUKA Agilus KR6 sixx.....	6
3.1.1	Arbeitsraum und zulässige Achswinkel .....	7
3.1.2	Roboterkonfigurationen.....	7
3.2	Modellierung des Roboters in Matlab/Simulink .....	9
3.3	Direkte Kinematik und Validierung des Modells.....	12
<b>4</b>	<b>Inverse Kinematik KUKA Agilus KR6 .....</b>	<b>14</b>
4.1	Geometrisch, analytisches Verfahren .....	14
4.1.1	Lösung für $q_1$ .....	15
4.1.2	Lösung für $q_2$ .....	16
4.1.3	Lösung für $q_3$ .....	18
4.1.4	Lösung für $q_4 - q_6$ .....	19
4.2	Numerisches Verfahren .....	20
<b>5</b>	<b>Bahnsteuerung .....</b>	<b>22</b>
5.1	Trajektorie .....	22
5.2	Geschwindigkeitsprofil.....	23
5.3	Implementierung der Bahnsteuerung ins Simulationsmodell.....	24
5.3.1	Initialisierung.....	24
5.3.2	Bahngenerator.....	26
5.3.3	Interpolation .....	27
<b>6</b>	<b>Simulationsmodell eines kooperativen Roboters .....</b>	<b>28</b>
6.1	Literaturrecherche.....	28
6.2	Aufbau und Anforderungen des Simulink-Modells .....	29
6.3	Multitask und Echtzeit-Synchronisation .....	31
6.4	Stateflow-Zustandsautomat .....	32

---

6.4.1	Abstandsüberwachung.....	33
6.4.2	Robotersteuerung.....	35
6.5	Generieren von Abstandsdaten.....	38
6.5.1	Laserscanner.....	38
6.5.2	Schnittstelle zu Simulink.....	38
6.5.3	Auswerten der Daten.....	40
6.6	Starten des Modells.....	43
6.7	Testen des Modells.....	44
<b>7</b>	<b>Ausblick.....</b>	<b>48</b>
	<b>Literaturverzeichnis.....</b>	<b>49</b>
	<b>Anhang A.....</b>	<b>51</b>
A.1	Formeln und Funktionen zur Berechnung des Polynom- Geschwindigkeitsprofils.....	51
A.2	Übersicht der Skripte und Funktionen.....	54
A.3	Matlab Programme.....	57

## Abbildungsverzeichnis

<i>Abbildung 3-1: KUKA Agilus KR6 sixx</i> .....	6
<i>Abbildung 3-2: Arbeitsraum des Roboters</i> .....	7
<i>Abbildung 3-3: SimMechanics Modell eines Doppelpendels</i> .....	9
<i>Abbildung 3-4: Animation des modellierten Doppelpendels</i> .....	10
<i>Abbildung 3-5: Animation des modellierten Roboters in Simulink</i> .....	11
<i>Abbildung 3-6: SimMechanics Blockschaltbild des modellierten KUKA Agilus</i> ..	11
<i>Abbildung 3-7: Koordinatensysteme nach DH-Notation [Frischgesell 2016]</i> .....	12
<i>Abbildung 4-2: Gelenkarmroboter mit Zentralhand</i> .....	15
<i>Abbildung 4-1: Bestimmen des Handwurzelpunktes <math>H</math></i> .....	15
<i>Abbildung 4-3: Winkeldefinition der atan2 Funktion</i> .....	15
<i>Abbildung 4-4: Gelenkwinkel <math>q_1</math></i> .....	16
<i>Abbildung 4-5: geom. Konstruktion für die Bestimmung des Gelenkwinkels <math>q_2</math></i> ..	16
<i>Abbildung 4-6: Winkel <math>q_2</math>, gebeugte Konfiguration</i> .....	17
<i>Abbildung 4-7: Winkel <math>q_2</math>, überstreckte Konfiguration</i> .....	17
<i>Abbildung 4-8: Winkel <math>q_3</math>, überstreckte Konfiguration</i> .....	18
<i>Abbildung 4-9: Winkel <math>q_3</math>, gebeugte Konfiguration</i> .....	18
<i>Abbildung 4-10: inverse Kinematik mittels Jacobi-Matrix</i> .....	21
<i>Abbildung 5-1: asynchrone, synchrone und vollsynchrone Bahnfahrt</i> .....	23
<i>Abbildung 5-2: Ablaufdiagramm: Bahnsteuerung</i> .....	24
<i>Abbildung 5-3: Baumdiagramm Initialisierung Teil I</i> .....	25
<i>Abbildung 5-4: Baumdiagramm Initialisierung Teil II</i> .....	26
<i>Abbildung 5-5: Komponenten der Bahnkoordinate <math>s(t)</math> bei vollsync. PTP Fahrt</i> ..	27
<i>Abbildung 6-1: Erste Ebene des Simulink Modells „AdvancedModell.slx“</i> .....	29
<i>Abbildung 6-2: Flussdiagramm Basis-Modell</i> .....	30
<i>Abbildung 6-3: Flussdiagramm Advanced-Modell</i> .....	31
<i>Abbildung 6-4: Stateflow-Zustandsautomat</i> .....	32
<i>Abbildung 6-5: Parallele Aufgabenstruktur im Stateflow-Zustandsautomaten</i> .....	33
<i>Abbildung 6-6: Stateflow Ablaufdiagramm der Distanzüberwachung</i> .....	34
<i>Abbildung 6-7: Stateflow Ablaufdiagramm der Robotersteuerung</i> .....	35
<i>Abbildung 6-8: Stateflow Transition Bedingung für Wendepunkt</i> .....	36
<i>Abbildung 6-9: Stateflow Transition Bedingung für Event</i> .....	36
<i>Abbildung 6-10: Messbereich Hokuyo Laserscanner</i> .....	38
<i>Abbildung 6-11: Hokuyo Laserscanner</i> .....	38
<i>Abbildung 6-12: Aufbau der Schnittstelle Laserscanner und Simulink</i> .....	39
<i>Abbildung 6-13: Simulink-Programm zur Auswertung der Laserscan-Daten</i> .....	40
<i>Abbildung 6-14: Zeitverlauf eines Messwertes des Laserscans</i> .....	41
<i>Abbildung 6-15: Abstandswerte eines sich nähernden Menschen</i> .....	42
<i>Abbildung 6-16: Laserscanner Tool: UrgBenriStandard</i> .....	43
<i>Abbildung 6-17: Matlab: Inputbereich des Initialisierungsskriptes</i> .....	44

<i>Abbildung 6-18: Simulation mit Kontroll-Submodell .....</i>	45
<i>Abbildung 6-19: Geschwindigkeitsreduktion um eine Stufe .....</i>	46
<i>Abbildung 6-20: Geschwindigkeitsreduktion um 3 Stufen in einem Zyklus.....</i>	46
<i>Abbildung 6-21: Auswertung bei sich bewegenden Objekten.....</i>	47
<i>Abbildung 0-1: Polynom Geschwindigkeitsprofil .....</i>	53

## **Tabellenverzeichnis**

<i>Tabelle 3-1: Kenngrößen des Roboters .....</i>	<i>7</i>
<i>Tabelle 3-2: Mögliche Konfigurationen des Roboters .....</i>	<i>8</i>
<i>Tabelle 3-3: DH Parameter für den KUKA Agilus KR6.....</i>	<i>12</i>



## Formelzeichenverzeichnis

$X$	Lage des Roboters in Umweltkoordinaten
$HM$	Homogene Matrix
$R$	Orientierungsmatrix
$q_i$	Gelenkwinkel
$\theta$	DH-Parameter, Rotation um Z-Achse
$d$	DH-Parameter, Translation in Z-Richtung
$\alpha$	DH-Parameter, Rotation um X-Achse
$a$	DH-Parameter, Translation in X-Richtung
$I_H$	Index für Handgelenkskonfiguration
$I_{UA}$	Index für Unterarmkonfiguration
$I_F$	Index für Fußkonfiguration
$A_{ij}$	Transformationsmatrix
$\vec{H}$	Handwurzelpunkt
$L_F$	Höhe des Fußes
$O_{F/UA}$	Offset Fuß/Oberarm
$L_{OA}$	Länge Oberarm
$O_{O/UA}$	Offset Oberarm/Unterarm
$L_{UA}$	Länge Unterarm
$L_{05z}$	Länge letztes Glied am Roboter
$J$	Jacobi-Matrix
$t_i$	Segmentzeiten
$t_b$	Beschleunigungszeit
$t_k$	Zeit der konstanten Fahrt
$t_v$	Verzögerungszeit
$t_{bm}$	Maximale Beschleunigungszeit
$t_{km}$	Maximale Zeit für konstante Fahrt
$t_{vm}$	Maximale Verzögerungszeit
$s_i$	Segmentlängen
$s_g$	Gesamtlänge
$s_b$	Beschleunigungsweg
$s_k$	Weg der konstanten Fahrt
$s_v$	Verzögerungsweg
$v_m$	Maximale Gelenkgeschwindigkeiten
$a_m$	Maximale Gelenkbeschleunigungen
$v_R$	Richtgeschwindigkeit
$q_{Koeff}$	Koeffizienten einer Polynomfunktion
$v_F$	Geschwindigkeitsfaktor

# 1 Einleitung

## 1.1 Motivation

Industrieroboter werden in diversen Branchen eingesetzt und in den letzten Jahren ist die Zahl der installierten Industrieroboter stark gestiegen. So sind zum Beispiel Montage-, Bestückungs-, Verpackungs- oder Schweißroboter typische Anwendungsbeispiele.

Industrieroboter haben sich bislang durch schwere und steife Konstruktionen mit hoher Präzision ausgezeichnet. Die Entwicklung von leistungsstarken Elektromotoren ermöglichten trotzdem schnelle Bewegungen und somit geringe Zykluszeiten. Um das sogenannte „Asimov’s 1st Law“: „Ein Roboter darf einen Menschen niemals verletzen“ zu gewährleisten, wurde bislang auf Sicherheitskonzepte gesetzt, die Roboter in einer abgesicherten Laborumgebung arbeiten lassen.

Die Forschung in den letzten Jahren zielt immer mehr auf ein interaktives Arbeiten von Mensch und Roboter ab. Die Vision von interagierenden Robotern, die in einem gemeinsamen Arbeitsraum mit Menschen arbeiten, wird vor allem durch die rasante Weiterentwicklung in der Sensorik, der erhöhten Rechenkapazitäten und neuer Regelkonzepte gestützt. Ein weiterer Ansatz sind hier auch Leichtbau-Roboter oder Roboter mit adaptiver Steifigkeit, die unfallfreies Kooperieren mit Menschen ermöglichen sollen. Kooperative Roboter sollen neben industriellen Anwendungsbereichen auch in Alltagsbereichen, wie zum Beispiel in der Haushaltshilfe oder im Gesundheitsbereich eingesetzt werden.

Ein erster Schritt in Richtung eines kooperativen Roboters ist es, wenn Roboter ihre Bewegung situationsbedingt an die Umgebung anpassen. Der Schwerpunkt dieser Bachelor-Arbeit liegt daher auf der Simulation eines Sechssachs-Industrieroboters mit einer adaptiven Trajektorienplanung. Dabei wird es in dieser Arbeit neben dem Entwickeln von Algorithmen zur Neuberechnung der Trajektorie auch um die Einbindung eines Laserscanners in die Simulation gehen. [Haddadin, 2014], [Biagiotti/Melchiorri 2008]

## 1.2 Aufgabenstellung

Im Zentrum für industrielle Robotik werden KUKA Agilus KR6 Roboter mit KRC4 Steuerung verwendet. Zur Simulation dient die Software KUKA SimPro 2.2. Mit dieser gelingt die Simulation von geplanten Trajektorien nur in der Form von PTP, Linear oder Spline. Ein direkter Eingriff in die Trajektorienplanung ist bisher nicht vorgesehen.

Zur alternativen Modellierung und Erweiterung um online geplante Trajektorien sollen mit der Software Matlab / Simulink / SimMechanics 2. Generation für den oben genannten Robotertyp Simulationsmodelle aufgebaut werden. In den Modellen werden Signale, die die Annäherung von Hindernissen oder Personen registrieren, aufgenommen und für eine neue Trajektorienplanung verwendet. Damit wird es möglich, sich dem Roboter zu nähern bzw. Hindernisse im Arbeitsraum zu bewegen und die Trajektorie online unmittelbar zu verändern.

Im ersten Schritt ist der Kuka Roboter zu modellieren und mittels direkter und indirekter Kinematik zu simulieren. Dazu sind analytische und numerische Lösungen mittels Jacobi Matrix zu erstellen. Die Ergebnisse bzgl. Pose und Figur sind mit dem im Labor vorhandenen Roboter abzugleichen. Die CAD Daten werden entsprechend konvertiert und in der SimMechanics Software implementiert, so dass der Roboter in 3D dargestellt wird.

Anschließend werden als Trajektorien klassische Trapezprofile für die Geschwindigkeit sowie Splines implementiert. Die entsprechenden Funktionen werden angelehnt an die KUKA Sim Pro Software parametrisiert. Für alle Interpolationsarten ist eine vollsynchroner Bahn vorzusehen.

Möglichkeiten der Trajektorienadaption sollen in der Literatur eruiert werden. Als externe Sensorik wird ein Linienscanner verwendet. Die Auswertung führt zu Distanzwerten, die für eine Neuberechnung der Trajektorie verwendet werden sollen. In diese neue Trajektorie soll ohne Stopp eingefahren werden.

Für die Vermeidung von Kollisionen sind Konzepte wie Zonenverletzung, Geschwindigkeitsreduzierung oder Neurechnung der Trajektorie zu entwickeln, die später direkt auf den KUKA übertragen werden können.

Optional: Dazu soll die Simulation aktuelle Positionswerte an die Kuka Sim Pro Software übergeben, so dass eine synchrone Simulation möglich wird.

### 1.3 Aufbau der Arbeit

Die Arbeit gliedert sich in sechs Kapitel. Die ersten 4 Kapitel beinhalten allgemeine Informationen zum Thema, während die Kapitel 5 und 6 die konkret entwickelten Simulationsmodelle vorstellen.

Nach der Einleitung befasst sich das zweite Kapitel mit einigen ausgewählten theoretischen Grundlagen, die in dieser Arbeit und der Robotertechnik eine große Bedeutung haben. Dazu gehören homogene Matrizen, der Aufbau einer kinematischen Struktur und die Denavit-Hartenberg Konvention.

Das dritte Kapitel stellt den simulierten KUKA Agilus KR6 900 Roboter vor und erläutert, wie der Roboter in der Simulink SimMechanics Umgebung modelliert wird. Das vorgestellte Modell kann die Stellung des Roboters unter Vorgabe der sechs Gelenkwinkel simulieren und mittels CAD Daten grafisch darstellen.

Im vierten Kapitel wird ein geometrisches Verfahren zur Ermittlung der inversen Kinematik erklärt. Die inverse Kinematik ist ein unabdingbares Werkzeug für die Bahnplanung, da sie es ermöglicht, Umweltkoordinaten in Roboterkoordinaten umzuwandeln. Im Rahmen dieser Arbeit wurde sowohl eine analytische, als auch eine numerische Variante erstellt. Es wird allerdings hauptsächlich auf die geometrisch/analytische Variante eingegangen, da diese auch im weiteren Verlauf genutzt wird.

Mit dem Modellierungsmodell aus Kapitel 3 und der inversen Kinematik aus Kapitel 4 kann eine erste einfache Bahnsteuerung vorgenommen werden. Im Kapitel 5 werden dafür die Grundlegenden Begriffe der Trajektorie und des Geschwindigkeitsprofils erläutert. Mit der in diesem Kapitel vorgestellten Bahnsteuerung kann eine einfache Trajektorie von A nach B in definiertem Geschwindigkeitsprofil (Trapez oder Polynom) und definierter Verfahrrart (PTP oder linear) abgefahren werden.

Das letzte Kapitel stellt das Simulationsmodell für einen kooperativen Roboter vor. Im Gegensatz zum im Kapitel 5 vorgestellten Basis-Modell wird dieses Modell mit einer komplexen Bahnsteuerung erweitert. Das Modell kann über einen Laserscanner mit der Umwelt interagieren und soll so ein Sicherheitskonzept in Bezug auf Mensch-Roboter Interaktion untersuchen.

Im Anhang werden die einzelnen Funktionen der Modelle „BasisModell (Kapitel 5)“ und „AdvancedModell (Kapitel 6)“ noch einmal detaillierter erläutert.

## 2 Theoretische Grundlagen

Im folgenden Kapitel werden ausgewählte mathematische Grundlagen zur Robotertechnik noch einmal genauer erläutert, um ein besseres Verständnis zu gewährleisten. Da nur eine Auswahl von Aspekten behandelt werden kann, sei an dieser Stelle schon einmal auf die Literatur von [Stark 2009] und [Corke 2013] verwiesen.

### 2.1 Homogene Matrizen

Das Konzept der homogenen Matrix (HM) hat in der Robotik eine große Bedeutung, da hiermit Koordinatentransformationen wie der Translation oder Rotation eines Koordinatensystems zu einem Referenzkoordinatensystem dargestellt werden [Stark 2009].

Die Grundlegende Gleichung einer homogenen Transformation ist dabei wie folgt definiert:

$$\vec{X}_0 = HM \cdot \vec{X}_1 \quad 2-1$$

Mit dieser Gleichung kann der Vektor  $X_1$ , definiert im kartesischen Koordinatensystem  $K_1$ , im Koordinatensystem  $K_0$  abgebildet werden: Man erhält  $X_0$ .

Die homogene Matrix ist eine 4x4 Matrix mit der Form

$$HM = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad 2-2$$

und kann wie folgt interpretiert werden:

1. Spalte (Eintrag 1-3): x-Vektor von  $K_1$  dargestellt in  $K_0$
  2. Spalte (Eintrag 1-3): y-Vektor von  $K_1$  dargestellt in  $K_0$
  3. Spalte (Eintrag 1-3): z-Vektor von  $K_1$  dargestellt in  $K_0$
  4. Spalte (Eintrag 1-3): Ursprungsvektor von  $K_1$  dargestellt in  $K_0$
4. Zeile: immer der Zeilenvektor [ 0 0 0 1 ]

Da die Homogene Matrix eine 4x4 Matrix ist müssen auch die Vektoren  $X_0$  und  $X_1$  um eine Zeile auf einen 4x1 Vektor erweitert werden. Der Eintrag in der 4. Zeile darf dabei nur 1 oder 0 sein. Wird für den Eintrag eine 1 gewählt erhält man den Ortsvektor von  $X_0$ , da nun die 4. Spalte der HM berücksichtigt wird. Wird der Eintrag dagegen auf 0 gesetzt, erhält man den freien Vektor  $X_0$ .

Homogenen Matrizen werden durch eine Kombination aus Translationen und Elementardrehungen definiert und durch Multiplikation der mathematischen Ausdrücke dieser

Transformationen erstellt. Ein einheitlich definiertes Schema bietet hierbei das Kinematikmodell nach Denavit-Hartenberg, auf welches im folgenden Kapitel eingegangen wird.

### 2.1.1 Bestimmung der Orientierungsmatrix

Die Orientierung eines Koordinatensystems wird in einer homogenen Matrix in einer 3x3 Matrix dargestellt. Da die Orientierung drei Freiheitsgrade im Raum besitzt, ist eine andere Möglichkeit die Orientierung durch drei Eulerwinkel darzustellen. Dabei wird ein Referenzkoordinatensystem nacheinander um drei definierte Achsen gedreht [Stark 2009]. Es ist zu beachten, dass es sich bei den Achsen der zweiten und dritten Transformation immer um die bereits transformierten Achsen handelt.

Zwischen Rotationsmatrix und Eulerwinkeln lässt sich also folgender Zusammenhang feststellen:

$$R = [\vec{n} \vec{o} \vec{a}] = \text{Rot}_z(\alpha) \cdot \text{Rot}_y(\beta) \cdot \text{Rot}_z(\gamma) \quad 2-3$$

Die Matrizen *Rot* sind dabei definierte homogene Matrizen für eine Elementardrehung um den jeweiligen Winkel.

## 2.2 Kinematische Struktur

Ein Roboter kann abstrakt betrachtet als ein System aus starren Körpern, die mit Gelenken verbunden sind, betrachtet werden. Die Gelenke sind durch ihren Freiheitsgrad definiert und beschreiben eine Art der möglichen Relativbewegung zwischen zwei Körpern. Anzahl und Anordnung der Gelenke mit ihren jeweiligen Freiheitsgraden sorgen für den Gesamtfreiheitsgrad eines Roboters [Stark 2009]. Im Falle des KUKA Agilus Roboters bilden die 6 Drehgelenke mit den starren Verbindungen eine offene kinematische Kette mit 6 Freiheitsgraden.

### 2.2.1 Denervit-Hartenberg Transformation

Eine mathematische Darstellung einer kinematischen Kette lässt sich mit Hilfe der Denervit-Hartenberg Transformation (DH-Transformation) beschreiben. Das Verfahren beschreibt die relative Transformation zwischen den einzelnen Koordinatensystemen mit Hilfe von homogenen Matrizen [Stark 2009]. Nach Anwendung des Verfahrens ist es möglich die Lage des Endeffektors im Basiskoordinatensystem abhängig von den Gelenkwinkeln  $q_i$  darzustellen. Folgende Konventionen beim Festlegen der Koordinatensysteme müssen bei der Denervit-Hartenberg Konvention eingehalten werden:

- Bewegungsachsen  $A_i$  ( $i = 0 \dots n$ ) festlegen
- Lage der  $z_i$  auf den Bewegungsachsen festlegen
- $x_i$  auf gemeinsamer Normalen von  $z_i$  und  $z_{i+1}$  festlegen

- $y_i$  nach Rechtssystem festlegen

Diese Definitionsweise ermöglicht es, die Transformation zwischen den einzelnen Koordinatensystemen auf vier standardisierte Transformationen festzulegen.

Nach jeder Transformation erhält das neue Koordinatensystem einen  $\prime$  als Index. Die vier Transformationen ergeben sich nach [Stark 2009] :

1. Rotation des Koordinatensystems  $K_{i-1}$  um die eigene z-Achse, um den Winkel  $\theta$  bis  $x_{i-1}'$  parallel zu  $x_i$  ist.
2. Translation entlang  $z_{i-1}'$  um den Betrag  $d_i$
3. Translation entlang  $x_{i-1}''$  um den Betrag  $a_i$
4. Rotation um  $x_{i-1}'''$  um den Winkel  $\alpha_i$  bis  $z_{i-1}''''$  mit  $z_i$  übereinstimmt.

Die ermittelten DH-Parameter werden dann in die homogenen Matrizen für die Elementartransformationen eingesetzt, sodass sich die Gesamttransformationsmatrix ergibt.

$$DH_{i-1,i}(\theta_i, d_i, a_i, \alpha_i) = Rot(z, \theta_i) \cdot Trans(z, d_i) \cdot Trans(x, a_i) \cdot Rot(x, \alpha_i) \quad 2-4$$

$$DH_{i-1,i}(\theta_i, d_i, a_i, \alpha_i) = \begin{bmatrix} \cos \theta_i & -\cos \alpha_i \sin \theta_i & \sin \alpha_i \sin \theta_i & a_i \cos \theta_i \\ \sin \theta_i & \cos \alpha_i \cos \theta_i & -\sin \alpha_i \cos \theta_i & a_i \sin \theta_i \\ 0 & \sin \alpha_i & \cos \alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad 2-5$$

## 3 Modellierung

In diesem Kapitel wird zuerst der modellierte Roboter und dessen Eigenschaften beschrieben. Danach wird erläutert, wie der Roboter in der Matlab/Simulink Toolbox „SimMechanics“ modelliert und visualisiert wird.

### 3.1 KUKA Agilus KR6 sixx

KUKA ist eines der führenden Unternehmen im Bereich der Robotertechnik und hat seinen Hauptsitz in Augsburg.

Mit der KUKA AGILUS Serie präsentiert KUKA eine Roboterfamilie mit verhältnismäßig kleinen Robotern mit einer Tragkraft von 6 bis 10 kg. Die Roboter zeichnen sich durch eine sehr hohe Geschwindigkeit und kurze Zykluszeiten aus. Alle KUKA Agilus Modelle sind entweder als 5-Achs oder 6-Achs Roboter ausgelegt und können auch in anspruchsvollen Umgebungen (z.B. Decken- oder Wandmontage) betrieben werden.

Außerdem zeichnet sich die AGILUS Familie durch Ihre Sicherheitsfunktionen aus, die eine Kooperation von Mensch und Maschine noch besser machen soll [Datenblatt KUKA 2016].

