Bachelorthesis

Hasibullah Shafaq

Gestengesteuerte Mensch-Roboter-Kollaboration

Fakultät Technik und Informatik *Faculty of Engineering and Computer Science*

Hasibullah Shafaq

Gestengesteuerte Mensch-Roboter-Kollaboration

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung
im Studiengang Mechatronik
an der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Erstprüfer: Prof. Dr. Ing Thomas Frischgesell
Zweitprüferin: Dipl. -Ing. Carolina Bohnert

Abgegeben am 17. August 2017

Hasibullah Shafaq

Thema der Bachelorthesis

Gestengesteuerte Mensch-Roboter-Kollaboration

Stichworte

Industrieroboter, Mensch-Roboter-Kollaboration, Gestensteuerung, MATLAB/Simulink, Kinect V2, TCP/IP, UDP, Server-Client, EthernetKRL, Robot Sensor Interface

Kurzzusammenfassung

Diese Arbeit beschäftigt sich mit der Gestensteuerung des „KUKA Agilus KR6 R900 sixx“-Industrieroboters. Über festgelegte Gesten können dem Industrieroboter definierte Anweisungen geben werden. Entsprechende Aufgaben können dann vom Industrieroboter ausgeführt werden. Zunächst wird ein kurzer Überblick bzgl. der verwendeten Hard- und Softwarekomponenten gegeben. Darauffolgend wird die Umsetzung der Arbeit betrachtet. Exemplarisch werden drei Anwendungsszenarien vorgestellt.

Die Gestenerkennung erfolgt über die Kinect V2 von Microsoft. Detektierte Gesteninformationen werden in MATLAB verarbeitet und anschließend an die KRC4-Robtoersteuerung gesendet. Der Datenaustausch zwischen MATLAB und der KRC4-Robotersteuerung findet über Ethernet und TCP/IP statt. XML dient dabei als Austauschmedium. Das EthernetKRL-Technologiepaket dient der KRC4-Robotersteuerung dabei als Kommunikationsschnittstelle zu MATLAB. Dessen Gebrauch wird ausführlich behandelt.

Hasibullah Shafaq

Title of the paper

Gesture-controlled human-robot collaboration

Keywords

Industrial robotics, Human-robot collaboration, gesture control, MATLAB/Simulink, Kinect V2, TCP-IP, UDP, Server-Client, EthernetKRL, Robot Sensor Interface

Abstract

This thesis deals with the gesture control of an “KUKA Agilus KR6 R900 sixx“-industrial robot. Gestures are detected via the Kinect V2. Instructions can be given to the industrial robot through defined gestures. First, a brief overview of the used hardware and software components is given. The implementation of this project is then considered. Three application scenarios are presented afterwards.

The Kinect V2 by Microsoft is used for the gesture recognition. Detected gestures are processed in MATLAB and are then sent to the KRC4 robot control. Ethernet and TCP/IP are used for the data exchange between MATLAB and the KRC4 robot control. XML serves as an exchange medium. The EthernetKRL technology package servers the KRC4 robot control as a communication interface to MATLAB. Its use discussed with in detail.

Danksagung

Die vorliegende Bachelorarbeit zum Thema „Gestengesteuerte Mensch-Roboter-Kollaboration“ entstand im Rahmen meines Studiums an der Hochschule für Angewandte Wissenschaften in Hamburg und wurde von Prof. Dr. Thomas Frischgesell betreut.

An dieser Stelle möchte ich mich bei Prof. Dr. Thomas Frischgesell für die Vergabe dieses spannenden Themas, das entgegengebrachte Vertrauen sowie die gute Zusammenarbeit bedanken. Dank dieser Möglichkeit konnte ich wertvolle und praktische Erfahrungen sammeln.

Bedanken möchte ich mich außerdem bei Dipl. -Ing. Carolina Bohnert und Ing. Robin Auffermann von der HAW Hamburg. Beide waren jederzeit ansprechbar und haben maßgeblich zur Umsetzung dieser Arbeit beigetragen.

Ein besonderer Danke geht an meine Eltern, meine Geschwister und meinen Freunden. Sie haben mich während meines Studiums aufopferungsvoll unterstützt.

Hamburg, 17. August 2017

Hasibullah Shafaq

Abkürzungsverzeichnis

C / C++	Programmiersprachen
CIRC	Kreisbewegung des Manipulators
EKI	EthernetKRL Interface
FOV	Field of View bzw. Sichtbereich
GUI	Bedieneroberfläche einer Software, auch General User Interface
KLI	KUKA Line Interface
KR 6	KUKA Robot 6
KRC4	KUKA Robot Control 4 (Compact)
KRL	KUKA Robot Language
LIN	Linearbewegung des Manipulators
MRK	Mensch-Roboter-Kollaboration
PTP	Point-To-Point Bewegung des Manipulators
RSI	Robot Sensor Interface
SDK	Software Development Kit
TCP	Tool Center Point bzw. Endeffektor des Manipulators
TOF	Time of Flight

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung

1.1 Motivation

Die deutsche Bundesregierung hat sich mit der Initiative „Industrie 4.0“ das Ziel gesetzt, die industrielle Produktion mit modernster Informations- und Kommunikationstechnik zu verzahnen. Die Versionsnummer „4.0“ in der Bezeichnung soll in diesem Zusammenhang auch die Nähe zur Software-Entwicklung besonders hervorheben. Diese industrielle „Revolution“ ist notwendig, um auf dem internationalen Markt konkurrenzfähig zu bleiben. Der steigende wirtschaftliche Druck führt dazu, dass stetig kürzere Produktionszyklen gefordert werden. Dabei darf die Qualität der Produkte aber nicht gemindert werden. Die „Smart Factory“ soll diese Bedingungen erfüllen. Dieser Begriff bezeichnet eine Produktionsumgebung, in der sich Fertigungs- und Logistikprozesse weitgehend selbstständig organisieren. Die direkte Kommunikation und Kooperation von Menschen, Maschinen, Logistik und Produkten soll eine effizientere und flexiblere Produktion ermöglichen [Kahlen 2017]. Die Mensch-Roboter-Kollaboration (MRK) spielt in der „Smart Factory“ eine zentrale Rolle und steht deshalb im Fokus dieser Arbeit.

Der Begriff der Kollaboration bezeichnet die Zusammenarbeit von mindestens zwei Individuen, bei der die Summen der Beiträge des Einzelnen zum Gesamtergebnis führen. Automatisierbare und nicht vom Menschen ausführbare Prozesse, wie z.B. das Bewegen eines schweren Werkstücks, werden vom Roboter ausgeführt. Der Mensch übernimmt entsprechend manuelle Prozesse bzw. Aufgaben, welche kognitive Fähigkeiten erfordern. Ein Beispiel aus der Industrie wäre z.B. die Montage von Automobilmotoren. Bei der Montage ist es erforderlich, dass der Motorblock in unterschiedliche Positionen bewegt wird. Das Gewicht des

Motorblocks ist für einen Menschen schwerer zu bewegen, als für einen Roboter. Montagearbeiten erfordern andererseits eine gewisse Feinfühligkeit und ein prüfendes Auge. Der Mensch kann solche Aufgaben effizienter erledigen.

Beim Betrieb von Industrierobotern, muss die körperliche Sicherheit von Personen im Umkreis jederzeit gewährleistet sein. Die Arbeitsräume des Industrieroboters werden üblicherweise durch diverse Schutzeinrichtungen gesichert. Zwei Schutzeinrichtungskategorien werden dabei unterschieden [CHV 2008]:

1. Trennende Schutzeinrichtung
    - Feststehende, trennende Schutzeinrichtungen
    - Bewegliche, trennende Schutzeinrichtungen
    - Umzäunungen bzw. Kabinen
    - Schutztüren, Klappen, Schutzfenster
2. Nicht trennende Schutzeinrichtungen:
    - Lichtvorhänge, Scanner, Schaltmatten, Schaltplatten, Zwei-Hand-Schaltungen

Der direkte oder indirekte Kontakt zwischen beiden Instanzen ist voraussetzend für die MRK. Dieser Aspekt führt dazu, dass absolut trennende Schutzeinrichtungen nicht in Frage kommen. Deshalb sind besonders nicht trennende Schutzeinrichtungen von Interesse. In dieser Arbeit wird der Industrieroboter über ein Bildererkennungsverfahren gesteuert. Es wird dem Anwender ermöglicht, den Industrieroboter über seine körperlichen Gesten zu befehligen. Dabei sollen alle vorhandenen Schutzeinrichtungen intakt bleiben.

In der Industrie werden momentan im Bereich der MRK vorwiegend Leichtbauindustrieroboter verwendet, wie z.B. der „KUKA LBR iiwa“. Derartige Leichtbauindustrieroboter detektieren Kollisionen durch Kraft-Moment-Sensoren.

Die Motivation dieser Arbeit liegt unter anderem auch darin, neuere und effizientere Ansätze bzgl. der MRK zu erforschen. In diesem Zusammenhang wird die Gestensteuerung für den „KUKA Agilus KR6 sixx“ mit Hilfe der „Kinect V2“ untersucht und realisiert.

Die Erfassung und Verarbeitung von Gesteninformationen wird in MATLAB erfolgen. MATLAB besitzt diverse Vorteile. Dank der interaktiven Befehlseingabe können wissenschaftliche Programme schnell und einfach erstellt werden. Außerdem ermöglicht der Interpreter die Modularisierung des Programms. Dadurch können einzelne Programmabschnitte entwickelt und getestet werden, ohne dass das ganze Programm kompiliert werden muss. MATLAB ermöglicht auch die Implementierung von anderen Programmiersprachen wie C, C++ oder FORTRAN. Ein zusätzlicher Vorteil sind die umfangreichen Softwarepakete. MATLAB bietet einige Toolboxen bzgl. der Simulation von Robotern an. Diese sind jedoch nicht für die Steuerung von realen Robotern gedacht. Deshalb wurden bereits diverse Anwendungen von Drittanbietern entwickelt, die die Steuerung von realen Robotern ermöglichen sollen.

Die Universität Wismar hat z.B. die *KUKA-KRL-Toolbox for Matlab® and Scilab*[1] entwickelt und publiziert. Diese ermöglicht den bidirektionalen Datenaustausch zwischen MATLAB und der KRC3-Robotersteuerung. Dafür wird eine serielle Schnittstelle genutzt. Diese Option kann in diesem Projekt nicht eingesetzt werden, da die KRC4-Robotersteuerung keine serielle Schnittstelle verfügt und dessen Programme nicht abwärtskompatibel sind.
Eine weitere Anwendung wurde von der Universität Siena[2] entwickelt. Diese basiert auf dem Robot Sensor Interface von KUKA. Der Datenaustausch erfolgt

[1] MatlabKukaKRL-Toobox: https://www.mb.hs-wismar.de/cea/Kuka_KRL_Tbx/Kuka_KRL_Tbx.html
[2] KUKA Control Toolbox (KCT): https://sourceforge.net/projects/kct/

über die Ethernet-Schnittstelle der Robotersteuerung. Jedoch ist auch diese Anwendung nur mit der KRC3-Robotersteuerung kompatibel.
Eine weitere einsetzbare Option, wäre die Anwendung *KUKAVARPROXY bzw. OpenShowVar*[3] von der Firma *IMTS S.r.L. Company*. Mit dieser Anwendung ist es möglich, Daten zwischen einem Rechner und der KRC4-Robotersteuerung auszutauschen. Jedoch basiert diese Anwendung auf der Programmiersprache *Java* und ist außerdem eine Stand-Alone Anwendung. Die Einbindung in MATLAB ist somit nicht möglich.

Insgesamt konnte keine Anwendung gefunden werden, die den Datenaustausch zwischen MATLAB und der KRC4-Robotersteuerung unterstützt. Zur Realisierung der Gestensteuerung ist dies jedoch ein unumgänglicher Aspekt. Deshalb wird in dieser Arbeit nach einer Möglichkeit gesucht, Daten zwischen diesen beiden Instanzen auszutauschen. Außerdem wird die Kinect V2 in MATLAB eingebunden werden, um schlussendlich die Gestensteuerung des Industrieroboters zu ermöglichen.

1.2 Ziel der Arbeit

In dieser Arbeit steht die Gestensteuerung des Industrieroboters im Fokus. Ein Sechsachsroboter (KUKA Agilus KR6 R900 sixx) wird anhand von menschlichen Bewegungsdaten bzw. Gesten gesteuert werden. Die Gestik des Anwenders wird mithilfe der Kinect V2 erfasst und in MATLAB verarbeitet werden. Anschließend werden diese Informationen an die KRC4-Robotersteuerung gesendet. Der Manipulator wird anhand der erfassten Körperdaten des Anwenders gesteuert.

Der KUKA Agilus befindet sich im Mechanik-Labor der HAW Hamburg. Bis zu Beginn dieses Projektes war es nicht möglich, Daten zwischen der

[3] OpenShowVar: https://sourceforge.net/projects/openshowvar/

Robotersteuerung und dem Rechner auszutauschen. Daher besteht das Ziel dieser Arbeit darin, den Datenaustausch zwischen diesen beiden Instanzen zu realisieren. Anschließend werden drei Anwendungsszenarien entworfen und realisiert. Diese dienen der Demonstration und veranschaulichen die Möglichkeiten dieser Anwendung bzgl. der MRK.

Im ersten Anwendungsszenario folgt der TCP des Manipulators einem sich ändernden Raumpunkt. Diesen gibt der Anwender durch seine Gestik vor. Dieses Anwendungsszenario stellt die allgemeine Reaktion des Roboters auf empfangene Befehle dar. Im zweiten Anwendungsszenario gibt der Anwender eine bestimmte Bahn bzw. Kontur durch seine Gestik vor. Der Roboter fährt anschließend die gleiche Bahn bzw. Kontur ab, ähnlich wie beim Playback-Verfahren. Dieses Anwendungsszenario zeigt, dass auch komplexe Bahnen vorgegeben und abgefahren werden können. Diese Methode kann z.B. für Lackierarbeiten genutzt werden. Das dritte Anwendungsszenario stellt die Objektübergabe zwischen Mensch und Roboter dar. Der Roboter verfügt über eine bestimmte Anzahl von Objekten. Der Anwender kann vorgeben, welches Objekt er erhalten möchte. Der Roboter erkennt anhand der Gestik, welches Objekt der Anwender erhalten möchte und übergibt es entsprechend. Dieses Szenario verdeutlicht den direkten und intelligenten Kontakt zwischen Menschen und Roboter.

1.3 Struktur der Arbeit

Diese Arbeit lässt sich in insgesamt sechs Hauptkapitel gliedern. Das erste beinhaltet die Einleitung in diese Arbeit. Darin werden die Motivation und das Ziel dieser Arbeit verdeutlicht.

Im zweiten Kapitel werden grundlegende Spezifikationen bzgl. der eingesetzten System- und Hardwarekomponenten dargestellt. Außerdem werden die beiden KUKA-Technologiepakete „Robot Sensor Interface“ und „EthernetKRL“ vorgestellt.

Während der Untersuchung stellt sich die Frage, welches der beiden Technologiepakete als Kommunikationsschnittstelle zum Arbeitsrechner geeignet ist. Beide Möglichkeiten werden anhand verschiedener Faktoren bewertet. EthernetKRL wird dabei als geeignete Methode ausgewählt.
Das dritte Kapitel thematisiert die Schnittstellen der Systemkomponenten und deren Kommunikation untereinander. Dabei liegt der Fokus nicht nur auf den Erkenntnissen, sondern auch auf dem methodischen Vorgehen während dieser Arbeit. Zunächst wird die Steuerung der Kinect V2 über MATLAB erläutert. Alle relevanten Methoden und Konfigurationen werden diskutiert. Anschließend liegt der Fokus auf den Datenaustausch zwischen der Robotersteuerung und dem Arbeitsrechner. Danach werden die fundamentalen Bewegungsfunktionen des Manipulators in der KUKA-Robot-Language (KRL) vorgestellt. Diese sollen in Kooperation mit MATLAB ausführt werden. Am Ende dieses Kapitels wird der grundsätzliche Kommunikationsablauf der Systemkomponenten bildhaft dargestellt und erklärt.

Im vierten Kapitel werden drei Anwendungsszenarien bzgl. der Gestensteuerung des Industrieroboters entwickelt und realisiert. Das erste Anwendungsszenario behandelt die PTP-Verfolgung der rechten Hand des Anwenders. Im zweiten Anwendungsszenario wird dem Roboter eine komplexe Bahn vorgegeben, die über einen Spline-Bewegungssatz abgefahren wird. Im dritten Anwendungsszenario wird die Objektübergabe zwischen Mensch und Roboter behandelt. Dem Roboter stehen mehrere Objekte mit bekannter Position zur Verfügung. Der Anwender entscheidet, welches Objekt er vom Roboter übergeben haben möchte.
Im fünften Kapitel wird die Arbeit zusammengefasst. Zudem werden alle Forschungsergebnisse präsentiert und ausgewertet. Das sechste und letzte Kapitel gibt einen Ausblick über mögliche Anschlussprojekte und bestehendes Verbesserungspotential.

2 Theoretische Grundlagen

2.1 System- und Hardwarekomponenten

2.1.1 Technische Systemanforderungen

Für die Umsetzung dieser Arbeit sind folgende technische Systemanforderungen bzgl. des Arbeitsrechners notwendig:

Tabelle 1: Technische Systemanforderungen

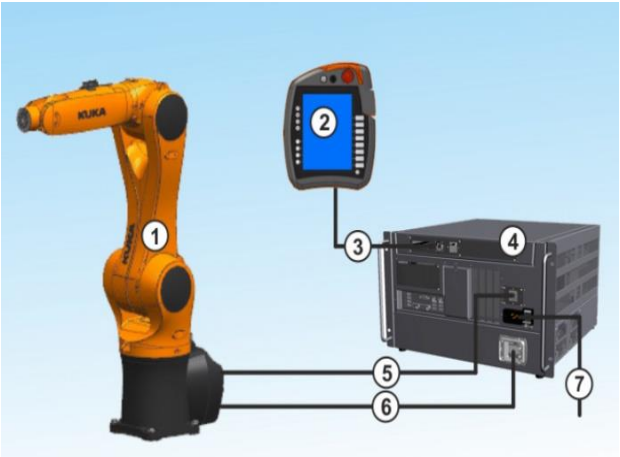
Betriebssystem	Windows 8 oder höher
Grafikkarte	DirectX 11.0 oder höher
USB-Port	1 x USB 3.0
MATLAB-Version	Getestet mit 2016b
C-Compiler für MATLAB	Visual C++ 2015 Compiler

Die Kinect V2 kann nur über einen USB 3.0-Anschluss betrieben werden. Dieser Anschluss wird erst ab „Windows 8“ unterstützt. Um die aufwändigen Grafikprozesse der Kinect V2 ausführen zu können, wird außerdem eine Grafikkarte mit „DirectX 11.0“ Kompatibilität benötigt. In dieser Arbeit wird die Kinect V2 in MATLAB über eine C-MEX-Bibliothek bedient. Diese muss vor dem Gebrauch mit einem C++-Compiler kompiliert werden. Dafür wurde der „Visual C++ 2015 Compiler“ genutzt. In Abschnitt 3.1 wird die Kompilierung von C-MEX-Bibliotheken genauer erklärt.

2.1.2 KUKA Agilus KR6 R900 sixx

In dieser Arbeit wird der „KUKA Agilus KR6 R900 sixx“ zusammen mit der Robotersteuerungseinheit KRC4 eingesetzt. Die Agilus-Modelle repräsentieren die Kleinroboterserie von KUKA. Diese sind als Fünf- und Sechsachsmodelle erhältlich und besitzen eine Traglastfähigkeit von bis zu 10 kg. KUKA wirbt damit, dass die erreichbare Geschwindigkeit im entsprechenden Traglastbereich einzigartig sei. Die Kombination aus hoher Präzision, integrierter Energiezuführung und diverser Einbaulagen des Manipulators, mache die Agilus-Serie flexibel und leistungsstark. Außerdem wird diese Serie, genau wie größere Modelle, über die KRC4-Robotersteuerung bedient. Dies ermöglicht eine Kooperation zwischen unterschiedlichen KUKA Robotermodellen [KUKA Roboter GmbH 2016a].
Die Agilus-Serie eigne sich dank der „Safe-Robot“-Funktionalität, besonders für die Mensch-Roboter-Kollaborationen. Im Vergleich zur KRC2, wird bei der KRC4 die Sicherheit nicht mehr ausschließlich durch Hardwarekomponenten, sondern zusätzlich durch Softwarealgorithmen gewährleistet [KUKA Roboter GmbH 2016b]. Dadurch eröffnen sich besonders im Rahmen der Forschung und Entwicklung neue Möglichkeiten und Freiheiten.

Ein Robotersystem besteht typischerweise aus diversen Einzelelementen. Das Robotersystem des KUKA Agilus umfasst einen Manipulator, die KRC4-Steuerungseinheit, Verbindungsleitungen, Werkzeuge und weitere Ausrüstungsteile. Alle Systemkomponenten sind in Abbildung 2-1 (S. 9) dargestellt und gekennzeichnet.



1. Manipulator: KR6 R900 sixx
2. Programmierhandgerät: SmartPad
3. Verbindungsleitung: SmartPad
4. Robotersteuerung: KRC4-Compact
5. Datenleitung
6. Motor- und Versorgungsleitung
7. Geräteanschluss-Leitung

Abbildung 2-1: KUKA Agilus Robotersystem
[KUKA Roboter GmbH 2015, S. 11]

Der Manipulator stellt den Bewegungsapparat des Robotersystems dar. Dieser zeichnet sich durch einen verhältnismäßig weiten Arbeitsraum aus. Eine weitere Besonderheit ist auch, dass selbst Raumkoordinaten die sich nah am Roboterfuß befinden, angefahren werden können.

In Abbildung 2-2 (S. 10) wird der Arbeitsraum des Manipulators dargestellt. Alle Maßangaben sind in Millimeter gültig. Aus den Angaben zum Arbeitsraum, lassen sich kinematische Gleichungen bzw. Modell herleiten. Diese können z.B. für die Bewegungssimulation und die Trajektorienplanung genutzt werden. An dieser Stelle sei erwähnt, dass der Fokus dieser Arbeit nicht auf der kinematischen Analyse des Roboters liegt. Deshalb wird diese nicht weiter ausgeführt. Für eine ausführlichere Ausarbeitung bzgl. der Roboterkinematik, wird an dieser Stelle auf die Arbeit von Schultz et al. verwiesen [Schultz, Frischgesell, Bohnert 2016].