In dieser Bachelor-Arbeit wird der KUKA AGILUS KR6 R900 sixx simuliert.



Abbildung 3-1: KUKA Agilus KR6 sixx



### 3.1.1 Arbeitsraum und zulässige Achswinkel

Die KUKA Agilus Serie bietet eine große Work-Envelope. So können vor allem auch Punkte dicht am Fuß erreicht werden:

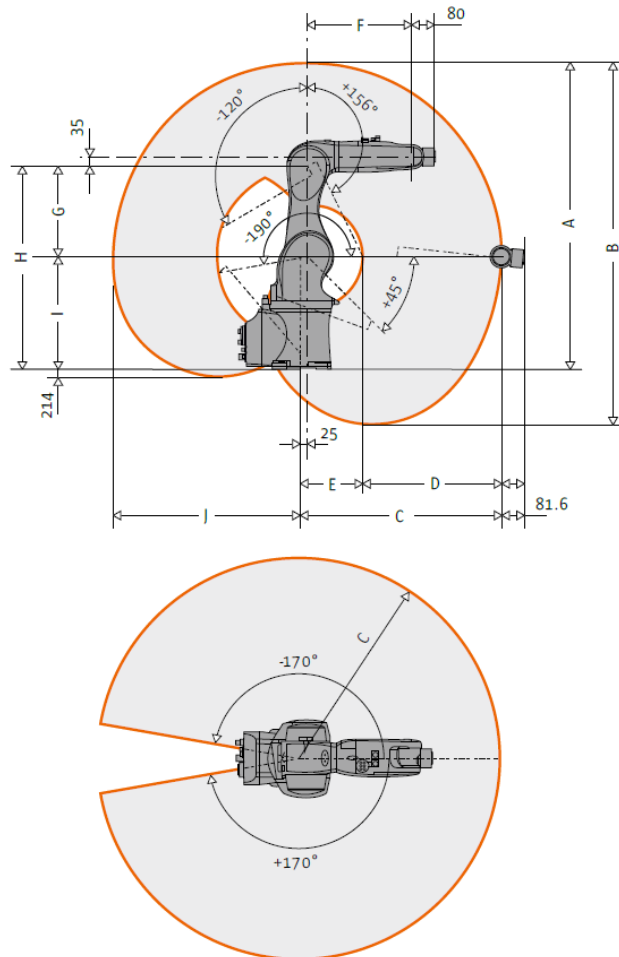


Abbildung 3-2: Arbeitsraum des Roboters

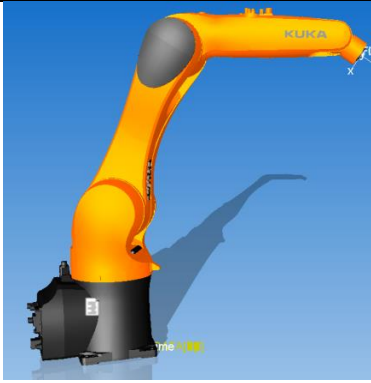
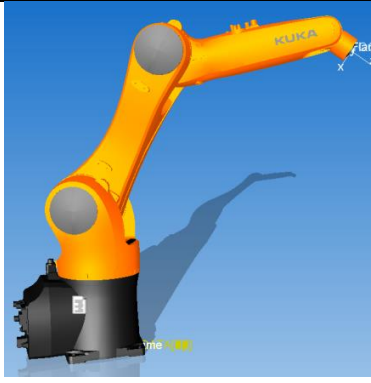
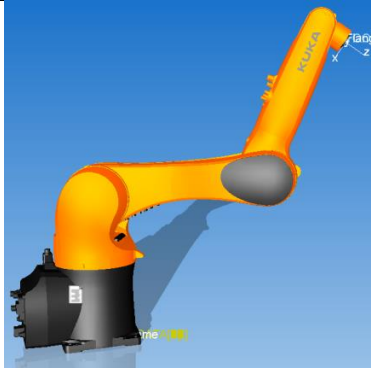
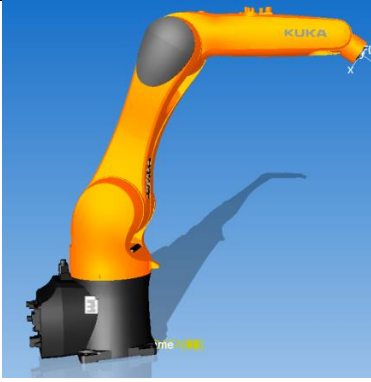
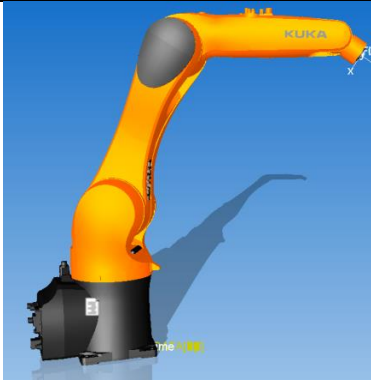

Tabelle 3-1: Kenngrößen des Roboters

Kenngrößen	
Dimension A	1276 mm
Dimension B	1620 mm
Dimension C	901,5 mm
Dimension D	656 mm
Dimension E	245,5 mm
Dimension F	851,5 mm
Dimension G	420 mm
Dimension H	455 mm
Dimension I	400 mm
Dimension J	855 mm
Zulässige Achswinkel	
Achse 1	$\pm 170^\circ$
Achse 2	$+45^\circ/-190^\circ$
Achse 3	$+165^\circ/-120^\circ$
Achse 4	$\pm 185^\circ$
Achse 5	$\pm 120^\circ$
Achse 6	$\pm 350^\circ$

### 3.1.2 Roboterkonfigurationen

Der kinematische Aufbau des KUKA Agilus KR6 lässt bei gegebener Position und Orientierung für den Endeffektor verschiedene Konfigurationen der Gelenkwinkel zu. Um bei der inversen Kinematik eine eindeutige Zuordnung von Umweltkoordinaten zu Roboterkoordinaten zu ermöglichen ist es notwendig alle möglichen Konfigurationen zu erfassen und mit Indizes eindeutig zu beschreiben. Diese sollen im Folgenden geometrisch verdeutlicht werden. Die Konfigurationen können mit drei Körpern des Roboters beschrieben werden, die jeweils zwei unterschiedliche Posen einnehmen können.

Tabelle 3-2: Mögliche Konfigurationen des Roboters

Fuß		
Beschreibung	Fuß in Richtung Endeffektor ausgerichtet	Fuß um 180° verdreht
Index	0	1
		
Unterarm		
Beschreibung	Unterarm ist überstreckt	Unterarm ist gebeugt
Index	0	1
		
Handgelenk		
Beschreibung	Handgelenk ist nach unten geklappt	Handgelenk ist nach oben geklappt
Index	0	1
		

Alle möglichen Konfigurationen lassen sich über ein Index Array mit drei Elementen für Fuß, Unterarm und Handgelenk beschreiben:

$$Konf = [I_H, I_{UA}, I_F] \quad 3-1$$

Bei drei Parametern mit jeweils zwei Möglichkeiten ergeben sich 8 verschiedene Konfigurationen, mit denen dieselbe Endeffektorposition und -orientierung erreicht werden kann.

### 3.2 Modellierung des Roboters in Matlab/Simulink

Für die graphische Simulation der Roboterbewegung ist es notwendig, den Roboter in Matlab/Simulink zu modellieren. Hierfür wird die „SimMechanics“ Bibliothek der 2. Generation verwendet.

Diese Simulationsumgebung ist extra für die Mehrkörpersimulation entwickelt worden. Mit Blöcken wie bodies, joints, constraints, force und sensor Elementen werden MKS Modelle aufgebaut. Auf Grundlage dieser Modelle erstellt SimMechanics Bewegungsgleichungen, die numerisch gelöst werden. Außerdem ist es möglich, Massen und Trägheiten für kinematische Untersuchungen zu implementieren, sowie CAD Dateien für einzelne Body-Elemente hinzuzufügen. Eine automatisch erstellte 3D Animation visualisiert das dynamische Modell [MathWorks SimMechanics].

In dieser Arbeit soll die kinematische Struktur des Roboters ohne Masseneffekte simuliert werden. Dafür wird die kinematische Kette analog zur Denavit-Hartenberg Transformation mit Koordinatensystemtransformationen abgebildet. Die Drehfreiheitsgrade zwischen den Koordinatensystemen werden durch „Joint-Blöcke“ definiert. Entlang dieser kinematischen Kette werden die starren Körper, wie z.B. der Oberarm, über ein körpereigenes KS fest mit einem Koordinatensystem der kinematischen Kette verbunden. Über die „Joint-Blöcke“ können dann die Gelenke angesteuert werden und eine Relativbewegung zwischen den Körpern erzeugt werden.

Folgendes Beispiel eines einfachen Pendels soll den grundlegenden Aufbau verdeutlichen:

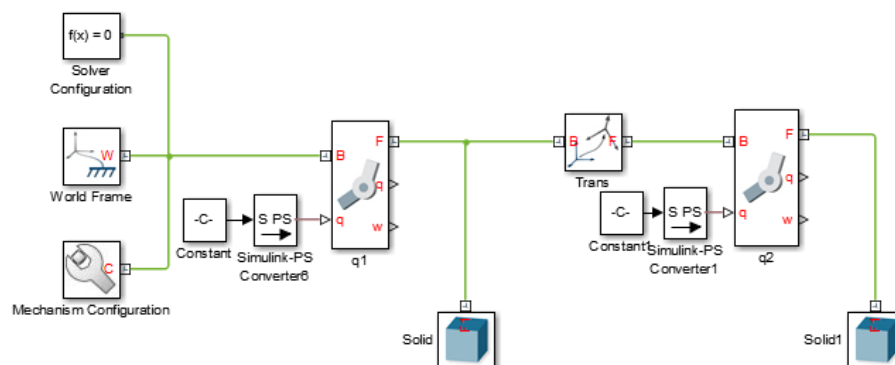


Abbildung 3-3: SimMechanics Modell eines Doppelpendels

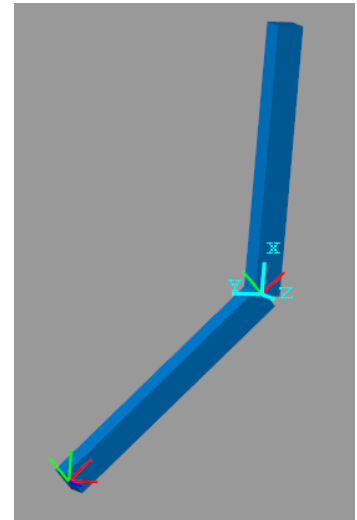
Die ersten drei Blöcke definieren das Welt-Koordinatensystem und die Richtung der Schwerkraft. Mit dem „revolute Joint“ Block „q1“ wird angegeben, dass eine relative Drehbewegung des World-Frame-KS zum Follower-KS um die gemeinsame Z-Achse möglich ist. Mit dem „Trans“ Block wird das Follower-KS des ersten Gelenks so transformiert, dass es im zweiten Gelenk liegt.

Die Solid-Blöcke definieren die Körper und deren Referenzkoordinatensysteme. In diesem Beispiel ist das Referenzkoordinatensystem des Balkens an der linken Grundseite. Dieses KS wird dann mit der kinematischen Kette verbunden, sodass es sich analog zum Follower-KS des ersten Gelenks bewegt.

Bei einer Drehung um jeweils  $45^\circ$  ergibt sich die in Abbildung 3-4 dargestellte Animation.

Unten links zu sehen ist das Welt-KS und das um  $45^\circ$  gedrehte Follower-KS auf dem analog zum zweiten Balken das Referenz-KS sitzt.

Das Follower-KS des ersten Gelenks wurde entlang der X-Achse um die Länge des ersten Balkens transformiert und dann wieder mit einem Drehgelenk mit dem Referenz-KS des zweiten Balkens (hellblaues KS) verbunden.



Analog zu diesem Beispiel wurde die Kinematische Kette des KUKA Agilus KR6 aufgebaut und visualisiert. Das Blockschaltbild und der visualisierte Roboter sind auf der folgenden Seite dargestellt:

*Abbildung 3-4:*  
Animation des  
modellierten  
Doppelpendels

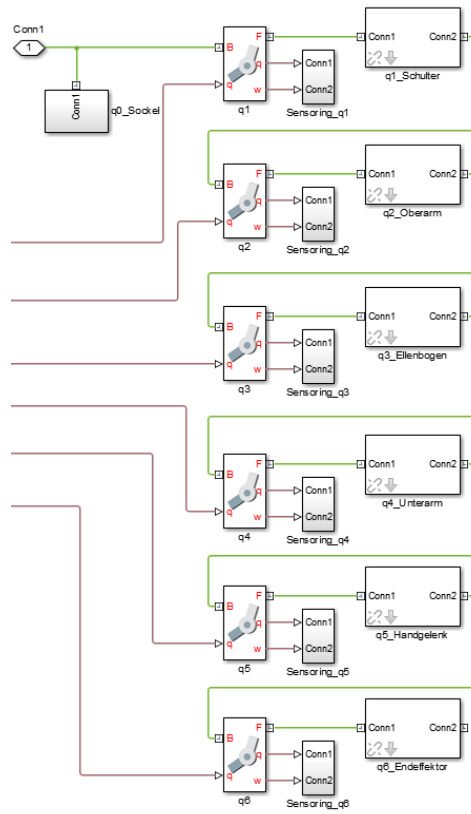


Abbildung 3-6: SimMechanics  
Blockschaltbild des modellierten  
KUKA Agilus



Abbildung 3-5: Animation des modellierten  
Roboters in Simulink

### 3.3 Direkte Kinematik und Validierung des Modells

Die direkte Kinematik gibt die Position des Endeffektors in Abhängigkeit der Gelenkwinkel an:

$$\vec{x} = f(\vec{q}) \quad 3-2$$

Mit Hilfe der Denavit-Hartenberg Transformation können homogene Matrizen, sogenannte A-Matrizen (Gleichung 2-7), erstellt werden, mit denen dann die Gesamttransformation vom Basiskoordinatensystem zum Endeffektorkoordinatensystem dargestellt werden kann.

$$A_{06} = A_{01} \cdot A_{12} \cdot A_{23} \cdot A_{34} \cdot A_{45} \cdot A_{56} \quad 3-3$$

Folgende Abbildung zeigt die Definition der Koordinatensysteme auf deren Grundlage dann die DH-Parameter definiert wurden. Die anschließende Tabelle gibt die dazugehörigen DH-Parameter an.

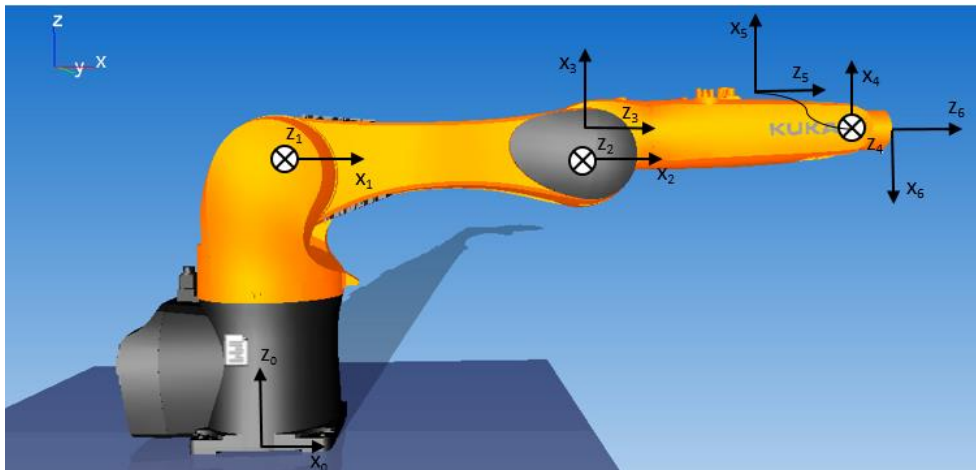


Abbildung 3-7: Koordinatensysteme nach DH-Notation [Frischgesell 2016]

Tabelle 3-3: DH Parameter für den KUKA Agilus KR6

	d	$\theta$	a	$\alpha$
0 → 1	$L_{01z}$	$-q_1$	$L_{01x}$	$-\frac{\pi}{2}$
1 → 2	0	$q_2$	$L_{12x}$	0
2 → 3	0	$q_3 - \frac{\pi}{2}$	$L_{23x}$	$-\frac{\pi}{2}$
3 → 4	$L_{34z}$	$-q_4$	0	$\frac{\pi}{2}$
4 → 5	0	$q_5$	0	$-\frac{\pi}{2}$
5 → 6	$L_{56z}$	$-q_6 + \pi$	0	0

Aus der Gesamttransformationsmatrix  $A_{06}$  lassen sich nun Position und Orientierung des Endeffektorkoordinatensystems bestimmen. Zur Bestimmung der Orientierung, angegeben in der Drehung um die raumfesten Achsen Z, Y, X, wurden folgende Gleichungen verwendet:

$$Rot_z = \text{atan2}(R(2,1), R(1,1)) \quad 3-4$$

$$Rot_y = -\text{asin}(R(3,1)) \quad 3-5$$

$$Rot_x = \text{atan2}(R(3,2), R(3,3)) \quad 3-6$$

In diesen Gleichungen entspricht die Matrix  $R$  der  $3 \times 3$  Rotationsmatrix aus der homogenen Matrix  $A_{06}$ .

Mit diesen Gleichungen wurde parallel zur Modellierung des Roboters die Position des Endeffektors über die direkte Kinematik bestimmt. In der Abbildung 3-6 ist gut zu sehen, dass der ermittelte Punkt (rot) mit der Position des Endeffektors übereinstimmt.

## 4 Inverse Kinematik KUKA Agilus KR6

In diesem Kapitel werden die Algorithmen erläutert, die die Gelenkwinkel für das Simulationsmodell liefern. In den meisten Anwendungsfällen werden dem Roboter globale Koordinaten im kartesischen Koordinatensystem vorgegeben, die dann in die roboterspezifischen Gelenkwinkel übersetzt werden müssen.

Die inverse Kinematik soll also die Gelenkwinkel  $q_i$  in Abhängigkeit von der Position und Orientierung des Endeffektors angeben:

$$q_i = f(X) \quad 4-1$$

Im Gegensatz zur direkten Kinematik ist die inverse Kinematik mehrdeutig. Vier Lösungsfälle können unterschieden werden [Stark 2009]:

1. Keine Lösung:  
X beschreibt eine Position außerhalb der Reichweite des Roboters
2. Eine Lösung:  
Durch X beschriebene Position kann nur im völlig gestreckten Zustand erreicht werden.
3. Mehrere Lösungen:  
Durch X beschriebene Position kann vom Roboter mit mehreren Kombinationen von  $q_i$  erreicht werden. Die einzelnen Posen des Roboters werden auch Roboterkonfigurationen genannt.
4. Unendlich viele Lösungen:  
Der Roboter befindet sich in einer sogenannten singulären Konfiguration. Dies ist dann der Fall, wenn zwei oder mehrere Drehachsen fluchten oder Schubachsen parallel sind.

Im Folgenden werden zwei verschiedenen Varianten zur Lösung der inversen Kinematik vorgestellt: Ein geometrisch/analytisches Verfahren und ein numerisches Verfahren. Beide Varianten können im Modell „KUKA\_Agilus\_KR6\_InDirKin.xls“ getestet werden.

### 4.1 Geometrisch, analytisches Verfahren

Um eine analytische Lösung für die inverse Kinematik zu erhalten, müsste man die Gleichungen 3-2 aus der direkten Kinematik nach den Gelenkwinkeln  $q_i$  auflösen. Dies ist aber im Allgemeinen nicht möglich, da es sich hier um nichtlineare Gleichungen handelt [Stark 2009]. Der folgende Ansatz beruht auf einem geometrischen Verfahren nach [Strak 2009] und [Lee/Ziegler 1983].

Eine Möglichkeit trotzdem eine analytische Lösung zu entwickeln ist es, die kinematische Struktur des Roboters einzuschränken. Dies ist der Fall beim Gelenkarmroboter mit Zentralhand bei dem folgende Bedingungen erfüllt sein müssen [Stark 2009]:

- Die Richtungsvektoren aller Achsen sind zueinander entweder orthogonal oder parallel.
- Die drei Handachsen schneiden sich in einem Punkt, dem Handwurzelpunkt H.

Folgende Abbildung 4-2 zeigt, dass es sich beim KUKA Agilus KR6 um einen Gelenkarmroboter mit Zentralhand handelt. Alle Achsen  $z_i$  stehen senkrecht oder parallel



zueinander und der Handwurzepunkt liegt im Gelenk  $q_5$ , wo sich die letzten drei Drehachsen  $z_3, z_4, z_5$  schneiden.

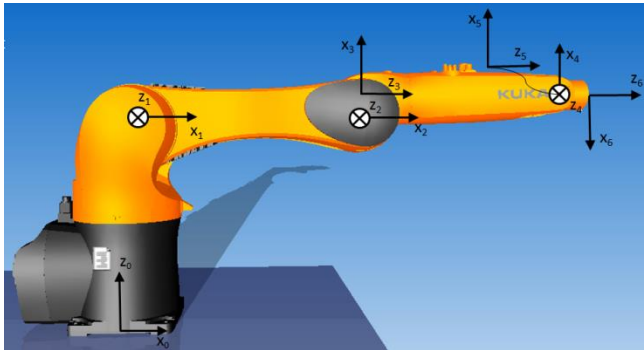


Abbildung 4-2: Gelenkarmroboter mit Zentralhand

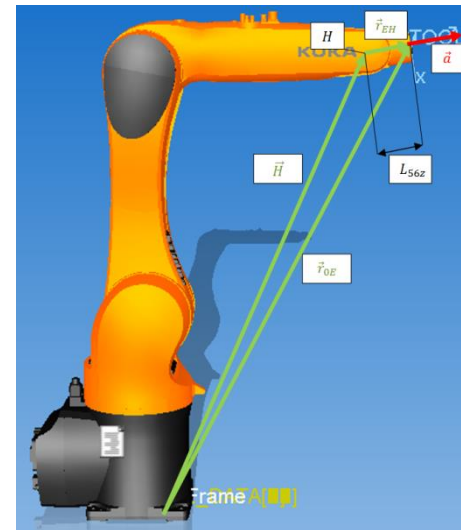


Abbildung 4-1: Bestimmen des Handwurzepunktes  $H$

Die Besonderheit dieser kinematischen Struktur ist es, dass mit den ersten drei Achsen ( $q_1 - q_3$ ) die Position des Endeffektors festgelegt werden kann. Die letzten drei Achsen ( $q_4 - q_6$ ) legen die Orientierung des Endeffektors fest.

Im ersten Schritt wird mit der bekannten Position und Orientierung des Endeffektors die Position des Handwurzepunktes  $\vec{H}$  bestimmt. Die Vektoren sind in Abbildung 4-1 noch einmal abgebildet.

$$\vec{H} = \vec{r}_{OE} - \vec{r}_{EH} \quad 4-2$$

mit

$$\vec{r}_{EH} = \vec{a} \cdot L_{05z}$$

#### 4.1.1 Lösung für $q_1$

Um den Winkel für  $q_1$  zu bestimmen wird der Roboter aus der Draufsicht (Projektion auf die Y-X Ebene betrachtet.

Der gesuchte Gelenkwinkel  $q_1$  ergibt sich durch:

$$q_1 = \text{atan2}(I_{Fu\beta} \cdot H_y, I_{Fu\beta} \cdot H_x) \quad 4-3$$

Die  $\text{atan2}(Y, X)$  Funktion ist für den Winkelbereich von  $\{-180^\circ \text{ bis } 180^\circ\}$  definiert: Befindet sich der Zeiger im ersten

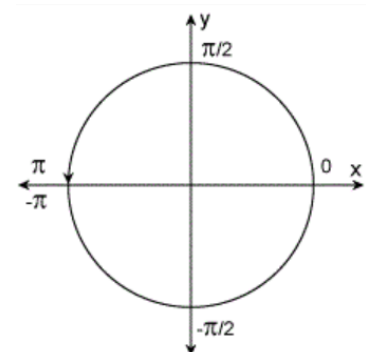


Abbildung 4-3:  
Winkeldefinition der  $\text{atan2}$   
Funktion

und zweiten Quadranten wird der positive Winkel ausgegeben. Im dritten und vierten Quadranten gibt die Funktion den negativen Winkel aus.

Sollte der Index  $I_{Fu\beta} \{1, -1\}$  mit  $-1$  belegt sein, erhält man den Winkel des gegenüberliegenden Quadranten. Dieser Winkel entspricht dann dem Winkel für den um  $180^\circ$  verdrehten Fuß.

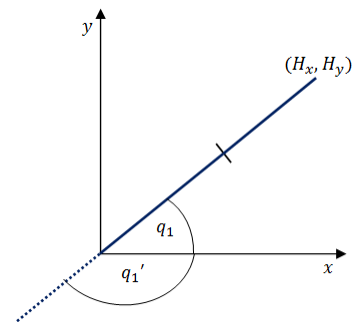


Abbildung 4-4:  
Gelenkwinkel  $q_1$

#### 4.1.2 Lösung für $q_2$

Für die Bestimmung des Gelenkwinkels  $q_2$  wird der Roboter aus der Seitenansicht (Projektion auf die Z-X Ebene des mit  $q_1$  gedrehten  $KS_1$ ) betrachtet. Die Offsets zwischen den Gelenkachsen von Fuß zu Oberarm und von Oberarm zu Unterarm ( $O_{F/OA}$  und  $O_{O/UA}$ ) erfordern zusätzliche Rechenschritte.

Für die Berechnung des Winkels  $q_2$  müssen die Längen der Vektoren  $r_{ges}$  und  $r_{OA}$  bestimmt werden. Mit diesen Vektoren und den entstandenen Dreiecken kann dann der Winkel  $q_2$  bestimmt werden.

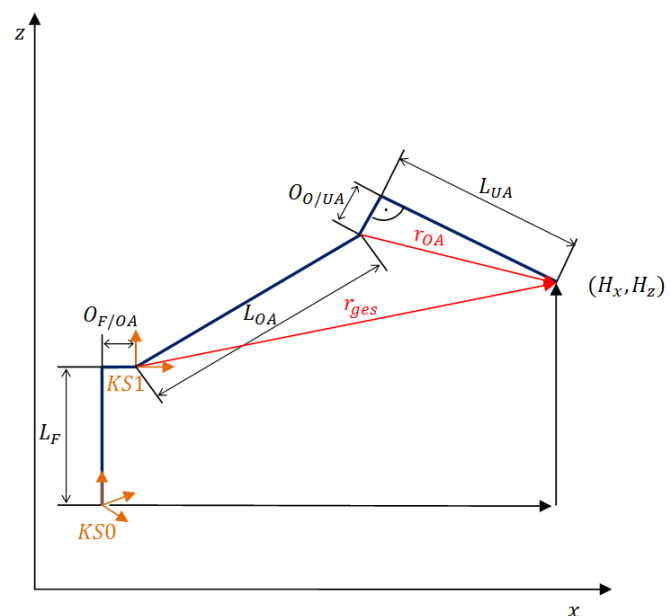


Abbildung 4-5: geom. Konstruktion für die Bestimmung des Gelenkwinkels  $q_2$

Um die  $X, Y, Z$  Komponenten von  $r_{ges}$  zu bestimmen wird eine Koordinatentransformation mit der homogenen Matrix  $A_{01}$  vom Basissystem 0 ins Koordinatensystem 1 durchgeführt. Der Vektor  $\vec{H}$  kann dann im  $KS1$  dargestellt werden, was den gesuchten Komponenten von  $r_{ges}$  entspricht. Die Hochgestellten Indizes drücken das Koordinatensystem aus, indem der Vektor dargestellt ist.

$$H^{(0)} = A_{01} \cdot H^{(1)}$$

$$H^{(1)} = \text{inv}(A_{01}) \cdot H^{(0)}$$

Die Länge des Vektors  $r_{ges}$  ergibt sich dann aus dem Satz des Pythagoras:

$$r_{ges} = \sqrt{H_x^{(1)2} + H_z^{(1)2}} \quad 4-4$$

Die Länge des Vektors  $r_{OA}$  ergibt sich ebenfalls aus dem Satz des Pythagoras:

$$r_{OA} = \sqrt{O_{O/UA}^2 + L_{UA}^2} \quad 4-5$$

Der gesuchte Winkel  $q_2$  ergibt sich dann aus der Addition von  $\beta$  und  $\delta$ . Die Abbildung 4-6 verdeutlicht diese Beziehung.

$$q_2 = \beta + \delta \quad 4-6$$

Mit

$$\beta = \arccos\left(\frac{L_{OA}^2 + r_{ges}^2 - r_{OA}^2}{2 \cdot L_{OA} \cdot r_{ges}}\right)$$

$$\delta = \arcsin\left(\frac{H_z^{(1)}}{r_{ges}}\right)$$

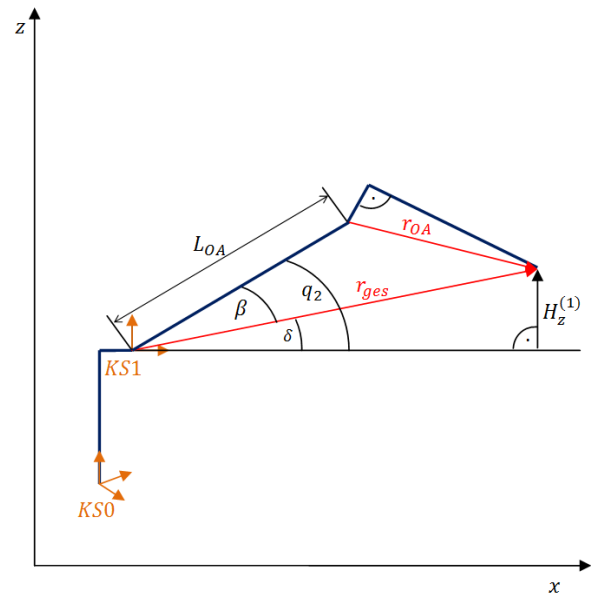


Abbildung 4-6: Winkel  $q_2$ , gebeugte Konfiguration

Die Formel 4-9 gilt allerdings nur für die Konfiguration des gebeugten Oberarms. Bei überstrecktem Oberarm gilt nach Abbildung 4-7:

$$q_2 = \delta - \beta \quad 4-7$$

Dieses negative Vorzeichen dreht sich allerdings wieder um, wenn der Fuß um  $180^\circ$  gedreht ist. Aus diesem Grund werden folgende Indizes für die Figur in die Formeln 4-6 und 4-7 implementiert:

$$q_2 = \delta + I_{OA} \cdot I_F \cdot \beta$$

Sollte der Fuß um  $180^\circ$  verdreht sein, muss zudem der Winkel  $q'_2 = \pi - q_2$  lauten. Für den Gelenkwinkel  $q_2$  ergeben sich folgende finale Formeln:

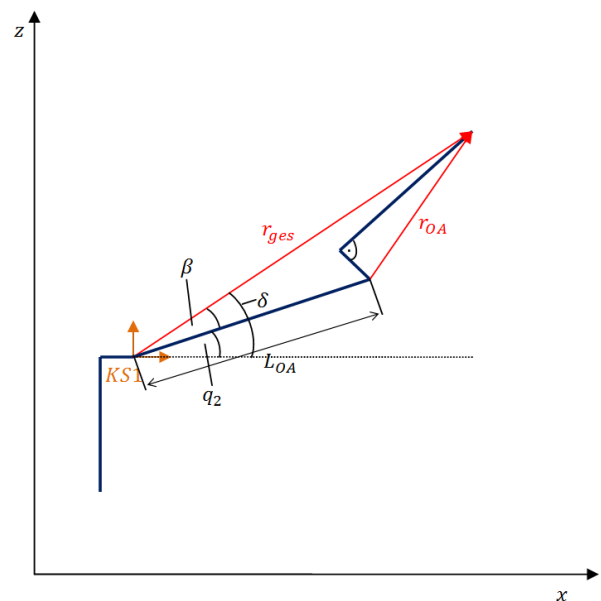


Abbildung 4-7: Winkel  $q_2$ , überstreckte Konfiguration

$$q_2 = \begin{cases} \beta + I_{OA} I_F \cdot \delta & ; \text{für } I_F = 1 \\ \pi - (\beta + I_{OA} I_F \cdot \delta) & ; \text{für } I_F = -1 \end{cases} \quad 4-8$$

### 4.1.3 Lösung für $q_3$

Für den Gelenkwinkel  $q_3$  müssen analog zum Winkel  $q_2$  Dreiecke gebildet werden und die unterschiedlichen Konfigurationen berücksichtigt werden.

Der Winkel  $q_3$  ergibt sich für die **überstreckte Oberarmkonfiguration** zu:

$$q_3 = 90^\circ - \alpha$$

mit

$$\alpha = \varphi - \beta$$

Die Winkel  $\varphi$  und  $\beta$  lassen sich analog zum Kapitel 4.1.2 mit einfachen trigonometrischen Funktionen oder dem Cosinussatz bestimmen.

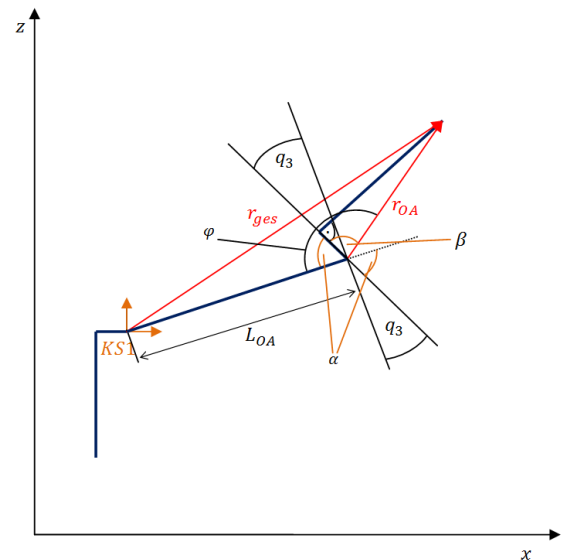


Abbildung 4-8: Winkel  $q_3$ , überstreckte Konfiguration

Für die **gebeugte Oberarmkonfiguration** ergibt sich der Winkel  $q_3$  zu:

$$q_3 = -(90^\circ - \alpha')$$

wobei diesmal  $\alpha'$  als:

$$\alpha' = \varphi' - \beta$$

definiert ist. Der Winkel  $\varphi'$  ist der stumpfe Winkel von  $\varphi$  und ergibt sich aus:

$$\varphi' = 360^\circ - \varphi$$

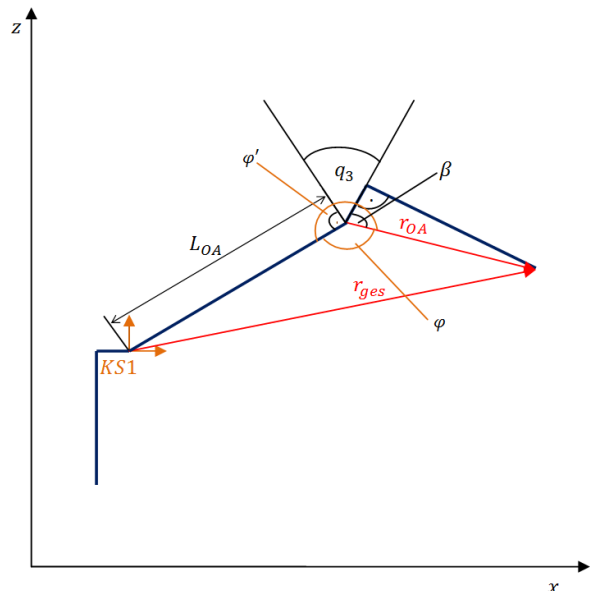


Abbildung 4-9: Winkel  $q_3$ , gebeugte Konfiguration

Da sich durch das Umdrehen der Fußstellung der Einfluss der Oberarmkonfiguration genau umdreht, erhält man folgende 4 Fälle:

Für  $I_F = 1$  (Fuß normal):

$$q_3 = \begin{cases} 90^\circ - \alpha & ; \text{für } I_{OA} = -1 \\ -(90^\circ - \alpha') & ; \text{für } I_{OA} = 1 \end{cases} \quad 4-9$$

Für  $I_F = -1$  (Fuß 180° verdreht):

$$q_3 = \begin{cases} 90^\circ - \alpha & ; \text{für } I_{OA} = 1 \\ -(90^\circ - \alpha') & ; \text{für } I_{OA} = -1 \end{cases} \quad 4-10$$

#### 4.1.4 Lösung für $q_4 - q_6$

Mit den Gelenkachsen  $q_1 - q_3$  wurde der Roboter so positioniert, dass der Endeffektor die Richtige Position hat.

Mit den Achsen  $q_4, q_5, q_6$  wird nun die Orientierung des Endeffektorkoordinatensystems eingestellt. Durch die ermittelten Winkel für  $q_1$  bis  $q_3$  ist die Lage und Orientierung vom  $KS3$  festgelegt.

Mit Hilfe der inversen Euler-Transformation können die drei Eulerwinkel berechnet werden, mit denen die Orientierung von einem  $KS$  mit drei aufeinanderfolgenden Drehungen in ein weiteres  $KS$  überführt werden. Im Falle des KUKA Agilus KR6 sind die drei Handachsen so angeordnet, dass eine Drehung um  $q_4, q_5, q_6$  einer Euler Transformation um  $ZYZ$  des  $KS3$  entspricht.

Mit Hilfe der bekannten Rotationsmatrizen  $R_{06}$  und  $R_{03}(q_1, q_2, q_3)$  kann die Rotationsmatrix  $R_{36}$  ermittelt werden:

$$R_{36} = \text{inv}(R_{03}) \cdot R_{06} \quad 4-11$$

In den Spalten der Matrix  $R_{36}$  sind die Vektoren  $\vec{a}, \vec{o}, \vec{n}$  des Endeffektorkoordinatensystems im  $KS3$  dargestellt.

Mit den Formeln für die  $ZYZ$  Euler-Transformation nach [Stark 2009] können nun die Gelenkwinkel berechnet werden.

$$q_4 = \text{atan2}(a_y, a_x) \quad 4-12$$

$$q_5 = \text{atan2}(\sqrt{a_x^2 + a_y^2}, a_z) \quad 4-13$$

$$q_6 = \text{atan2}(y_{n1}, y_{o1}) \quad 4-14$$

mit

$$yn_1 = -n_x \sin(q_4) + n_y \cos(q_4)$$

$$yo_1 = -o_x \sin(q_4) + o_y \cos(q_4)$$

Die inverse Eulertransformation ist mehrdeutig und kann am KUKA Agilus KR6 anschaulich dargestellt werden. So kann sich der Unterarm um  $180^\circ$  drehen, wenn zugleich das Handgelenk um den negativen Winkel in die entgegengesetzte Richtung geklappt wird. Es ergeben sich folgende Winkel:

$$q'_4 = \begin{cases} q_4 - \pi & \text{für } q_4 \geq 0 \\ q_4 + \pi & \text{für } q_4 < 0 \end{cases} \quad 4-15$$

$$q'_5 = -q_5 \quad 4-16$$

$$q'_6 = \begin{cases} q_6 - \pi & \text{für } q_6 \geq 0 \\ q_6 + \pi & \text{für } q_6 < 0 \end{cases} \quad 4-17$$

Um eindeutig zu bestimmen nach welcher Lösung gesucht ist, wird über den Handgelenk Index  $I_H$  angegeben, ob das Handgelenk nach oben oder unten geklappt ist. Im Algorithmus wird als erstes der Winkel  $q_5$  bestimmt und dann mit der Vorgabe aus dem Index  $I_H$  abgeglichen. Sollte der Winkel nicht mit der Handgelenkkonfiguration übereinstimmen, werden die Winkelvorschriften der anderen Konfiguration gewählt.

## 4.2 Numerisches Verfahren

Auf den numerischen Ansatz wird in dieser Arbeit nur kurz verwiesen, da er für die weiteren Modelle nicht verwendet wurde.

Das Verfahren beruht auf den differentiellen Zusammenhängen, die über die Jacobi-Matrix hergestellt werden. So gilt:

$$\dot{X} = J \cdot \dot{Q} \quad 4-18$$

$$\dot{Q} = J^{-1} \cdot \dot{X} \quad 4-19$$

Die Gleichung 4-19 wird verwendet um mithilfe von numerischer Integration die gesuchten Gelenkwinkel  $q$  zu erhalten. Nachteil dieser Variante ist es, dass dem Integrationsblock die Startwinkel vorgegeben werden müssen. Zudem können keine exakten Vorgaben für die Konfiguration gemacht werden und in den singulären Konfigurationen liefert das Gleichungssystem keine Lösung.

Die folgende Abbildung 4-10 zeigt den Aufbau des Submodells der numerischen inversen Kinematik in Simulink. Das Eingangssignal, die Position  $x$ , wird einmal abgeleitet und mit der inversen Jacobi-Matrix multipliziert. Die Jacobi-Matrix ist abhängig von der aktuellen Position des Roboters:  $J(q)$ . Die erhaltene Ableitung der Gelenkwinkel  $\dot{q}$  wird wieder integriert um die Gelenkwinkel  $q$  zu erhalten.

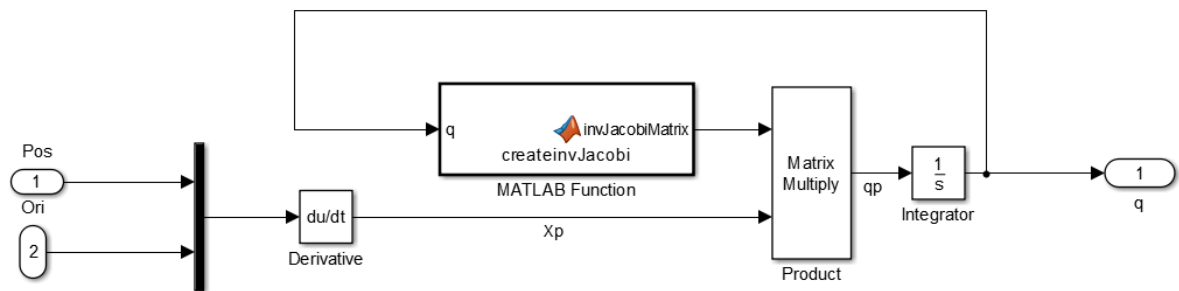


Abbildung 4-10: inverse Kinematik mittels Jacobi-Matrix

## 5 Bahnsteuerung

Industrieroboter müssen in der Lage sein, bahngeführte Aufgaben wie sie zum Beispiel beim Bahnschweißen relevant sind, auszuführen.

„Bahnsteuerungen (...) müssen die vorgegebene Trajektorie während der Bewegung generieren, im allgemeinen eine Transformation zwischen dem Bezugs- und dem gelenkspezifischen Koordinatensystem berechnen und die Roboterhand exakt mit definierter Geschwindigkeit hierauf führen.“ [J.Olomski 1989]

Zwei Aufgabenbereiche lassen sich zu diesem Thema definieren: Zum einen die Bahnplanung und zum anderen die Bahngenerierung oder auch Interpolation genannt.

Im Schritt der Bahnplanung werden der geometrische Verlauf, die Trajektorie, sowie das dazugehörige Geschwindigkeitsprofil festgelegt. Das Geschwindigkeitsprofil der einzelnen Achsen muss, je nach Forderung einer asynchronen, synchronen oder voll-synchronen Ausführung der Bewegung, angepasst werden.

Bei der Interpolation werden je nach Verfahrrart und Geschwindigkeitsprofil Achssollwerte in Abhängigkeit der Zeit für die Roboterregelung bereitgestellt.

### 5.1 Trajektorie

Der räumliche Verlauf einer Bewegung wird als Trajektorie bezeichnet und lässt sich durch folgende Parameter definieren:

- Start und Zielpunkt
- Verfahrrart
- Zwischenpunkte (optional)
- Information zum Abrunden der Trajektorie bei Zwischenpunkten (optional)

Bei der Verfahrrart wird zwischen zwei Modellen unterschieden:

#### ***Planung in Roboterkoordinaten:***

Ein Start- und Zielpunkt wird definiert und mit Roboterkoordinaten abgebildet. Die Planung der Trajektorie erfolgt in Roboterkoordinaten, die über die Zeit interpoliert werden. Eine solche Bewegung wird als „Point to Point“ Bewegung (**PTP**) bezeichnet. Der Vorteil dieser Planung ist, dass eine zeitoptimale Lösung für das Erreichen des Zielpunktes gefunden wird und Grenzwerte für Ort, Geschwindigkeit und Beschleunigung der Achsen nicht überschritten werden.

Nachteil dieser Planung ist, dass der genaue Fahrweg nicht bekannt ist und so Kollisionsprüfungen nur schwer durchführbar sind. [Stark, 2008]



### ***Planung in Weltkoordinaten:***

Bei der Planung in Weltkoordinaten ist der gesamte Trajektorienverlauf exakt durch den Anwender definiert. Diese Form der Bewegung wird als „Continuous Path“ (**CP**) bezeichnet. Mögliche geometrische Formen eines „Continuous Path“ ist eine Gerade, eine Kreisbahn oder ein Spline, der die definierten Punkte verbindet.

Der Vorteil dieser Bahnplanung ist, dass die Bewegung so exakt an ein Bauteil angepasst werden kann. Außerdem haben roboterspezifische Eigenschaften, wie die Kinematik des Roboters keinen Einfluss auf die Ausführung der Trajektorie.

Der Nachteil wiederum ist, dass zulässige Achsgeschwindigkeiten überschritten werden können, besonders im Bereich von singulären Konfigurationen des Roboters. [Stark, 2008]

## **5.2 Geschwindigkeitsprofil**

Das Geschwindigkeitsprofil gibt eine Information über den zeitlichen Verlauf der Bewegung. Aus der Integration des Geschwindigkeitsprofils lässt sich ein Verlauf der Bahnkoordinate  $s(t)$  ermitteln.

Im Allgemeinen lässt sich das Geschwindigkeitsprofil in drei Phasen einteilen: eine Beschleunigungsphase, eine Phase in der die Geschwindigkeit gehalten wird und eine Verzögerungs- oder Abbremsphase.

Im einfachsten Fall werden lineare Verläufe gewählt, die dann zu einem Trapez zusammengesetzt werden. Um keine Unstetigkeiten in den Beschleunigungsverläufen zu haben und so ein ruckfreies Anfahren zu gewährleisten, werden auch Polynome oder trigonometrische Funktionen als Geschwindigkeitsprofil verwendet.

Unter der Synchronisation der Bewegung versteht man das Angleichen der Teilbewegungen auf gemeinsame Zeitpunkte. So ist eine synchrone Bewegung durch gleichzeitiges Anfahren und Stoppen der Einzelbewegungen definiert. Eine vollsynchrone Bewegung hat zudem gleiche Zeiten für die Beschleunigungsphase, die konstante Phase und die Verzögerungsphase. Die folgende Abbildung [Strak 2009] verdeutlicht dies noch einmal:

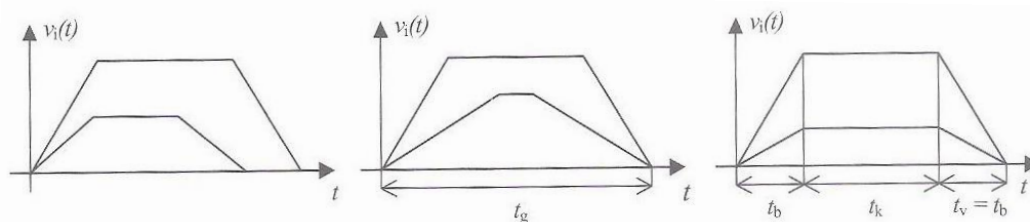


Abbildung 5-1: asynchrone, synchrone und vollsynchrone Bahnfahrt

### 5.3 Implementierung der Bahnsteuerung ins Simulationsmodell

Bei einem realen Robotersystem liefert die Bahnsteuerung Sollwerte für die Roboterachsen. In der Simulation übergibt die Bahnsteuerung also die Achssollwerte an das modellierte Kinematikmodell des Roboters.

Während die Bahnplanung in einem Initialisierungsskript vor der Simulationsschleife abgerufen wird, wird die Bahngenerierung und die Interpolation in der Simulationsschleife ausgeführt.

Folgendes Ablaufdiagramm visualisiert den Aufbau des Modells noch einmal mit den Input- und Outputgrößen der einzelnen Abschnitte.