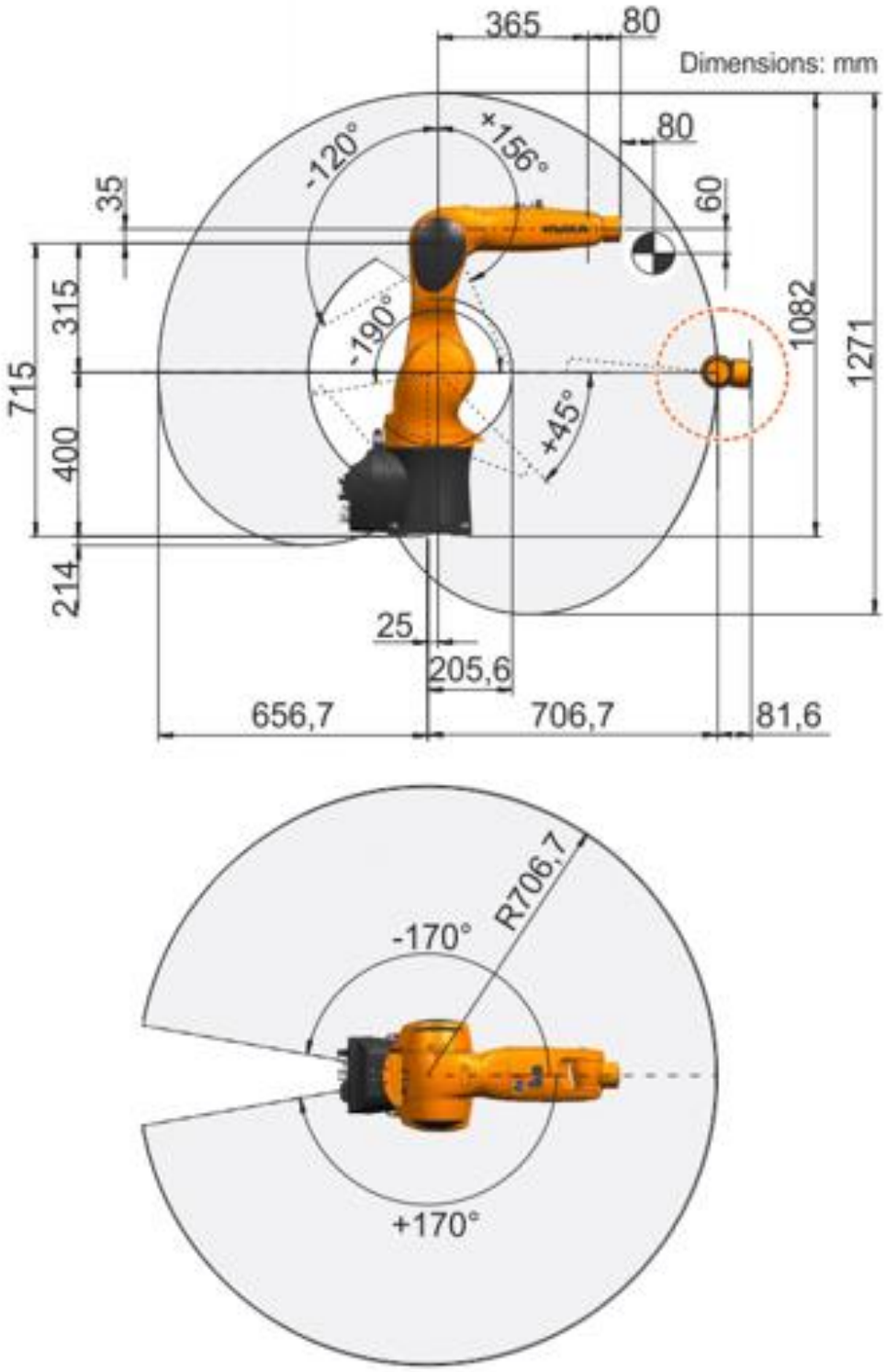


Abbildung 2-2: Arbeitsraum des KUKA Agilus KR6
[KUKA Roboter GmbH 2015, S. 20]

2.1.3 KR C4 Compact

Das zentrale Element des Robotersystems ist die „KRC4“-Robotersteuerung. Diese kommt bei moderneren KUKA-Industrierobotern zum Einsatz. In diesem Projekt wird die kleinere Version der KRC4 verwendet, die „KRC4 Compact“. Diese wird zur Steuerung von Kleinrobotern eingesetzt. Im weiteren Verlauf ist der Begriff KRC4 gleichbedeutend mit der KRC4-Compact. Die KRC4 besitzt diverse Schnittstellen, welche in Abbildung 2-3 dargestellt und gekennzeichnet sind.

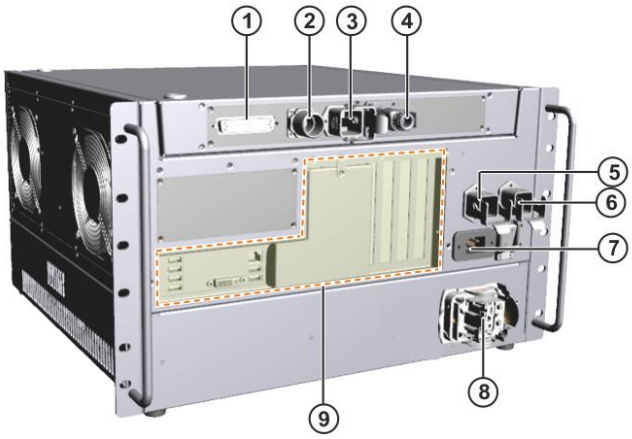


Abbildung 2-3: KR C4 compact Schnittstellen [KUKA Roboter GmbH 2014a, S. 16]

1. X11 Sicherheits-Schnittstelle
2. X19 SmartPad-Anschluss
3. X65 Extension Interface
4. X69 Service Interface
5. X21 Manipulator Schnittstelle
6. X66 Ethernet-Schnittstelle
7. X1 Netz-Anschluss
8. X20 Motorstecker
9. Steuerungs-PC Schnittstellen

In dieser Arbeit wird die Ansteuerung des Industrieroboters über die Ethernet-Schnittstelle des PCs angestrebt. Der Arbeitsrechner soll entsprechend mit der X66-Ethernet-Schnittstelle des KRC4 verbunden werden. Die wichtigsten Einstellungen und Anwendungsmöglichkeiten bzgl. der KRC4 werden im Laufe dieser Arbeit erläutert. Für weitere Informationen bzgl. der KRC4, z.B. zu den technischen Spezifikationen oder der Installation, wird hier auf die Bedienungsanleitung von KUKA verwiesen [KUKA Roboter GmbH 2014a].

KUKA bietet zur Steuerung des Roboters und der KRC4-Robotersteuerung das „SmartPad“-Programmierhandgerät. Es bietet „alle Bedien- und Anzeigemöglichkeiten die für die Bedienung und Programmierung des

Industrieroboters erforderlich sind“ [KUKA Roboter GmbH 2014a, S. 30]. Auf die Bedienung des SmartPads wird - soweit nötig - im Laufe dieser Arbeit eingegangen. Informationen bzgl. der grundsätzlichen Bedienung des SmartPads lassen sich in der entsprechenden Dokumentation finden.

2.1.4 Kinect V2

Die Kinect V2 (siehe Abbildung 2-4) von Microsoft ist ein sog. „Motion Sensing Input Device“, das zusammen mit der „Xbox One“-Spielekonsole zum Einsatz kommt. Ursprünglich wurde es für interaktive Spiele entwickelt, bei denen der Spieler anhand seiner Körperbewegung bestimmte Aktionen auslösen kann. Dabei werden Bewegungen und Gesten mit einer Farbkamera und einem Tiefensensor erfasst. Die Veröffentlichung der ersten Kinect SDK [Microsoft 2012], ermöglichte die Inbetriebnahme und den Gebrauch in Windows. Dies eröffnete besonders im Bereich der Entwicklung viele Möglichkeiten. Der zusätzlich geringe Kaufpreis führte dazu, dass die Kinect in vielen Projekten Einsatz fand und selbst Jahre nach der Veröffentlichung noch großen Anklang findet.

Abbildung 2-4: Microsoft Kinect V2
[Sugiura 2014]

Die Kinect V2 bietet die sog. „Skeleton Tracking“-Methode. Sie ermöglicht die Erfassung von Gesten und Körpergelenkkoordinaten im Raum. In diesem Projekt soll diese dazu genutzt werden, um Personen im Umkreis des Industrieroboters zu detektieren und Anweisungen über dessen Gestik entgegen zu nehmen. Die Körpergelenkkoordinaten sollen in MATLAB oder Simulink erfasst, verarbeitet und an die Robotersteuerung gesendet werden. In Tabelle 2 werden die technischen Spezifikationen der Kinect V1 und der Kinect V2 dargestellt.

Tabelle 2: Technische Spezifikationen zur Kinect V1 & V2
[Matthew Szymczyk 2014] & [Microsoft]

Attribute	**Kinect V1**	**Kinect V2**
Farbbildaufnahme	640 x 480 @ 30 FPS	1920 x 1080 @ 30 FPS
Tiefenbildaufnahme	320 x 240 @ 30 FPS	512 x 424 @ 30 FPS
Aufnahmedistanz	0.4 m – 4.5 m	0.5 m – 4.5 m
FOV	57 x 43 Grad	70 x 60 Grad
Infrarotaufnahme	Keine	512 x 424
Erfassungsmethode	Streifenprojektion	TOF-Kamera
Audioaufnahme	4-Mic.-Array @ 16 kHz	4-Mic.-Array @ 48 kHz
Detektierbare Skelette	2	6
Detektierbare Joints	20	26
Ausrichtungsmotor	Ja	Nein
USB	2.0	3.0

Für weitere Informationen bzgl. der Inbetriebnahme oder der Funktionsweise, sei auf die Arbeit „Schnelle Aufnahme von Bewegungen mit Scannern“ verwiesen [Shafaq, Frischgesell 2017, S. 17–22].

2.2 KUKA Technologiepakete

KUKA bietet zwei Technologiepakete an, mit denen der Datenaustausch realisiert werden kann. Zu Beginn sollen beide Möglichkeiten analysiert und vorgestellt werden. Anschließend werden beide bewertet.

2.2.1 Robot Sensor Interface

KUKA bietet mit dem „Robot Sensor Interface“ (RSI) ein erweiterbares Technologiepaket an, mit dem der Datenaustausch zwischen der Robotersteuerung und einem Sensor- bzw. externem System ermöglicht wird. Der Datenaustausch kann sowohl über Ethernet als auch über das IO-System der Robotersteuerung erfolgen. In dieser Arbeit wird die Kommunikation zwischen dem Arbeitsrechner und der KRC4-Robotersteuerung über Ethernet erfolgen. Im RSI werden Signale im festen Sensortakt zyklisch verarbeitet und ausgewertet. Dadurch wird die Ausführung von echtzeitfähigen Programmen ermöglicht. Roboterbewegungen lassen sich somit auch während des Programmablaufs durch Sensorsignale beeinflussen.

Für die Datenübertragung wird das verbindungslose UDP-Protokoll genutzt. Im Gegensatz zum TCP/IP-Protokoll, ist dieses grundsätzlich für die Echtzeitübertragung geeignet. Das UDP-Protokoll ist aber im Vergleich dazu eher unzuverlässig. Daten werden zwischen den Netzwerkteilnehmern nur „gestreamt“. Einen „Handshake“, wie es eines beim TCP/IP-Protokoll gibt, existiert nicht. Somit kann die Vollständigkeit der übertragenen Daten nicht garantiert bzw. geprüft werden. Der Einsatz des UDP-Protokolls ist nur empfehlenswert, wenn das auszuführende Programm tolerant gegenüber unvollständigen bzw. fehlerhaften Daten ist. Im Zweifelsfall muss während der Programmierung dafür gesorgt werden, dass fehlerhafte bzw. unvollständige Daten nicht zu Fehlern führen.

Durch die fehlende Prüfung der ausgetauschten Daten und der verbindungslosen Datenübertragung, kann das UDP-Protokoll höhere Übertragungsraten erreichen. Dies macht es besonders für Echtzeitanwendungen attraktiv.

Das Verhalten von RSI-Anwendungen kann im grafischen RSI-Visual Editor über Signalflussdiagramme definiert werden, ähnlich wie in Simulink. Die Signalflussdiagramme setzen sich aus RSI-Objekten zusammen. Ein RSI-Objekt ist ein Funktionsbaustein mit einer definierten Anzahl von Ein- und Ausgängen. Eingangssignale werden entsprechend der Funktionalität des Bausteins verarbeitet und an den Signalausgängen bereitgestellt. Der RSI-Visual Editor verfügt über eine umfangreiche Bibliothek mit implementierten Funktionsbausteinen bzw. RSI-Objekten. Der Begriff RSI-Kontext bezeichnet die Verknüpfung der einzelnen RSI-Objekte zu einem gesamten Signalflussdiagramm (siehe Abbildung 2-5). Dadurch wird das Verhalten der Anwendung definiert. Der RSI-Kontext lässt sich in ein „KUKA Robot Language (KRL)“-Programm laden und dort benutzen. KRL bezeichnet die allgemeine Programmiersprache, mit der KUKA-Roboter bedient und gesteuert werden.

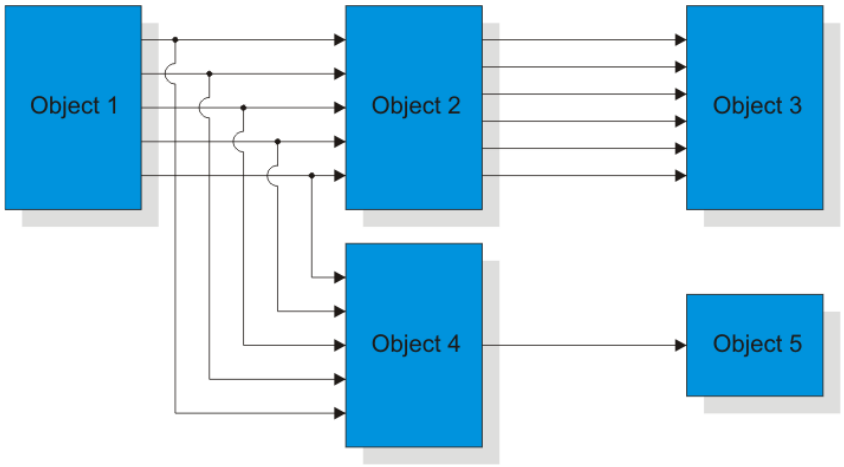


Abbildung 2-5: Schematischer Aufbau eines RSI-Kontext
[KUKA Roboter GmbH 2013, S. 12]

In RSI werden XML-Datenpakete zwischen der Robotersteuerung und dem Arbeitsrechner ausgetauscht. Die Struktur des XML-Datenpakets wird in einer entsprechenden XML-Konfigurationsdatei definiert. Diese wird automatisch aus dem jeweiligen RSI-Kontext bzw. Signalflussdiagramm generiert. Alle variablen Sensorinformationen werden in die Struktur des XML-Datenpakets geladen und gesendet. Die Robotersteuerung besitzt einen XML-Parser, der die enthaltenen Sensorinformationen aus dem empfangenen XML-Datenpaket herausfiltern und interpretieren kann. Diese können dann als Variablen im KRL-Programm verwendet werden.

Das RSI-Technologiepaket unterstützt die Manipulation des Roboters in Echtzeit. Ein typisches Anwendungsbeispiel ist z.B. die Feinjustierung des „Tool Center Points“ (TCP) bei der Bewegung auf einer rauen Oberfläche. Die Oberflächenkontur kann dabei mit einem Sensor detektiert werden. Somit kann der Roboter während der Bewegung größere Oberflächenunebenheiten rechtzeitig erkennen und umfahren. Prinzipiell ist der Einsatz des RSI nur für kleinere Bewegungskorrekturen gedacht.

In dieser Arbeit geht es vorrangig um die Steuerung der Roboterbewegungen im Allgemeinen. Eine Echtzeitkorrektur der Roboterbewegung ist nicht unbedingt erforderlich. Für die Ausführung einer RSI-Anwendung ist außerdem eine echtzeitfähige Netzwerkkarte oder ein echtzeitfähiges Computersystem notwendig. Diese sind während des Projektes nicht gegeben, weshalb der Einsatz des RSI-Paketes nicht möglich ist. Entsprechend wird dieses Unterkapitel nicht weiter vertieft. Für weitere Informationen wird auf die KUKA Robot Sensor Interface 3.2 Betriebsanleitung [KUKA Roboter GmbH 2013] verwiesen.

2.2.2 EthernetKRL

EthernetKRL ist, genau wie das Robot Sensor Interface, ein nachladbares Technologiepaket. Dieses kann zusätzlich von KUKA erworben werden. Es bietet dem Benutzer die Möglichkeit, die Robotersteuerung über ein externes System bzw. einen Rechner zu bedienen. Die Verbindung zum Arbeitsrechner kann über eine übliche Ethernet-Verbindung hergestellt werden. Dabei wird dieser an die KLI-Schnittstelle bzw. an den X66-Anschluss der KRC4-Robotersteuerung angeschlossen. Der Datenaustausch zwischen den Instanzen kann über das TCP/IP- oder dem UDP-Protokoll erfolgen. Für EthernetKRL-Anwendungen wird im Allgemeinen das TCP/IP-Protokoll empfohlen. Das ist, im Vergleich zum UDP-Protokoll, zuverlässiger bzgl. der Übertragung von vollständigen Datensätzen. Das UDP-Protokoll bietet hingegen eine schnellere Datenübertragung. Die Kommunikationszeit des TCP/IP-Protokolls ist aber im Wesentlichen vom KRL-Programm und dem gesendeten Datenvolumen abhängig. Bei entsprechender Programmierung kann eine Umlaufzeit von 2 Millisekunden pro Paket erreicht werden [vgl.KUKA Roboter GmbH 2014b, S. 9].

Die Robotersteuerung und der Arbeitsrechner müssen sich beim TCP/IP-Protokoll in einer Server-Client-Beziehung befinden. Dabei muss jeder Instanz eine feste Rolle zugewiesen werden. Die KRC4-Robotersteuerung kann gleichzeitig Server und Client sein. Gleiches gilt für externe Systeme (siehe Abbildung 2-6, S. 18). Dadurch ist es möglich, ein Netzwerk aus Server- und Client-Systemen aufzubauen. Das ermöglicht die Verzahnung von mehreren Robotersystemen, z.B. wie in einer Produktionsstraße. Zu jeder Instanz, sei es Server oder Client, sind bis zu 16 aktive Verbindungen möglich. Die Software-Schnittstelle zwischen der Robotersteuerung und den externen Systemen stellt das EthernetKRL Interface (EKI) dar. Jedoch kann die EKI nur einen Client besitzen, falls sie als Server konfiguriert ist. Es ist aber auch möglich, innerhalb der Robotersteuerung

mehrere EKI-Server oder EKI-Clients zu betreiben [KUKA Roboter GmbH 2014b, S. 12]. So kann die KRC4-Robotersteuerung mittels der EKI für ein externes System als Server dienen, für ein anderes externes System aber als Client. In dem Fall muss jeder EKI-Kanal bzw. jede Verbindung in einer gesondert XML-Konfigurationsdatei eingestellt werden.

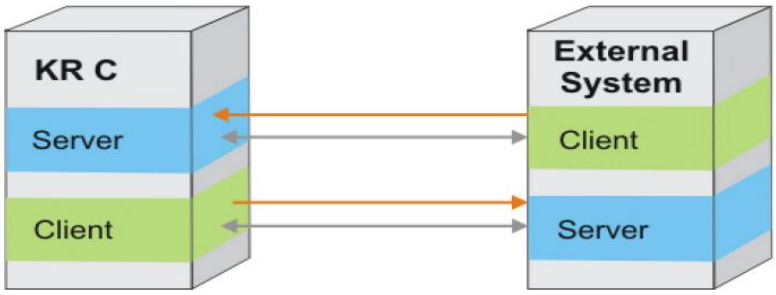


Abbildung 2-6: Server-Client-Beziehung zwischen einer KRC4 und externem System [KUKA Roboter GmbH 2014b, S. 12]

Die Ethernet-Verbindung wird mittels einer XML-Datei konfiguriert. Jede einzelne Verbindung bzw. jeder einzelne EKI-Kanal muss in einer eigenen XML-Konfigurationsdatei definiert werden. Diese XML-Konfigurationsdateien müssen dann im entsprechenden Verzeichnis[4] hinterlegt werden. Das Verzeichnis befindet sich innerhalb der KRC4-Robotersteuerung. Die Konfigurationsdateien werden bei der Initialisierung der EKI-Kanäle aufgerufen und eingelesen. Der genaue Aufbau der XML-Konfigurationsdateien wird in dieser Arbeit analysiert und in Abschnitt 3.2 beschrieben.

Um während der Programmausführung eine stabile Verbindung zwischen Robotersteuerung und Arbeitsrechner aufrechterhalten zu können, bietet EthernetKRL einige Möglichkeiten. Die Verbindung wird über einen Ping auf den Arbeitsrechner geprüft. Dieser wird in regelmäßigen Zeitabständen von der Robotersteuerung ausgesendet. Bei erfolgreicher Verbindung kann ein „Flag“ gesetzt werden, welches im KRL-Programm genutzt werden kann. Bei

[4] EthernetKRL Konfigurationsverzeichnis: C:\ KRC\ ROBOTER\ Config\ User \Common \EthernetKRL

Verbindungsabbruch wird dieses „Flag“ automatisch wieder deaktiviert und muss nicht explizit zurückgesetzt werden. Auf diese Weise kann ein Verbindungsabbruch kontrolliert werden. Der Grund für einen Verbindungsabbruch kann auf dem SmartPad angezeigt werden [KUKA Roboter GmbH 2014b, S. 9–10].

Hauptsächlich liegt die Absicht in der Verwendung des EthernetKRL-Technologiepakets darin, Daten zwischen der Robotersteuerung und dem Arbeitsrechner auszutauschen. In Abbildung 2-7 wird das Konzept des Datenaustauschs zwischen der EKI und einem externen System bzw. Rechner dargestellt. Dabei ist der Ablauf des Datenempfangs rot, der des Datenversands grün gekennzeichnet.

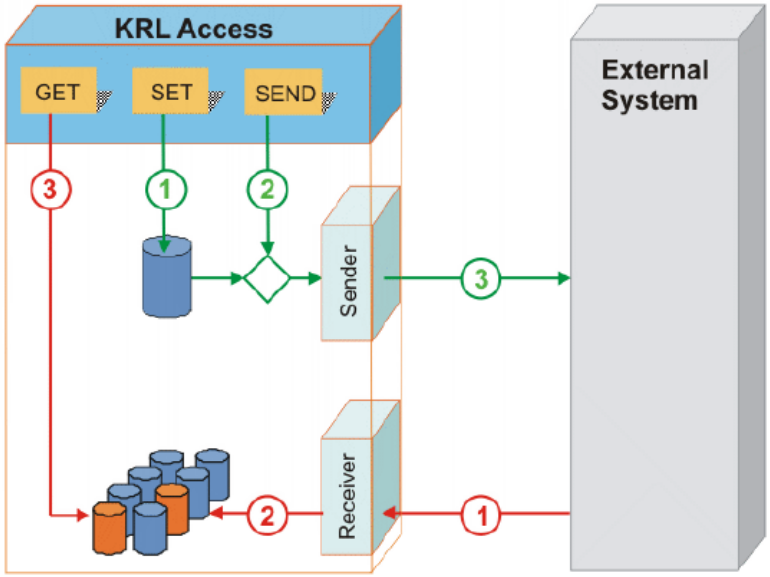


Abbildung 2-7: Datenaustausch zwischen EKI und externem System
[KUKA Roboter GmbH 2014b, S. 10]

Beim Datenversand werden die zu versendenden Informationen zuerst strukturiert in einen Zwischenspeicher geladen. Zum Versenden der Informationen ist im KRL-Programm ein gesonderter Sendebefehl notwendig.

Dieser liest die Daten aus dem Zwischenspeicher aus und sendet sie an das externe System.

Beim Datenempfang sendet ein externes System Informationen an die EKI-Schnittstelle. Die empfangenen Daten werden vom Empfangstask aufgefangen. Dieser filtert alle beinhalteten Informationen (entsprechend der XML-Elemente aus der XML-Konfigurationsdatei) aus dem XML-Datenpaket aus und legt diese strukturiert in einen Datenspeicher ab. Diese verbleiben im Datenspeicher, bis sie im KRL-Programm aufgerufen werden. Entsprechend können keine neuen Daten empfangen werden, solange der Datenspeicher belegt ist.