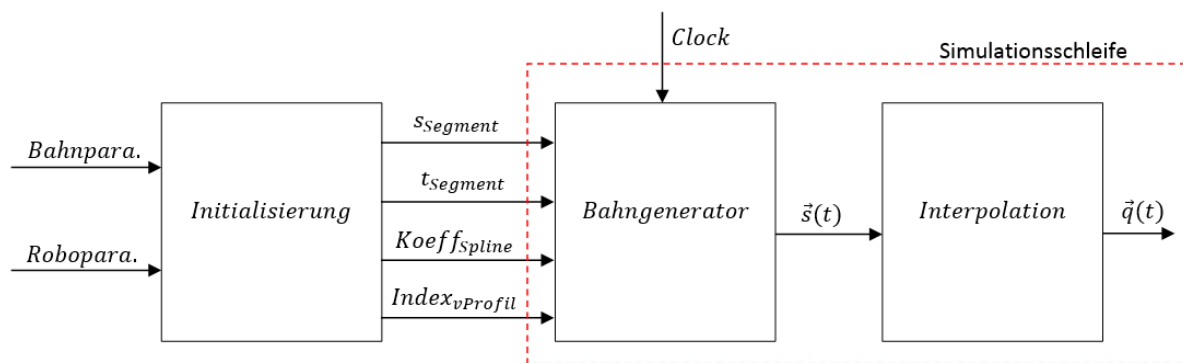


Abbildung 5-2: Ablaufdiagramm: Bahnsteuerung

#### 5.3.1 Initialisierung

Das Matlab-File Robo\_ini.m erstellt basierend auf den Bahn- und Roboterparametern die Bahnplanung.

Die **Inputgrößen** - Bahn- und Roboterparameter - setzen sich aus folgenden Variablen zusammen:

Bahnparameter:

- Startpunkt, -Startorientierung und Startkonfiguration
- Zielpunkt, Zielorientierung und Zielkonfiguration
- Interpolationsart (PTP oder Line)
- Geschwindigkeitsprofil (Trapez oder Polynom)

Roboterparameter:

- Maximale Geschwindigkeit der Achsen
- Maximale Beschleunigung der Achsen
- Faktor (zwischen 0 und 1) mit der die Geschwindigkeit angepasst wird

Die **Outputgrößen** sind die Längen  $\vec{s}$  und Zeiten  $\vec{t}$  der Bewegungssegmente: Beschleunigung, konstante Fahrt und Verzögerung. Im Falle eines Polynom-Geschwindigkeitsprofils werden die

Koeffizienten des integrierten Polynoms übergeben. Der Index  $I_{v_{profil}}$  übergibt der nächsten Funktion, welches Geschwindigkeitsprofil gewählt wurde.

Das **Programm Robo\_ini** unterteilt sich in einen allgemeinen und einen spezifischen Teil für die jeweilige Kombination aus Interpolationsart und Geschwindigkeitsprofil.

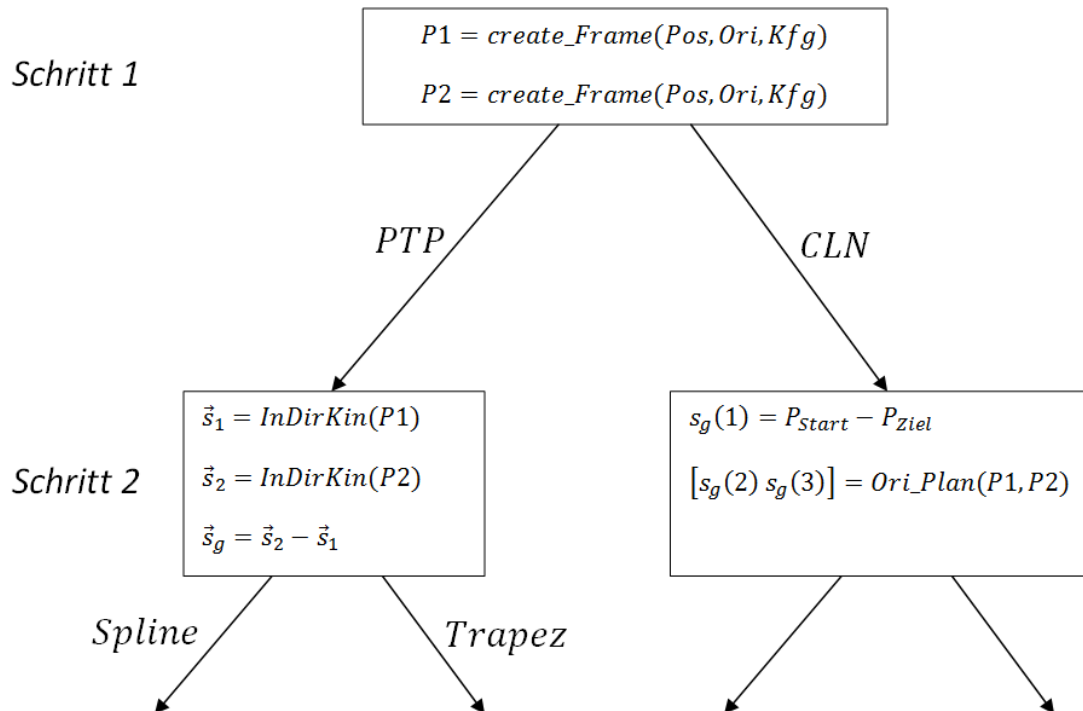


Abbildung 5-3: Baumdiagramm Initialisierung Teil I

Im ersten Schritt, im allgemeinen Teil, werden die homogenen Matrizen der Start und Zielpunkte aus Position und Orientierung gebildet. Dafür steht die Funktion createFrame.m zur Verfügung.

Je nachdem, ob es sich um eine PTP oder CP Bewegung handelt, wird die Bahnkoordinate  $\vec{s}$  in Roboterkoordinaten oder Weltkoordinaten beschrieben. Im Falle einer PTP Bewegung ist die Bahnkoordinate ein  $[n_{Achsen} \times 1]$  Vektor, da jede Achse eine Teilbewegung darstellt. Bei einer CP Bewegung hat die Bahnkoordinate immer drei Einträge für drei Teilbewegungen: Der erste Eintrag beschreibt die Position in kartesischen Koordinaten; die Einträge zwei und drei beschreiben die Orientierung um zwei definierte Achsen. Diese Achsen sind so gewählt, dass jede Orientierung durch eine Drehung um diese beiden raumfesten Achsen beschrieben werden kann.

Im zweiten Schritt wird dann der Vektor  $s_g$  definiert, der den Betrag der Komponenten der Bahn definiert.

Im Falle der PTP Bewegung werden über die indirekte Kinematik (mit der Funktion InDirKin.m) die Achswinkel vom Start und Zielpunkt ermittelt und die Differenz gebildet.

Im Falle einer CLN Bewegung kann die Strecke der Bewegung direkt über den kartesisch definierten Ziel- und Startpunkt ermittelt werden. Die Überführung der Orientierung von P1 zu P2 erfolgt mit der Funktion  $Ori\_Plan(P1, P2)$ . Die Funktion  $Ori\_Plan(P1, P2)$  ist dem Buch [Stark 2008] entnommen und wird hier nicht näher erläutert.

Im dritten Schritt werden je nach gewähltem Geschwindigkeitsprofil die Segmentzeiten (Beschleunigung, konstante Fahrt, Verzögerung) und die resultierenden Segmentlängen  $\vec{s}_i$  ermittelt. Um eine vollsynchronen Ausführung der Bewegung zu generieren, werden bei der PTP Bewegung die maximalen Segmentzeiten ermittelt und die anderen Teilbewegungen daran angepasst. Bei der CLN Bahn werden die Segmentzeiten der Position auch für die Orientierung vorgeschrieben. Nachdem die Zeiten definiert sind, werden die Segmentlängen der vollsynchronen Bahn bestimmt. Das folgende Schaubild verdeutlicht die Schritte noch einmal:

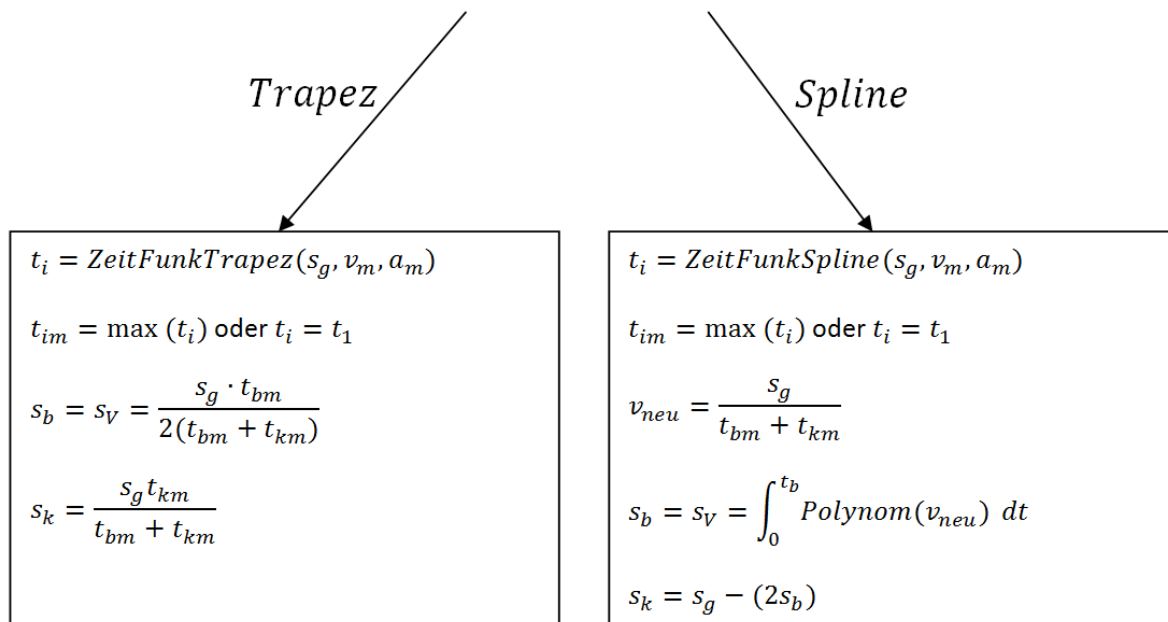


Abbildung 5-4: Baumdiagramm Initialisierung Teil II

Die Formeln für die Bestimmung der Segmentzeiten und Wege für das Trapezprofil sind nach [Stark 2008] definiert und werden nicht weiter erläutert. Die Funktionen und Formeln für das Polynom-Geschwindigkeitsprofil werden im Anhang genauer erklärt, da sich in den weiteren Modellen erst einmal auf Trapezprofile beschränkt wird.

### 5.3.2 Bahngenerator

Im Funktionsblock „Bahngenerator“ werden aus den definierten Segmentlängen und Segmentzeiten sowie den Informationen über das Geschwindigkeitsprofil die Bahnfunktion  $\vec{s}(t)$  definiert.

Die Funktion wird aus der Integration der Geschwindigkeitsprofile der Teilbewegungen abgeleitet. Bei einer PTP Bewegung mit Trapez-Geschwindigkeitsprofil ergibt sich für die Segmentbewegungen folgender Graph:

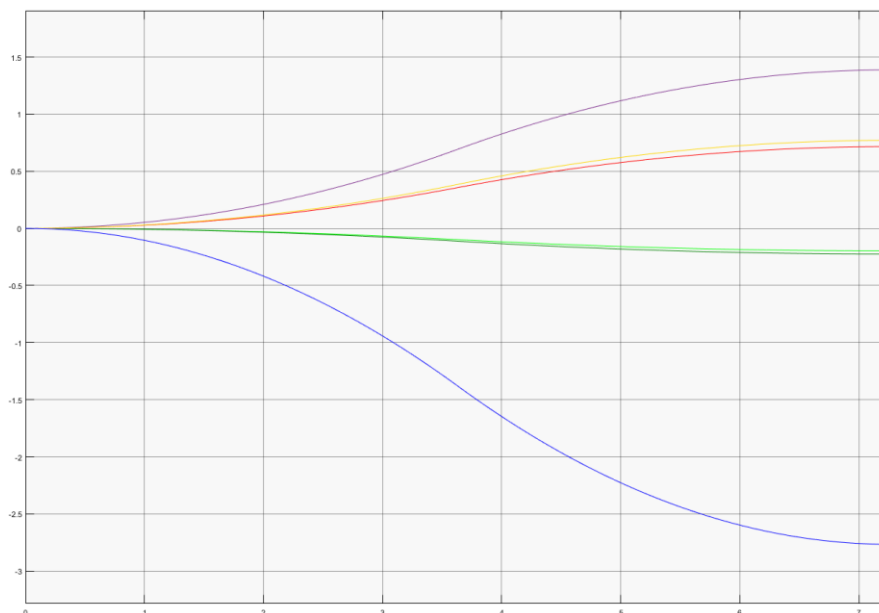


Abbildung 5-5: Komponenten der Bahnkoordinate  $\vec{s}(t)$  bei vollsync. PTP Fahrt

### 5.3.3 Interpolation

Im Funktionsblock der Interpolation werden abhängig von der Interpolationsart Zwischenpunkte definiert, aus denen dann die Achsstellungen für die Simulation abgeleitet werden.

Mögliche Interpolationsarten für zwei Punkte sind zum Beispiel die lineare Interpolation oder für drei Punkte die Kreisinterpolation. Eine größere Anzahl von Punkten kann mit Hilfe der Spline-Interpolation zu einer stetigen Funktion verbunden werden.

Eingangsgrößen für die Interpolation sind die Bahnparameter (Start- und Zielpunkt) sowie die Bahnkoordinate  $s(t)$ , die als Funktionsstelle für die Interpolation zwischen Start- und Zielpunkt dient.

Der Interpolationstakt ist in diesem Fall die Simulationsschrittweite, mit der die Funktion  $s(t)$  abgetastet wird.

Wenn die Bahnkoordinate  $s(t)$  in Roboterkoordinaten vorliegt, können die interpolierten Gelenkwinkel direkt ausgegeben werden. Sollte die Bahnkoordinate in Weltkoordinaten vorliegen, wird die Position und Orientierung in Weltkoordinaten interpoliert und dann mittels der inversen Kinematik in Roboterkoordinaten umgewandelt.

Die Formeln für die lineare Interpolation sind der Literatur [Stark 2008] entnommen und werden hier nicht weiter erläutert.

## 6 Simulationsmodell eines kooperativen Roboters

Um einen Roboter auf Ereignisse in seiner Umgebung reagieren lassen zu können, ist es nötig, dass die Trajektorie online beeinflusst und geändert werden kann. Es müssen Konzepte entwickelt werden, die es erlauben, dass der Roboter ohne Stopp einer neuen Trajektorie folgt und so Hindernissen ausweicht, einen Sicherheitsabstand zum Anwender einhält oder seine Geschwindigkeit bei Verletzung bestimmter Sicherheitszonen reduziert. Man spricht in diesem Zusammenhang von Trajektorienadaptation.

In der Literatur lassen sich bereits einige Ansätze finden, auf die im Folgenden kurz näher eingegangen wird.

### 6.1 Literaturrecherche

#### *Continuous genetic algorithm (CGA)*

Diese Algorithmen sind speziell dafür entwickelt, online auf Ereignisse zu reagieren und passen somit ihre Lösung kontinuierlich an. In dem recherchierten Beispiel handelt es sich um ein CGA zur Kollisionsvermeidung bei einer kartesischen Bahnplanung.

Bei diesem Ansatz wird die Pfadplanung in Form eines Optimierungsproblems formuliert, das den Abstand zum gewünschten Pfad minimiert, jedoch als Randbedingung einen Mindestabstand zu den Hindernissen einhält. In diesem speziellen Fall wird zur Lösung des Problems ein „genetic algorithm“ angewendet, der angelehnt an die Evolution mit Wahrscheinlichkeiten, Vererbung, Mutation und Populationen arbeitet. Auf diesen interessanten, aber hoch mathematischen Ansatz wird in dieser Arbeit nicht weiter eingegangen. Es sei auf die Literatur [Abo-Hammour, Alsmadi; 2011] verwiesen.

#### *Multipoint Trajectories*

Ein weiterer Ansatz ist die Definition von mehreren Punkten, die dann der Reihe nach, zum Beispiel in einer PTP Bewegung abgefahren werden. Bei Ereignissen ändert sich die Referenz-Trajektorie mit einer Sprungfunktion und die Ist-Trajektorie passt sich nach dem jeweiligen Regelverhalten an. Für dieses Modell wird ein dynamischer nichtlinearer Filter genutzt, um online die Ist-Trajektorie zu beeinflussen. Für weitere Informationen sei auf [Biagiotti, Melchiorri; 2008] verwiesen.

#### *Event-Driven Systems: Stateflow – Matlab/Simulink*

Im Gegensatz zu den kontinuierlichen Algorithmen kann das Problem der Trajektorienadaptation auch mit einem reaktiven Algorithmus gelöst werden. Wenn ein Event auftritt, müssen Zustandsvariablen, wie zum Beispiel Bahnpunkte, Segmentzeiten und -geschwindigkeiten unmittelbar neu bestimmt werden.

In der Matlab/Simulink Umgebung wurde extra für Zustandsabhängige Systeme die Stateflow-Toolbox entwickelt. Die Toolbox ermöglicht es, mit Hilfe eines Automaten während der Simulation zustandsabhängig Variablen zu definieren.

Normalerweise simuliert Simulink das Verhalten von Systemen auf kontinuierliche Änderungen. Mit der Stateflow-Toolbox erweitern sich die Möglichkeiten, von Simulink nun auch auf unmittelbare Änderungen zu reagieren. [MathWorks Model-Event-Driven System]

Das Feder-Dämpfer Verhalten eines Fahrzeuges könnte in der Simulink Umgebung simuliert werden. Soll in diese Simulation nun auch noch das Wechseln der Gänge eingebunden werden, müsste ein Stateflow Automat eingebunden werden, um die unmittelbaren Änderungen auf Zustandsvariablen zu berücksichtigen.

Als Einstieg zum Arbeiten mit Stateflow empfiehlt sich das Einstiegstutorial „Getting started with stateflow“ [Mathworks, Stateflow 2016].

## 6.2 Aufbau und Anforderungen des Simulink-Modells

Das Hauptmodell lässt sich in zwei Submodelle einteilen:

Auf der einen Seite die Bahnsteuerung, die die Gelenkwinkel  $q$  in Abhängigkeit der Zeit bereitstellt und auf der anderen Seite die Modellierung des Roboters, die die Roboterposition in der SimMechanics Umgebung visualisiert.

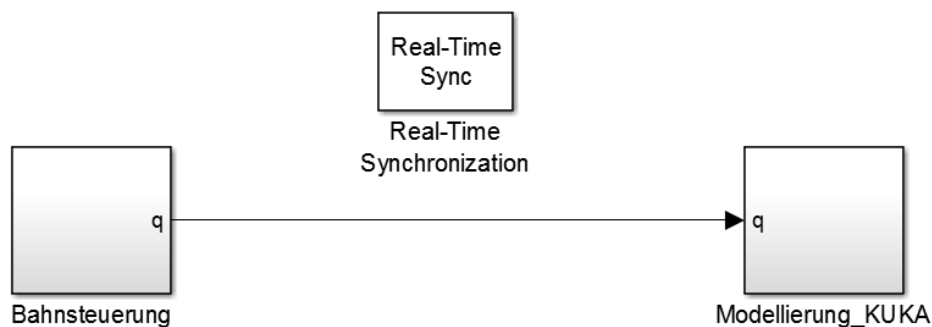


Abbildung 6-1: Erste Ebene des Simulink Modells „AdvancedModell.slx“

Die Modellierung des Roboters wurde in Kapitel 3.2 beschrieben, der Aufbau der Bahnsteuerung für eine nicht-zyklische Bahnfahrt in Kapitel 5. Zur Vereinfachung wird das Modell aus Kapitel 5 im Folgenden nur „Basis-Modell“ und dieses Modell „Advanced-Modell“ genannt.

Das Basis-Modell wird nun dahingehend erweitert, dass der Roboter eine zyklische Bewegung zwischen definierten Punkten ausführt und bei der Unterschreitung von bestimmten Sicherheitsabständen die Geschwindigkeit stufenweise bis zum Stillstand reduziert. Dabei

werden als Geschwindigkeitsprofile Trapezprofile und als Interpolationsart eine PTP-Bewegungen verwendet.

Die beschriebenen Erweiterungen des Basis-Modells betreffen nur den Block der *Bahnsteuerung*. Es lassen sich aus den zusätzlichen Funktionen der Zyklusfahrt und der online Geschwindigkeitsadaption folgende Forderungen für das Modell ableiten:

### Zyklusfahrt

1. Es muss erkannt werden, wenn der Roboter einen Zielpunkt erreicht hat und dann unmittelbar die Parameter für die Rückfahrt definiert werden.
2. Sowohl das Verlangsamen als auch der Stopp der Bewegung haben keinen Einfluss auf Start und Zielpunkt der Zyklusfahrt.

### Online-Geschwindigkeitsadaption

1. Das Initialisierungsskript, das die Bahnparameter definiert, muss zu jedem Zeitpunkt gestartet werden können und so unmittelbar dem Bahngenerator eine neue Trajektorie vorgeben.
2. Es muss eine Schnittstelle zwischen dem Simulinkmodell und dem Laserscanner geschaffen werden, sodass Abstandsdaten ausgewertet werden können auf deren Grundlage Events ausgelöst werden.
3. Die Simulation muss in Echtzeit, jedoch unabhängig von der Taktrate des Laserscanners laufen. So ist eine flüssige Simulation gewährleistet, die trotzdem direkt auf Events reagieren kann

Grundlegende Änderung im Block der Bahnsteuerung im Vergleich zum Basismodell ist, dass ein Zustandsautomat aus der Stateflow-Bibliothek eingebunden wird. Dieser Automat steuert die Zyklusbewegung und startet das Initialisierungsskript beim Erreichen von Zielpunkten oder Sicherheitsabstands-Events. Der Automat besitzt zusätzlich eine Logik, die die Abstandsdaten auswertet und den jeweiligen Sicherheitszustand definiert.

Zum Vergleich sind hier noch einmal beide Flussdiagramme des Basis-Modells und des Advanced-Modells dargestellt:

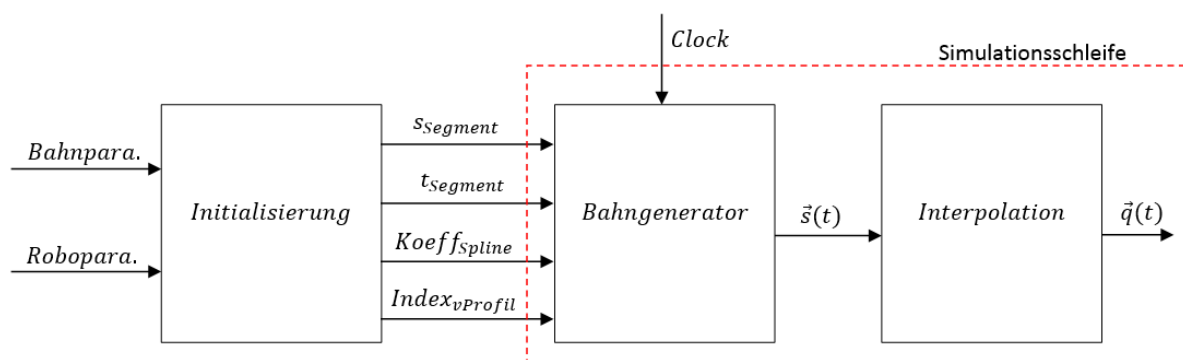


Abbildung 6-2: Flussdiagramm Basis-Modell



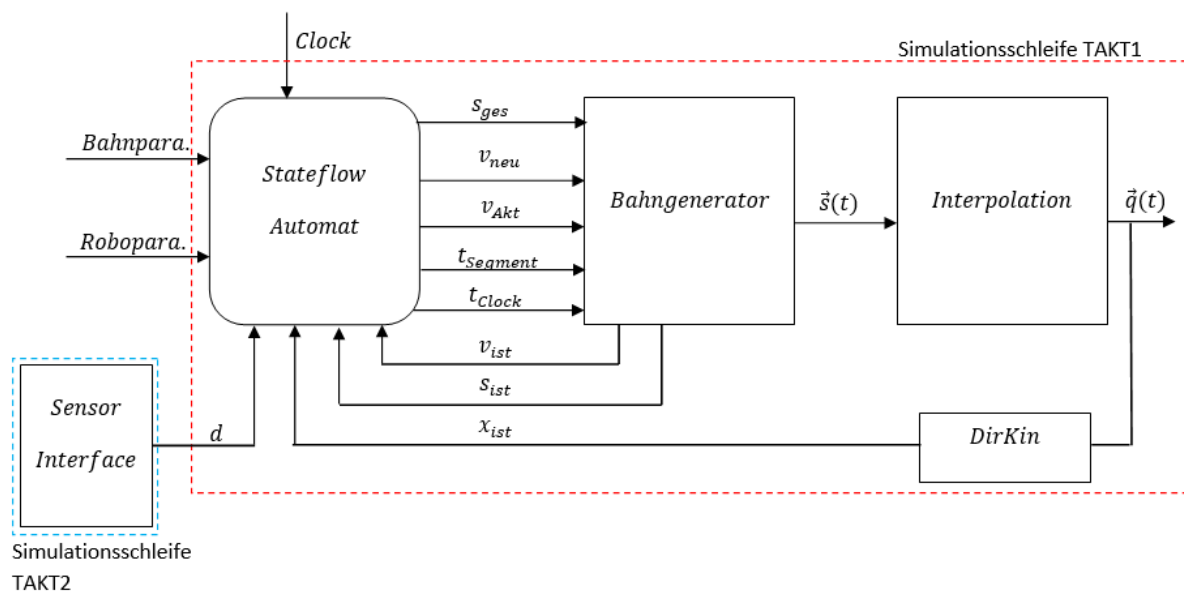


Abbildung 6-3: Flussdiagramm Advanced-Modell

Durch die Grafiken wird noch einmal deutlich, dass der Stateflow-Automat den Roboter überwacht. Die Rückführung der Signale, aktuelle Geschwindigkeit ( $v_{ist}$ ), aktuelle Bahnkoordinate ( $s_{ist}$ ) und aktueller Position im Raum  $x_{ist}$  liefern die dafür notwendigen Signale.