Empfangene Informationen werden, je nach XML-Element, stapelweise in die Datenspeicher abgelegt (siehe Abbildung 2-8). Die Informationen können über die FIFO- (First In First Out) oder über LIFO (Last In Last Out) -Reihenfolge aus den Stapelspeichern ausgelesen werden. Standardmäßig gilt die FIFO-Reihenfolge.

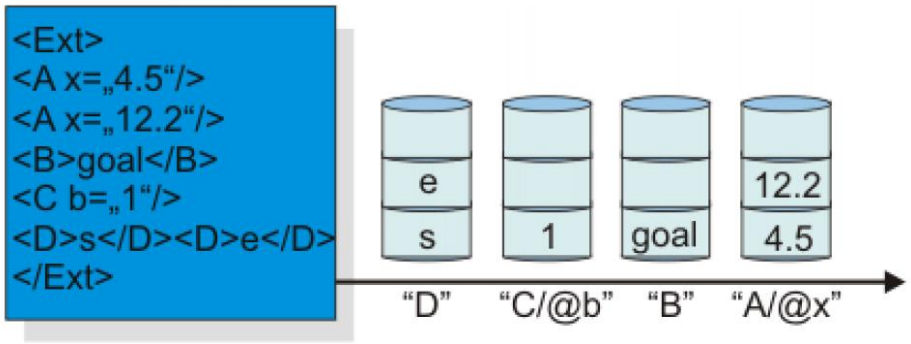


Abbildung 2-8: XML-Empfangsdatenspeicher
[KUKA Roboter GmbH 2014b, S. 11]

Die Daten werden mit Hilfe des Roboter-Interpreters ausgelesen und interpretiert. Dieser weist die empfangenen Informationen im KRL-Programm den entsprechenden Variablen zu. Die genaue Verwendung innerhalb von KRL-Programmen wird in Abschnitt 3.3 erläutert.

Beim Datenaustausch zwischen mehreren Instanzen ist es üblich, ein Dateiformat bzw. Protokoll zu vereinbaren, um Eindeutigkeit und Vollständigkeit zu bewahren. In EthernetKRL werden dem Benutzer zwei vordefinierte Dateiformate vorgestellt. Zum einen können Informationen im binären Format gesendet werden, zum anderen über die XML-Struktur. Keine der beiden Dateiformate bieten wesentliche Vor- oder Nachteile. Die XML-Struktur ist für den Menschen im Allgemeinen jedoch besser zu lesen. Deshalb liegt der Fokus im Folgenden auch auf der Verwendung der XML-Formats. Die Anwendungsprinzipien sind analog auf das binäre Datenformat übertragbar. Hier sei vermerkt, dass EthernetKRL das „XPath"-Format unterstützt. Dieses Format ist eine Unterkategorie des allgemeinen XML-Schemas.

Eine XML-Struktur wird aus einzelnen XML-Elementen aufgebaut. Die Syntax eines Elements setzt sich wie folgt zusammen:

```
<Elementname> Informationen </Elementname>
```

Beispiel: `<GreiferOffen> 1 </GreiferOffen>`

Zwischen den Elementnamen bzw. den sog. „XML-Tags", können Informationen hinterlegt werden, die übertragen und interpretiert werden können. Eine weitere Möglichkeit, Informationen in XML zu hinterlegen, sind die XML-Attribute. Diese können einem XML-Element zugewiesen werden. Bei der Zuweisung von Attributen wird folgende Syntax genutzt:

```
<Elementname Attribut= „Wert" />
```

Beispiel: `<GreiferOffen Status= „1" />`

Außerdem entfällt bei der Attributzuweisung das Abschlusstag [KUKA Roboter GmbH 2014b, S. 27–28]. Für die Datenübertragung sind beide Möglichkeiten gleichwertig.

Die XML-Struktur der zu empfangenden Daten wird in der XML-Konfigurationsdatei[5] festgelegt. Dem Roboter-Interpreter muss mitgeteilt werden, welche XML-Struktur ein zu erwartendes XML-Datenpaket besitzt. Das externe bzw. das sendende System muss sich exakt an die Syntaxvorgaben halten. Im Falle einer fehlerhaften Syntax, kann der Roboter-Interpreter die Informationen nicht lesen bzw. interpretieren. Dieser ist gegenüber fehlerhafter Syntax äußerst intolerant. Ansonsten wird dem Anwender aber die Freiheit gegeben, XML-Elemente und -Attribute nach Belieben selbst zu deklarieren, zu definieren und zu sortieren.

In der XML-Konfigurationsdatei werden neben den Überragungsstrukturen auch alle Konfigurationen bzgl. der Verbindung zwischen dem EKI und dem externen System getätigt. Folgende Grundstruktur ist dabei einzuhalten:

```
<ETHERNETKRL>
     <CONFIGURATION>
          <EXTERNAL></EXTERNAL>
          <INTERNAL></INTERNAL>
     </CONFIGURATION>
     <RECEIVE>
          <ELEMENTS></ELEMENTS>
     </RECEIVE>
     <SEND>
          <ELEMENTS></ELEMENTS>
     </SEND>
</ETHERNETKRL>
```

[5] EthernetKRL Konfigurationsverzeichnis: C:\ KRC\ ROBOTER\ Config\ User\ Common\ EthernetKRL

Unter *CONFIGURATION* werden die Verbindungseinstellungen definiert. Unter *EXTERNAL* können Einstellungen bzgl. der Beziehung zum externen System vorgenommen werden. Unter *INTERNAL* werden interne Verbindungsparameter definiert.
Im Abschnitt *RECEIVE* werden die zu erwartenden XML-Elemente und deren Struktur definiert. Gleiches gilt für die gesendeten Elemente unter *SEND*. Alle Einstellungsparameter werden im EthernetKRL-Handbuch [KUKA Roboter GmbH 2014b, S. 21–28] ausführlich beschrieben. Im Abschnitt 3.2 wird der Datenaustausch über XML genauer untersucht werden. Anwendungsbeispiele lassen sich im Abschnitt 4 finden.

Anders als das RSI-Technologiepaket erfordert das EthernetKRL keine echtzeitfähige Netzwerkkarte bzw. System. Der Datenaustausch wird entsprechend über EthernetKRL erfolgen. Die Realisierung des Datenaustausches wird in den folgenden Kapiteln erforscht und dargestellt.

3 Kommunikation und Schnittstellen der Systemkomponenten

Das Ziel dieser Arbeit besteht in der Umsetzung der Gestensteuerung des KUKA Agilus mit Hilfe der Kinect V2. Bei der Analyse der Systemkomponenten hat sich ergeben, dass sich die Umsetzung des Ziels in drei grundsätzliche Module unterteilen lässt. Folgende Unterteilung ist möglich:

1. Steuerung der Kinect V2 über MATLAB und Ausgabe der Gesteninformationen
2. Verbindungsaufbau und Datenaustausch zwischen Rechner und Robotersteuerung
3. Bewegungssteuerung des Manipulators

Diese Punkte werden in den folgenden Abschnitten erforscht und realisiert.

3.1 Steuerung der Kinect V2 über MATLAB

Die Erfassung der Gesteninformationen spielt bei der Gestensteuerung des Industrieroboters eine zentrale Rolle. Für diesen Zweck wird die Kinect V2 von Microsoft eingesetzt. Diese soll in MATLAB eingebunden und bedient werden. Dessen gelieferte Informationen sollen verarbeitet und an die KRC4-Robotersteuerung gesendet werden. In diesem Abschnitt wird eine Lösung bzgl. der Erfassung von Gesteninformationen in MATLAB gesucht.

Die Kinect V2 kann bis zu sechs Personen gleichzeitig erkennen und alle voneinander unterscheiden. Jede Person bekommt einen „Body-Index“ von 1 bis 6 zugewiesen. Die Zuweisung erfolgt zufällig, falls sich mehrere Personen im Detektionsbereich aufhalten. Befindet sich nur eine Person im Bild, bekommt

diese den Index „1“ zugewiesen. Jede hinzukommende Person bekommt einen höheren Body-Index.

Zusätzlich zur Personenunterscheidung kann die Kinect bis zu 26 Körpergelenke pro Person detektieren. Die einzelnen Körpergelenke bekommen einen sog. „Joint-Index“ zugewiesen. Die Zuweisungen sind in Abbildung 3-1 (S. 25) und in Tabelle 3 dargestellt. Body- und Joint-Indizes können im Programmcode verwendet werden, um einzelne Personen bzw. Körpergelenke auszuwerten. Jedes Körpergelenk kann über den jeweiligen Joint-Index ausgewertet werden. Dabei können verschiedene Informationen abgerufen werden, wie z.B. die entsprechenden Raumkoordinaten.

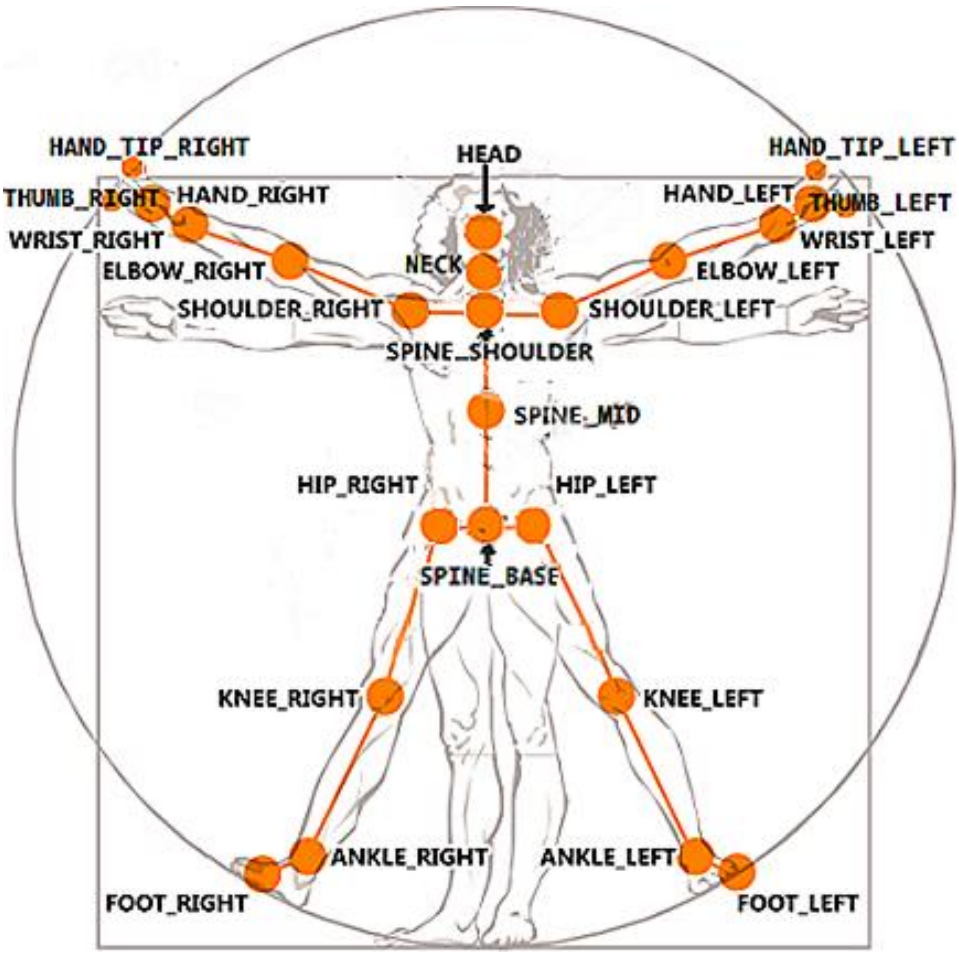


Abbildung 3-1: Körpergelenke und deren Bezeichnungen [Microsoft 2013]

Tabelle 3: Joint-Index Zuweisungen
[Microsoft 2013]

Joint	Index	Joint	Index
Ankle Left	14	Knee Left	13
Ankle Right	18	Knee Right	17
Elbow Left	5	Neck	2
Elbow Right	9	Shoulder Left	4
Foot Left	15	Shoulder Right	8
Foot Right	19	Spine Base	0
Hand Left	7	Spine Mid	1
Hand Right	11	Spine Shoulder	20
Hand Tip Left	21	Thumb Left	22
Hand Tip Right	23	Thumb Right	24
Head	3	Wrist Left	6
Hip Left	12	Wrist Right	10
Hip Right	16		

Die Kinect bietet neben der Auswertung von Körpergelenkinformationen, auch die Möglichkeit, diverse Handzeichen zu erkennen. Jedem Handzeichen ist ein festgelegter Index zugewiesen. Die Zuweisung ist in Tabelle 4 dargestellt.

Tabelle 4: Hand-State-Index Zuweisung
[Microsoft 2014]

Hand-State	Index	Beschreibung
Unknown	0	Hand-State ist unbekannt
NotTracked	1	Hand wird nicht erkannt
Open	2	Hand ist offen
Closed	3	Hand ist geschlossen
Lasso	4	Hand zeigt Lasso

Sowohl für die rechte, als auch für die linke Hand kann festgestellt werden, welches Handzeichen ausgedrückt werden soll. Bei aktiven Handzeichen kann zwischen offen, geschlossen und dem sog. „Lasso“ (auch bekannt als „Victory“-Handzeichen) unterschieden werden. Diese Handzeichen können beliebig im Programmcode genutzt werden. *Unknown* und *NotTracked* sind eher passive

Handzeichen und signalisieren, dass das Handzeichen nicht bzw. nicht ordnungsgemäß detektiert wird.

MATLAB bietet für die Kinect V2 eine eigene Softwarebibliothek mit vorgefertigten Markos an, die „Image Acquisition Toolbox“ [The Mathworks 2016]. Diese basiert auf der Kinect SDK 2.0 von Microsoft. Zu Beginn wurde versucht die Kinect V2 über die „Image Acquisition Toolbox“ in MATLAB einzubinden. Alle Arbeitsschritte wurden wie in der Dokumentation beschrieben ausgeführt. Die Inbetriebnahme blieb jedoch erfolglos. Um ein allgemeines Fehlverhalten des Arbeitsrechners bzw. des Systems auszuschließen, wurde die Kinect an unterschiedliche Rechner angeschlossen. An den jeweiligen Rechnern wurden die MATLAB Versionen 2016a, 2016b, sowie 2017a installiert. Die Inbetriebnahme blieb in allen Fällen erfolglos. Der Grund für dieses Fehlverhalten konnte nicht ermittelt werden. Deshalb wurde diese Herangehensweise verworfen.

Eine geeignetere Alternative bietet die „Microsoft Kinect SDK 2.0“. Diese ist zwar nicht direkt mit MATLAB kompatibel, lässt sich jedoch in C/C++ verwenden. Daher liegt es nahe, die entsprechenden Bibliotheken in eine sog. C-MEX-Bibliothek kompilieren.

C-MEX-Dateien ermöglichen die Einbindung und den Gebrauch von C/C++-Programmen in MATLAB. Genau wie bei üblichen C/C++-Programmen, werden diese als Ganzes kompiliert. Eine kompilierte C-MEX-Datei kann alle beinhalteten Funktionen in MATLAB zu Verfügung stellen. Sie sind mit den „Dynamic Link Library“-Dateien (.DLL) in Windows vergleichbar. Außerdem kann dadurch eine signifikante Verbesserung der Laufzeit des Programms erreicht werden. Ein zu langsames Programm könnte sich z.B. negativ auf die Aufnahmefrequenz der Kinect auswirken. Dies ist deshalb ein vorteilhafter Nebeneffekt der C-MEX-Bibliotheken.

Zunächst muss geprüft werden, ob die Kinect überhaupt über C/C++ bedienbar ist. Erst wenn dies garantiert werden kann, sollte daraus eine C-MEX-Bibliothek generiert werden. Um ein grundsätzliches Verständnis über die möglichen Funktionen der „Microsoft Kinect 2.0 SDK“ zu erhalten, empfiehlt sich ein Blick in die Dokumentation [Microsoft 2017].

In „Visual Studio 2015“ wurde ein entsprechendes Programm geschrieben (siehe Anhang 1). Zum Ausführen muss dem Programm der Pfad zur Kinect-Library bekannt sein. Deshalb müssen die entsprechenden Pfade innerhalb der Projekteigenschaften (in *Visual Studio 2015)* hinterlegt werden. Diese sind im Programmcode in den obersten Zeilen als Kommentar zu finden.
Das Programm aus Anhang 1 kann die Raumkoordinaten der rechten und der linken Hand von jeder Person im Detektionsbereich erfassen und ausgeben. Außerdem können Handzeichen erkannt und ausgegeben werden. Es ist in C++ geschrieben und konnte erfolgreich ausgeführt werden. Damit ist bewiesen, dass die Ansteuerung der Kinect V2 in C/C++ einwandfrei funktioniert.

Das Programm aus Anhang 1 wird nach der erfolgreichen Inbetriebnahme über C/C++, nun in MATLAB implementiert. C-MEX-Bibliotheken besitzen eine fest vorgegebene Syntax. C-Code kann nicht wie üblich übernommen werden. MATLAB greift über Gateway- und Callback-Funktionen auf die C-MEX-Bibliothek zu. Dieser Faktor führt dazu, dass das gesamte Programm aus Anhang 1 umgeschrieben und umstrukturiert werden müsste.

Während der Recherche wurde jedoch ein MATLAB Add-In gefunden, das die „Microsoft Kinect 2.0 SDK“ bereits als kompilierbare C-MEX-Bibliothek anbietet. Das „Kinect 2 Interface for Matlab“-Add-In [Juan R. Terven 2015] konnte erfolgreich in Betrieb genommen werden. Während dieser Arbeit werden einzelne Anweisungen aus der kompilierten C-MEX-Bibliothek dieses Add-Ins genutzt.

Detaillierte Anwendungen werden in Kapitel 4 diskutiert. Die Generierung einer C-MEX-Bibliothek aus dem Programm aus Abschnitt 1 ist nicht nötig und wird während dieser Arbeit nicht weiterverfolgt.

Um das Add-In nutzen zu können, muss zuerst die entsprechende C-MEX-Bibliothek kompiliert werden. Die Kompilierung findet über MATLAB statt. Dafür benötigt dieser jedoch einen integrierten C/C++-Compiler. Die Kompilierung wurde mit dem „Visual C++ 2015 Compiler“ erfolgreich durchgeführt. Andere Compiler garantieren keinen Erfolg. So wurde z.B. auch der „GCC Compiler“ von *GNU* [6] getestet. Die Kompilierung blieb beim Einsatz dieses Compilers erfolglos, weshalb der „Visual C++ 2015 Compiler“ dringend empfohlen wird.
Um den „Visual C++ 2015 Compiler“ zu installieren, wurden folgende Schritte ausgeführt:

1. Installation von „Visual Studio 2015“
    - Bei der Installation ist darauf zu achten, dass nicht die Standardinstallation durchgeführt wird. Nur in der benutzerdefinierten Installation ist der „Visual C++ 2015 Compiler“ enthalten.
2. Im MATLAB Kommandofenster den Befehl „mex -setup“ eingeben und den entsprechenden Compiler auswählen
3. Im „Kinect 2 Interface for Matlab“-Ordner (standardmäßig unter Dokumente/ MATLAB)
    - *Compile_cpp_files.m* auswählen und in MATLAB öffnen
    - Include- und Lib-Path der „Microsoft Kinect 2.0 SDK“ eintragen. Beispielskript mit den entsprechenden Verzeichnissen befindet sich im Anhang in der beigefügten CD. (Angegebene Verzeichnisse können jedoch bei anderen Rechnern abweichen)

[6] GCC Compiler: https://gcc.gnu.org/

- *Compile_cpp_files.m* ausführen
- Im selben Verzeichnis befindet sich der Ordner *Mex*. Wenn die Kompilierung erfolgreich war, sollte sich nun die Datei *Kin2_mex.mexw64* in dem Ordner befinden.

4. C-MEX-Bibliothek im MATLAB-Skript implementieren
    - Dies ist mit dem Befehl *addpath('Mex')* möglich
    - C-MEX-Funktionen können wie gewohnt in MATLAB aufgerufen werden

3.2 Datenaustausch zwischen Rechner und Robotersteuerung

In diesem Kapitel wird die Verbindung zwischen dem Arbeitsrechner und der KRC4-Robotersteuerung untersucht und hergestellt. Der Datenaustausch zwischen beiden Einheiten wird angestrebt. Die KRC4-Robotersteuerung bietet zusammen mit dem EthernetKRL-Technologiepaket die Möglichkeit des Datenaustausches. Diese Anwendung wird untersucht und umgesetzt. Auf dem Arbeitsrechner wird der Datenaustausch über MATLAB gesteuert.

Wie bereits in Abschnitt 2.2.2 erläutert wurde, erfolgt der Datenaustausch über die Ethernet-Schnittstelle des Rechners und der KLI-Schnittstelle (X66-Anschluss) der Robotersteuerung. Dabei werden XML-Datenpakete über das TCP/IP-Protokoll ausgetauscht. Außerdem soll jeder Netzwerkteilnehmer (Robotersteuerung und Arbeitsrechner) eine bestimmte Netzwerkrolle (Server oder Client) zugewiesen bekommen. Entsprechende Netzwerkeinstellungen können in einer Konfigurationsdatei festgelegt werden. Alle Konfigurationsmöglichkeiten lassen sich in der EthernetKRL Dokumentation [vgl.KUKA Roboter GmbH 2014b, S. 21–26] finden.

3.2.1 Verbindungskonfiguration des EthernetKRL-Testservers

KUKA liefert, zusammen mit dem EthernetKRL-Softwarepaket, auch eine entsprechende „Testserver“-Anwendung. Anhand derer ist es möglich, Robotersteuerung und Rechner miteinander zu verbinden und bereitgestellte Beispielprogramme auszuführen. Dadurch kann das Verhalten der Kommunikationspartner während des Datenaustauschs getestet und untersucht werden. Um die Verbindung zwischen der Robotersteuerung und dem Arbeitsrechner herzustellen, wird das Beispielprogramm *XmlCallBack* (siehe

Anhang 2) [KUKA Roboter GmbH 2014b, S. 49–51] ausgeführt. Bei dessen Ausführung auf dem Rechner, agiert dieser als Server und die Robotersteuerung als Client. Anhand dieses Beispiels wird die allgemeine Herangehensweise während der Verbindungsherstellung untersucht und hier dargestellt.

Um die Verbindung zwischen Robotersteuerung und Rechner herzustellen, muss zuerst der Testserver gestartet werden. Abbildung 3-2 stellt die Bedieneroberfläche des EthernetKRL-Testservers dar. Die Bedieneroberfläche ist in der EthernetKRL Dokumentation ausführlich beschrieben [KUKA Roboter GmbH 2014b, S. 42–43].