Außerdem werden nun andere Parameter an den Bahngenerator übergeben. Das Wegfallen von den Parametern „KoeffSpline“ und „IndexVProfil“ ist nur der Tatsache geschuldet, dass in diesem Modell keine Polynom-Geschwindigkeitsprofile zur Auswahl stehen.

Während im Basis-Modell noch die Integration vom Geschwindigkeitsprofil zur Bahnkoordinate analytisch durchgeführt wurde, wird im Advanced-Modell nur das Trapez-Geschwindigkeitsprofil vorgegeben und die Bahnkoordinate dann über numerische Integration bestimmt. Dies hat den Vorteil, dass auch komplexere Trapezprofile mit mehreren Verzögerungen oder Beschleunigungen in einem Zyklus realisiert werden können.

### 6.3 Multitask und Echtzeit-Synchronisation

Simulink simuliert ein Modell in einer bestimmten Abtastrate. Dabei werden alle Blöcke einmal durchlaufen und danach die Simulationszeit um die Abtastrate erhöht. Bei einfachen Modellen und leistungsstarken Rechnern kann deswegen das Systemverhalten für einen langen Zeitraum innerhalb weniger Sekunden simuliert werden.

Im konkreten Fall soll allerdings über den Laserscanner mit der Umwelt, das heißt mit der Echtzeit, interagiert werden. Deswegen ist es notwendig, dass der Computer in Echtzeit und nicht schneller simuliert. Der Block, der hierfür Abhilfe schafft, ist in der „Simulink Realtime Desktop“ Bibliothek zu finden und heißt „Real-Time Synchronization“.

Bei Nutzung dieses Blockes müssen folgende Einstellungen vorgenommen werden:

1. „Configuration Parameters → Code Generation → System Target File → sldr.tlc
2. „Configuration Parameters → Sim Scape → Log simulation data → none

Danach muss die Simulation einmal im “external-mode” gestartet werden, damit ein Real-Time cernal installiert wird. Wenn dies erfolgt ist, kann die Simulation wieder im „normal mode“ ausgeführt werden.

Dass alle Blöcke einmal durchlaufen werden, bevor die Simulationszeit erhöht wird, bedeutet allerdings auch, dass die Simulation bei einem komplexen Block erheblich gedrosselt wird.

Da das Auswerten der Laserscandaten ca. 0.1 Sekunden dauert, müssen unterschiedliche Taktraten für die einzelnen Systeme (Roboter und Laserscanner) eingestellt werden. Die Simulation des Roboters erfolgt mit einem fixed-step-solver in einer Abtaste von 0.001. Wenn man die beiden Systeme nicht entkoppeln würde, würde die Robotersimulation in einer Sekunde nur 10 Schritte ausführen, was einer Zeit von 0.01 Sekunden entspräche.

Um das System trotzdem in Echtzeit simulieren zu können, wurde für das System des Laserscanners eine andere Sample Rate definiert als für das Robotersimulationssystem.

Für solch eine Simulation mit unterschiedlichen Sample Rates muss in den „Configuration Parameters → Diagnostics → sample Time → Multi Task rate Transition“ auf „warning“ gestellt werden.

## 6.4 Stateflow-Zustandsautomat

Der Stateflow Zustandsautomat stellt das Herzstück des Advanced-Modells dar, da hier der Sicherheitsabstand zu Personen überwacht und die Robotersteuerung realisiert wird.

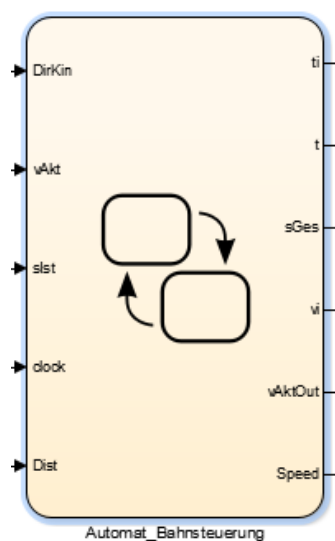


Abbildung 6-4:

Stateflow-Zustandsautomat

Inputgrößen sind:

- Distanzwerte vom Laserscanner → Sicherheitsüberwachung
- Aktuelle Position (DirKin) → Zielpunkterkennung und Bewegungsumkehr
- Aktuelle Geschwindigkeit → Neurechnung der Trajektorie (Events)
- Aktuelle Bahnkoordinate → Neurechnung der Trajektorie (Events)
- Simulationszeit

Die Output-Größen dienen alle der Definition des Geschwindigkeits-Trapezprofils:

- Segmentzeiten, von Beschleunigung, konstanter Fahrt und Verzögerung
- Geschwindigkeit der konstanten Fahrt
- Startgeschwindigkeit für Trapezprofil bei Events
- Simulationszeit; diese wird immer auf 0 gesetzt bei neuer Trajektorie
- Gesamtstrecke der Bahnkoordinate
- Aktueller Geschwindigkeitsfaktor (für Kontrollzwecke)

Da der Zustandsautomat zwei Aufgaben (Abstandsüberwachung und Robotersteuerung) gleichzeitig ausführen muss, sind in der ersten Ebene zwei parallele Zustände definiert. Das parallele Verhalten der Zustände ist durch die gestrichelte Linie gekennzeichnet. Die Nummern bedeuten, welcher Zustand innerhalb eines Simulationsschrittes als erstes abgearbeitet wird.

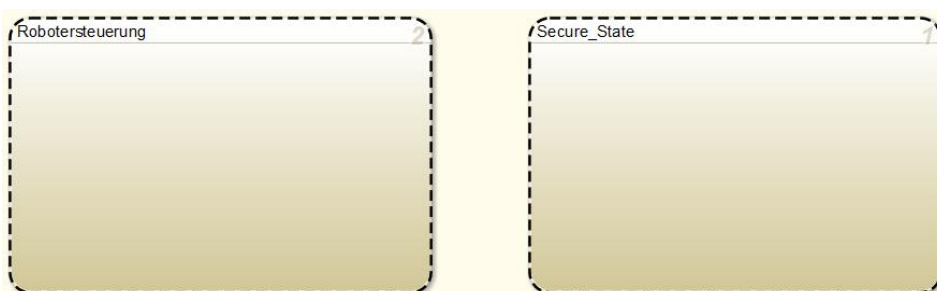


Abbildung 6-5: Parallele Aufgabenstruktur im Stateflow-Zustandsautomaten

#### 6.4.1 Abstandsüberwachung

Der Aufbau des Zustandsautomaten für die Abstandsüberwachung eignet sich sehr gut, um grundlegende Eigenschaften eines Stateflow-Diagramms deutlich zu machen. Auf der folgenden Seite ist das Subchart der Abstandsüberwachung dargestellt.

Die abgerundeten Kästchen stellen Zustände (States) dar, die über Verbindungen (Transitions) miteinander verbunden sind. An den Verbindungen stehen in eckigen Klammern Bedingungen, die erfüllt sein müssen, damit von einem in den andern Zustand gewechselt wird. Zusätzlich können auch noch Aktionen (in geschweiften Klammern) definiert werden, die ausgeführt werden, wenn eine Transition passiert wird.

Innerhalb der Zustände können drei Bereiche definiert werden:

Der Entry-Bereich: Alle Aktionen die hier definiert werden, werden nur einmal zu Beginn des Zustands ausgeführt.

Der During-Bereich: Solange der Zustand aktiv ist, wird dieser Bereich mit der Simulationsschleife durchlaufen.

Der Close-Bereich: Dieser Bereich wird nur einmal beim Verlassen des Zustands ausgeführt.

Über eine „default-Transmission“ wird definiert, welcher Zustand als erstes aktiv ist.

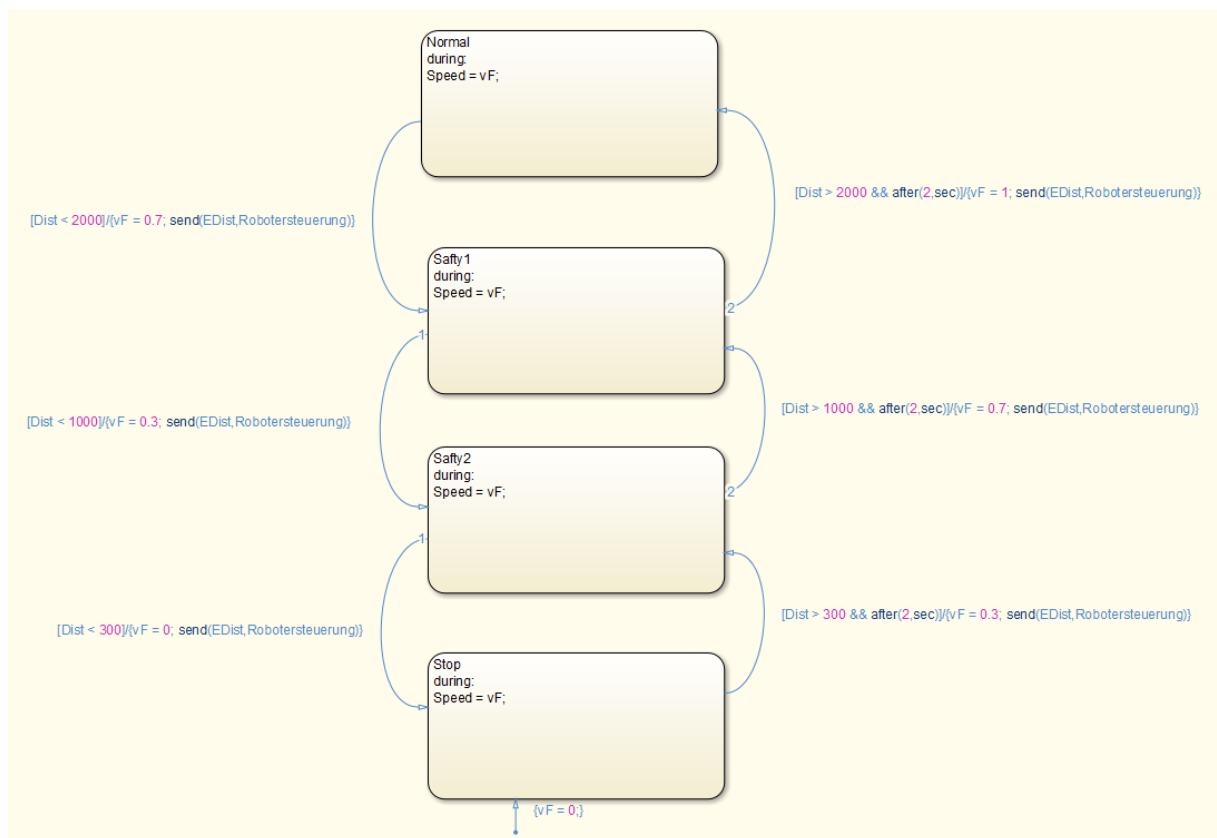


Abbildung 6-6: Stateflow Ablaufdiagramm der Distanzüberwachung

Im konkreten Fall wurden vier Zustände definiert, die jeweils eine Sicherheitsstufe mit entsprechend gedrosselter Geschwindigkeit repräsentieren. Über die „default-Transmission“ wird der Zustand „Stop“ aktiv gesetzt und gleichzeitig der Geschwindigkeitsfaktor  $vF = 0$  gesetzt. Somit ist sichergestellt, dass der Roboter erst losfährt, wenn ein ausreichender Sicherheitsabstand gewährleistet ist. Ein Wechsel in die niedrigeren Sicherheitsstufen erfolgt, wenn die Abstandsvariable *Dist* über einen bestimmten Wert steigt. Wenn eine solche Transmission passiert wird, wird der Geschwindigkeitsfaktor entsprechend verändert. In diesem Beispiel wird die Geschwindigkeit bei Unterschreiten von 2000mm, 1000mm und 300mm jeweils auf 0.7, 0,3 oder das 0-fache der maximal Geschwindigkeit gesetzt. Ein Wechsel in den schnelleren Zustand erfolgt beim Überschreiten der Abstandswerte und zusätzlich einem vorgeschriebenen „delay“ von 2 Sekunden, um ein Flackern zwischen den Zuständen zu verhindern.

Zur Überwachung des aktuellen Geschwindigkeitsfaktors wird im During-State die Ausgangsvariable *Speed* mit dem aktuellen Geschwindigkeitsfaktor  $vF$  belegt.

Die Kommunikation zwischen Subchart der Abstandsüberwachung mit der Robotersteuerung erfolgt über das Senden von Events. Wenn eine Transmission passiert wird, sich also der

Zustand ändert, wird das Event „EDist“ an den Zustand Robotersteuerung gesendet. Der Befehl hierfür lautet: „*send(EDist, Robotersteuerung)*;“

Auf der folgenden Seite wird nun das Subchart der Robotersteuerung dargestellt:

### 6.4.2 Robotersteuerung

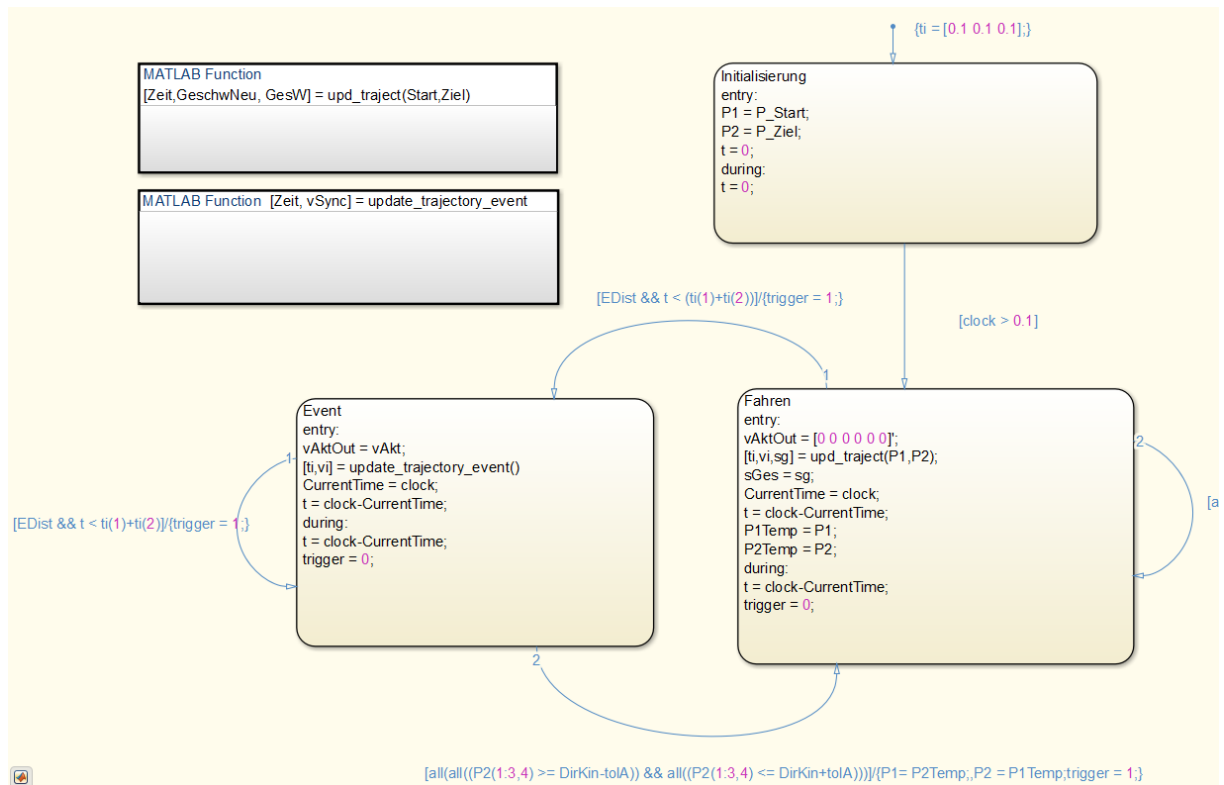


Abbildung 6-7: Stateflow Ablaufdiagramm der Robotersteuerung

Das Subchart enthält drei Zustände; neben dem Fahren und dem Event-Zustand gibt es noch einen Initialisierungsblock, der aber nur aus Simulations-Stetigkeitsaspekten vor die Hauptblöcke Fahren und Event geschaltet ist.

Bei **normaler Fahrt ohne Sensor-Events** wird die Roboterbewegung aus dem Block „Fahren“ gesteuert.

Im Entry Bereich werden folgende Funktionen ausgeführt:

1. Zu Anfang einer Fahrt befindet sich der Roboter in Ruhe  $\rightarrow vAkt = [0\ 0\ 0\ 0\ 0]$ .
2. In einem Stateflow Chart können Matlab-Funktionen aufgerufen werden. In diesem Schritt wird die Matlab Funktion *upd\_traject(P1, P2)* aufgerufen, die die Parameter  $t_i$  und  $v_i$  für das Trapezprofil, sowie die Gesamtlänge der Fahrt  $s_g$  bereitstellt. Die Formeln sind hier dieselben, wie bereits im Kapitel 5 zur vollsynchronen Bahnfahrt vorgestellt.
3. Über die lokale Variable *CurrentTime* wird ein Zeitstempel der Simulationszeit genommen.

4. Der Start und Zielpunkt wird in den temporären Variablen  $P1Temp$  und  $P2Temp$  gespeichert.

Im During-Bereich wird dann nur noch die Simulationszeit ausgegeben, die aber durch den Zeitstempel mit Beginn der Bahnfahrt auf den Wert Null gesetzt wird. Zusätzlich wird eine Trigger-Variable auf 0 gesetzt. Jedes Mal, wenn eine Transition passiert wird, wird diese Trigger Variable auf 1 gesetzt, was einen „Reset“ des Integrationsblocks zur Folge hat.

Wie in Abbildung 6-3 zu sehen, erhält der Zustandsautomat Rückführungen über den aktuellen Zustand des Roboters. Über das Signal der direkten Kinematik kann die Ist-Position des Endeffektors mit der Zielposition abgeglichen werden. Dieser Vergleich ist Bedingung für die „self-transition“, einer Transition, die wieder in den selben Block führt.

In der folgenden Abbildung ist die Bedingung für die Transition noch einmal dargestellt:

```
[all((P2(1:3,4) >= DirKin-toIA)) && all((P2(1:3,4) <= DirKin+toIA))]/{P1= P2Temp;,P2 = P1Temp;trigger = 1;}
```

Abbildung 6-8: Stateflow Transition Bedingung für Wendepunkt

Da es sich um eine numerische Simulation handelt, wird ein gewisser Toleranzbereich für den Zielpunkt zugelassen. Wird die Transition durchlaufen, werden die Aktionen in den geschweiften Klammern ausgeführt. Hier werden die Punkte  $P1$  und  $P2$  vertauscht und der Trigger für den Reset des Integrator-Blocks gesetzt. Da es sich um eine „self-transition“ handelt, beginnt der Zyklus nun wieder von vorn mit dem Durchlaufen des Entry-Bereichs des Fahren-Zustands und somit dem Definieren der Rückfahrt.

**Im Falle von Sensor-Events** wird aus dem Subchart „Abstandsüberwachung“ ein Event gesendet, dass im Subchart „Robotersteuerung“ die Transition zum Event-Zustand aktiviert. Als Extra-Bedingung ist lediglich definiert, dass sich der Roboter nicht schon sowieso im Abbremsvorgang befindet:

```
[EDist && t < (ti(1)+ti(2))]/{trigger = 1;}
```

Abbildung 6-9: Stateflow Transition Bedingung für Event

Der Zustandsblock „Event“ unterscheidet sich vom Aufbau nur marginal von dem des Blocks „Fahren“:

Im Entry Bereich wird die aktuelle Geschwindigkeit eingelesen, um dann in der Funktion `update_trajektory_event()` das neue Trapezprofil für eine Geschwindigkeitsreduktion oder Geschwindigkeitserhöhung zu definieren.

Analog zum Fahren-Zustand wird ein Zeitstempel genommen, um die Simulationszeit wieder von 0 starten zu können.

Der „Event-Zustand“ wird entweder über die Transition 2 verlassen, wenn der Zielpunkt erreicht wurde oder es wird während der Fahrt im Event-Zustand ein neues Event detektiert, sodass die Geschwindigkeit noch einmal korrigiert werden muss (Transition 1). Sollte der Zielpunkt erreicht worden sein, geht es wieder normal im „Fahren-Zustand“ weiter.

Im Folgenden werden die Formeln für die Ermittlung des neuen Geschwindigkeitsprofils im Event-Fall vorgestellt:

Es werden die Rest-Wege für alle Achsen und die Zeiten und Wege der Einzelkomponenten der Bewegung ermittelt: erste Verzögerung, konstante Fahrt, zweite Verzögerung.

$$s_{Rest} = s_{ges} - s_{Ist} \quad 6-1$$

$$t_1 = \frac{|v_{Neu} - v_{Akt}|}{a_{max}} \quad 6-2$$

$$t_2 = \frac{v_{Neu}}{a_{max}} \quad 6-3$$

$$s_1 = v_{neu} \cdot t_1 + 0.5 \cdot a_{max} \cdot t_1^2 \quad 6-4$$

$$s_2 = 0.5 \cdot a_{max} \cdot t_2^2 \quad 6-5$$

$$t_k = \frac{|s_{Rest}| - (s_1 + s_2)}{v_{neu}} \quad 6-6$$

Für eine vollsynchronen Fahrt werden die maximalen Segmentzeiten  $t_{1m}$ ,  $t_{km}$  und  $t_{2m}$  ermittelt. Aus dem Ansatz, dass die Rest-Strecke sich aus den einzelnen Segmentwegen zusammensetzt, lässt sich dann die Geschwindigkeit der vollsynchronen Fahrt für die einzelnen Teilbewegungen ermitteln:

$$s_k = s_{Rest} - s_1 - s_2 \quad 6-7$$

$$t_{km} \cdot v_{neu} = s_{Rest} - v_{neu} \cdot t_{1m} - 0.5 \cdot (v_{Akt} - v_{neu}) \cdot t_{1m} - 0.5 \cdot v_{neu} \cdot t_2 \quad 6-8$$

$$v_{neu} = \frac{s_{Rest} - 0.5 \cdot v_{Akt} \cdot t_{1m}}{t_{km} + t_{1m} + 0.5 \cdot t_{1m} + 0.5 \cdot t_{2m}} \quad 6-9$$

Mit den definierten Geschwindigkeiten und den Segmentzeiten sind alle Parameter gegeben, die der Bahngenerator braucht, um das Geschwindigkeitsprofil zu definieren.

## 6.5 Generieren von Abstandsdaten

Um eine abstandsabhängige Robotersteuerung zu realisieren, müssen über einen Laserscanner Abstandswerte eingelesen werden.

### 6.5.1 Laserscanner

Als Hardware wurde der 2D Laserscanner URG-04LX von der Firma Hokuyo verwendet.