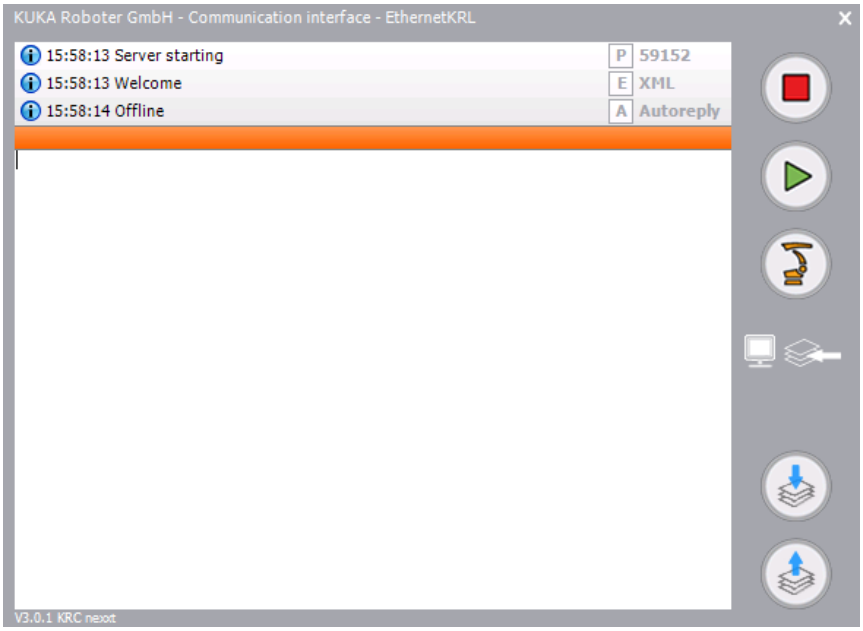


Abbildung 3-2: Bedieneroberfläche des EthernetKRL-Testservers

Um den Testserver in Betrieb zu nehmen, müssen einige Konfigurationen vorgenommen werden. Unter *Communication Properties* müssen folgende Parameter gesetzt werden. Diese sind in Abbildung 3-3 (S.33) dargestellt.

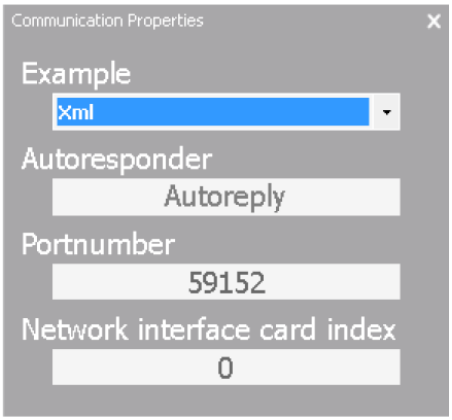


Abbildung 3-3: Kommunikationsparameter

Unter *Example* kann ausgewählt werden, welches Datenformat übertragen werden soll. Es stehen XML, BinärFixed (mit fester Länge) und BinärStream (variable Länge mit Endzeichenfolge) zur Verfügung. In dieser Arbeit wird für die Datenübertragung das XML-Format genutzt. *Autoreply* setzt fest, ob der Testserver sofort auf Anfrage des Clients antworten soll oder ob vorher eine manuelle Bestätigung des Anwenders erforderlich sein soll. Unter *Portnumber* wird der Port des Servers bzw. des Rechners festgelegt. Dieser Port dient während des Serverbetriebs dem Datenaustausch. Diese Portnummer muss auch in der entsprechenden XML-Konfigurationsdatei[7] des EKI-Kanals hinterlegt werden. Diese befindet sich auf der Robotersteuerung. Des Weiteren muss der richtige Netzwerkkartenindex angegeben werden. Dieser ist individuell und vom Rechner abhängig, liegt in der Regel aber im Zahlenbereich von 0 bis 5.

Neben den Einstellungen innerhalb des Testservers müssen außerdem einige Konfigurationen bzgl. des Arbeitsrechners vorgenommen werden. Die folgenden Einstellungen gelten nur für die direkte Ethernet-Verbindung zwischen der Robotersteuerung und dem Arbeitsrechner. Der Zugriff über ein Netzwerk kann von dieser Methode abweichen.

[7] EthernetKRL Konfigurationsverzeichnis: C:\ KRC\ ROBOTER\ Config\ User\ Common\ EthernetKRL

Dem Rechner muss in *Windows* unter *Netzwerk- und Freigabecenter* eine feste IP-Adresse vergeben werden. In den Verbindungseigenschaften kann eine feste TCP/IPv4-Adresse vergeben werden. Während dieser Arbeit wurde die `100.100.100.101` für den Server bzw. Arbeitsrechner gewählt und `100.100.100.100` für den Client bzw. Robotersteuerung. Die IP-Adressen können beliebig vergeben werden. In Abbildung 3-4 ist die Vergabe einer festen IP-Adresse dargestellt.

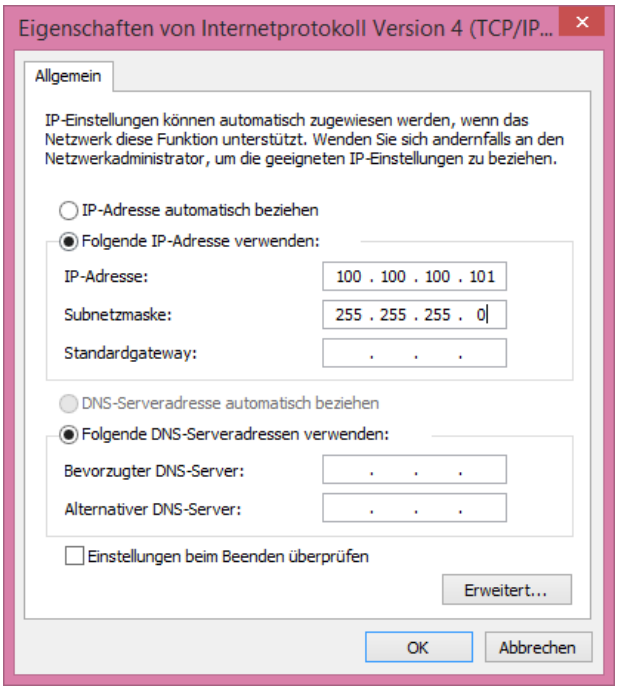


Abbildung 3-4: Vergabe einer festen IP-Adresse in Windows

3.2.2 Verbindungskonfiguration der Robotersteuerung

Neben den Verbindungskonfigurationen des Testservers und des Rechners muss die Verbindung auch auf der Robotersteuerung eingestellt werden. In EthernetKRL wird jede Verbindung in einer entsprechenden XML-Konfigurationsdatei definiert (siehe Abschnitt 2.2.2). Für das *XmlCallBack* Beispielprogramm ist diese im Anhang 2 zu finden. Grundsätzlich lässt sich der Aufbau der Konfigurationsdatei in drei Abschnitte aufteilen: *Configuration, Receive* und *Send*.

Im Abschnitt <CONFIGURATION> lassen sich allgemeine Verbindungsparameter setzen. Dabei werden interne und externe Verbindungsparameter unterschieden. Dadurch, dass in diesem Beispiel der Rechner als Server agiert, sind dem Client nur externe Verbindungsparameter zu übergeben. Die Übergabe der Server-IP und des Server-Ports reichen hier vollkommen aus. Die Parameterübergabe über XML wird in Abschnitt 2.2.2 erklärt. Weitere Einstellungen werden vom System festgelegt, wie z.B. Übertragungsprotokolle. In der EthernetKRL Dokumentation sind alle verfügbaren Verbindungsparameter aufgelistet und kommentiert [KUKA Roboter GmbH 2014b, S. 21–26].

Nachdem die allgemeinen Verbindungsparameter konfiguriert wurden, muss der EKI mitgeteilt werden, welche XML-Pakete generiert und versendet werden sollen und welche XML-Pakete zu erwarten sind. Im Abschnitt <RECEIVE> werden die zu erwartenden XML-Elemente und deren -Struktur definiert. Empfangene XML-Datenpakete, die nicht mit der festgelegten Datenstruktur konform sind, können nicht gelesen bzw. ausgewertet werden. Das macht die Verbindung sehr empfindlich gegenüber fehlerhaften und unvollständigen Datenpakete. Das „EthernetKRL Interface“ kann die empfangenen XML-Datenpakete anhand der Strukturdefinitionen richtig „parsen“ und den entsprechenden KRL-Variablen zuweisen. Dazu wird ein implementierter XML-Parser genutzt.

Im <SEND> Abschnitt der XML-Konfigurationsdatei werden die zu sendenden Elemente definiert. Hier kann ebenfalls eine Struktur definiert werden. Die Grundstruktur[8] der XML-Konfigurationsdatei muss zwar erhalten bleiben, jedoch kann der Benutzer innerhalb der verschiedenen Abschnitte beliebige XML-Elemente definieren. Das Verhältnis zwischen den XML-Strukturdefinitionen und den tatsächlich generierten bzw. gesendeten und den empfangenen

[8] Grundstruktur der XML-Konfigurationsdatei: <EthernetKRL>, <CONFIGURATION>, <RECEIVE>, <SEND> oder siehe Abschnitt 2.2.2

Datenpaketen wird aus der Dokumentation nicht ersichtlich. Deshalb wird deren Verhältnis in Abschnitt 3.2.4 erforscht und dargestellt.

In der XML-Konfigurationsdatei werden alle Attribute bzgl. des EKI-Kanals eingestellt. Die allgemeinen Netzwerkkonfigurationen der Robotersteuerung sind im SmartPad unter *Inbetriebnahme/Netzwerkkonfigurationen* vorzunehmen. Zuerst muss ein Interface hinzugefügt und eingestellt werden (siehe Abbildung 3-5, S. 36). Im Empfangstask muss der Empfangsfilter auf *Ziel-Subnetz* gesetzt werden. Zusätzlich zum Interface muss ein NAT-Port hinzugefügt werden. Die Portnummer muss die gleiche wie in der XML-Konfigurationsdatei zum EKI-Kanal sein. In dieser Arbeit wurde der Port *54600* genutzt (siehe Abbildung 3-6). An diesem Port empfängt die Robotersteuerung die XML-Datenpakete. Außerdem kann festgelegt werden, welches Protokoll genutzt werden soll, hier ist es das TCP/IP-Protokoll.

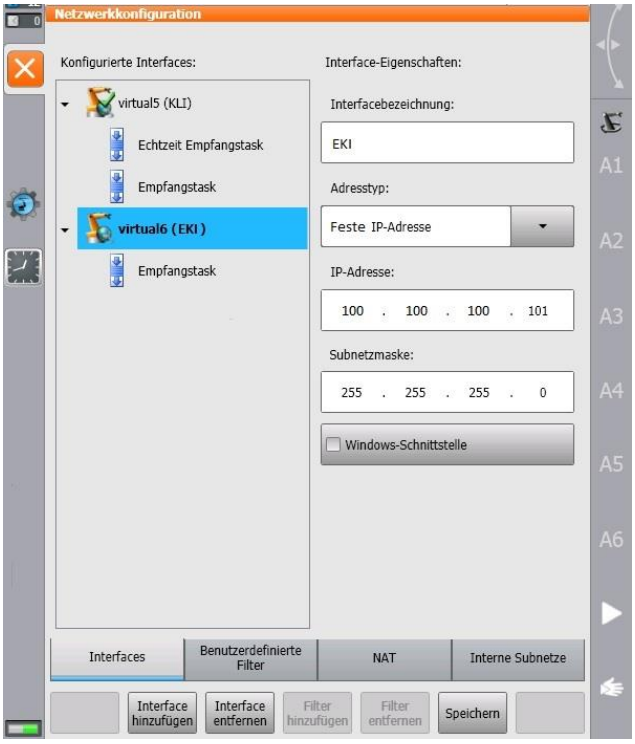


Abbildung 3-5: Netzwerkkonfiguration des EKI-Interfaces

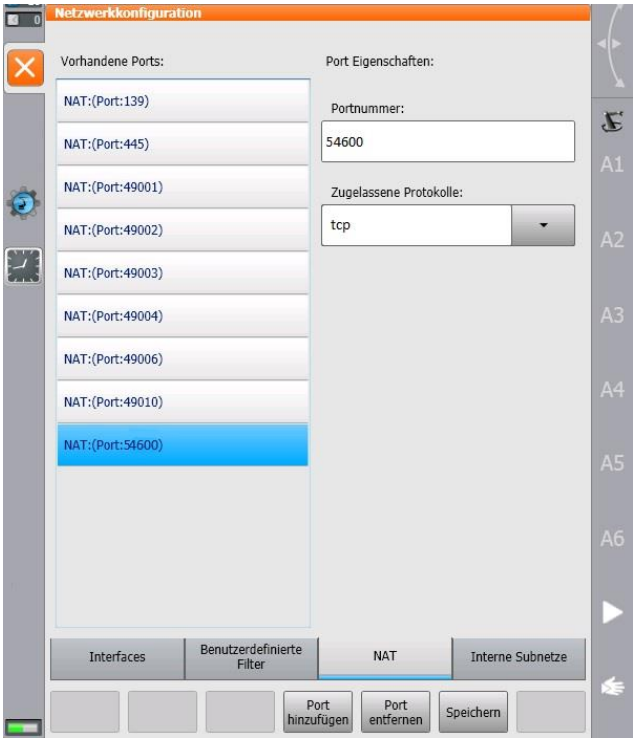


Abbildung 3-6: Netzwerkkonfiguration des NAT-Ports

3.2.3 Verbindungsaufbau zwischen dem EthernetKRL-Testserver und der Robotersteuerung

Sobald alle Konfigurationen getätigt wurden, kann die Verbindung zwischen dem EthernetKRL-Testserver und der KRC4-Robotersteuerung hergestellt werden. Dazu wird zunächst der Testserver mit dem *Play*-Button gestartet. Im *Meldungsfenster* sollte nun angezeigt werden, dass eine bestimmte IP-Adresse und Port abgehört wird. Der Testserver steht zum Verbindungsaufbau bereit. Jetzt kann das *XmlCallBack* Programm (siehe Anhang 2) auf der Robotersteuerung ausgewählt und gestartet werden. Die Robotersteuerung versucht nun die Verbindung zum Arbeitsrechner aufzubauen. Der Testserver sollte, wenn alle Schritte korrekt ausgeführt wurden, die Verbindungsanfrage des Clients nun akzeptieren und die Verbindung aufbauen. Sobald die Verbindung steht, sendet die Robotersteuerung ein XML-Datenpaket an den Testserver. Dieser antwortet dem Client bzw. der Robotersteuerung dann ebenfalls mit einem XML-Datenpaket. In dieser Arbeit konnte das Beispielprogramm erfolgreich ausgeführt werden. Des Weiteren ist die Arbeitsweise des Beispielprogramms in der EthernetKRL Dokumentation [KUKA Roboter GmbH 2014b, S. 50–51] ausreichend beschrieben und wird hier nicht näher betrachtet. Sie wird lediglich dazu genutzt, um den Datenaustausch zwischen der Robotersteuerung und dem Arbeitsrechner zu analysieren.

3.2.4 Analyse der ausgetauschten XML-Datenpakete

Anhand des Beispielprogramms und des EthernetKRL-Testservers, kann nun die Verbindung und der Datenaustausch analysiert werden. Für die Analyse wird „WireShark[9]“ genutzt. „WireShark“ ist ein sog. „Netzwerk-Sniffer“, mit dem der Datenverkehr in einem Netzwerk überwacht und ausgewertet werden kann.

[9] WireShark: https://www.wireshark.org/download.html

Dieser kann sowohl Kommunikationspartner und dessen IP-Adressen anzeigen, als auch Datenpakete die zwischen ihnen ausgetauscht wird. Vor allem bietet es aber die Möglichkeit, Fehler bzgl. der Verbindung zu untersuchen und zu beheben.

Mit Hilfe von „WireShark“ konnte außerdem das Verhältnis zwischen der XML-Strukturdefinition und den generierten bzw. empfangenen XML-Datenpaketen genauer analysiert werden. Das folgende Beispiel, stellt die Analogie zwischen der XML-Strukturdefinition aus der XML-Konfigurationsdatei und dem tatsächlich gesendeten XML-Datenpaketen zwischen Robotersteuerung und Arbeitsrechner dar:

```
...
<XML>
<RECEIVE>
<ELEMENT Tag="Sensor/Message" Type="STRING"/>
<ELEMENT Tag="Sensor/Positions/Current/@X" Type="REAL"/>
<ELEMENT Tag="Sensor/Positions/Before/X" Type="REAL"/>
<ELEMENT Tag="Sensor/Nmb" Type="INT"/>
<ELEMENT Tag="Sensor/Status/IsActive" Type="BOOL"/>
<ELEMENT Tag="Sensor/Read/xyzabc" Type="FRAME"/>
<ELEMENT Tag="Sensor/Show/@error" Type="BOOL"/>
<ELEMENT Tag="Sensor/Show/@temp" Type="INT"/>
<ELEMENT Tag="Sensor/Show" Type="STRING"/>
<ELEMENT Tag="Sensor/Free" Type="INT" Set_Out="998"/>
<ELEMENT Tag="Sensor" Set_Flag="998"/>
</RECEIVE>
</XML>
...
```

```
<Sensor>
 <Message>Example message</Message>
 <Positions>
  <Current X="4645.2" />
  <Before>
    <X>0.9842</X>
  </Before>
 </Positions>
 <Nmb>8</Nmb>
 <Status>
  <IsActive>1</IsActive>
 </Status>
 <Read>
  <xyzabc X="210.3" Y="825.3" Z="234.3"
A="84.2" B="12.3" C="43.5" />
 </Read>
 <Show error="0" temp="9929" />
 <Free>2912</Free>
</Sensor>
```

Der linksseitige XML-Code ist ein Auszug aus der *XmlCallBack*-Konfigurationsdatei. Im Abschnitt *RECEIVE* wird der Robotersteuerung bekannt gegeben, welche Struktur das zu empfangene XML-Datenpaket haben wird und welche Datentypen zu erwarten sind. Jedes XML-Element beginnt mit der Syntax *ELEMENT*, gefolgt von *Tag*. Anschließend wird die XML-Struktur des Datenpakets beschrieben. Die XML-Struktur der Datenpakete ist frei wählbar. EthernetKRL folgt der „XPath“-Syntax. Diese ist in Abschnitt 2.2.2 beschrieben.

Informationen können zwischen den sog. XML-Tags hinterlegt werden. Es gibt jedoch auch die Möglichkeit, Informationen über die Attribute der XML-Elemente zu übergeben. Die Übergabe findet über das @-Zeichen statt. Attribute bieten den Vorteil, dass mehrere Variablen über ein XML-Element übergeben werden können. Auf diese Weise kann das XML-Datenpaket strukturiert werden. Die Übergabe zwischen den Tags eines XML-Elements erlaubt nur die Angabe einer Variable. Des Weiteren unterscheiden sich diese beiden Verfahren jedoch nicht voneinander.

Um diesen Sachverhalt zu veranschaulichen, sei auf die Zeilen *„Sensor/Message“* und *„Sensor/Show“* verwiesen. Die Zeile *Message*, beinhaltet nur eine Variable vom Typ *String*. Eine zweite, getrennte Variable kann nicht hinzugefügt werden. Die Zeile *Show* beinhaltet hingegen zwei Variablen beinhalten. Zum einen beinhaltet sie *Error* vom Typ BOOL, zum anderen auch *Temp* vom Typ INT. Bei größeren Datenmengen ist die Übergabe über Attribute vorzuziehen. Dies garantiert einen übersichtlicheren und besser lesbareren Programmcode.

3.2.5 Datenaustausch zwischen der KRC4-Robotersteuerung und MATLAB/Simulink

Das Verhältnis zwischen der XML-Konfigurationsdatei und dem tatsächlich gesendeten bzw. empfangenen XML-Datenpaket konnte erfolgreich analysiert und bestimmt werden. Auf dieser Grundlage kann nun die Verbindung zu MATLAB/Simulink hergestellt werden. Der EthernetKRL-Testserver wird im weiteren Verlauf nicht mehr benötigt. Stattdessen sollen XML-Datenpakete zwischen der Robotersteuerung und MATLAB/Simulink ausgetauscht werden.

Im Folgenden soll der Datenaustausch zwischen der Robotersteuerung und Simulink realisiert werden. Simulink bietet im Vergleich zu MATLAB diverse Vorteile. Die verwendeten Blockschaltbilder bieten dem Anwender z.B. weit mehr Übersicht als ein MATLAB-Skript. Außerdem können bereits erstellte Simulink-Blöcke ohne größeren Aufwand wiederverwendet werden. Dies ermöglicht die Modularisierung und Erweiterung von komplexen Programmabläufen.
In Simulink werden die Blöcke *TCP/IP-Receive* und *TCP/IP-Send* in der „Instrument Control Toolbox“ angeboten. Diese ermöglichen die Kommunikation mit externen Systemen über Ethernet und dem TCP/IP-Protokoll. In Abbildung 3-7 (S. 41) ist die Bedienoberfläche des *TCP/IP-Receive*-Blocks dargestellt. Diese ermöglicht den Empfang von Informationen über TCP/IP. Vorher sind jedoch diverse Parameter einzustellen.

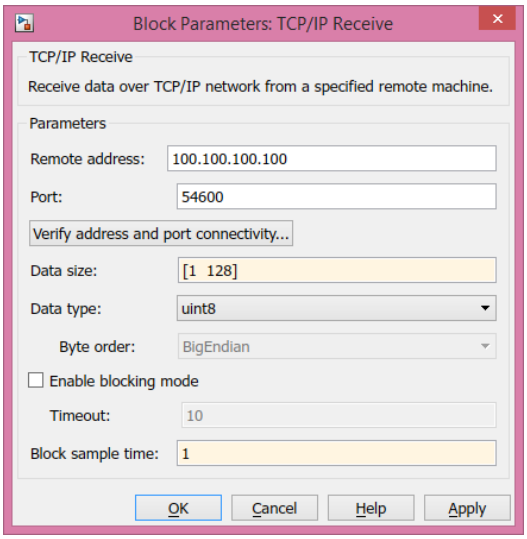


Abbildung 3-7: Bedienoberfläche des TCP/IP-Blocks in Simulink

Unter *Remote adress* ist die IP-Adresse der Robotersteuerung bzw. des EKI-Kanals anzugeben. Diese muss mit der angegeben IP-Adresse aus der jeweiligen XML-Konfigurationsdatei der Robotersteuerung übeinstimmen. Der *Port* der Robotersteuerung lässt sich in der gleichen XML-Konfigurationsdatei finden. Im Feld *Data size* muss angegeben werden, welche Länge das zu erwartende Datenpaket besitzen wird. Die Angabe erfolgt über einen zweidimensionalen Vektor. In diesem Beispiel besitzt das erwartete XML-Datenpaket 128 Zeichen. Die Anzahl der Zeichen in einem Datenpaket kann über „Wireshark“ ermittelt werden. Ob dieser Betrag an erster oder zweiter Stelle des Vektors angegeben wird, hängt davon ab, ob die Ausgabe des TCP/IP-Blocks als Zeilen- bzw. Spaltenvektor gewünscht ist. Unter *Data type* kann festgelegt werden, welchen Datentyp das auszugebene Datenpaket am Blockausgang besitzen soll.

Der Datenaustausch zwischen der Robotersteuerung und dem Arbeitsrechner ist strikt sequenziell. Nur ein Verbindungsteilnehmer kann zur selben Zeit auf dem EKI-Kanal senden. Im *Blocking Mode* wird die Ausführung des Simulink Modells so lange gestoppt, bis neue Datenpakete zum Empfang bereitstehen. Ist der *Blocking Mode* deaktiviert, kommt ein weiterer Blockausgang mit der Bezeichnung *Status* hinzu. Dieser signalisiert mittels einer steigenden Flanke, dass neue Datenpakete

empfangen wurden. In dieser Arbeit wird der TCP/IP-Block jedoch im *Blocking Mode* betrieben.

Mit Hilfe des TCP/IP-Blocks konnte die Verbindung und der Datenaustausch zwischen der Robotersteuerung und dem Arbeitsrechner bzw. Simulink realisiert werden. Die gewünschten Variablen konnte aus dem empfangenen XML-Datenpaketen herausgefiltert werden. So wurden z.B. die aktuellen Raumkoordinaten des Manipulators an Simulink gesendet. Das XML-Datenpaket wurde vom TCP/IP-Receive-Block als uint8-Vektor ausgegeben. Anhand der folgenden MATLAB-Funktion war es beispielsweise möglich, den Wert für die X-Koordinate herauszufiltern:

```
function y = fcn(u)
% uint8-XML-Datenpaket nach char bzw. string konvertieren
xml_str = char(u);
% Position der Variable 'X' im Datenpaket finden
pos_of_x = strfind(xml_str, 'X=');
% Wert für 'X' herausfiltern
if ~isempty(pos_of_x(1))
    str_x_val = str((pos_of_x(1)+3):(pos_of_x(1)+8));
    double_x_val = str2double(str_x_val);
    y = double_x_val;
else
% Sonst soll am Blockausgang NULL ausgegeben werden
    y = 0.0;
end
```

Diese Funktion konvertiert den vom TCP/IP-Block ausgegebenen uint8-Vektor in einen String. Über *strfind* kann die Position der X-Variable im String herausgesucht werden. Dessen Inhalt wird anschließend entnommen und nach *double* konvertiert und ausgegeben. MATLAB bietet keinen integrierten XML-Parser. Deshalb können alle weiteren Variablen nach diesem Schema herausgefiltert werden.

Des Weiteren ist erwähnenswert, dass der TCP/IP-Block nur als Client agieren kann. Simulink bietet keine Möglichkeit, einen TCP/IP-Server zu betreiben. Dies führt dazu, dass die Robotersteuerung die Netzwerkrolle des Servers übernehmen muss.

Die Realisierung der Robotersteuerung in Simulink konnte aus leider zeitlichen Gründen nicht weiter untersucht. Deshalb wird die Gestensteuerung in MATLAB umgesetzt. Damit jedoch alle Anwendungen in dieser Arbeit auch für zukünftige Erweiterungen bzgl. Simulink kompatibel bleiben, wird im Folgenden die Robotersteuerung als Server betrieben. MATLAB agiert im weiteren Verlauf dieser Arbeit als Client und könnte künftig einfach durch Simulink ersetzt werden.

MATLAB bietet in der „Data Import and Analysis Toolbox“ die Funktion *TCPCLIENT* [The Mathworks 2017] an. Damit ist der Datenaustausch über die Ethernet-Schnittstelle und dem TCP/IP-Protokoll möglich. MATLAB übernimmt dabei die Netzwerkrolle des TCP/IP-Clients. Um die Verbindung zur Robotersteuerung herzustellen, muss zuerst ein TCPCLIENT-Objekt erstellt werden. Dieser verlangt zwei Übergabeparameter: Der erste gibt die IP-Adresse des Servers bzw. der Robotersteuerung an. Der zweite gibt dessen Port an. In dieser Arbeit ergibt dies folgendes:

```
t = tcpclient('100.100.100.100',54600);
```

Sobald die Verbindung zur Robotersteuerung aufgebaut wurde, können Daten vom MATLAB-Client gelesen oder gesendet werden. Zum Lesen wird der Befehl `read(t)` genutzt, zum Schreiben der Befehl `write(t, data)`. Der zweite Parameter des Schreibbefehls beinhaltet die zu sendenden Daten.

3.3 Bewegungssteuerung des Manipulators in KRL

In diesem Abschnitt liegt der Fokus auf der Bewegungssteuerung des Manipulators. Dabei werden Gesteninformationen des Anwenders über die Kinect aufgenommen, in MATLAB verarbeitet und anschließend über Ethernet an die Robotersteuerung gesendet. Alle Anweisungen und Raumkoordinaten werden als XML-Datenpakete vom MATLAB-Client an die KRC4-Robotersteuerung gesendet. Dieser filtert alle relevanten Informationen aus dem XML-Datenpaket heraus und steuert bzw. bewegt anhand derer den Manipulator.

KUKA bietet für die Steuerung seiner Roboter die hauseigene KRL-Programmiersprache. Die KRL-Syntax ist der Programmiersprache Pascal ähnlich. Ein KRL-Programm, auch Modul genannt, besteht üblicherweise aus zwei Einheiten, der SRC- und der DAT-Datei. Die SRC-Datei ist das auszuführende Skript bzw. das eigentliche Programm. Es enthält alle Anweisungen und arbeitet diese in chronologischer Reihenfolge ab.
In der DAT-Datei können permanent verfügbare Variablen und Konstanten deklariert und definiert werden. Diese sind dann global in der gleichnamigen SRC-Datei bekannt und können benutzt werden. In der DAT-Datei werden zudem alle Teach-in Punkte aus den Inline-Bewegungsformularen abgelegt. Die SRC- und die DAT-Datei bilden zusammen ein sog. Modul. Eine SRC-Datei beginnt typischerweise mit der Deklaration und der Initialisierung von Variablen und Flags. Danach kann das eigentliche Programm geschrieben werden. Die allgemeine Syntax von KRL-Programmen wird im Folgenden nicht näher erläutert. Diese ist ausführlich in der entsprechenden Dokumentation [KUKA Roboter GmbH 2014c] beschrieben.
Um den Manipulator durch Gesteninformationen bewegen zu können, müssen empfangene XML-Datenpakete bzw. Variablen im KRL-Programm verfügbar sein; dies wird mit EthernetKRL erreicht. Deshalb werden im Folgenden die

Anweisungen und die Funktionsweise von EthernetKRL in KRL-Programmen vorgestellt.

Um Daten zwischen der Robotersteuerung und dem Arbeitsrechner auszutauschen, muss zuerst im KRL-Programm ein EKI-Kanal initialisiert und geöffnet werden. Wenn der EKI-Kanal nicht mehr benötigt wird, muss dieser explizit geschlossen und gelöscht werden. Dazu werden die Funktionen aus Tabelle 5 genutzt. Eine Variable vom Datentyp *EKI_STATUS* kann Rückgabewerte der Funktionen annehmen. Diese können z.B. Informationen darüber enthalten, ob die jeweilige Funktion erfolgreich ausgeführt werden konnte. Diese Variablen können im Laufe des KRL-Programms weiter genutzt werden, z.B. um Bedingungen zu prüfen.

Den Funktionen selbst wird die EKI-Kanalbezeichnung als Parameter übergeben. Diese kann entweder direkt als Übergabeparameter ausgeschrieben oder als Variable vom Datentyp *CHAR* übergeben werden. Die Bezeichnung eines Kanals kann frei ausgewählt werden, sollte jedoch einheitlich sein.

Tabelle 5: EthernetKRL Verbindungsfunktionen
[KUKA Roboter GmbH 2014b, S. 28]

Verbindung initialisieren, öffnen, schließen und löschen	
EKI_STATUS = EKI_Init(CHAR[])	→ Initialisieren des EKI-Kanals
EKI_STATUS = EKI_Open(CHAR[])	→ Öffnen des EKI-Kanals
EKI_STATUS = EKI_Close(CHAR[])	→ Schließen des EKI-Kanals
EKI_STATUS = EKI_Clear(CHAR[])	→ Buffer des EKI-Kanals löschen

EthernetKRL bietet die Möglichkeit, empfangene XML-Datenpakete zu „parsen“ und die entsprechenden Informationen für die Weiterverarbeitung in KRL vorzubereiten. Um XML-Datenpakete auszulesen oder zu senden, werden einige Funktionen bereitgestellt. Dabei unterscheiden sich diese bzgl. ihres Datentyps, der ausgelesen bzw. gesendet werden soll. In Tabelle 6 sind alle verfügbaren

Funktionen, die zum Senden von Daten genutzt werden können, aufgelistet. Dabei werden zuerst Informationen in einen Zwischenspeicher geschrieben, bevor sie anschließend über einen gesonderten Sendebefehl gesendet werden können.

Tabelle 6: EthernetKRL Sendefunktionen
[KUKA Roboter GmbH 2014b, S. 28]

Daten senden
EKI_STATUS = EKI_Send(CHAR[], CHAR[])
Daten schreiben
EKI_STATUS = EKI_SetReal(CHAR[], CHAR[], REAL)
EKI_STATUS = EKI_SetInt(CHAR[], CHAR[], INTEGER)
EKI_STATUS = EKI_SetBool(CHAR[], CHAR[], BOOL)
EKI_STATUS = EKI_SetFrame(CHAR[], CHAR[], FRAME)
EKI_STATUS = EKI_SetString(CHAR[], CHAR[], CHAR[])

Dem EKI-Sendebefehl müssen zwei Parameter übergeben werden. Der erste Parameter gibt die Bezeichnung des EKI-Kanals an. Im Beispielprogramm (siehe Anhang 2) wäre dies z.B. die Bezeichnung *XmlCallBack*. Der zweite Parameter gibt an, welcher Teil der definierten XML-Datenstruktur ausgewählt werden soll. Die XML-Datenstruktur wird bekanntlich in der entsprechenden XML-Konfigurationsdatei definiert.

Es ist z.B. möglich, einzelne Elemente bzw. Attribute zu versenden oder auch das gesamte XML-Datenpaket. Im Beispielprogramm *XmlCallBack* wird z.B. das gesamte XML-Datenpaket versendet (siehe Anhang 2, KRL-Programm, Zeile 80). Mit der Übergabe von *Robot*, wird der gesamte XML-Tag mit sämtlichen Inhalten ausgewählt. Mit der Angabe von „*Robot/Data/ActPos/@X*" würde nur dieses Attribut des Elementes *ActPos* ausgewählt und versendet werden. Dieses Auswahlprinzip gilt für alle EKI-Funktionen. Es bezieht sich jedoch immer auf die entsprechende XML-Datenstruktur aus der EKI-Konfigurationsdatei.

Die EKI-Schreibfunktionen besitzen im Vergleich zur EKI-Sendefunktion einen weiteren, dritten Übergabeparameter. Für die Übergabe muss zuerst im ersten Übergabeparameter der richtige EKI-Kanal ausgewählt werden. Der zweite Übergabeparameter bezieht sich auf das entsprechende XML-Element bzw. XML-Attribut, welches den Wert zugewiesen bekommen soll. Der dritte Übergabeparameter gibt letztendlich den Übergabewert an. Dies kann eine Variable, eine Zahl oder ein String sein.

In der Tabelle 7 sind alle EKI-Auslesefunktionen aufgelistet. Sie funktionieren genau wie die EKI-Schreibfunktionen, außer, dass der dritte Übergabeparameter eine KRL-Variable sein muss, die die entsprechenden Informationen übergeben bekommt und abspeichert. Des Weiteren sind diese jeweils nach dem zu lesenden Datentyp kategorisiert. Genau wie bei den EKI-Schreibfunktionen, muss hier ebenfalls die richtige Funktion entsprechend dem zu lesenden Datentyps ausgewählt werden. Die wichtigsten EKI-Funktionen wurden dargestellt und erklärt. Weitere EKI-Funktionen sind in der Dokumentation [KUKA Roboter GmbH 2014b, S. 28] beschrieben.

Tabelle 7: EthernetKRL Auslesefunktionen
[KUKA Roboter GmbH 2014b, S. 28]

Daten auslesen
EKI_STATUS = EKI_GetBool(CHAR[], CHAR[], BOOL)
EKI_STATUS = EKI_GetBoolArray(CHAR[], CHAR[], BOOL[])
EKI_STATUS = EKI_GetInt(CHAR[], CHAR[], Int)
EKI_STATUS = EKI_GetIntArray(CHAR[], CHAR[], Int[])
EKI_STATUS = EKI_GetReal(CHAR[], CHAR[], Real)
EKI_STATUS = EKI_GetRealArray(CHAR[], CHAR[], Real[])
EKI_STATUS = EKI_GetString(CHAR[], CHAR[], CHAR[])
EKI_STATUS = EKI_GetFrame(CHAR[], CHAR[], FRAME)
EKI_STATUS = EKI_GetFrameArray(CHAR[], CHAR[], FRAME[])

KUKA bietet in der KRL diverse Bewegungsfunktionen, mit denen der Manipulator bewegt werden kann. Der Endeffektor bzw. der TCP kann bestimmte Raumkoordinaten direkt anfahren. Dabei gibt es unterschiedliche Bewegungsarten. Diese sind im Folgenden aufgelistet und kurz erklärt:

- PTP-Bewegung
  - TCP bewegt sich entlang der schnellsten Bahn zum Zielpunkt
  - Bewegung zwischen Start- und Zielpunkt ist nicht linear und undefiniert
- LIN-Bewegung
  - TCP fährt linear und mit definierter Geschwindigkeit zum Zielpunkt
- CIRC-Bewegung
  - TCP fährt mit definierter Geschwindigkeit entlang einer Kreisbahn zum Zielpunkt
  - Kreisbahn ist durch Start-, Hilfs- und Zielpunkt definiert
- Spline-Bewegungen
  - TCP kann komplexe Bahn abfahren
  - Einzelne Punkte bzw. Spline-Segmente werden „geteacht“, daraus wird der optimale Bahnverlauf errechnet
  - Dieser wird mit der schnellstmöglichen Geschwindigkeit abgefahren

Diese Bewegungsarten können entweder direkt im KRL-Programm über *WorkVisual* oder über Inline-Formulare programmiert werden. Die Inline-Formulare werden empfohlen. Diese können direkt über das SmartPad in das KRL-Programm implementiert und genutzt werden. Die Anwendungsszenarien aus Abschnitt 4 sollen als Beispiel für die Nutzung der Bewegungsfunktionen dienen. Des Weiteren können alle nötigen Informationen in der entsprechenden Dokumentation [KUKA Roboter GmbH 2014c, S. 287–316] gefunden werden.

3.4 Grundlegendes Kommunikationsprinzip

Bei der Kommunikation zwischen der Robotersteuerung und dem Arbeitsrechner, können diverse Konfigurationen und Parameter gesetzt werden. Das kann komplex werden und während der Umsetzung zu Verwirrungen führen. Deshalb ist es wichtig, diese Prozesse möglichst übersichtlich zu dokumentieren. Auf diese Weise können Fehler vermieden und ggf. nachvollzogen werden.

Im Folgenden wird das Kommunikationsprinzip zwischen der Robotersteuerung und dem Arbeitsrechner dargestellt und erläutert. Außerdem wird ein Grundkonzept entwickelt, auf dessen Basis dann weitere Anwendungsszenarien aufgebaut werden können. Zunächst ist in Abbildung 3-8 der hardwaretechnische Aufbau des Robotersystems dargestellt. Alle Anwendungsszenarien werden bzgl. dieser Konfiguration entwickelt und durchgeführt.

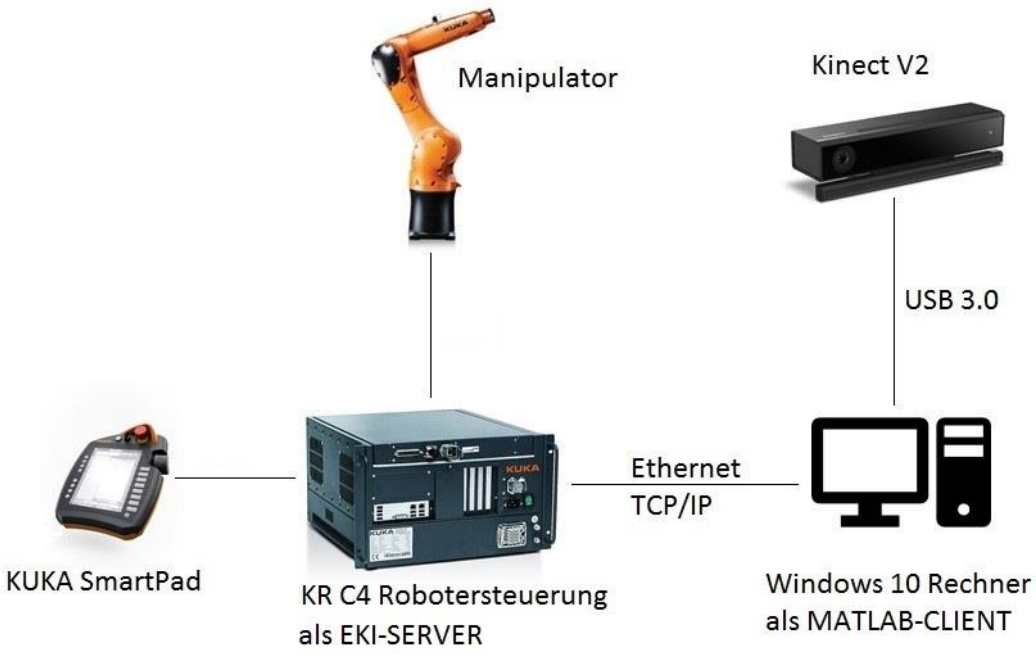


Abbildung 3-8: Aufbau und Konfiguration des Robotersystems

Um die softwaretechnische Kommunikation zwischen den Systemkomponenten darzustellen, wird in dieser Arbeit ein Sequenzdiagramm nach der UML 2.0 Modellierungssprache genutzt. Abbildung 3-9 (S. 50) stellt den allgemeinen Ablauf

der Bedienung des Robotersystems und der Kommunikation zwischen den Systemkomponenten dar. Auf dieser Grundlage können diverse Anwendungsfälle entwickelt und umgesetzt werden.

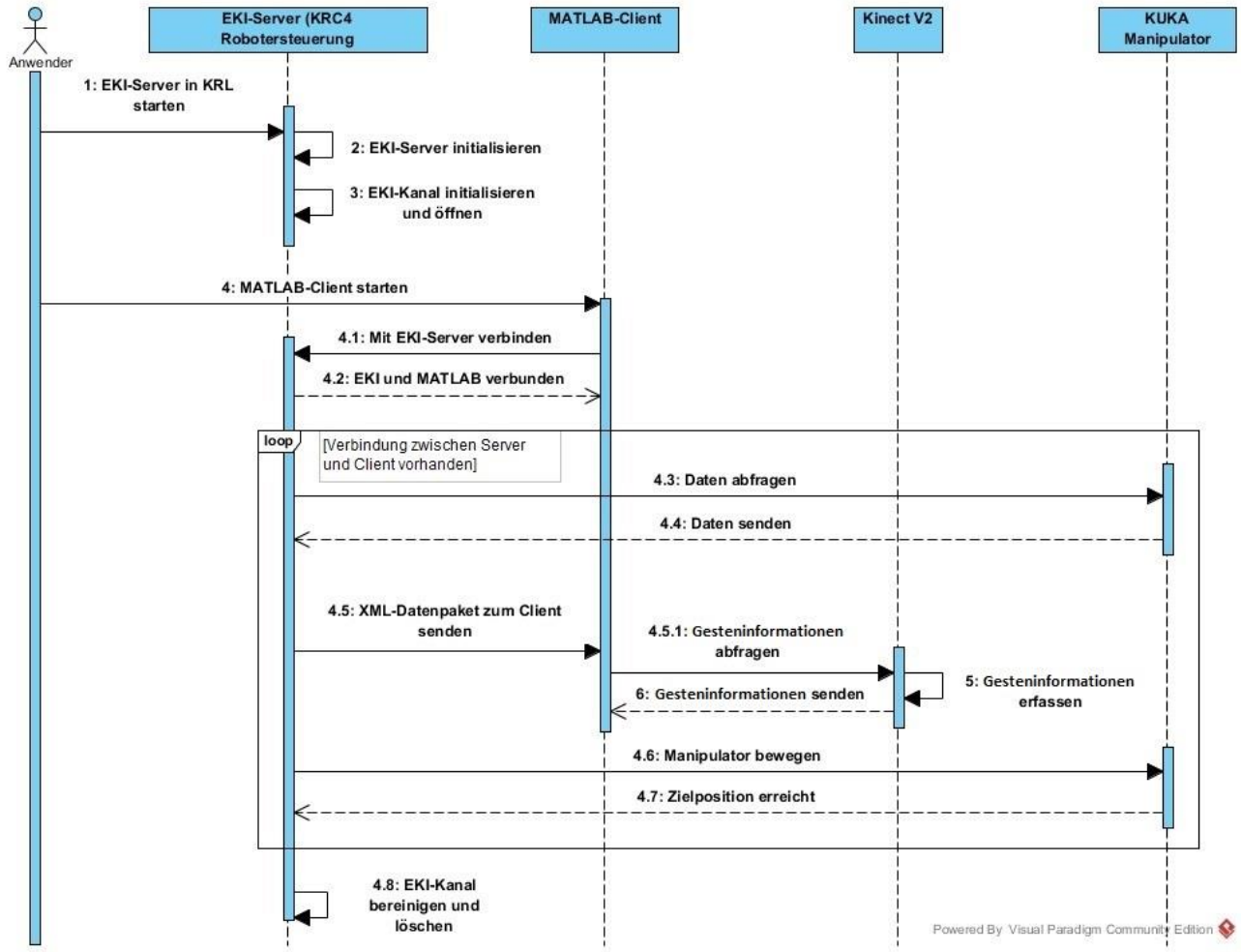


Abbildung 3-9: Grundlegender Kommunikationsablauf zwischen den Systemkomponenten

Zu Beginn muss der Anwender den EKI-Server im KRL-Programm auf der Robotersteuerung starten. Dabei müssen zunächst alle relevanten Variablen, Flags und Ausgänge deklariert und initialisiert werden. Danach kann der EKI-Server initialisiert und geöffnet werden. Der MATLAB-Client kann sich nun mit diesem verbinden. Der EKI-Server wartet an dieser Stelle so lange, bis eine Verbindung zum MATLAB-Client besteht. Währenddessen kann der Anwender den MATLAB-Client starten. Beide Netzwerkteilnehmer verbinden sich miteinander und können nun Daten austauschen.

Sobald die Verbindung zwischen der EKI-Server und dem Arbeitsrechner aufgebaut wurde, sendet dieser das definierte XML-Datenpaket an den MATLAB-Client. Im Anschluss werden die Gesteninformationen des Anwenders mit der Kinect erfasst und in MATLAB verarbeitet. Aus diesen wird dann ein XML-Datenpaket generiert, das über TCP/IP an den EKI-Server versendet wird. Das vom MATLAB-Client versendete XML-Datenpaket muss mit der XML-Strukturdefinition der EKI-Konfigurationsdatei konform sein. Der EKI-Server empfängt das XML-Datenpaket, filtert alle empfangen Informationen heraus und weist diese den entsprechenden KRL-Variablen zu. Diese können für die Steuerung des Manipulators genutzt werden, wie z.B. Körpergelenkkoordinaten oder Hand-States.

Der Datenaustausch zwischen den beiden Netzwerkteilnehmern findet so lange statt, bis die Verbindung zwischen diesen abbricht. Sobald die Verbindung nicht mehr nachgewiesen werden kann, wird der EKI-Kanal von der Robotersteuerung aus geschlossen. Informationen, die sich noch im EKI-Zwischenspeicher befinden, können noch abgearbeitet werden. Nach dem Verbindungsabbruch empfiehlt es sich, den EKI-Zwischenspeicher zu löschen. Die Anwendung und der Datenaustausch werden beendet.

4 Entwicklung und Realisierung demonstrativer Anwendungsszenarien

In diesem Kapitel sollen diverse Anwendungsszenarien entwickelt werden. Diese sollen die Anwendungsmöglichkeiten dieser Arbeit verdeutlichen. Alle Anwendungsszenarien basieren auf dem Kommunikationsprinzip aus Abbildung 3-9 (S. 50). Die folgenden Anwendungsszenarien befassen sich jedoch ausführlicher mit dem Datenaustausch und der Ausführung. Abschnitt 4.1 ist ausführlich beschrieben. Einige Thematiken wiederholen sich in den Abschnitten 4.2 und 4.3 und werden deshalb nur kurz erwähnt. Alle erwähnten Programmschritte sind zusätzlich in den Programmcodes im Anhang kommentiert.