Der Laserscanner liefert Abstandswerte in einem Winkelbereich von  $240^\circ$  und einer Schrittweite von  $0.36^\circ$  bei einer maximalen Reichweite von 4095mm.

In den folgenden Abbildungen ist der Laserscanner und sein Arbeitsbereich dargestellt.

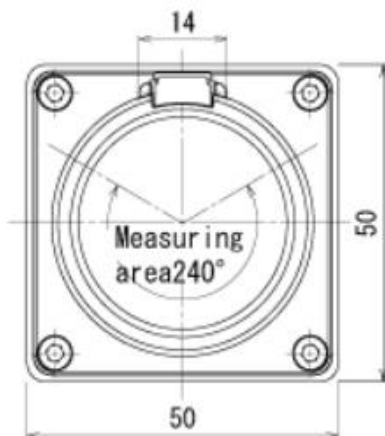


Abbildung 6-10: Messbereich Hokuyo Laserscanner  
Abbildung 6-11: Hokuyo Laserscanner

### 6.5.2 Schnittstelle zu Simulink

Für die Simulation müssen die ermittelten Abstandsdaten online in einer möglichst hohen Taktrate zur Verfügung gestellt werden.

Zuerst wurde der Laserscanner über eine open-source Toolbox aus dem Matlab Fileexchange mit Matlab/Simulink verbunden [Shrestha 2012]. Mit der Toolbox konnten allerdings nur Taktzeiten von über einer Sekunde realisiert werden, was für ein sicherheitsrelevantes System, das auf Echtzeit-Events reagieren soll, eine viel zu große Zeitspanne darstellt. Als Hauptursache für die langsame Auswertung wurde identifiziert, dass das Tool in Matlab geschrieben ist. Ein besseres Ergebnis würde ein C-Code liefern, der dann allerdings noch in Matlab/Simulink eingebunden werden müsste.

Eine Lösung des Problems konnte über das Einbinden einer Auslesesoftware vom Hersteller Hokuyo gefunden werden.



Mit dem Tool „UrgBenriStandard“ [Hokuyo, Data viewing tool] können Abstandswerte schnell eingelesen werden und in eine Datei geschrieben werden. Diese Datei wird dann über ein Matlab-Skript zum Lesen geöffnet und die relevanten Daten werden ausgelesen. So lassen sich Taktzeiten von unter 0.1 Sekunde realisieren.

Das Programm zum Auslesen der Daten geht immer ans Ende der Datei und liest dann die letzte Zeile ein. Dem Einlesealgorithmus ist bekannt, dass die Abstandswerte durch ein Semikolon getrennt sind.

Beispiel für diese Ausleseprogramme sind „ReadScanData.m“ oder „CreateEnviScan.m“.

Da diese Programme Matlab Funktionen, wie zum Beispiel *fseek* nutzen, können diese nicht in einem Simulink Block (Matlab Function) kompiliert werden. Eine Lösung bietet die *addListener* Konstruktion, bei der immer dann, wenn ein Simulink-Block seine Output-Function ausführt, ein Matlab-Skript gestartet wird. In diesem Matlab-Skript wird dann über den Befehl „*set\_parameter*“ der ermittelte Parameter in das Simulink-Modell implementiert. Die *addListener* Funktion muss in den Model-Properties unter den callback Startfunctions definiert sein. Die folgende Grafik visualisiert nochmal die Schnittstelle vom Laserscanner bis zum Simulink-Modell:

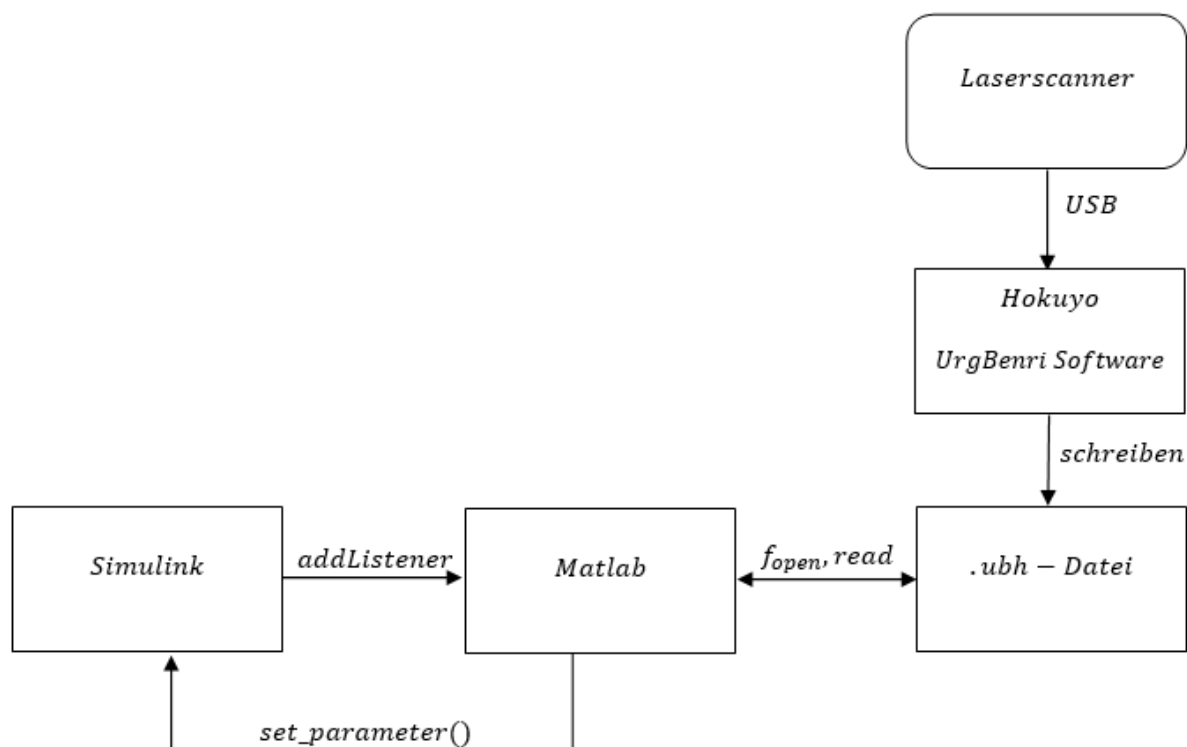


Abbildung 6-12: Aufbau der Schnittstelle Laserscanner und Simulink

### 6.5.3 Auswerten der Daten

Über die vorgestellte Schnittstelle wird somit im Simulink Modell mit einer Taktrate von 0.1 Sekunden ein Array mit 682 Distanzwerten zur Verfügung gestellt. Diese 682 Distanzwerte entsprechen der Auflösung von  $0,36^\circ$  auf einen Gesamtbereich von  $240^\circ$ .

Es muss nun ein Algorithmus entwickelt werden, der anhand der Daten erkennt, wenn sich ein Mensch dem Roboter nähert.

Dabei treten folgende Schwierigkeiten auf, die es zu lösen gilt:

1. Fest installierte Gegenstände, die dicht am Roboter stehen, dürfen das Ergebnis nicht beeinflussen
2. Messwert-Ausreißer müssen als solche erkannt und ignoriert werden
3. Bei einer Mittelwertbildung über mehrere Scans ist die Verzögerung zu beachten, mit der ein Verletzten einer Zone erkannt wird.

Folgendes Simulink-Blockschaltbild zeigt den Aufbau zur Distanzdatenauswertung:

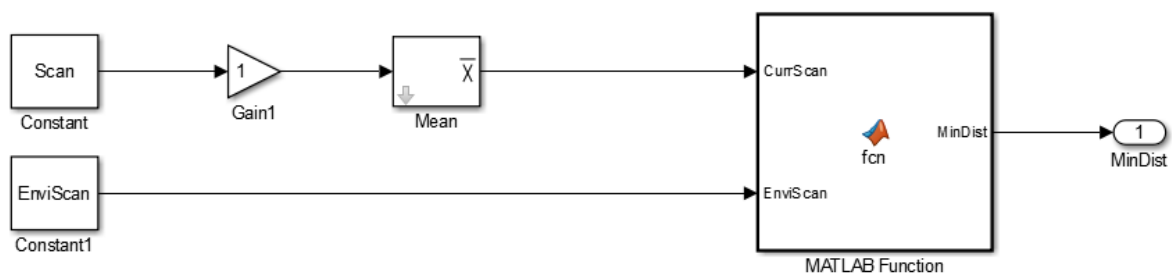


Abbildung 6-13: Simulink-Programm zur Auswertung der Laserscan-Daten

Zur Lösung des ersten Problems wird, bevor die Simulation startet, ein Referenzscan von der Umgebung ohne Menschen erstellt. So kann der Algorithmus zwischen fest installierten Objekten und neuen sich im Raum bewegend Objekten unterscheiden. Sollte im Laserscanbereich kein Objekt innerhalb der 4095 mm Reichweite detektiert werden, gibt der Laserscanner eine Null zurück. Diese Nullen werden vom Programm auf 4095 mm gesetzt. Da der Referenzscan vor der Simulation gemacht wird, gibt es hier keine Zeitbegrenzung, weswegen über 100 Scans gemittelt wird, um einen möglichst genauen Referenzscan zu erhalten.

Der laufende Scan, hier „current Scan“ genannt, unterliegt Messfehlern, die durch mehrere Verfahren ausgeglichen bzw. ignoriert werden können.

Es kann zwischen zwei Mittelwertverfahren unterschieden werden, die jeweils eine Glättung des Signals erwirken, wobei jedoch auch Informationen verloren gehen:

1. *Mitteln der Distanzwerte über die Zeit:*

Vorteil dieser Variante ist, dass die hohe Auflösung des Scans erhalten bleibt. Der erhebliche Nachteil ist, dass Ereignisse erst mit einer Verzögerung detektiert werden.

2. *Mitteln der Distanzwerte mit benachbarten Punkten:*

Vorteil dieser Methode ist, dass keine zeitliche Verzögerung beim Auswerten der Messwerte auftritt. Jedoch wird die Auflösung reduziert und scharfe Kanten werden verschmiert dargestellt.

Nach mehreren Versuchen ist als optimale Lösung ein Kompromiss aus beiden Verfahren entstanden: Über einen minimalen Zeitraum von einer zehntel Sekunde werden Distanzscans gemittelt und zudem wird jeder Distanzwert mit 2 benachbarten Punkten auf jeder Seite gemittelt.

Folgendes Bild zeigt das Signal für einen zufällig ausgewählten Winkel im Messbereich.

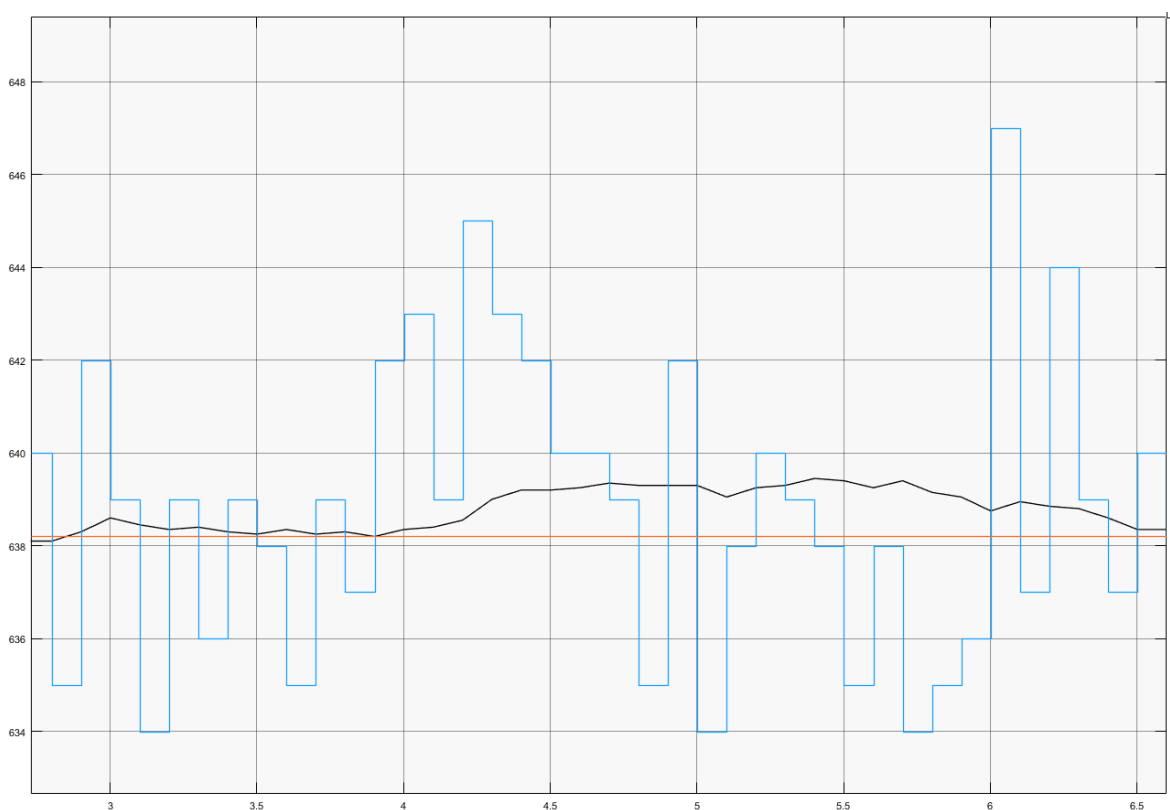


Abbildung 6-14: Zeitverlauf eines Messwertes des Laserscans

Rot dargestellt ist der Distanzwert aus dem Environment-Scan, also dem Referenzscan, der vor der Simulation aus 100 Scans erstellt wurde. Das blaue Signal ist das ungefilterte Signal und das schwarze das gemittelte Signal.

Das Grundprinzip des Algorithmus, der in Abbildung 6-13 in der Matlab-Funktion definiert ist, besteht darin, den Referenzscan mit dem aktuellen Scan zu vergleichen und die Indizes der Punkte, die eine Differenz um 50 mm überschreiten, zu speichern. 50 mm erscheinen zunächst

relativ viel, aber man muss bedenken, dass Menschen sich immer um 5 cm von den Umweltgegenständen abheben. Unter diesen ermittelten Indizes wird dann der kleinste Wert aus dem aktuellen Scan ermittelt.

Mit diesem Grundprinzip war das Ergebnis allerdings immer noch nicht ausreichend genau. Das Problem bestand darin, dass weitrhin noch einzelne Werte, die trotz Mittelwertverfahren ausgerissen sind, das Ergebnis beeinflussen konnten.

Zur Lösung wurde ausgenutzt, dass man bei einem Menschen als zu detektierendes Objekt davon ausgehen kann, dass ein bestimmter Winkelbereich vom Referenzscan abweichen muss. Dies führt dazu, dass zum Beispiel eine Fliege nicht mehr erkannt werden würde, ein Mensch jedoch schon. Der Winkelbereich von  $240^\circ$  wurde somit in 68 Sektionen eingeteilt, die jeweils  $3,5^\circ$  abdecken. Erst wenn alle Werte einer Sektion eine bestimmte Differenz zum Referenzscan aufweisen, wird die Sektion berücksichtigt und mit anderen relevanten Sektionen auf den minimalsten Wert verglichen.

Ein weiterer Vorteil dieser Methode ist, dass je näher man dem Laserscanner kommt, desto höher die Auflösung des Scans ist. Somit können dicht vor dem Roboter auch kleinere Bewegungen detektiert werden.

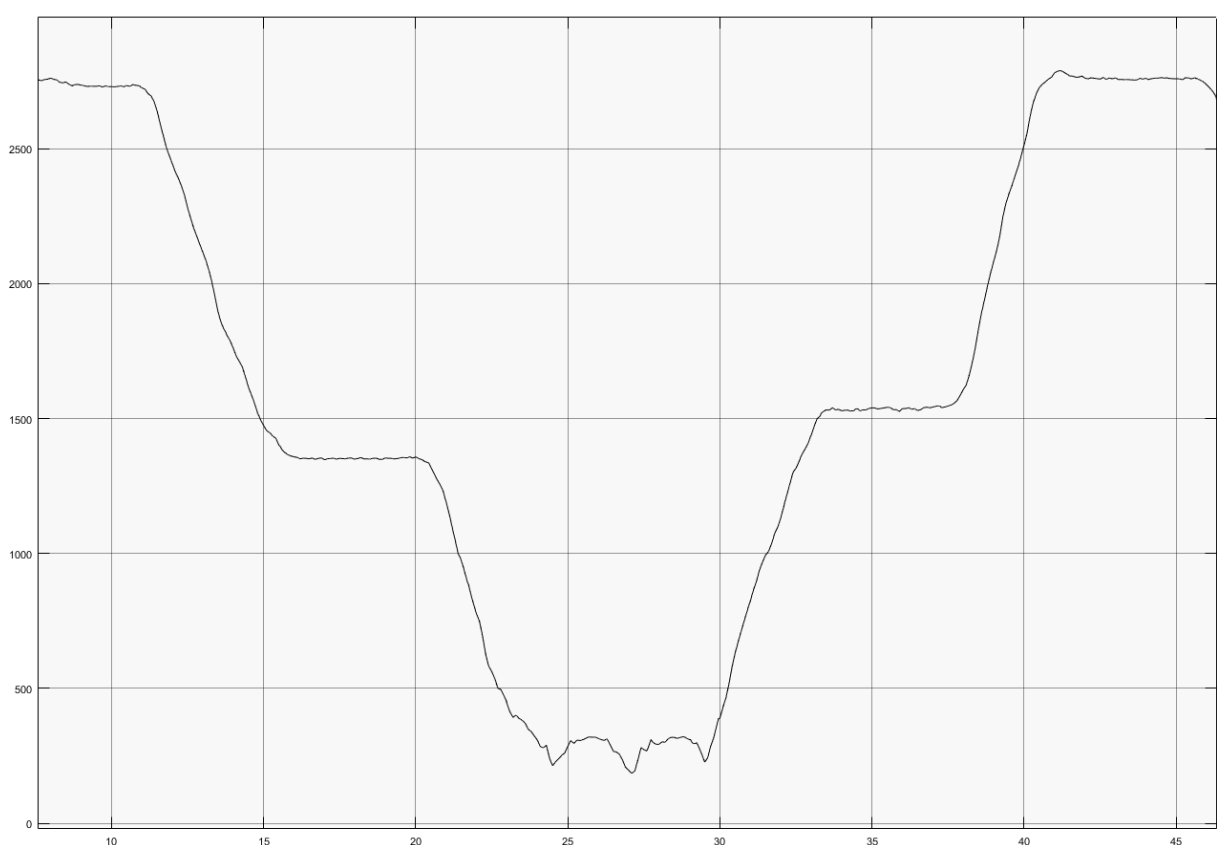


Abbildung 6-15: Abstandswerte eines sich nähernden Menschen

Die Abbildung 6-15 verdeutlicht diese Eigenschaft noch einmal: In einem Probescan wurde sich dem Laserscanner genähert und vor dem Körper auf Höhe des Scanners ein Stift vom

Körper weg und wieder zurückbewegt. Erst unmittelbar vor dem Scanner wird diese Bewegung erkannt.

## 6.6 Starten des Modells

Bevor das Advanced-Modell gestartet werden kann, muss das Programm zum Auslesen des Laserscanners gestartet werden. Dafür muss der Laserscanner verbunden werden (Button 1) und das „Data-recoding“ gestartet werden (Button 2). Als Datei ist die Datei „ScanData.ubh“ im Ordner der Simulink Datei auszuwählen.

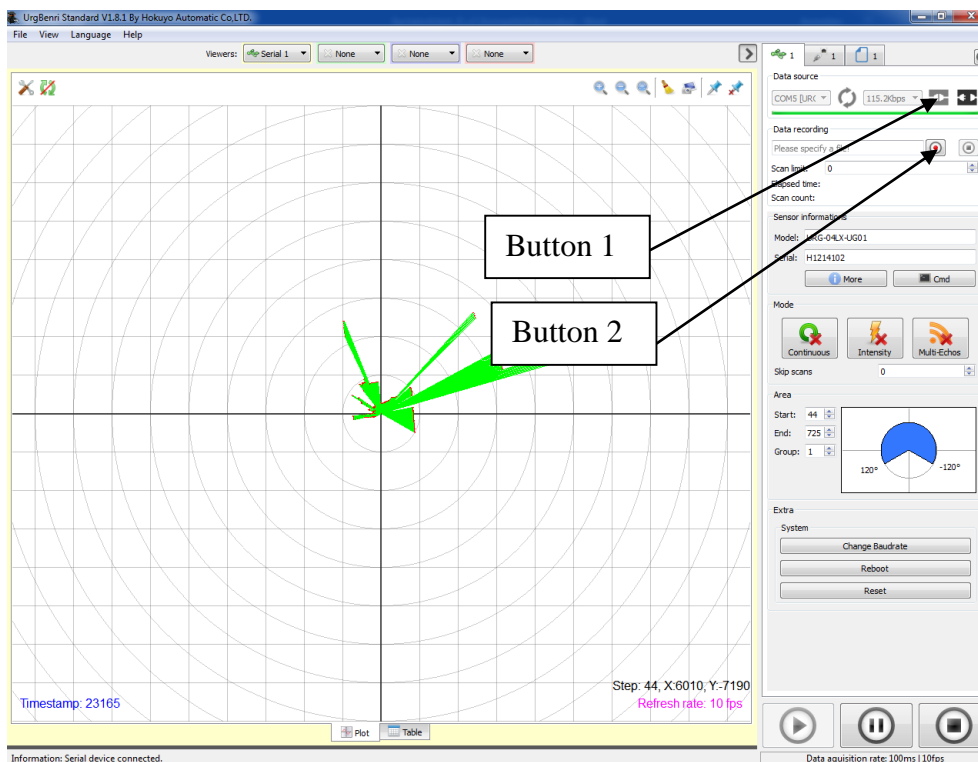


Abbildung 6-16: Laserscanner Tool: UrgBenriStandard

Als nächstes müssen im Initialisierungsskript „ini\_Automat.m“ die Parameter für die Bahnfahrt vorgegeben werden.

```

% ### --- Input --- ###

Start = [650; 0; 435];   % Startpunkt in [mm]
OriStart = [0,180,0];   % Orientierung am Startpunkt in [°]
kf1 = [0 1 0];         % [Fuß: 0->normal, Unterarm: 0->überstreckt, Handgelenk: 0->nach unten geklappt]

Ziel = [350; 0; 435];   % Zielpunkt
OriZiel = [0,90,90];
kf2 = [0 1 0];

BahnTyp = 1;           % 1: PTP; 2:CPL (noch nicht implementiert!)
GeschwPrf = 1;        % 1: Trapez; 2: Spline 3.Grades (noch nicht implementiert!)

maxV = [360 300 360 381 388 615]*pi/180/2; % maximale Winkelgeschwindigkeiten der Achsen
maxA = [1200 1200 1200 1200 1200 1200]*pi/180; % maximale Winkelbeschleunigungen der Achsen
tolA = 1;              % Tolleranz mit der ein Zielpunkt erreicht werden muss

% ###--- End Input --- ###

```

Abbildung 6-17: Matlab: Inputbereich des Initialisierungsskriptes

Danach kann das Simulink Modell „KUKA\_Agilus\_KR6\_AdvancedModell.slx“ gestartet werden.

In den Modelproperties sind in den Callbacks als Initialisierungsfunktionen das vom User definierte Skript „ini\_Automat.m“ vorgegeben und das Skript zum Erstellen des Referenzscans „CreateEnviScan“. Aus diesem Grund ist es auch wichtig, dass sich beim Starten des Programms kein bewegtes Objekt im überwachten Bereich befindet.

In der Startfunction der Callbacks wird dann noch der „Listener“ für das kontinuierliche Scansignal mit dem Skript „addListener.m“ aufgesetzt.

## 6.7 Testen des Modells

Während der Simulation wird der Roboter im SimMechanics Explorer grafisch animiert. Zusätzlich kann im Submodell „Bahnsteuerung → Controll-Interface“ der aktuelle Distanzwert und der zugehörige Geschwindigkeitsfaktor eingesehen werden.