Im ersten Anwendungsszenario folgt der TCP des Manipulators der rechten Hand des Anwenders. Es soll demonstrieren, dass Gesteninformationen schnell genug erfasst werden und den Manipulator mit kurzer Verzögerung beeinflussen können. Im Allgemeinen soll sie die Reaktion des Roboters auf erhaltene Befehle demonstrieren.

Das zweite Anwendungsszenario werden Bahnen bzw. Konturen über die Gestik des Anwenders vorgegeben. Der Manipulator kann die vorgegebene Bewegung abfahren. Es soll zeigen, dass auch komplexe Bewegungsabläufe erkennt und ausführt werden können, ähnlich wie beim Playback-Verfahren von Lackierrobotern.

Das dritte Anwendungsszenario demonstriert den direkten Kontakt zwischen Mensch und Roboter. Der Anwender entscheidet durch seine Gestik, welche verfügbaren Objekte er übergeben bekommen möchte. Der Roboter besitzt die Fähigkeit, diese Befehle zu interpretieren und das verlangte Objekte an den Anwender zu übergeben.

4.1 PTP-Verfolgung einer vorgegebenen Raumkoordinate

Im ersten Anwendungsfall wird der TCP des Manipulators, der rechten Hand des Anwenders folgen. Außerdem soll erkannt werden, ob die rechte Hand geöffnet oder geschlossen ist. Anhand dessen öffnet oder schließt sich der Greifer. Mit der linken Hand werden dem Roboter definierte Kommandos gegeben. Das Aktivitätsdiagramm aus Abbildung 4-1 (S. 54) stellt die Funktionsweise des Programms aus Anhang 3 dar. Es wurde während Entwurfsphase hergeleitet und nach dessen praktischer Erprobung ergänzt.

Als erstes muss der Anwender das KRL-Server-Programm (siehe Anhang 3) ausführen. In diesem werden alle nötigen Variablen deklariert und definiert. Dabei handelt es sich um eine Variable vom Typ FRAME, die die Raumkoordinaten und die -orientierung des TCP's speichert. Außerdem wird eine Variable vom Typ INTEGER benötigt, die den Status des Greifers speichert und zur Verfügung stellt. Anschließend fährt der TCP in die Home-Position. Dies ist die festgelegte Ruhelage des Manipulators und sollte standardmäßig vor jeder Programmausführung angefahren werden.

Als nächstes wird ein EKI-Kanal geöffnet und initialisiert. Hier ist darauf zu achten, dass der EKI-Kanal die gleiche Bezeichnung wie dessen XML-Konfigurationsdatei besitzt. Der EKI-Server sollte nun betriebsbereit sein und Client-Verbindungen akzeptieren. Jetzt kann das MATLAB-Client Programm ausgeführt werden. Dieser erstellt vor dem Verbindungsversuch ein Kinect-Objekt mit allen nötigen Parametern. Die Kinect-Hardware wird über dieses Objekt gesteuert. Als nächstes verbindet sich der MATLAB-Client mit dem EKI-Server.

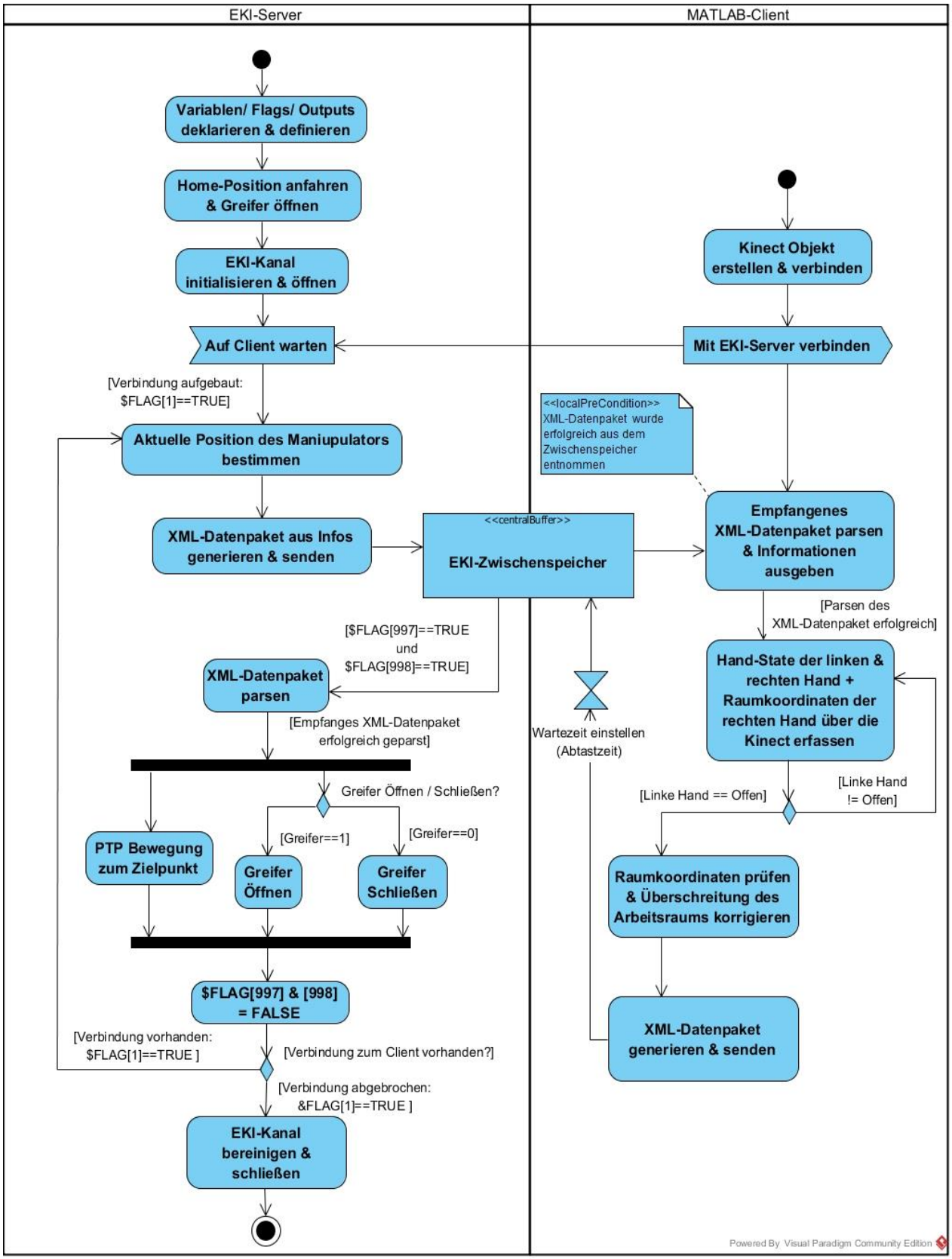


Abbildung 4-1: Aktivitätsdiagramm zur PTP-Verfolgung

EthernetKRL bietet die Möglichkeit einer periodischen Verbindungüberprüfung zwischen der Robotersteuerung und jeglichen externen Systemen. Mit Hilfe des XML-Elementes *ALIVE* aus der XML-Konfiguration kann festgelegt werden, ob eine Verbindung besteht und in welchen zeitlichen Abständen diese überprüft werden soll. Außerdem können z.B. Flags im KRL-Programm gesetzt werden, wenn die Verbindung vorhanden ist. In diesem Anwendungsszenario wird bei bestehender Verbindung *FLAG[1]* auf *TRUE* gesetzt. Sobald die Verbindung abbricht, wird *FLAG[1]* automatisch wieder auf *FALSE* gesetzt. Flags sind standardmäßig auf *FALSE* gesetzt, welches eine Initialisierung erspart. Der EKI-Server wartet deshalb so lange, bis die Verbindung zum MATLAB-Client aufgebaut wurde bzw. bis *FLAG[1]* gesetzt wird.

Sobald eine stabile Verbindung hergestellt wurde, analysiert die Robotersteuerung die aktuelle Position und Orientierung des TCPs. Diese Informationen werden zu einem XML-Datenpaket zusammengefasst und an den EKI-Zwischenspeicher gesendet. Das gesendete XML-Datenpaket muss der XML-Konfiguration entsprechen. Folgende XML-Datenpakete werden zwischen dem EKI-Server und dem MATLAB-Client ausgetauscht:

Tabelle 8: Ausgetauschte XML-Datenpakete während der PTP-Verfolgung

MATLAB-Client → EKI-Server	EKI-Server → MATLAB-Client
<Sensor> <Read> <RechteHandFrame X="..." Y="..." Z="..." A="-6.5" B="86.8" C="-6.4" /> <Greifer>...</Greifer> </Read> </Sensor>	<Robot> <Data> <ActPos X="..." Y="..." Z="..." A="..." B="..." C="..." /> </Data> </Robot>

RechteHandFrame beinhaltet die Raumkoordinate und die Orientierung der rechten Hand des Anwenders. Die Orientierung (A, B und C) wird jedoch kontant an die Robotersteuerung übergeben. Lediglich die Raumposition der rechten Hand ist variabel. Greifer übergibt entweder den Wert 1 oder 0, analog zu

geöffnetem und geschlossenem Greifer. Der EKI-Server sendet nur die aktuelle Position und Orientierung des TCPs an den MATLAB-Client.

Sobald diese Informationen im EKI-Zwischenspeicher liegen, werden sie vom MATLAB-Client abgerufen. Darauffolgend wird das empfangene XML-Datenpaket in MATLAB gefiltert und analysiert. Alle empfangenen Informationen werden in entsprechende Variablen geladen und zur Weiterverarbeitung bereitgestellt.
Um Gesteninformationen über die Kinect erfassen zu können, müssen zuerst Momentaufnahmen erzeugt und gespeichert werden. Aus dem Bild können alle benötigten Informationen herausgefiltert werden, wie z.B. die Raumkoordinaten der Gelenke. In dieser Anwendung ist es vorgesehen, dass die linke Hand des Anwenders als sog. „Kommandohand“ dient. Darüber können bestimmte Aktionen und Bewegungen ausgelöst werden. Die rechte Hand wird zum Positionieren des TCPs und zum Öffnen und Schließen des Greifers genutzt. Folgendes Anwendungsschema ist vorgesehen:

- Linke Hand: Geschlossen
    - TCP verweilt in der aktuellen Position
    - MATLAB-Client sendet keine XML-Datenpakete
- Linke Hand: Geöffnet und rechte Hand: Geöffnet
    - TCP bewegt sich bzgl. der Y- und der Z-Achse des Manipulators zur rechten Hand des Anwenders
    - TCP bleibt bzgl. der X-Achse konstant
    - Der Greifer ist geöffnet
- Linke Hand: Geöffnet und rechte Hand: Geschlossen
    - TCP bewegt sich bzgl. der Y- und der Z-Achse des Manipulators zur rechten Hand des Anwenders
    - TCP bleibt bzgl. der X-Achse konstant
    - Der Greifer ist geschlossen

Entsprechend dieses Anwendungsschemas wird ersichtlich, dass der MATLAB-Client nur Informationen an den EKI-Server sendet, wenn sich die linke Hand des Anwenders öffnet. Im geschlossenen Zustand werden keine Daten gesendet und der MATLAB-Client wartet, bis sich die linke Hand öffnet. Außerdem hat EKI-Server ein reaktives Verhalten, d.h. immer, wenn Daten im EKI-Zwischenspeicher liegen, wird das entsprechende KRL-Programm ausgeführt, ansonsten wird gewartet. Sendet der MATLAB-Client zu häufig, könnte dies zu unnötigem Datenverkehr und zu entsprechenden Überlastungen führen. Die eingestellte Wartezeit definiert demnach nicht nur die Sendefrequenz des MATLAB-Clients, sondern auch die Abtastfrequenz der Kinect. Hier steht es dem Anwender frei, eine angemessene Wartezeit einzustellen. In dieser Arbeit wurde eine Wartezeit von 0,5 s bis 1 s gewählt. Je kürzer die Wartezeit, desto höher ist die Abtastfrequenz der Kinect bzw. die Sendefrequenz in MATLAB. Die Robotersteuerung müsste alle empfangenen XML-Datenpakete in kürzester Zeit verarbeiten. Dies kann zu ungleichmäßigen und verzögerten Bewegungen des Manipulators führen.

Bevor die erfassten Raumkoordinaten der rechten Hand an den EKI-Server gesendet werden, werden diese mit einer festgelegten Begrenzung bzgl. des Arbeitsraumes des Manipulators verglichen und ggf. angepasst. So wird verhindert, dass die Robotersteuerung evtl. Raumkoordinaten anfährt, die nicht möglich sind oder zu Kollisionen führen. Die integrierte Sicherheitseinrichtung des Robotersystems würde dies zwar verhindern, ggf. jedoch zu Fehlern führen und die Steuerung verlangsamen bzw. unterbrechen. Es empfiehlt sich daher, diese Werte in MATLAB zu prüfen und erst dann an den EKI-Zwischenspeicher zu versenden.

Sobald *FLAG[997]* und *FLAG[998]* auf *TRUE* gesetzt wurden, ist dem EKI-Server bekannt, dass XML-Datenpakete im EKI-Zwischenspeicher zur Abholung

bereitstehen. Beide Flags signalisieren die Verfügbarkeit von unterschiedlichen Informationen im Zwischenspeicher:

- FLAG[997]: Raumkoordinaten der rechten Hand wurden empfangen
- FLAG[998]: Status des Greifers wurde empfangen

Es empfiehlt sich für jedes XML-Element einen gesonderten Flag zu setzen. Dadurch können XML-Elemente einzeln herausgefiltert werden. Nachdem die XML-Elemente aus dem Zwischenspeicher entnommen wurden, müssen die entsprechenden Flags manuell auf *FALSE* gesetzt werden. Dieser Schritt ist unbedingt auszuführen, da sich die genannten Flags sonst dauerhaft auf *TRUE* befinden und nicht mehr erkannt werden kann, ob sich neue XML-Datenpakete im EKI-Zwischenspeicher befinden. Neue Anweisungen bzw. Raumkoordinaten würden in dem Fall nicht mehr erkannt werden.

Das XML-Datenpaket wird von der Robotersteuerung aus dem EKI-Zwischenspeicher entnommen und mit einem integrierten XML-Parser analysiert. Alle beinhalteten Informationen werden herausgefiltert und in entsprechende KRL-Variablen geladen. Die Bewegung des Manipulators wird gestartet, sobald alle nötigen Informationen erfolgreich empfangen wurden.
In diesem Anwendungsszenario bewegt sich der TCP des Manipulators über eine PTP-Bewegung zur empfangenen Raumkoordinate der rechten Hand des Anwenders. Dem PTP-Befehl wird eine Variable vom Typ FRAME übergeben. Diese beinhaltet die Raumorientierung und die Raumposition, die der TCP anfahren soll. Parallel dazu öffnet bzw. schließt sich der Greifer. Sobald der Zielpunkt erreicht wurde und der Greifer den Status übernommen hat, kann das Programm fortgeführt werden und neue Raumkoordinaten anfahren.

Das Programm kann nur fortgesetzt werden, wenn die Verbindung zwischen EKI-Server und MATLAB-Client noch besteht. Deshalb wird zunächst geprüft, ob *FLAG[1]* auf *TRUE* oder *FALSE* gesetzt ist. Dieser signalisiert den Verbindungsstatus zwischen den Netzwerkteilnehmern. Ist die Verbindung vorhanden (*FLAG[1]==TRUE*), wird das Programm fortgesetzt. Als nächstes wird die aktuelle Position des Manipulators bestimmt und über ein XML-Datenpaket an den EKI-Zwischenspeicher gesendet. Das Programm wird vollständig beendet, falls die Verbindung abgebrochen ist (*FLAG[1]==FALSE*). Zum Fortsetzen wäre in dem Fall ein Neustart erforderlich.

Bei der Anwendung ist aufgefallen, dass der Manipulator bzw. dessen TCP schwingt, obwohl sich die rechte Hand des Anwenders dauerhaft und konstant an einer Position befand. Ein Grund könnte die eingeschränkte Wiederholgenauigkeit der Kinect bzgl. der Erfassung von Körpergelenkkoordinaten sein. Außerdem tendiert die menschliche Hand dazu, selbst bei bewusster Haltung eines Raumpunktes, minimale Bewegungen auszuführen. Ein weiterer Grund könnte sein, dass sich die Kinect während der Ausführung, auf dem gleichen Podest befand, wie der Manipulator. Beide Einheiten waren somit mechanisch gekoppelt. Die ruckhaften Bewegungen des Manipulators lösten ebenfalls Ruckbewegungen bzw. Schwingungen an der Kinect aus und führten somit zu einer unruhigen Aufnahme.

Um zu verhindern, dass der TCP während der Anwendung schwingt, wurden diverse Maßnahmen getestet. Zum Beispiel wurde ein *Average Moving Filter* in MATLAB genutzt, um die dauerhaft erfassten Raumkoordinaten der rechten Hand des Anwenders zu glätten und schnelle und kurze Bewegungen herauszufiltern. Dies führte zu einer Verfälschung der erfassten Raumkoordinaten und wurde deshalb nicht in das Programm implementiert.

Im Endeffekt wurde die Kinect auf ein eigenes Podest gestellt, das mechanisch von dem des Manipulators getrennt ist. Außerdem wird empfohlen, die Ausführungsgeschwindigkeit des KRL-Programms zu reduzieren. Auf diese Weise werden ruckartige Bewegungen des Manipulators verringert. Insgesamt konnte dieses Anwendungsszenario erfolgreich ausgeführt werden.

4.2 Spline-Bewegung des Industrieroboters

In diesem Abschnitt soll der Manipulator eine Spline-Bewegung anhand einer vorgegebenen Bahn abfahren. Die Spline-Bewegung kann in KRL über den Spline-Block realisiert werden, der aus einzelnen Spline-Segmenten bzw. Raumpunkten besteht. In diesem Anwendungsszenario werden Spline-Segmente über die Kinect erfasst und anschließend „geteacht“ werden. Die Robotersteuerung berechnet den optimalen Bahnverlauf anhand der Spline-Segmente. Die Bahn wird anschließend als ein Bewegungssatz ausgeführt. Die gleiche Bahn könnte auch über LIN- und CIRC-Bewegungen abgefahren werden, jedoch wäre die Bewegung nicht so flüssig wie bei einer Spline-Bewegung.

Das Aktivitätsdiagramm in Abbildung 4-2 (S. 61) stellt den Ablauf des Programms dar. Der Verbindungsaufbau und der Datenaustausch sind identisch zur PTP-Verfolgung aus Abschnitt 4.1. Die ausgetauschten Informationen sind jedoch nicht gleich und sind wie gewohnt in der entsprechenden EKI-Konfigurationsdatei (siehe Anhang 4) definiert. Sobald die Verbindung zwischen EKI-Server und MATLAB-Client erfolgreich hergestellt worden ist, sendet der EKI-Server das konfigurierte XML-Datenpaket. Der MATLAB-Client versucht Daten aus dem EKI-Zwischenspeicher zu entnehmen und diese auszugeben.

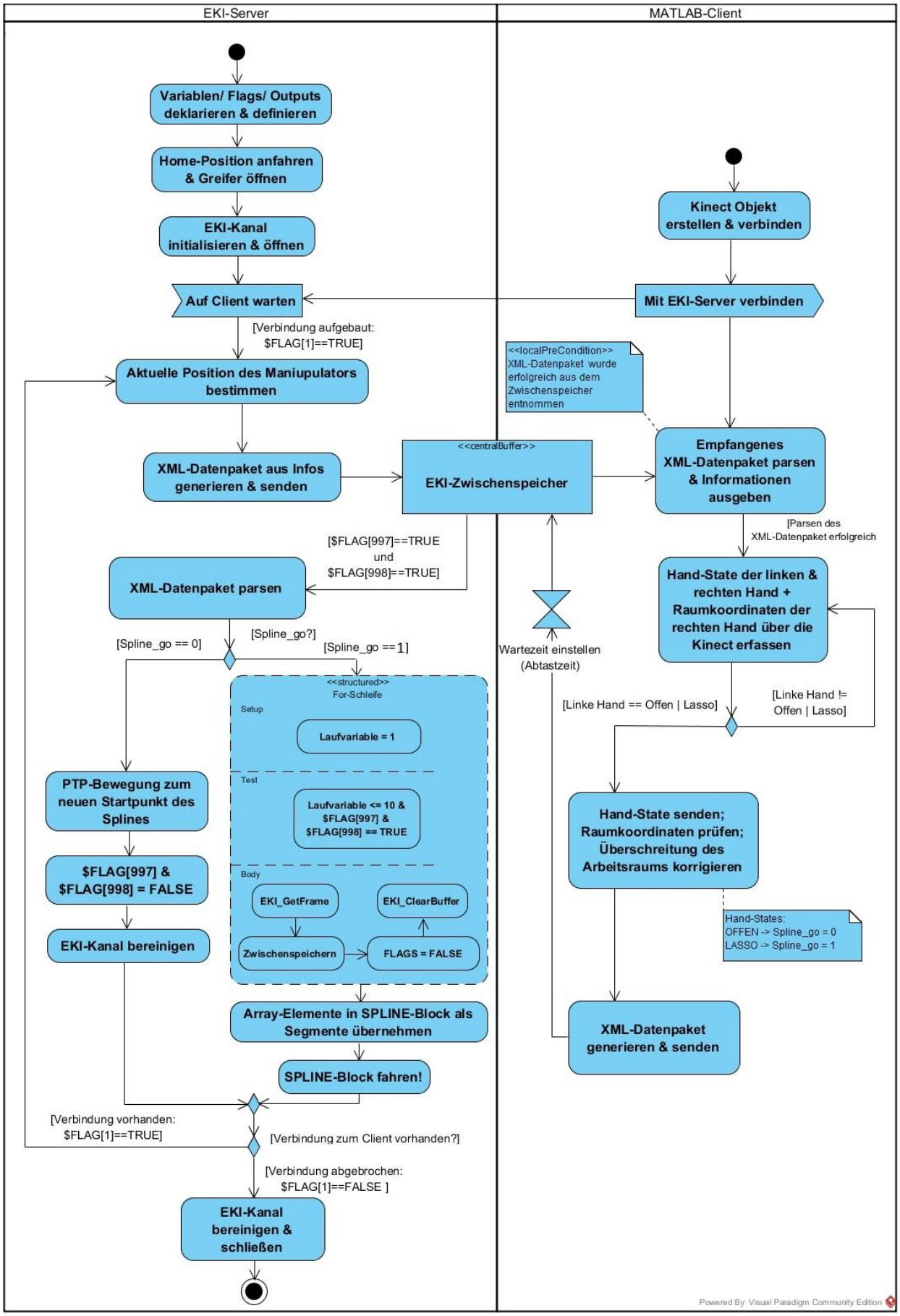


Abbildung 4-2: Aktivitätsdiagramm zur Spline-Bewegung

Nachdem der MATLAB-Client alle vorhandenen XML-Datenpakete aus dem EKI-XML-Zwischenspeicher entnommen, deren Informationen herausgefiltert und diese ausgegeben hat, wird das aktuelle Frame von der Kinect angefordert. Im Gegensatz zur PTP-Verfolgung, in der nur geprüft wird, ob die linke Kommandohand des Anwenders geöffnet oder geschlossen ist, wird in dieser Anwendung zusätzlich geprüft, ob diese ein Lasso-Zeichen bildet. Falls die linke Hand des Anwenders geöffnet ist, wird *Spline_go* auf 0 gesetzt. Bildet diese ein Lasso, dann wird *Spline_go* auf 1 gesetzt. Die Raumposition der rechten Hand wird ebenfalls erfasst. Anschließend wird geprüft, ob erfassten Raumkoordinaten im Arbeitsraum des Manipulators befindet. Bei Überschreitung wird der Wert angepasst. Alle erfassten Informationen werden zu einem XML-Datenpaket zusammengefasst und an den EKI-Server gesendet.

Vor kann eine gewisse Wartezeit festgelegt werden. Auf diese Weise kann die Aufnahmefrequenz von Spline-Elementen reguliert werden. Ansonsten bestimmt diese Wartezeit auch die Ausführungsgeschwindigkeit der Anwendung, genau wie im ersten Anwendungsfall. Es wird eine Wartezeit von 0,5s bis 1s empfohlen.

In diesem Anwendungsszenario werden folgende XML-Datenpakete zwischen dem EKI-Server und dem MATLAB-Client, ausgetauscht:

Tabelle 9: Ausgetauschte XML-Datenpakete während der Spline-Bewegung

MATLAB-Client → EKI-Server	EKI-Server → MATLAB-Client
<Sensor> <Read> <RechteHandFrame X="..." Y="..." Z="..." A="-6.5" B="86.8" C="-6.4" /> </Read> <Status> <Spline_go>BOOL-Wert</Spline_go> </Status> </Sensor>	<Robot> <Data> <ActPos X="..." Y="..." Z="..." A="..." B="..." C="..." /> </Data> </Robot>

Das Frame *RechteHandFrame* enthält die Raumkoordinaten und die Orientierung der rechten Hand des Anwenders. Die Raumkoordinaten X, Y und Z sind variabel

und ändern sich mit der Position der rechten Hand. Die Orientierung der rechten Hand und folglich die Orientierung des TCPs ist genau wie im ersten Anwendungsszenario fest vorgegeben. *Spline_go* beinhaltet einen Bool-Wert und legt fest, ob mit der Aufnahme von Spline-Segmenten begonnen werden soll oder nicht.

Der Robotersteuerung wird anhand von Flags bekannt gegeben, dass Informationen aus dem EKI-Zwischenspeicher abgerufen werden können. Jedem XML-Element ist ein individuelles Flag zugewiesen. Dies erleichtert die Unterscheidung der empfangen Daten.

- FLAG[997]: Informationen bzgl. des Hand-States der linken Hand stehen bereit
- FLAG[998]: Informationen bzgl. der Raumposition der rechten Hand stehen bereit

Die Informationen aus dem XML-Datenpaket werden herausgefiltert und in entsprechende KRL-Variablen geladen. Der Manipulator wird anhand der erhaltenen Informationen gesteuert.

Dabei gilt für die Steuerung der Spline-Verfolgung folgendes Kommandoschema:

- Linke Hand geöffnet:
  - TCP fährt zum Startpunkt der folgenden Spline-Bewegung
  - Startpunkt der Spline-Bewegung wird über die Raumposition der rechten Hand vermittelt
  - Falls bereits Spline-Segmente erfasst wurden: Keine PTP-Bewegung
- Linke Hand zeigt Lasso:
  - Raumposition der rechten Hand wird in der Robotersteuerung als Spline-Segment übernommen
  - Wenn Spline-Block gefüllt ist: TCP fährt vorgegebene Bahn ab

Bei Spline-Bewegungen fährt der TCP immer eine vorgegebene Bahn bzw. Kontur ab. Der Startpunkt der Spline-Bewegung ist immer die momentane Raumposition des TCPs. Falls der Wunsch bestehen sollte, die vorgegebene Bahn von einer bestimmten Startposition aus zu beginnen, muss der TCP vorher in die gewünschte Position bewegt werden.
Dem TCP wird die gewünschte Startposition über die rechte Hand des Anwenders übergeben. Den Fahrbefehl gibt die offene linke Hand. Zum Anfahren der Raumposition wird eine PTP-Bewegung verwendet. In diesem Fall wäre die Variable *Spline_go* in KRL gleich 0. Eine PTP-Bewegung wird nur dann ausgeführt, wenn momentan keine Spline-Segmente aufgenommen werden. Nachdem die Zielposition über PTP angefahren wurde, werden *FLAG[997]* und *FLAG[998]* explizit auf *FALSE* gesetzt. Es ist wichtig, dass die Flags manuell zurückgesetzt werden, da diese sonst dauerhaft *TRUE* signalisieren. Dies führt dazu, dass die Bereitstellung von neuen Informationen im EKI-Zwischenspeicher nicht mehr von der Robotersteuerung erkannt wird.

Der TCP kann so lange bewegt werden, bis das erste Spline-Segment aufgenommen wird. Die Aufnahme von Spline-Segmenten wird über das Lasso-Zeichen der linken Hand initialisiert. Ist dies der Fall, wird *Spline_go* auf 1 gesetzt. Das KRL-Programm nimmt die erwarteten Spline-Segmente in einer FOR-Schleife auf. Mit Hilfe der Laufvariable innerhalb der FOR-Schleife werden insgesamt 10 Spline-Segmente aufgenommen. Die Anzahl der aufgenommen Spline-Segmente kann beliebig geändert werden. Der Maximalwert der Laufvariable definiert die Menge der Spline-Segmente in einem Spline-Block. Falls dieser Wert noch nicht erreicht wurde, bleibt das Programm in der FOR-Schleife. Außerdem wird geprüft, ob *FLAG[997]* und *[998]* auf *TRUE* gesetzt sind. Dies signalisiert, dass neue Spline-Segmente im EKI-Zwischenspeicher bereitstehen.

Sind diese Voraussetzungen erfüllt, wird die Raumposition über *GetFrame* aus dem EKI-Zwischenspeicher entnommen und in einem KRL-Array zwischengespeichert. Dieses Array muss die gleiche Anzahl von Speicherplätzen wie der Spline-Block besitzen. Danach wird der EKI-Zwischenspeicher entleert, um Platz für neue Daten zu schaffen. Dieser Vorgang wird solange wiederholt, bis das Array 10 Frames bzw. Raumkoordinaten beinhaltet. Diese werden anschließend dem Spline-Block übergeben, die anhand der erhaltenen Raumkoordinaten die optimale Bahn berechnet und diese dann abfährt.

Nachdem die PTP-Bewegung angefahren oder der Spline-Block abgefahren wurde, wird geprüft, ob die Verbindung zwischen dem EKI-Server und dem MATLAB-Client vorhanden ist. Ist die Verbindung abgebrochen, wird der EKI-Kanal geleert und geschlossen. Die Anwendung wird beendet. Falls die Verbindung noch vorhanden ist, wird die Anwendung weiter fortgeführt.

Am Ende soll noch erwähnt werden, dass die Splines in dieser Anwendung sich nicht beliebig im Raum bewegen können. In Abbildung 4-3 (S. 66) wird diese Einschränkung bildhaft dargestellt. Zunächst sind zwei Hälften des Arbeitsraumes zu unterscheiden. Wird der Manipulator von vorne betrachtet, dann befindet sich links von der mittleren Achse des Manipulators die eine Hälfte des Arbeitsraums, rechts die andere. Spline-Bewegungen können innerhalb der einen Hälfte beliebig vorgegeben werden. Der Manipulator fährt diese im Regelfall ab, wie z.B. die Spline-Bewegungssätze 1 und 2. Die Spline-Bewegung bricht den Bewegungssatz ab, falls die mittlere Achse überschritten wird. Als Beispiel ist der Spline-Bewegungssatz 3 gegeben. Das erste Spline-Segment des dritten Bewegungssatzes kann abgefahren werden (grün gekennzeichnet). Das zweite Spline-Segment des dritten Bewegungssatzes sieht eine Überschreitung der mittleren Achse vor und kann entsprechend nicht abgefahren werden (rot gekennzeichnet).

Der Grund für dieses Verhalten liegt in der Roboterkinematik. Dadurch dass die Orientierung des TCPs in dieser Anwendung vorgegeben ist, kommt es bei der Überschreitung der mittleren Achse zu einer ungültigen Achsstellung. Um dieses Verhalten zu umgehen, könnte die Orientierung des TCPs ignoriert werden. Diese wäre dann zwar nicht mehr konstant, jedoch würde eine Überschreitung der mittleren Achse nicht mehr zum Abbruch der Spline-Bewegung führen.

In KRL gibt es auch die Möglichkeit, die Orientierung von einzelnen Spline-Segmenten zu ignorieren (mit $ORI_TYPE = #IGNORE), diese aber für die restlichen Spline-Segmente beizubehalten [KUKA Roboter GmbH 2014c, S. 305–306]. Aus demonstrationsgründen wird eine feste Orientierung des TCPs beabsichtigt. Dieses Verhalten wird deshalb in Kauf genommen.

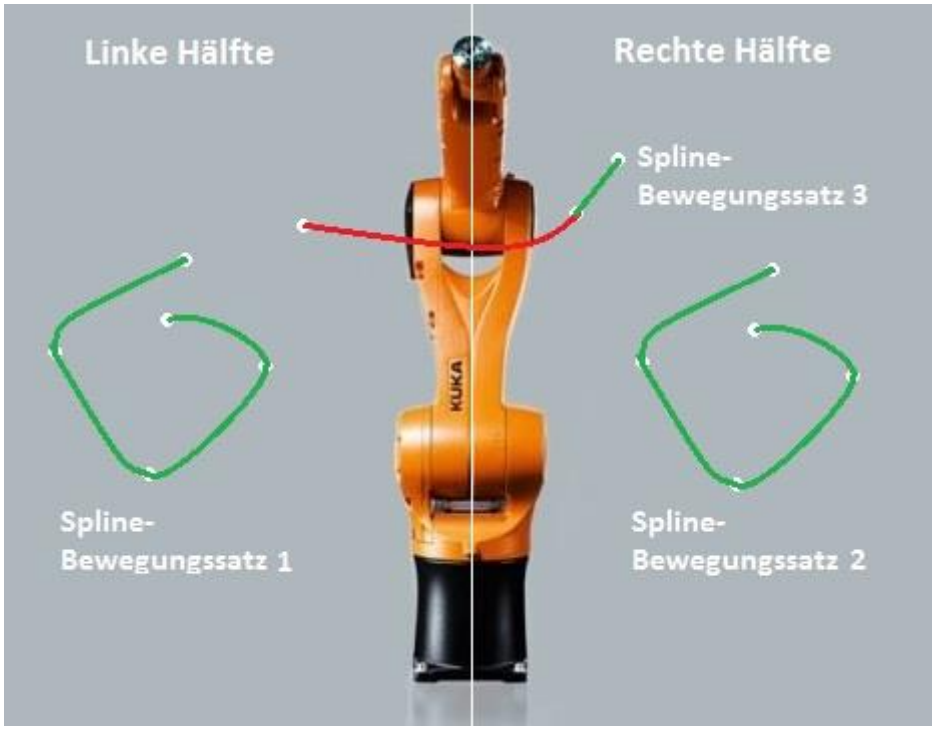


Abbildung 4-3: Einschränkung der Spline-Bewegung

4.3 Objektübergabe zwischen Mensch und Roboter

Das dritte Anwendungsszenario befasst sich mit der Objektübergabe. Dem Industrieroboter stehen an einer bestimmten Position drei Fächer zur Verfügung (siehe Abbildung 6-1, S. 76). Das linke und das rechte Fach beinhalten handgroße Stäbe in verschiedenen Farben, das mittlere soll leer bleiben. Der Anwender soll über definierte Gesten entscheiden können, welches Objekt er übergeben bekommen möchte. Der Roboter soll dann das entsprechende Fach anfahren, den Stab entnehmen und ihn zur Raumposition der rechten Hand des Anwenders transportieren und übergeben.

Der direkte Kontakt zwischen Mensch und Roboter steht ganz im Sinne der Mensch-Roboter-Kollaboration. Jedoch kommt es aufgrund der Überschreitung der Sicherheitslichtschranke während der Objektübergabe, dass dieses Anwendungsszenario nur im T1-Testmodus des Industrieroboters ausgeführt werden kann. Im Testbetrieb und ist der Roboter entsprechend langsam. Für die Praxis ist eine schnellere Ausführung zwar wünschenswert, wegen der Sicherheitsvorgaben durch KUKA jedoch eingeschränkt. Der automatische Betrieb während der MRK ist deshalb nur möglich, wenn kein direkter Kontakt zwischen Mensch und Roboter besteht. Hier besteht weiterhin Forschungs- und Optimierungspotential.

Das Aktivitätsdiagramm aus Abbildung 4-4 (S. 68) stellt den Programmablauf während der Objektübergabe dar. Zu Beginn steht wie üblich die Deklaration und die Definition der nötigen Variablen, Flags und Outputs. Als nächstes wird zuerst die Home-Position angefahren und dann der Greifer geöffnet. Diese Reihenfolge ist einzuhalten, weil das Öffnen des Greifers in einer anderen Position dazu führen könnte, dass dieser eventuell mit umgebenden Objekten kollidiert. Sobald diese Schritte erfolgreich durchgeführt wurden, kann der EKI-Kanal initialisiert und geöffnet werden. Verbindungsanfragen von externen Clients können nun angenommen werden.

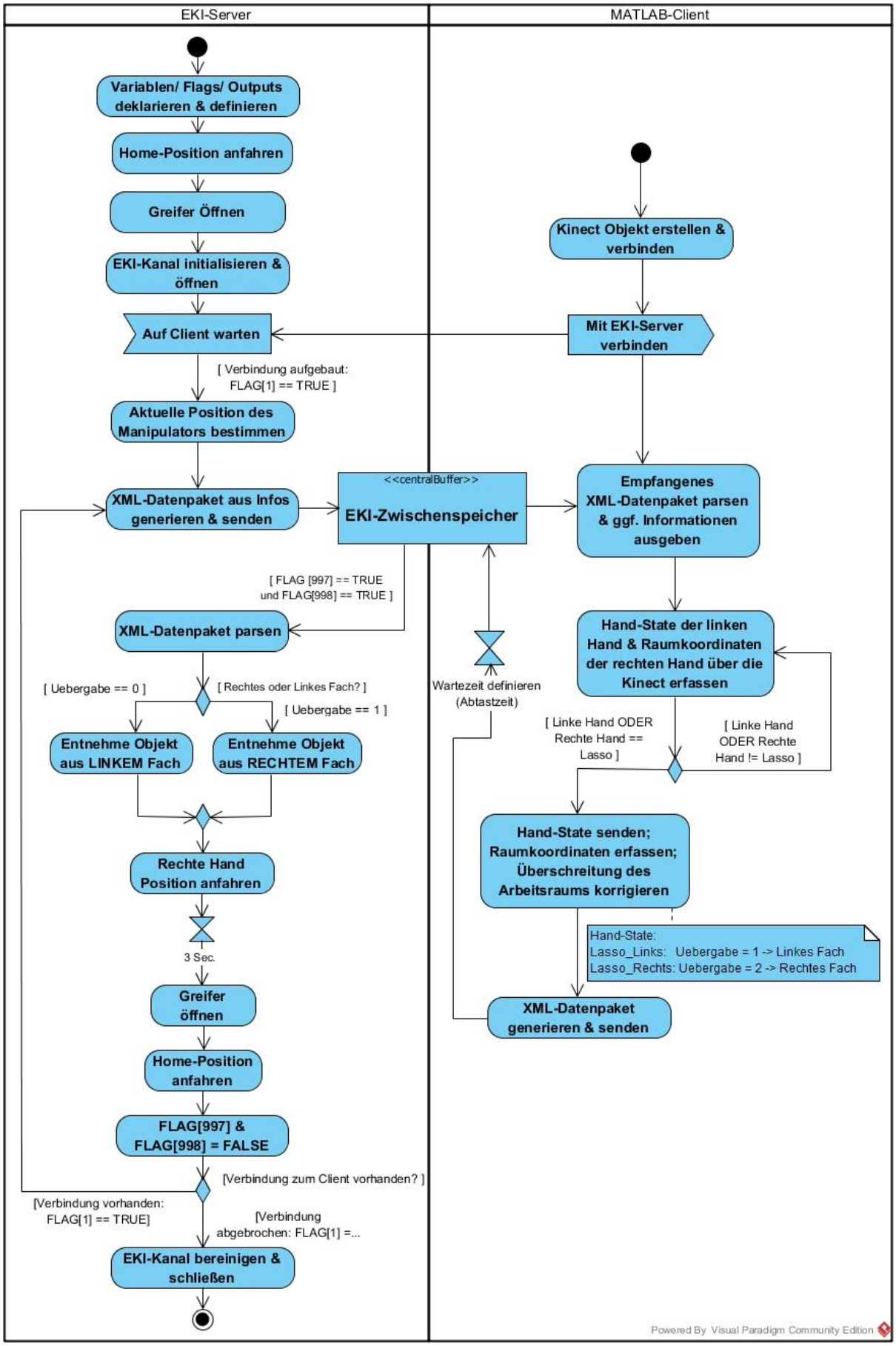


Abbildung 4-4: Aktivitätsdiagramm zur Objektübergabe

Der Anwender startet nun den MATLAB-Client. Nach erfolgreicher Initialisierung verbindet sich dieser mit dem EKI-Server. Sobald die Verbindung steht, wird $FLAG[1] im KRL-Programm auf TRUE gesetzt. Zunächst ermittelt die Robotersteuerung die aktuelle Position des TCPs, generiert daraus ein XML-Datenpaket und sendet dieses an den EKI-Zwischenspeicher. Das XML-Datenpaket wird anhand der Einstellungen in der XML-Konfigurationsdatei erstellt. Der MATLAB-Client kann das XML-Datenpaket nun aus dem Zwischenspeicher entnehmen, alle beinhalteten Informationen herausfiltern und diese ausgegeben. In diesem Anwendungsszenario werden folgende XML-Datenpakete, zwischen dem EKI-Server und dem MATLAB-Client, ausgetauscht:

Tabelle 10: Ausgetauschte XML-Datenpakete während der Objektübergabe

MATLAB-Client → EKI-Server	EKI-Server → MATLAB-Client
<Sensor> <Read> <RechteHandFrame X="..." Y="..." Z="..." A="-6.5" B="86.8" C="-6.4" /> <Uebergabe>...</Uebergabe> </Read> </Sensor>	<Robot> <Data> <ActPos X="..." Y="..." Z="..." A="..." B="..." C="..." /> </Data> </Robot>

Als nächstes werden alle nötigen Gesteninformationen des Anwenders über die Kinect erfasst. In diesem Anwendungsszenario wären dies die Hand-States der linken und rechten Hand, sowie die Raumposition der rechten Hand. Anhand der Hand-States wird die weitere Vorgehensweise entschieden. Bildet die linke Hand ein Lasso-Zeichen, so soll der Roboter zum linken Fach fahren und das dortige Objekt entnehmen (*Uebergabe=1)*. Bildet die rechte Hand ein Lasso-Zeichen, soll der Roboter das Objekt aus dem rechten Fach entnehmen (*Uebergabe=2)*. Diese Informationen werden in ein definiertes XML-Datenpaket zusammengefasst und an den EKI-Zwischenspeicher gesendet.

Vor dem Absenden empfiehlt es sich jedoch, eine gewisse Wartezeit einzustellen, genau wie im ersten Anwendungsfall. Hier wird auch eine Wartezeit zwischen 0,5s und 1s empfohlen.

Sobald ein XML-Datenpaket im EKI-Zwischenspeicher bereitsteht, werden im KRL-Programm *FLAG[997]* und *FLAG[998]* auf *TRUE* gesetzt. Mit Hilfe eines integrierten XML-Parsers werden alle Informationen aus dem empfangene XML-Datenpaket herausgefiltert. Als nächstes wird anhand des Wertes der Variable *Uebergabe* entschieden, ob das Objekt aus dem linken oder rechten Fach entnommen und übergeben werden soll. Ist der Variable *Uebergabe* der Wert 1 zugewiesen, wird das Objekt aus dem linken Fach entnommen. Beinhaltet sie den Wert 2, dann wird das Objekt aus dem rechten Fach entnommen.

Nach erfolgreicher Objektentnahme, fährt der TCP zur Raumposition der rechten Hand des Anwenders. Dort wartet der TCP drei Sekunden lang und öffnet anschließend den Greifer. Der Anwender besitzt somit genügend Zeit, um das Objekt anzunehmen. Es wird nicht geprüft, ob der Anwender tatsächlich das Objekt erhalten hat. Der Manipulator lässt den Stab in jedem Fall los. Nach der Übergabe fährt der Manipulator wieder in seine Home-Position. Als nächstes ist ein manuelles Zurücksetzen der *FLAGS[997]* und *FLAG[998]* auf *FALSE* erforderlich. Geschieht dies nicht, kann das Programm nicht erkennen, ob sich ein neues XML-Datenpaket im EKI-Zwischenspeicher befindet.

Um die Anwendung weiter ausführen zu können, wird zunächst geprüft, ob die Verbindung zwischen EKI-Server und MATLAB-Client vorhanden ist. Ist die Verbindung abgebrochen, d.h. *FLAG[1]* steht auf *FALSE*, wird der EKI-Kanal bereinigt und gelöscht. Um die Anwendung fortzusetzen, wäre ein kompletter Neustart erforderlich. Steht *FLAG[1]* allerdings auf *TRUE*, kann die Anwendung fortgeführt werden.

5 Zusammenfassung und Fazit

Insgesamt konnte die Gestensteuerung des KUKA Industrieroboters erfolgreich umgesetzt werden. Der Datenaustausch zwischen der KRC4-Robotersteuerung und dem Arbeitsrechner konnte realisiert werden. Die Robotersteuerung konnte mit Hilfe des EthernetKRL Technologiepakets von KUKA als Server konfiguriert werden und XML-Datenpakete empfangen und senden.
Auf dem Arbeitsrechner wurde MATLAB als Client konfiguriert. Über MATLAB konnte die Kommunikation zur Robotersteuerung hergestellt werden. Die Kinect V2 wurde über eine C-MEX-Bibliothek in MATLAB integriert und bedient. Diese basiert auf der „Microsoft Kinect 2.0 SDK“. Gesteninformationen wurden über MATLAB abgefragt und über die Ethernet-Schnittstelle an die Robotersteuerung gesendet. Für die Datenübertragung wurde das TCP/IP-Protokoll genutzt.

Zum Abschluss wurden demonstrative Anwendungsszenarien entworfen und realisiert. Im ersten Anwendungsszenario wurde eine einfache PTP-Verfolgung der rechten Hand des Anwenders umgesetzt. Der Greifer des Manipulators lässt sich über definierte Gesten bedienen. Im zweiten Anwendungsszenario konnte dem Industrieroboter eine beliebige Bahn bzw. Kontur im Raum vorgegeben werden. Diese wurde dann über einen SPLINE-Bewegungssatz abgefahren bzw. nachgeahmt. Das dritte Anwendungsszenario steht ganz im Sinne der Mensch-Roboter-Kollaboration. Sie demonstriert die intelligente Objektübergabe zwischen dem Industrieroboter und dem Anwender. Dabei konnte der Anwender entscheiden, welche Objekte er übergeben bekommen möchte.
Alle genannten Anwendungsszenarien konnten erfolgreich umgesetzt werden. Diese veranschaulichen die Möglichkeiten der Gestensteuerung in MRK-Anwendungen.

Die körperliche Sicherheit des Anwenders spielt bei der MRK eine entscheidende Rolle. Deshalb lag die Motivation in dieser Arbeit auch darin, dies vollständig zu gewährleisten. Durch den Datenaustausch über die gesonderte EKI-Schnittstelle, konnten alle vorhandenen Sicherheitsmaßnahmen ungestört weiterarbeiten.