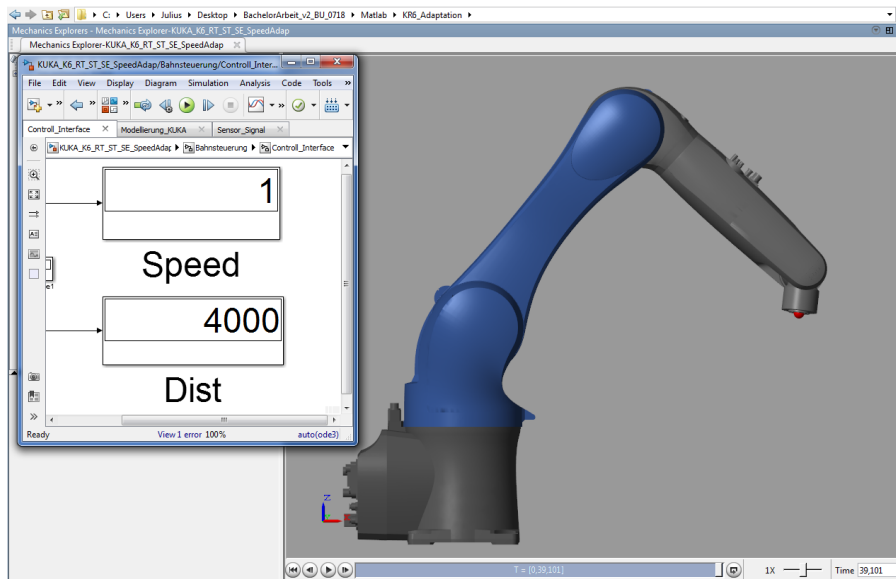


Abbildung 6-18: Simulation mit Kontroll-Submodell

Im Folgenden werden nun einige Simulationsverläufe zur Validierung der online Geschwindigkeitsadaption betrachtet. Als erste ist nach einer Fahrt auf höchster Geschwindigkeitsstufe eine Reduktion der Geschwindigkeit um eine Stufe dargestellt. Es werden die Geschwindigkeitsverläufe und der Geschwindigkeitsfaktor geplottet.

Der darauf folgende Plot zeigt eine Geschwindigkeitsreduktion von maximaler Geschwindigkeit bis zum Stillstand in einer Bewegung.

Im dritten Diagramm ist eine Annäherung durch eine Person bis zum Stillstand des Roboters und danach die Distanzierung der Person mit Wiederaufnahme der Bewegung des Roboters dargestellt. Zusätzlich zu den Geschwindigkeitsverläufen und dem Geschwindigkeitsfaktor wird der aktuelle Distanzwert dargestellt.

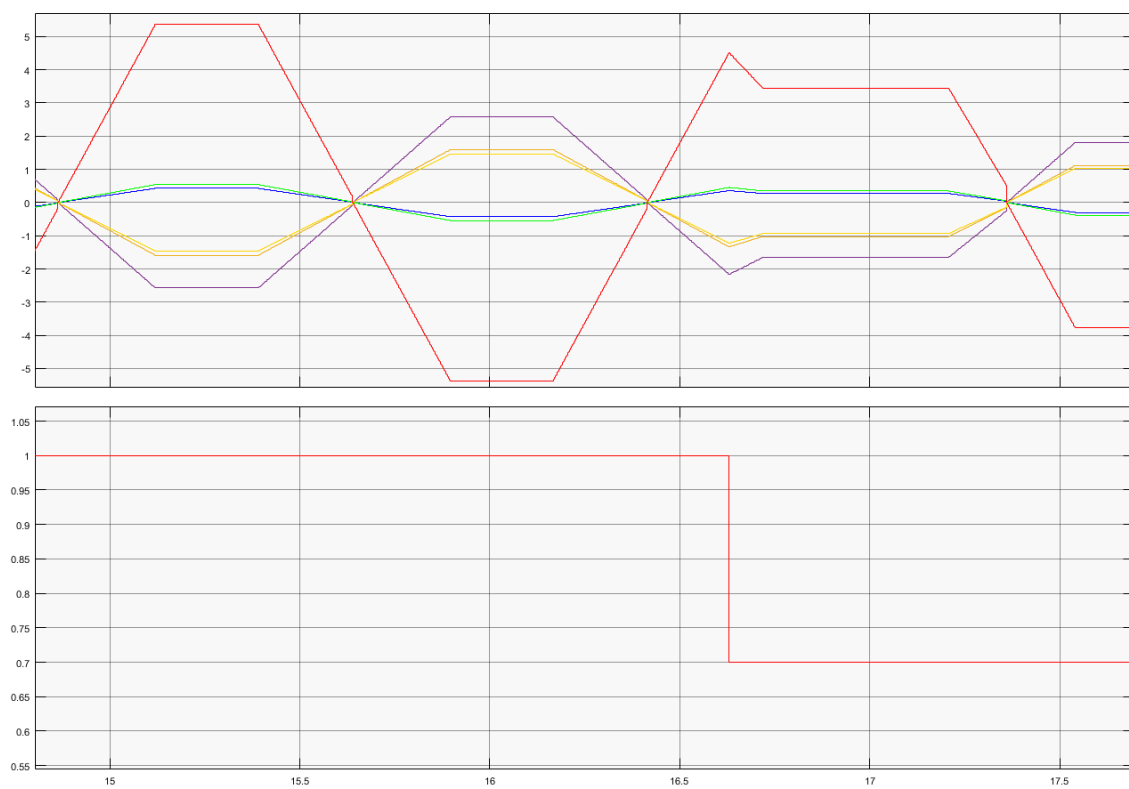


Abbildung 6-19: Geschwindigkeitsreduktion um eine Stufe

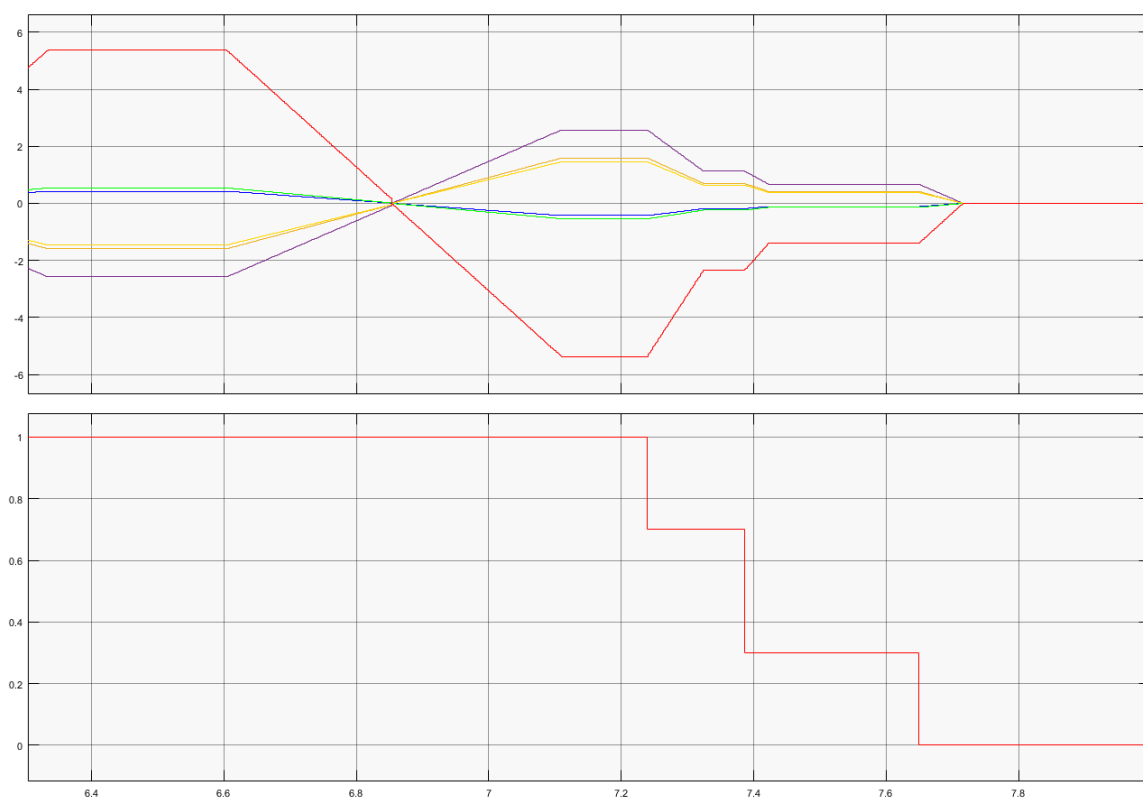


Abbildung 6-20: Geschwindigkeitsreduktion um 3 Stufen in einem Zyklus



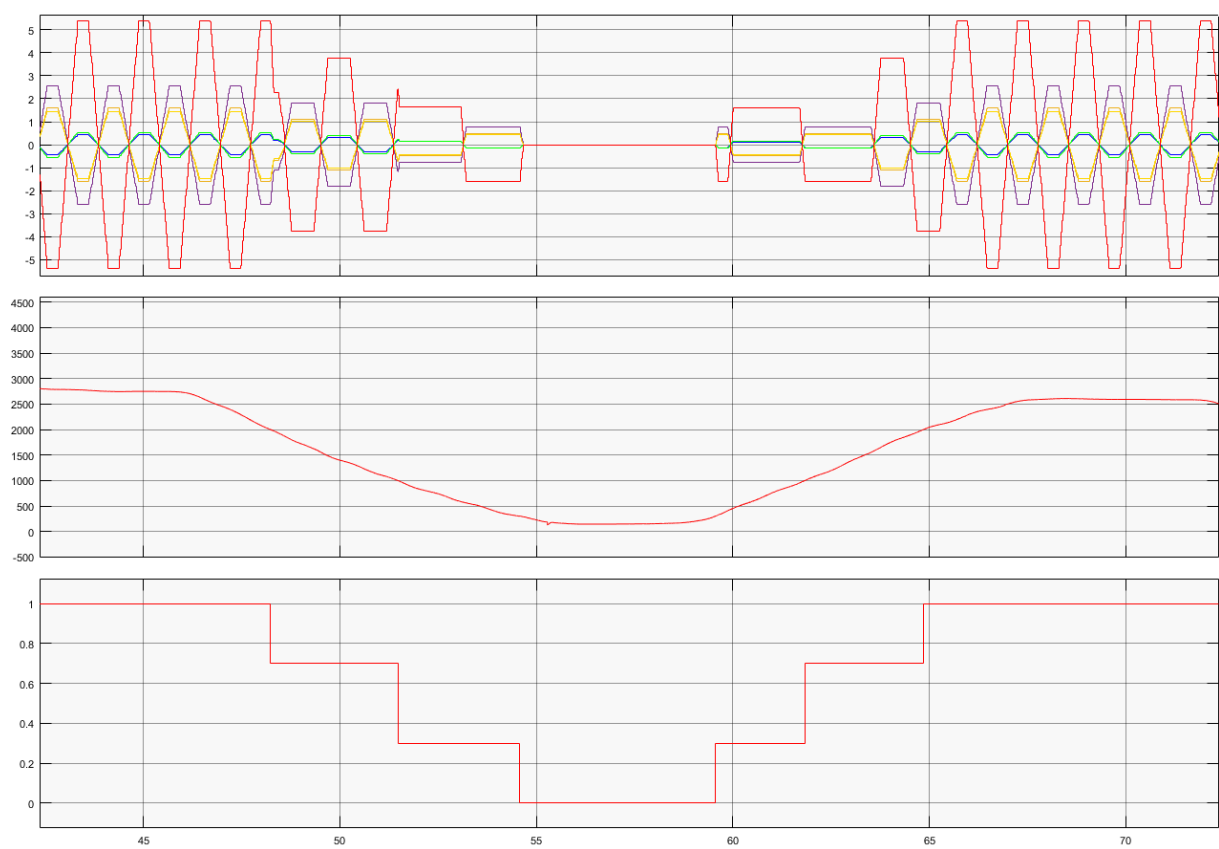


Abbildung 6-21: Auswertung bei sich bewegenden Objekten

## 7 Ausblick

Diese Bachelor-Arbeit stellt alle Komponenten für die Simulation des KUKA Agilus KR6 sixx zur Verfügung, die nötig sind, um das Roboterverhalten unter Einfluss von Umweltfaktoren zu simulieren.

Neben der Modellierung des Roboters mit CAD-Daten, kann das Simulationsmodell über direkte und indirekte Kinematik angesteuert werden. Die Robotersteuerung besitzt eine Schnittstelle zum Laserscanner und kann eine Zyklusbewegung des Roboters überwachen und steuern. Mit Hilfe des Stateflow-Zustandsautomaten ist es möglich, die Trajektorie unmittelbar bei detektierten Ereignissen anzupassen.

Mit diesem Modell als Basis können nun weitere Vorhaben realisiert werden. In einem nächsten Schritt sollte die Robotersteuerung dahingehend erweitert werden, dass mehrere Punkte hintereinander in einem Zyklus abgefahren werden können. Die bereits im Basis-Modell realisierten Verfahrenarten „linear“ und weitere Geschwindigkeitsprofile können in das Advanced-Modell übertragen werden.

Die Trajektorienadaption, die durch Sensorevents ausgelöst wird, kann mit komplexeren Algorithmen ausgestattet werden. So wird es möglich sein, dass der Roboter neben einer Geschwindigkeitsreduktion auch neue Punkte in die Trajektorie mitaufnimmt, um einen Mindestabstand zum Anwender einzuhalten.

Für ein anwenderfreundlicheres Nutzen des Simulationsprogramms müsste eine GUI-Oberfläche programmiert werden und zudem Kontrollalgorithmen implementiert werden, die zulässige Gelenkwinkel, den Arbeitsraum und singuläre Konfigurationen überprüfen. Langfristig ist es auch anzustreben, das Einlesen der Laserscan-Daten direkt über ein in Simulink programmiertes C-Programm zu realisieren und nicht den Umweg über das externe Tool von Hokuyo zu nehmen.

Abschließend sollte das Fernziel dieser Arbeit nicht aus den Augen verloren werden, die simulierten Trajektorien auf den realen Roboter übertragen zu können. Dafür müsste in einem ersten Schritt eine Schnittstelle von der Matlab/Simulink Umgebung auf die KUKA SimPro Software realisiert werden.

## Literaturverzeichnis

[Haddadin, 2014] Sami Haddadin, Towards safe robotics, Approaching Asimov's 1<sup>st</sup> Law, 1. Auflage: 2014, Springer-Verlag

[Biagiotti/ Melchiorri 2008] Dr. Luigi Biagiotti, Prof. Claudio Melchiorri, Trajectory Planning for Automatic Machines and Robots, 1.Auflage: 2008, Springer-Verlag

[Stark 2009] Georg Stark: Robotik mit Matlab, 1. Auflage: 2009, Carl-Hansa Verlag München

[Datenblatt KUKA 2016] Datenblätter KR6 R900 sixx, Url:  
[www.kuka-robotics.com/res/sps/6b77eeca-cfe5-42d3-b736-af377562ecaa\\_PB0001\\_KR\\_AGILUS\\_de.pdf](http://www.kuka-robotics.com/res/sps/6b77eeca-cfe5-42d3-b736-af377562ecaa_PB0001_KR_AGILUS_de.pdf)

[MathWorks SimMechanics] Simscape Multibody – Model and simulate multibody mechanical systems Url:  
[de.mathworks.com/products/simmechanics/](http://de.mathworks.com/products/simmechanics/)

[Lee / Ziegler 1983] C.S.G. Lee, M.Ziegler, a geometrical approach in solving the inverse kinematics of PUMA robots, Departement of Electrical and computer Engineering, The University of Michigan

[J.Olmski 1989] Jürgen Olmski, Fortschritte der Robotik – Band 4, Bahnplanung und Bahnführung von Industrierobotern, 1.Auflage: Vieweg & Sohn Verlagsgesellschaft mbH, Braunschweig

[Abo-Hammour, ALsmadi 2011] Za'er S. Abo-Hammour, Othman MK. Alsmadi Continous Gnetic Algorithims for Collision-Free Cartesian Path Planning of Robot Manipulators – Regular Paper , Departement of Mechatronics Engineering - University of Jordan

[Mathworks ModelEvent-Driven Systems] Model Event-Driven System Url:  
[de.mathworks.com/help/stateflow/ug/programming-your-chart-with-matlab-syntax.html](http://de.mathworks.com/help/stateflow/ug/programming-your-chart-with-matlab-syntax.html)

[Mathworks Starteflow] Introduction to Stateflow Url:  
[de.mathworks.com/video/introduction-to-stateflow-81549.html](http://de.mathworks.com/video/introduction-to-stateflow-81549.html)

[Shrestha 2012] S.Shrestha: Hokuyo URG-04LK Lidar Driver for Matlab, Mai 2012 –  
URL: <http://www.mathworks.com/matlabcentral/fileexchange/36700-hokuyo-urg-04lx-lidar-driver-for-matlab> – Abruf: 2015-08-27

[Hokuyo Data viewing tool] Tool zum Auslesen der Scandaten Url:  
<https://www.hokuyo-aut.jp/02sensor/07scanner/download/data/UrgBenri.htm>

## Anhang A

### A.1 Formeln und Funktionen zur Berechnung des Polynom-Geschwindigkeitsprofils

Im ersten Schritt werden die Segmentzeiten über die Funktion `ZeitFunkSpline(sg, vR).m` festgelegt. Da beim Polynom-Geschwindigkeitsprofil der Verlauf der Beschleunigung, der mindestens quadratisch ist, ohne Kenntnis der Beschleunigungszeit nicht exakt beschrieben werden kann, wird für die Ermittlung der Beschleunigungszeit ein iterativer Prozess vorgenommen. Die Segmentzeiten werden über eine Abschätzrechnung vorgegeben und dann wird geprüft, ob maximale Achsbeschleunigungen überschritten wurden. Im Falle einer CLN Bewegung ist dies sowieso notwendig, da hier in kartesischen Koordinaten gerechnet wird, aus denen sich dann die Achsbewegungen ableiten.

Der Abschätzalgorithmus ist wie folgt programmiert:

Über eine Richtgeschwindigkeit, wird die Zeit ermittelt, wenn der Weg ohne Beschleunigungs- und Verzögerungsphase zurückgelegt wird.

$$t_{k,fiktiv} = \frac{s_g}{v_R} \quad \text{A-1}$$

Danach wird ein definierter Anteil dieser Zeit für Beschleunigung und Verzögerung definiert:

$$t_b = t_v = 0.2 \cdot t_{k,fiktiv} \quad \text{A-2}$$

Mit diesem festgelegten Wert und der Richtgeschwindigkeit  $v_K$  kann nun ein Polynom definiert werden. Die Gleichungen ergeben sich aus den Werten an den Rändern und den Randbedingungen, dass das Polynom an den Grenzen  $t = 0$  und  $t = t_B$  die Steigung 0 hat.

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ t_b^3 & t_b^2 & t_b & 1 \\ 0 & 0 & 1 & 0 \\ 3t_b^2 & 2t_b & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix} = \begin{bmatrix} 0 \\ v_R \\ 0 \\ 0 \end{bmatrix} \quad \text{A-3}$$

Sobald das Polynom definiert ist, kann geprüft werden, ob Achsbeschleunigungen überschritten werden. Falls dieser Fall eintritt, muss die Zeit für den Beschleunigungsvorgang verlängert werden.

Über die Matlab-Funktion `polyint` lassen sich die Koeffizienten des integrierten Polynoms ausgeben.

$$q_{Koeff} = \text{polyint}(a_i)$$

Über die Funktion `polyval` kann nun die zurückgelegte Strecke während der Beschleunigung und Verzögerungsphase ermittelt werden:

$$s_b = s_v = \text{polyval}(q_{Koeff}, t_b)$$

Mit diesen Größen ergibt sich die Zeit für das konstante Segment zu

$$t_k = \frac{s_g - (s_v + s_b)}{v_R} \quad \text{A-4}$$

Die Zeiten der Teilbewegungen sind damit definiert. Um eine Vollsynchroner Ausführung der Bewegung zu erreichen werden für die PTP-Bewegung die maximalen Segmentzeiten aus den Teilbewegungen und für die CLN Bewegung die Segmentzeiten der Position für alle Teilbewegungen festgelegt. Daraus resultiert zwangsläufig, dass einige Teilbewegungen nun mit reduzierter Richtgeschwindigkeit ausgeführt werden.

Für die neue angepasste Geschwindigkeit wird sich folgende Eigenschaft der Polynomfunktion zu nutzen gemacht. Der Weg, also das Integral der Geschwindigkeitsfunktion, lässt sich auch durch eine lineare Trapezform exakt abbilden. So kann eine einfache Beziehung zwischen Zeiten, Weg und Geschwindigkeit aufgestellt werden:

$$s_g = \frac{1}{2} t_b \cdot v_M + t_k \cdot v_M + \frac{1}{2} t_v \cdot v_M = (t_b + t_k) \cdot v_M \quad \text{A-5}$$

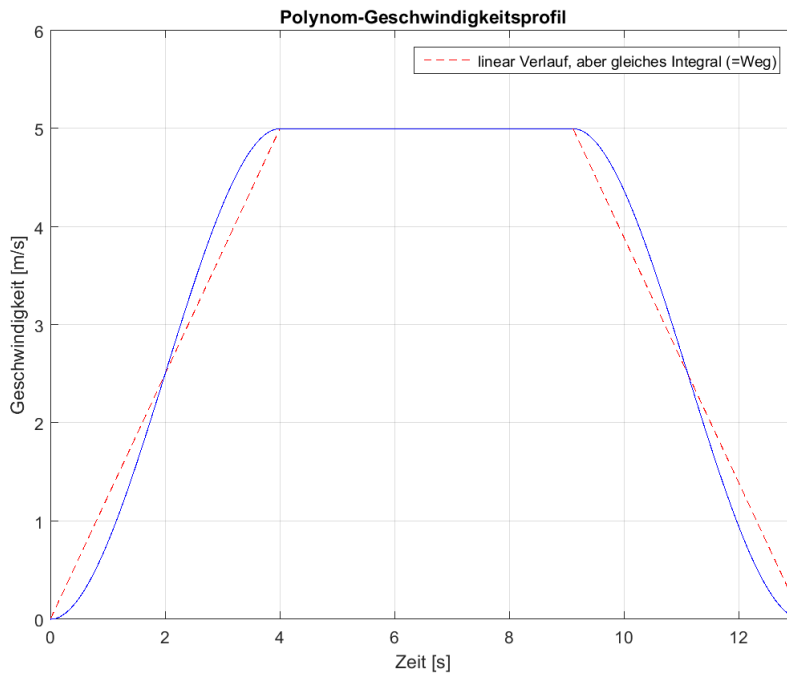


Abbildung A-1: Polynom Geschwindigkeitsprofil

Die neue Richtgeschwindigkeit, bei vorgegebenen Segmentzeiten und aus der Anwendung vorgegebenem Weg, ergibt sich zu:

$$v_{R,sync} = \frac{s_g}{t_{b,sync} + t_{k,sync}} \quad \text{A-6}$$

Mit der angepassten Richtgeschwindigkeit und den gegebenen synchronisierten Segmentzeiten können nun die Segmentlängen ermittelt werden. Dafür steht die Funktion  $WegFunkPolynom(tbm, tkm, tvn, vRsync)$  zur Verfügung. Die Funktion erstellt analog zum oben vorgestellten Lösungsweg Polynom-Funktionen und ermittelt über die integrierten Polynome die Längen der Segmente.

## A.2 Übersicht der Skripte und Funktionen

Name	Modell	Input	Output	Beschreibung
<b>HomogeneDHMatrixZX.m</b>	InDirKin BasisModell AdvaModell	$d, \theta, a, \alpha$	$A_{Matrix}$	Die Funktion generiert auf Basis der DH Parameter die Transformationsmatrix $A_{ij}$ , die die Transformation zwischen zwei Koordinatensystemen beschreibt.
<b>CreateFrame.m</b>	BasisModell AdvaModell	Pos,Ori	Frame	Die Funktion ermöglicht die Transformation zweier Schreibweisen zur Darstellung der Position und Orientierung eines KS. Input sind die kartesischen Koordinaten und die Orientierung in ZYX Eulerwinkeln. Output ist ein sogenanntes Frame: eine homogene 4x4 Matrix.
<b>CreateRotMat.m</b>	BasisModell AdvaModell	Rz,Ry,Rx	R	Diese Funktion erstellt nur ein Teil des Ergebnisses der CreateFrame.m Funktion. So wird hier nur die Orientierung von Eulerwinkeldarstellung in 3x3 Matrizendarstellung umgewandelt
<b>DirKin.m</b>	BasisModell AdvaModell	$q$	$x_{Ist}$	Mit Hilfe der in Kapitel 3.3 vorgestellten Formeln für die direkte Kinematik werden die Gelenkwinkel in die Umweltkoordinaten transformiert.
<b>InDirKin.m</b>	InDirKin BasisModell AdvaModell	Frame, Figur	$q$	Die Funktion InDirKin.m nutzt die Gleichung der analytisch geometrischen Indirekten Kinematik. Als Input werden ein Frame zur



				Bestimmung der Position und Orientierung des EE benötigt, sowie ein [3,1] Array zur Definition der Konfiguration. Output sind dann die Gelenkwinkel des Roboters
<b>Ori_Plan.m</b>	BasisModell	$P1, P2$ als Frame	$s_2, s_3$	Die Funktion ori_Plan.m ist für die Interpolation einer linearen Bewegung nötig. Mit ihr wird die Überführung der Orientierung von Start zu Zielpunkt realisiert. Genaue Erläuterungen finden sich in der Literatur von [Stark 2009].
<b>Robo_ini.m</b>	BasisModell	%	%	Das Skript ist das Initialisierungsskript für das Basis-Modell. In einem Input-Block werden Startpunkt, Zielpunkt mit der jeweiligen Konfiguration bestimmt. Zudem wird der Bahntyp, das Geschwindigkeitsprofil und der Geschwindigkeitsfaktor festgelegt. Außerhalb des Input-Blocks können noch Roboterspezifische Parameter wie die maximalen Gelenkgeschwindigkeiten und die maximalen Beschleunigungen definiert werden. Das Skript liefert dann die nötigen Parameter für die Generierung des Geschwindigkeitsprofils.
<b>WegFunk_Spline.m</b>	BasisModell	$t_b, t_k, t_v, v_m$	$s_b, s_k s_v, q_b, q_v$	Die Funktion wird im Robo_ini.m Skript verwendet um Segmentlängen der Polynombewegung zu ermitteln und die Koeffizienten der Funktionen zu ermitteln.

<b>ZeitFunk_Spline.m</b>	BasisModell	$s, v_m$	$t_b, t_k, t_v$	Die Funktion wird im Robo_ini.m Skript verwendet um Segmentzeiten der Polynombewegung zu ermitteln.
<b>ZeitFunk_Trapez.m</b>	BasisModell	$s, v_m, a_m$	$t_b, t_k, t_v$	Die Funktion wird im Robo_ini.m Skript verwendet, um Segmentzeiten der Trapezbewegung zu ermitteln.
<b>addList.m</b>	AdvaModell	%	%	Skript zum Erstellen eines Listener-Objekts, das bei jedem Simulationsblock aktiv wird.
<b>CreateEnviScan.m</b>	AdvaModell	%	%	Wird vor der Simulation gestartet und erstellt 100 Scans der Umgebung, die dann als Referenzscan an das Modell übergeben werden.
<b>Ini_Automat</b>	AdvaModell	%	%	Initialisierungsskript für das Advanced Modell. Hier werden Start und Zielpunkt sowie deren Konfiguration festgelegt. Als Bahntyp und Bahn-geschwindigkeitsprofil sind erstmal nur PTP und Trapez zulässig.
<b>ReadScanData</b>	AdvaModell	%	%	Script, dass während der Simulation laufend Scan-Daten einliest und über die Funktion set_parameter ins Simulink-Modell überträgt.

## A.3 Matlab Programme

### HomogeneDHMatrixZX

```
function A01 = HomogeneDHMatrixZX(d,theta,a,alpha)
A01 = [cos(theta),-sin(theta)*cos(alpha), sin(theta)*sin(alpha) , a*cos(theta);...
      sin(theta), cos(theta)*cos(alpha), -cos(theta)*sin(alpha), a*sin(theta);...
      0 , sin(alpha) , cos(alpha) , d ;...
      0 , 0 , 0 , 1];
```

### CreateFrame

```
function Frame = CreateFrame(Pos,Ori)

% Erstellen der Rotationsmatix R des Endeffektors
Rz = Ori(1)*pi/180;
Ry = Ori(2)*pi/180;
Rx = Ori(3)*pi/180;

MRz = [cos(Rz) -sin(Rz) 0; sin(Rz) cos(Rz) 0; 0 0 1];
MRy = [cos(Ry) 0 sin(Ry);0 1 0;-sin(Ry) 0 cos(Ry)];
MRx = [1 0 0;0 cos(Rx) -sin(Rx); 0 sin(Rx) cos(Rx)];

R = MRz*MRy*MRx;
Frame = [R,Pos; 0 0 0 1];
```

### CreateRotMatrix

```
function R = CreateRotMat(Rz,Ry,Rx)

% Erstellen der Rotationsmatix R des Endeffektors

MRz = [cos(Rz) -sin(Rz) 0; sin(Rz) cos(Rz) 0; 0 0 1];
MRy = [cos(Ry) 0 sin(Ry);0 1 0;-sin(Ry) 0 cos(Ry)];
MRx = [1 0 0;0 cos(Rx) -sin(Rx); 0 sin(Rx) cos(Rx)];

R = MRz*MRy*MRx;
```

### DirKin

```
function xIst = DirKin(q)

L01z = 400;
L01x = 25;
L12x = 455;
L23x = 35;
L34z = 420;
L56z = 80;
%DH Parameter
% % Die folgende Matrix enthält die DH Parmeter des Roboters ?
% %
Matrix = [ L01z , -q(1) , L01x , -pi/2 ;
          0 , q(2) , L12x , 0 ;
          0 , q(3)-pi/2 , L23x , -pi/2 ;
          L34z , -q(4) , 0 , pi/2 ;
          0 , q(5) , 0 , -pi/2 ;
          L56z , -q(6)+pi , 0 , 0 ];

d = Matrix(:,1);
theta = Matrix(:,2);
a = Matrix(:,3);
alpha = Matrix(:,4);
```

```

i = 1;
A01 = [cos(theta(i)), -sin(theta(i))*cos(alpha(i)), sin(theta(i))*sin(alpha(i)), a(i)*cos(theta(i));...
       sin(theta(i)), cos(theta(i))*cos(alpha(i)), -cos(theta(i))*sin(alpha(i)), a(i)*sin(theta(i));...
       0, sin(alpha(i)), cos(alpha(i)), d(i);...
       0, 0, 0, 1];

i = 2;
A12 = [cos(theta(i)), -sin(theta(i))*cos(alpha(i)), sin(theta(i))*sin(alpha(i)), a(i)*cos(theta(i));...
       sin(theta(i)), cos(theta(i))*cos(alpha(i)), -cos(theta(i))*sin(alpha(i)), a(i)*sin(theta(i));...
       0, sin(alpha(i)), cos(alpha(i)), d(i);...
       0, 0, 0, 1];

i = 3;
A23 = [cos(theta(i)), -sin(theta(i))*cos(alpha(i)), sin(theta(i))*sin(alpha(i)), a(i)*cos(theta(i));...
       sin(theta(i)), cos(theta(i))*cos(alpha(i)), -cos(theta(i))*sin(alpha(i)), a(i)*sin(theta(i));...
       0, sin(alpha(i)), cos(alpha(i)), d(i);...
       0, 0, 0, 1];

i = 4;
A34 = [cos(theta(i)), -sin(theta(i))*cos(alpha(i)), sin(theta(i))*sin(alpha(i)), a(i)*cos(theta(i));...
       sin(theta(i)), cos(theta(i))*cos(alpha(i)), -cos(theta(i))*sin(alpha(i)), a(i)*sin(theta(i));...
       0, sin(alpha(i)), cos(alpha(i)), d(i);...
       0, 0, 0, 1];

i = 5;
A45 = [cos(theta(i)), -sin(theta(i))*cos(alpha(i)), sin(theta(i))*sin(alpha(i)), a(i)*cos(theta(i));...
       sin(theta(i)), cos(theta(i))*cos(alpha(i)), -cos(theta(i))*sin(alpha(i)), a(i)*sin(theta(i));...
       0, sin(alpha(i)), cos(alpha(i)), d(i);...
       0, 0, 0, 1];

i = 6;
A56 = [cos(theta(i)), -sin(theta(i))*cos(alpha(i)), sin(theta(i))*sin(alpha(i)), a(i)*cos(theta(i));...
       sin(theta(i)), cos(theta(i))*cos(alpha(i)), -cos(theta(i))*sin(alpha(i)), a(i)*sin(theta(i));...
       0, sin(alpha(i)), cos(alpha(i)), d(i);...
       0, 0, 0, 1];

A06 = A01*A12*A23*A34*A45*A56;

R = A06(1:3, 1:3);
xT = A06(1:3, 4);

RotA = atan2( R(2,1), R(1,1) ); %Drehung um z
RotB = - asin( R(3,1) ); %Drehung um y
RotC = atan2( R(3,2), R(3,3) ); %Drehung um x

xR = [RotC; RotB; RotA];
xIst = [xT; xR];

```

## InDirKin

```

function q = InDirKin(Frame, Figur)

% Übersetzung der Figur in Rechenindex -1 und 1
if Figur(1) == 0
    Figur(1) = -1;
end
if Figur(2) == 0
    Figur(2) = -1;
end
if Figur(3) == 1
    Figur(3) = -1;
else
    Figur(3) = 1;
end

% geometrische Daten des Roboters
L01z = 400; % Höhe des Fusses
L01x = 25; % Versatz Achse 1 / Achse 2
LOA = 455; % Länge Oberarm

```

```

L23x = 35;      % Versatz Gelenk 3 - Unterarm
LUA = 420;     % Länge Unterarm
L56z = 80;     % Abstand Gelenk 5/6 - EE (Flansch)

% 1) Erstellen der Rotationsmatrix R des Endeffektors
A06 = Frame;
Pos = A06(1:3,4);
R06 = A06(1:3,1:3);

% 2) Bestimmen des Punktes P
a = A06(1:3,3);      % Richtungsvektor a der Hand
P = Pos - L56z * a;

% 3) Bestimmen der Gelenkwinkel
% 3.1) ### --- q1 --- ###

    q1 = Figur(3)*-atan2(P(2),Figur(3)*P(1));
    q1_Grad = q1*180/pi;

% 3.2) ### --- q2 --- ###
%Versatz zeigt in Richtung Zielpunkt: Figur = 1 oder Versatz zeigt von
Zielpunkt weg: Figur = -1;

d = L01z;
theta = -q1;
a = L01x;
alpha = 0;

A01rr = HomogeneDHMatrixZX(d,theta,a,alpha);

PReal = inv(A01rr)*[P;1];
rr = sqrt(PReal(1)^2+PReal(2)^2+PReal(3)^2);

deltaS = PReal(3)/rr;
deltaC = sqrt(PReal(1)^2+PReal(2)^2)/rr;
delta = atan2(deltaS,deltaC);

gammaC = (LOA^2+rr^2-(L23x^2+LUA^2))/(2*LOA*rr);
gammaS = sqrt(1-gammaC^2);
gamma = atan2(gammaS,gammaC);

if Figur(3) == 1
    q2 = -(delta + Figur(2)*Figur(3)*gamma);
    q2_Grad = q2*180/pi;
else
    q2 = -(pi-((delta + Figur(2)*Figur(3)*gamma)));
    q2_Grad = q2*180/pi;
end

% 3.3) ### --- q3 --- ###
thetaC = (LOA^2+(L23x^2+LUA^2)-rr^2)/(2*LOA*sqrt(L23x^2+LUA^2));
thetaS = sqrt(1-thetaC^2);

if Figur(3) > 0
    if Figur(2) > 0
        theta = 2*pi-atan2(thetaS,thetaC);      % stumpfer Winkel
        thetaGrad = theta*180/pi;
    else
        theta = atan2(thetaS,thetaC);          % spitzer Winkel
        thetaGrad = theta*180/pi;
    end
end

```

```

    end
else if Figur(3) < 0
    if Figur(2) > 0
        theta = 2*pi-atan2(thetaS,thetaC);    % stumpfer Winkel
        thetaGrad = theta*180/pi;
    else
        theta = atan2(thetaS,thetaC);        % spitzer Winkel
        thetaGrad = theta*180/pi;
    end
end
end

betaS = LUA/sqrt(L23x^2+LUA^2);
betaC = L23x/sqrt(L23x^2+LUA^2);

beta = atan2(betaS,betaC);
betaGrad = beta*180/pi;

q3 = -(pi/2-(theta-beta));
q3_Grad = q3*180/pi;

% 3.5) ### --- q5 --- ###
%Berechnen von n, o, a

d = L01z;
theta = -q1;
a = L01x;
alpha = -pi/2;

A01 = HomogeneDHMatrixZX(d,theta,a,alpha);

d = 0;
theta = q2;
a = LOA;
alpha = 0;

A12 = HomogeneDHMatrixZX(d,theta,a,alpha);

d = 0;
theta = q3-pi/2;
a = L23x;
alpha = -pi/2;

A23 = HomogeneDHMatrixZX(d,theta,a,alpha);
A03 = A01*A12*A23;

R03 = A03(1:3,1:3);
R36 = inv(R03)*R06;
n = R36(1:3,1);
o = R36(1:3,2);
a = R36(1:3,3);

y5 = sqrt(a(1)^2+a(2)^2);
q5 = -atan2(y5,a(3));
q5_Grad = q5*180/pi;

TestVar = Figur(1)*q5;
Umkehr = 1;
if TestVar > 0
    Umkehr = -1;
end

```

```

    q5 = -q5;
    q5_Grad = q5*180/pi;
end

%3.4) ### --- q4 --- ###
q4 = -atan2(Umkehr*a(2),Umkehr*a(1));
q4_Grad = q4*180/pi;

% 3.6) ### --- q6 --- ###
yn1 = -n(1)*sin(-q4)+n(2)*cos(-q4);
yo1 = -o(1)*sin(-q4)+o(2)*cos(-q4);

q6 = atan2(yn1,-yo1);
q6_Grad = q6*180/pi;
q = [q1,q2,q3,q4,q5,q6];

```

## oriPlan

```

function [s2, s3] = ori_Plan(P1,P2)

kleinsterWinkel = pi/1000;

z1 = P1(1:3,3);
z2 = P2(1:3,3);

dvz = cross(z1,z2);
s2 = acos(dot(z1,z2));

PO1 = P1;
PO1(1:3,4) = [0 0 0]';

if s2 > kleinstWinkel
    POZ = rotv(dvz,s2)*PO1;
else
    POZ = PO1;
end

x1_z = POZ(1:3,1);
x2 = P2(1:3,1);

s3 = acos(dot(x1_z,x2));

```

## Robo\_ini

```

clear all

% ### --- Input --- ###
bs.Start = [650; 0; 435]; % Startpunkt
OriStart = [0,180,0];
kf1 = [0 1 0]; % [Fuß: 0->normal, Unterarm: 0->überstreckt, Handgelenk: 0->nach unten geklappt]

bs.Ziel = [400; 0; 435]; % Zielpunkt
OriZiel = [0,90,90];
kf2 = [0 1 0];

```

```

BahnTyp = 1; % 1: PTP; 2:CPL
GeschwPrf = 1; % 1: Trapez; 2: Spline 3.Grades
bs.geschw = 0.8; % Geschwindigkeitsparameter

% ###--- End Input --- ###

% Robo Parameter
robo.vm = [360 300 360 381 388 615]*pi/180;
robo.am = [1200 1200 1200 1200 1200 1200]*pi/180/100;
posv = 100;
posa = 100;
oriv = 100*pi/180;
oria = 100*pi/180;

posv_prg = posv * bs.geschw;
oriv_prg = oriv * bs.geschw;
vm_prog = robo.vm*bs.geschw;
am = robo.am;

% Start und Ziel Frame erzeugen
P1 = CreateFrame(bs.Start,OriStart);
P2 = CreateFrame(bs.Ziel,OriZiel);

sg = zeros(1,6);

% Winkel oder Weg Differenzen definieren
if BahnTyp == 1
    Q1 = InDirKin(P1,kf1);
    Q2 = InDirKin(P2,kf2);
    sg = Q2-Q1;

elseif BahnTyp == 2
    u1 = bs.Start;
    u2 = bs.Ziel;
    sg(1) = norm(u2-u1);
    [sg(2), sg(3)] = ori_Plan(P1,P2);
end

tb = zeros(1,6);
tk = zeros(1,6);
tv = zeros(1,6);
sb = zeros(1,6);
sk = zeros(1,6);
sv = zeros(1,6);
qb = zeros(5,6);
qv = zeros(5,6);

% ### --- Trapezprofil --- ###

if GeschwPrf == 1

    if BahnTyp == 1 % PTP
        for n = 1:6
            [tb(n), tk(n), tv(n)] =
ZeitFunk_Trapez(sg(n),vm_prog(n),am(n));
        end

        tbm = max(tb);
    end
end

```



```

tkm = max(tk);
tvm = max(tv);

sb = sg*tbm/(2*(tbm+tkm));
sk = sg*tkm/(tbm+tkm);
sv = sb;

elseif BahnTyp == 2           %CLN

    [tb(1), tk(1), tv(1)] = ZeitFunk_Trapez(sg(1),posv_prg,posa);
    [tb(2), tk(2), tv(2)] = ZeitFunk_Trapez(sg(2),oriv_prg,oria);
    [tb(3), tk(3), tv(3)] = ZeitFunk_Trapez(sg(3),oriv_prg,oria);

    tbm = tb(1);               % Zeiten richten sich nach der Weg
    Koordinate s
    tkm = tk(1);
    tvm = tv(1);

    sb = sg*tbm/(2*(tbm+tkm));
    sk = sg*tkm/(tbm+tkm);
    sv = sb;
end

% ### --- Spline-Profil --- ###
elseif GeschwPrf == 2

    if BahnTyp == 1           % PTP

        for n = 1:6;
            [tb(n), tk(n), tv(n)] = ZeitFunk_Spline(sg(n),vm_prog(n));
        end

        tbm = max(tb);
        tkm = max(tk);
        tvm = max(tv);

        vm = sg./(tbm+tkm);

        for i = 1:6
            [sb(i), sk(i), sv(i), qb(:,i), qv(:,i)] =
WegFunk_Spline(tbm,tkm,tvm,vm(i));
        end

    elseif BahnTyp == 2       % CLN

        for n = 1:3;
            [tb(n), tk(n), tv(n)] = ZeitFunk_Spline(sg(n),posv_prg);
        end

        tbm = tb(1);         % Zeiten richten sich nach der Weg
        Koordinate s
        tkm = tk(1);
        tvm = tv(1);

        vm = sg./(tbm+tkm);

        for i = 1:3
            [sb(i), sk(i), sv(i), qb(:,i), qv(:,i)] =
WegFunk_Spline(tbm,tkm,tvm,vm(i));

```

```

        end
    end
end

tg = tbm+tkm+tv;

```

## WegFunk\_Spline

```

function [sb, sk, sv, qb, qv] = WegFunk_Spline(tb,tk,tv,vm)

% Ermitteln der Polynomkoeffizienten für die Anfahrt und Abbremsbewegung
M = [ 0      0      0 1;
      tb^3 tb^2  tb 1;
      0      0      1 0;
      3*tb^2 2*tb 1 0];

Ergb = [0;vm;0;0];

ab = M\Ergb;

qb = polyint(ab');
sb = polyval(qb,tb);
sv = sb;

sk = vm*tk;

Ergv = [vm; 0; 0; 0];

av = M\Ergv;
qv = polyint(av');
end

```

## ZeitFunk\_Spline

```

function [tb, tk, tv] = ZeitFunk_Spline(s,vm)

s = abs(s);

% Zeit für tk ohne Anfahren und Bremsen ermitteln
tk_p = s/vm;

% Festlegen der Zeit für Anfahrvorgang
tb = 0.3*tk_p;
tv = tb;

% Ermitteln des Wegs der beim Anfahrvorgang zurückgelegt wird
Mb = [ 0      0      0 1;
      tb^3 tb^2  tb 1;
      0      0      1 0;
      3*tb^2 2*tb 1 0];

Ergb = [0;vm;0;0];

ab = Mb\Ergb;

qb = polyint(ab');
sb = polyval(qb,tb);
sv = sb;

```

```
% Genaue Bestimmung der Zeit des konstanten Segments
tk = (s-(2*sb))/vm;
```

## ZeitFunk\_Trapez

```
function [tb, tk, tv] = ZeitFunk_Trapez(s,vm,am)

% Ermitteln der Wege und Zeiten der Segmente: Anfahren,Konstante,
Abbremsen

sg = abs(s);

skrit = vm^2/am;

if sg-skrit>eps
    tb = vm/am;
    tv = tb;
    tk = sg/vm-vm/am;
else
    tb = sqrt(sg/am);
    tv = tb;
    tk = 0;
end
```

## addList

```
blk1 = 'KUKA_Agilus_KR6_AdvancedModell/Bahnsteuerung/Sensor_Signal/Gain1';
% Schreibt die Werte von q1 aus dem Bock:
% 'modell/Sub1/q1' aus der Simulation
h1 = add_exec_event_listener(blk1, 'PreOutputs', @ReadScanData);
```

## CreateEnviScan

```
global EnviScan

DistMatrix = [];

for n = 1:100
    ScanFile = fopen('ScanData.ubh','r');
    lastLine = '';
    offset = 1;
    fseek(ScanFile,-offset,'eof');
    newChar = fread(ScanFile,1);

    while (~strcmp(newChar,char(10))) || (offset == 1)
        offset = offset+1;
        fseek(ScanFile,-offset,'eof');
        newChar = fread(ScanFile,1,'*char');
    end

    text = fread(ScanFile,'uint8=>char');
    DistCell = textscan(text,'%d','delimiter',';');
    DistArray = double(cell2mat(DistCell));

    DistArray(DistArray == 0) = 4095;
    DistArray = medfilt1(DistArray,5);
```

```

if length(DistArray) == 682
    DistMatrix = [DistMatrix; DistArray];
end

minDist = min(DistArray);
fclose(ScanFile);

end

EnviScan = mean(DistMatrix);

% Plot
% NumEl = length(AveDist);
% deg = (-NumEl/2:1:NumEl/2-1)*240/NumEl*pi/180;
% polar(deg,AveDist)

```

## iniAutomat

```

% ### --- Input --- ###

Start = [650; 0; 435]; % Startpunkt in [mm]
OriStart = [0,180,0]; % Orientierung am Startpunkt in [°]
kf1 = [0 1 0]; % [Fuß: 0->normal, Unterarm: 0->überstreckt,
Handgelenk: 0->nach unten geklappt]

Ziel = [350; 0; 435]; % Zielpunkt
OriZiel = [0,90,90];
kf2 = [0 1 0];

BahnTyp = 1; % 1: PTP; 2:CPL (noch nicht implementiert!)
GeschwPrf = 1; % 1: Trapez; 2: Spline 3.Grades (noch nicht
implementiert!)

maxV = [360 300 360 381 388 615]*pi/180/2; % maximale
Winkelgeschwindigkeiten der Achsen
maxA = [1200 1200 1200 1200 1200 1200]*pi/180; % maximale
Winkelbeschleunigungen der Achsen
tolA = 0.1; % Tolleranz mit der
ein Zielpunkt erreicht werden muss

% ###--- End Input --- ###

Scan = zeros(1,682);
oldScan = zeros(1,682);
count = 0;
a = 0;
% Start und Ziel Frame erzeugen
P_Start = CreateFrame(Start,OriStart);
P_Ziel = CreateFrame(Ziel,OriZiel);

Q1 = InDirKin(P_Start,kf1);

```

## iniScript

```

lobal Scan oldScan
Scan = zeros(1,682);
oldScan = zeros(1,682);

```

## ReadScanData

```
function ReadScanData(block,ei)

global Scan oldScan count

copyfile('ScanData.ubh','ScanData1.ubh');
ScanFile = fopen('ScanData1.ubh','r');
offset = 1;
fseek(ScanFile,-offset,'eof');
newChar = fread(ScanFile,1);

    while (~strcmp(newChar,char(10))) || (offset == 1)
        offset = offset+1;
        fseek(ScanFile,-offset,'eof');
        newChar = fread(ScanFile,1,'*char');
    end

text = fread(ScanFile,'uint8=>char');
DistCell = textscan(text,'%d','delimiter',';');
DistArray = double(cell2mat(DistCell));

DistArray(DistArray == 0) = 4095;

DistArray = medfilt1(DistArray,5);

minDist = min(DistArray);

if length(DistArray) == 682
    Scan = DistArray;
    oldScan = Scan;

set_param('KUKA_Agilus_KR6_AdvancedModell/Bahnsteuerung/Sensor_Signal/
Constant','Value','Scan');
else

set_param('KUKA_Agilus_KR6_AdvancedModell/Bahnsteuerung/Sensor_Signal/
Constant','Value','oldScan');
    count = count + 1;
end
fclose(ScanFile);
```