Diese Arbeit hat gezeigt, dass die Gestensteuerung eines Industrieroboters möglich ist. Die entwickelten Anwendungen können die Kollaboration zwischen Mensch und Roboter positiv ergänzen bzw. unterstützen.
Im Vergleich zu bereits veröffentlichen Anwendungen bzgl. Gestensteuerung eines KUKA-Industrieroboters (siehe Motivation, S. 1), ermöglicht die Lösung in dieser Arbeit den Datenaustausch zwischen der KRC4-Robotersteuerung und MATLAB. Andere Lösungen waren entweder nicht mit der KRC4-Robtersteuerung oder mit MATLAB kompatibel. Außerdem benötigt die Umsetzung über EthernetKRL kein Echtzeitsystem zur Gestenkontrolle benötigt. Andere Anwendungen basierten fast ausschließlich auf dem RSI-Technologiepaket. Diese benötigen entweder eine echtzeitfähige Netzwerkkarte oder einen echtzeitfähigen xPC. In dieser Arbeit waren diese nicht erforderlich.

Diese Arbeit soll als Basis für künftige Projekte dienen und einen Einblick in die Schnittstellen der KRC4-Robotersteuerung geben. Trotz allem besteht noch Erweiterungspotential. In Kapitel 6 sollen einige Punkte vorgestellt und diskutiert werden.

6 Ausblick

In diesem Abschnitt sollen Optimierungs- und Ergänzungsmöglichkeiten bzgl. dieser Arbeit vorgestellt werden. Diese werden aufgelistet und erläutert.

1. Realisierung in Simulink

   Während dieser Arbeit wurde bereits versucht, den Datenaustausch und die Steuerung der Kinect V2 in Simulink zu realisieren. Diese wurde aus zeitlichen Gründen verworfen. Die Umsetzung in Simulink hätte jedoch diverse Vorteile. Zum einen wäre es die Übersichtlichkeit des Blockschaltdiagramms. Außerdem könnten sich außenstehende Personen schneller in das Programm einarbeiten. Ein weiterer Punkt wäre die Wiederverwendbarkeit von bestimmten Blöcken. Diese könnten für weitere Anwendungen verwendet werden, ohne dass ganze Programmcodes neu geschrieben werden müssten.

2. Entwicklung weiterer Hand-State Zeichen

   Während der Entwurfs- und Realisierungsphase der drei Anwendungsszenarien konnte festgestellt werden, dass weitere Hand-States bzw. Handzeichen wünschenswert wären. Die Kinect V2 bietet zwar fünf Hand-States, diese sind jedoch nicht aussagekräftig genug. Zum einen sind die Hand-States *Unknown* und *NotTracked* verfügbar. Diese beiden sind jedoch eher passiv und bedeuten lediglich, dass Hand-States nicht erkannt wurden. Sie sind ungeeignet, um dem Roboter zu signalisieren, dass bestimmte Aktionen ausgeführt werden sollen. Des Weiteren sind die Hand-States *Open*, *Closed* und *Lasso* verfügbar. Dies sind aktive Hand-States und entsprechend gut geeignet, um bestimmte Aktionen auszulösen. Jedoch sind die beiden Hand-States *Open* und *Closed* nicht ergonomisch genug. Dadurch, dass diese beiden Hand-States auch eher unbewusst vom

Anwender ausgeführt werden könnten, könnte dies zu ungewollten Aktionen führen. Das *Lasso* Handzeichen wird im Gegensatz dazu aktiv und bewusst vom Anwender ausgeführt. Um Befehle zu signalisieren, ist dieses Handzeichen am besten geeignet. Daher wären weitere, ähnliche Handzeichen wünschenswert. Diese könnten z.B. mit dem Bilderverarbeitungstool *OpenCV* definiert und über die Kinect V2 erkannt werden.

3. Verbesserung der Bilderkennung

   Während dieser Arbeit wurde die Kinect V2 von Microsoft genutzt. Sie ist vergleichsweise günstig und fällt unter die Kategorie der Consumer-Elektronik. Die Bilderkennung ist nicht immer zuverlässig und teilweise zu ungenau. Außerdem ist diese stark von den gegebenen Lichtverhältnissen abhängig. Ein Verbesserungsansatz wäre z.B. der Einsatz einer Industriekamera. Es sollte untersucht werden, wie diese sich bzgl. der Bilderkennung verhält. Außerdem könnte getestet werden, ob die Nutzung von zusätzlichen Kameras dazu führt, dass Personen und Gesten zuverlässiger erkannt werden.

4. Sprachsteuerung

   Bei MRK-Anwendungen kann es sein, dass der Anwender in bestimmten Situationen nicht in der Lage ist, Handzeichen für die Befehlsübergabe zu nutzen. Diverse Situationen sind denkbar, wie z.B. bei Montagearbeiten, bei denen beide Hände zum Einsatz kommen. Sprachbefehle könnten die Anwendung optimieren und die Kollaboration effizienter gestalten.

5. Bedienoberfläche bzw. GUI

   Eine Bedienoberfläche bzw. ein sog. GUI, könnten die Bedienung der EKI-Schnittstelle vereinfachen. Dem Anwender wäre es möglich, die Schnittstelle zwischen der Robotersteuerung und dem Rechner zu

bedienen, ohne die genauen Hintergrundprozesse zu kennen. Die Konfiguration der ausgetauschten XML-Datenpakete ist sehr statisch. Kleinste Tippfehler in der Konfigurationsdatei können dazu führen, dass der Datenaustausch nicht ordnungsgemäß ausgeführt wird. Dies könnte mit einer entsprechenden Bedienoberfläche vermieden werden. Außerdem könnte damit die Übersicht bzgl. des Datenaustausches gesteigert werden. Letztendlich führt dies dazu, dass kompliziertere Anwendungen könnten einfacher realisiert werden könnten.

6. Echtzeitsteuerung und RSI

   In dieser Arbeit wurde das EthernetKRL-Softwarepaket von KUKA genutzt. Diverse Gestenerkennungsanwendungen (siehe Einführung) basieren jedoch auf dem RSI-Softwarepaket von KUKA. Mit dieser ist es möglich, Echtzeitanwendungen bzgl. der Robotersteuerung auszuführen. Dazu wird ein echtzeitfähiges System, aber vor allem eine echtzeitfähige Netzwerkkarte benötigt. Dieser Ansatz wurde während dieser Arbeit nicht weiter erforscht, wäre aber einer Untersuchung würdig.

7. Übergabe und Korrektur von komplexen Spline-Bewegungen

   Die Möglichkeit, eine komplexe Bahn bzw. Kontur im Raum vorzugeben, wurde in dieser Arbeit erfolgreich behandelt. Vorteilhaft wäre aber die nachgehende Korrektur der aufgenommenen Spline-Segmente und letztendlich die abzufahrende Bahn. Dem Anwender sollte die Möglichkeit gegeben werden, abweichende Spline-Segmente in ihrer Position zu korrigieren. Dies kann in MATLAB geschehen.

Diese Arbeit soll im November 2017 während der *Nacht des Wissens* in Hamburg präsentiert werden. Für diesen Zweck wurde eine transportfähige Plattform entworfen und aufgebaut. Auf diesem befinden sich die KRC4-Robotersteuerung und der Manipulator des Roboters.

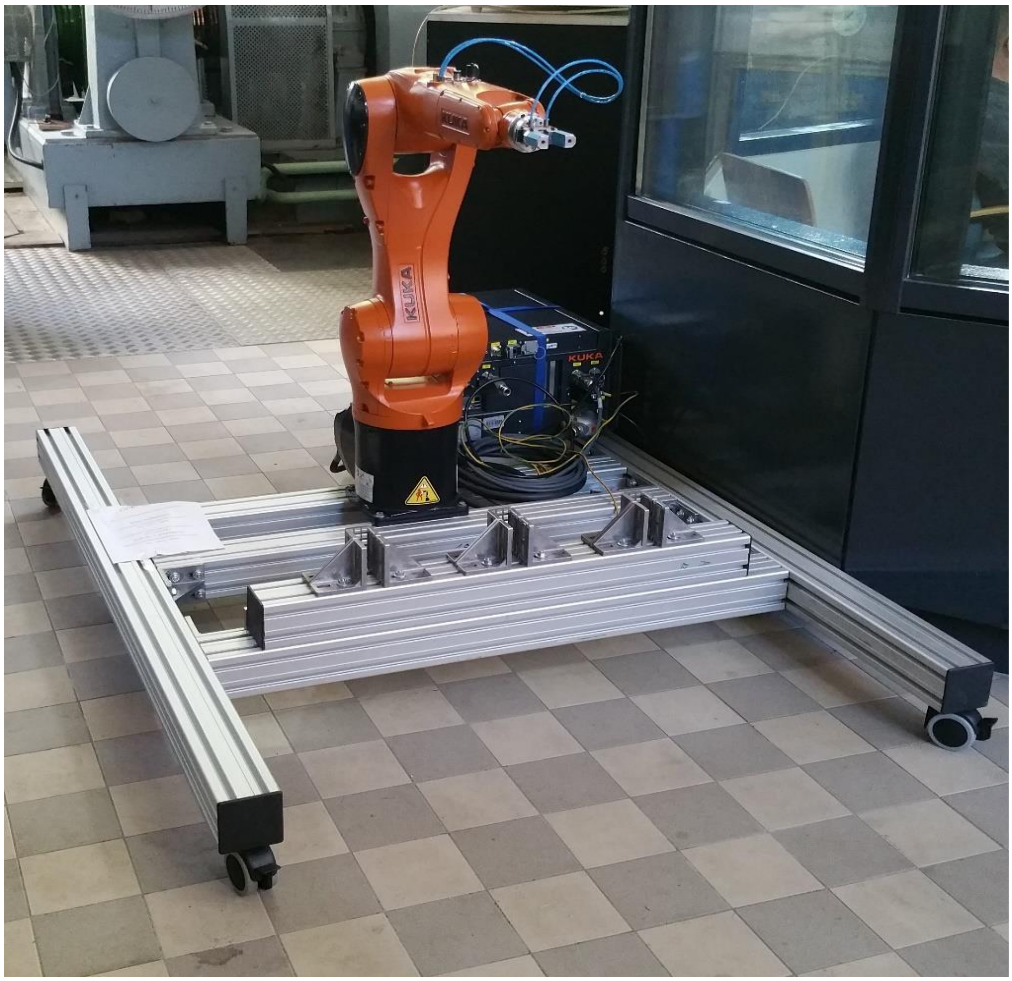


Abbildung 6-1: Mobiler Robotertisch

Literaturverzeichnis

CHV, HVBG: *BG-Information : Industrieroboter* (2008). URL https://www.arbeitssicherheit.de/media/pdfs/bgi_5123.pdf – Überprüfungsdatum 2017-06-14

JUAN R. TERVEN: *Kinect 2 Interface for Matlab.* URL https://de.mathworks.com/matlabcentral/fileexchange/53439-kinect-2-interface-for-matlab. – Aktualisierungsdatum: 2017 – Überprüfungsdatum 2017-06-29

KAHLEN, Christine: *Was ist Industrie 4.0?* URL http://www.plattform-i40.de/I40/Navigation/DE/Industrie40/WasIndustrie40/was-ist-industrie-40.html – Überprüfungsdatum 2017-06-14

KUKA ROBOTER GMBH (HRSG.): *KUKA.RobotSensorInterface 3.2.* Augsburg, 2013

KUKA ROBOTER GMBH (HRSG.): *KR C4 compact : Bedienungsanleitung.* Augsburg, 2014a

KUKA ROBOTER GMBH (HRSG.): *KUKA.EthernetKRL 2.2.* Augsburg, 2014b

KUKA ROBOTER GMBH (HRSG.): *System Software 8.3.* Augsburg, 2014c

KUKA ROBOTER GMBH (HRSG.): *KR Agilus sixx.* Mit W- und C-Variante. Augsburg, 2015

KUKA ROBOTER GMBH (HRSG.): *KR AGILUS sixx.* URL https://www.kuka.com/de-de/produkte-leistungen/robotersysteme/industrieroboter/kr-agilus-sixx – Überprüfungsdatum 2017-06-14

KUKA ROBOTER GMBH (HRSG.): *KUKA.SafeOperation.* URL https://www.kuka.com/de-de/produkte-leistungen/robotersysteme/software/querschnittstechnologien/kuka_safeoperation. – Aktualisierungsdatum: 2017b – Überprüfungsdatum 2017-06-14

MATTHEW SZYMCZYK: *How Does The Kinect 2 Compare To The Kinect 1?* URL http://zugara.com/how-does-the-kinect-2-compare-to-the-kinect-1 – Überprüfungsdatum 2017-06-25

MICROSOFT (HRSG.): *Kinect-Hardware.* URL https://developer.microsoft.com/de-de/windows/kinect/hardware – Überprüfungsdatum 2017-06-25

MICROSOFT (HRSG.): *Kinect for Windows SDK v1.0.* URL https://www.microsoft.com/en-us/download/details.aspx?id=28782 – Überprüfungsdatum 2017-06-20

MICROSOFT (HRSG.): *JointType Enumeration.* URL https://msdn.microsoft.com/en-us/library/microsoft.kinect.jointtype.aspx – Überprüfungsdatum 2017-06-26

MICROSOFT (HRSG.): *HandState Enumeration.* URL https://msdn.microsoft.com/en-us/library/windowspreview.kinect.handstate.aspx – Überprüfungsdatum 2017-06-26

MICROSOFT (HRSG.): *Kinect for Windows v2 Windows Runtime API Reference.* URL https://msdn.microsoft.com/en-us/library/dn758675.aspx. – Aktualisierungsdatum: 2017 – Überprüfungsdatum 2017-06-29

SCHULTZ, Julius ; FRISCHGESELL, Thomas ; BOHNERT, Carolina: *Trajektorienplanung für kooperative Industrieroboter.* Hamburg, 2016

SHAFAQ, Hasibullah ; FRISCHGESELL, Thomas: *Schnelle Aufnahme von Bewegungen mit Scannern.* Hamburg, 2017

SUGIURA, Tsukasa: *Kinect V2 Introduction and Tutorial.* URL https://www.slideshare.net/SugiuraTsukasa/kinect-v2-introduction-and-tutorial – Überprüfungsdatum 2017-06-20

THE MATHWORKS (HRSG.): *Image Acquisition Toolbox Support Package for Kinect For Windows Sensor.* URL https://de.mathworks.com/help/supportpkg/kinectforwindowsruntime/index.html. – Aktualisierungsdatum: 2016 – Überprüfungsdatum 2017-06-26

THE MATHWORKS (HRSG.): *TCP/IP Communication.* URL https://de.mathworks.com/help/matlab/tcpip-communication.html. – Aktualisierungsdatum: 2017 – Überprüfungsdatum 2017-07-02

Anhang

1 Body_Joint_Tracking.cpp

```
// Unter Projekteigenschaften -> C/C++ -> Allgemein:
// C:\Program Files\Microsoft SDKs\Kinect\v2.0_1409\inc
// Unter Projekteigenschaften -> Linker -> Allgemein:
// C:\Program Files\Microsoft SDKs\Kinect\v2.0_1409\Lib\x86

#include<Kinect.h>
#include<iostream>
void processBodies(const unsigned int &bodyCount, IBody **bodies);

// Standard Template zur sicheren Freigabe von belegtem Memoryspeicher
template<class Interface>
static inline void safeRelease(Interface *interfaceToRelease)
{
	if (interfaceToRelease != nullptr) {
		interfaceToRelease->Release();
		interfaceToRelease = nullptr;
	}
}

int main(int argc, char *argv[])
{
	IKinectSensor *sensor = nullptr;
	IBodyFrameReader *bodyFrameReader = nullptr;

	// Default Kinect Sensor finden
	HRESULT hr = GetDefaultKinectSensor(&sensor);

	// Wenn Kinect erfolgreich gefunden, führe folgenden Code aus
	if (SUCCEEDED(hr)) {
		hr = sensor->Open();

		if (SUCCEEDED(hr)) {
			// BodyFrameSource lesen
			IBodyFrameSource *bodyFrameSource = nullptr;
			hr = sensor->get_BodyFrameSource(&bodyFrameSource);

			// Wenn BodyFrameSource erfolgreich gelesen, BodyFrameReader lesen
			if (SUCCEEDED(hr)) {
				hr = bodyFrameSource->OpenReader(&bodyFrameReader);
			}

			// BodyFrameSource kann freigegeben werden
			safeRelease(bodyFrameSource);
		}
	}

	if (sensor == nullptr || FAILED(hr)) {
		std::cerr << "Kinect konnte nicht gefunden werden.\n";
		return E_FAIL;
	}

	while (bodyFrameReader != nullptr) {
		IBodyFrame *bodyFrame = nullptr;
		hr = bodyFrameReader->AcquireLatestFrame(&bodyFrame);

		if (SUCCEEDED(hr)) {
			IBody *bodies[BODY_COUNT] = { 0 };
			hr = bodyFrame->GetAndRefreshBodyData(_countof(bodies), bodies);
```

```
                    if (SUCCEEDED(hr)) {
                        processBodies(BODY_COUNT, bodies);
                    // Körperdaten werden berechnet und BodyFrame wird freigegeben


for (unsigned int bodyIndex = 0; bodyIndex < _countof(bodies); bodyIndex++) {
                                safeRelease(bodies[bodyIndex]);
                            }

                            safeRelease(bodyFrame);
                        }
                    }
                    else if (sensor) {
                        BOOLEAN isSensorAvailable = false;
                        hr = sensor->get_IsAvailable(&isSensorAvailable);

                        if (SUCCEEDED(hr) && isSensorAvailable == false) {
                            std::cerr << "Kinect nicht verfügbar.\n";
                        }
                    }
                    else {
                        std::cerr << "Body Frame konnte nicht gelesen werden.\n";
                    }
                }

                return 0;
}

void processBodies(const unsigned int &bodyCount, IBody **bodies)
{
                for (unsigned int bodyIndex = 0; bodyIndex < bodyCount; bodyIndex++) {
                    IBody *body = bodies[bodyIndex];

// Wenn Person nicht detektiert wird, wird die FOR-Schleife verlassen
                    BOOLEAN isTracked = false;
                    HRESULT hr = body->get_IsTracked(&isTracked);
                    if (FAILED(hr) || isTracked == false) {
                        continue;
                    }

// Wenn Personen detektiert worden sind, dann Körpergelenkkoordinaten auslesen
                    Joint joints[JointType_Count];
                    hr = body->GetJoints(_countof(joints), joints);
                    if (SUCCEEDED(hr)) {
// Ausgabe der Handkoordinaten (Rechts und Links)
const CameraSpacePoint &rightHandPos = joints[JointType_HandRight].Position;
const CameraSpacePoint &leftHandPos = joints[JointType_HandLeft].Position;

std::cout << "X: " << rightHandPos.X << " Y: " << rightHandPos.Y << " Z: " <<
rightHandPos.Z << "\n";

std::cout << "Y: " << leftHandPos.X << " Y: " << leftHandPos.Y << " Z: " <<
leftHandPos.Z << "\n";

// Ausgabe der Handzeichen (Rechte Hand)
                        HandState rightHandState;
                        hr = body->get_HandRightState(&rightHandState);

                        if (SUCCEEDED(hr)) {
                            if (rightHandState == HandState_Closed) {
                                std::cout << "CLOSED\r\n";
                            }
```

```
                        else if(rightHandState == HandState_Open) {
                            std::cout << "OPEN\r\n";
                        }
                        else if(rightHandState == HandState_Lasso) {
                            std::cout << "LASSO\r\n";
                        }
                        else if(rightHandState == HandState_NotTracked) {
                            std::cout << "NOT TRACKED\r\n";
                        }
                        else if(rightHandState == HandState_Unknown) {
                            std::cout << "UNKNOWN\r\n";
                        }
                    }
                }
            }
}
```

2 XmlCallBack

2.1 XmlCallBack: XML-Konfiguration

```
<ETHERNETKRL>
  <CONFIGURATION>
    <EXTERNAL>
      <IP>100.100.100.101</IP>                    // IP-Adresse des Arbeitsrechners
      <PORT>59152</PORT>                          // Netzwerkport des Arbeitsrechners
    </EXTERNAL>
  </CONFIGURATION>
  <RECEIVE>
    <XML>
      <ELEMENT Tag="Sensor/Message" Type="STRING"/>
      <ELEMENT Tag="Sensor/Positions/Current/@X" Type="REAL"/>
      <ELEMENT Tag="Sensor/Positions/Before/X" Type="REAL"/>
      <ELEMENT Tag="Sensor/Nmb" Type="INT"/>
      <ELEMENT Tag="Sensor/Status/IsActive" Type="BOOL"/>
      <ELEMENT Tag="Sensor/Read/xyzabc" Type="FRAME"/>
      <ELEMENT Tag="Sensor/Show/@error" Type="BOOL"/>
      <ELEMENT Tag="Sensor/Show/@temp" Type="INT"/>
      <ELEMENT Tag="Sensor/Show" Type="STRING"/>
      <ELEMENT Tag="Sensor/Free" Type="INT" Set_Out="998"/>
      <ELEMENT Tag="Sensor" Set_Flag="998"/>
    </XML>
  </RECEIVE>
  <SEND>
    <XML>
      <ELEMENT Tag="Robot/Data/LastPos/@X"/>
      <ELEMENT Tag="Robot/Data/LastPos/@Y"/>
      <ELEMENT Tag="Robot/Data/LastPos/@Z"/>
      <ELEMENT Tag="Robot/Data/LastPos/@A"/>
      <ELEMENT Tag="Robot/Data/LastPos/@B"/>
      <ELEMENT Tag="Robot/Data/LastPos/@C"/>
      <ELEMENT Tag="Robot/Data/ActPos/@X"/>
      <ELEMENT Tag="Robot/Data/ActPos/@Y"/>
      <ELEMENT Tag="Robot/Data/ActPos/@Z"/>
      <ELEMENT Tag="Robot/Status"/>
      <ELEMENT Tag="Robot/Mode"/>
      <ELEMENT Tag="Robot/RobotLamp/GrenLamp/LightOn"/>
    </XML>
  </SEND>
</ETHERNETKRL>
```

2.2 XmlCallBack: KRL-Programm

```
&ACCESS RVP
&REL 6
&PARAM SensorITMASK = *
&PARAM TEMPLATE = C:\KRC\Roboter\Template\vorgabe
&PARAM DISKPATH = KRC:\R1\Program
DEF XmlCallBack( )
;FOLD Declaration
INT i
DECL EKI_STATUS RET
$FLAG[1]=FALSE
;ENDFOLD
;FOLD Communicated data
;FOLD receive from external program
; <Sensor>
  ; <Message>Example message</Message>
  ; <Positions>
    ; <Current X="4645.2" />
    ; <Before>
      ; <X>0.9842</X>
    ; </Before>
  ; </Positions>
  ; <Nmb>8</Nmb>
  ; <Status>
    ; <IsActive>1</IsActive>
  ; </Status>
  ; <Read>
    ; <xyzabc X="210.3" Y="825.3" Z="234.3" A="84.2" B="12.3" C="43.5" />
  ; </Read>
  ; <Show error="0" temp="9929">Taginfo in attributes</Show>
  ; <Free>2912</Free>
; </Sensor>
;ENDFOLD
;FOLD send to external program
; <Robot>
  ; <Data>
    ; <ActPos X="1000.12">
    ; </ActPos>
    ; <LastPos A="..." B="..." C="..." X="..." Y="..." Z="...">
    ; </LastPos>
  ; </Data>
  ; <Mode>ConnectSensor</Mode>
  ; <RobotLamp>
    ; <GrenLamp>
      ; <LightOn>1</LightOn>
    ; </GrenLamp>
```

```
  ; </RobotLamp>
  ; <Status>12345678</Status>
; </Robot>
;ENDFOLD
;ENDFOLD
;FOLD INI
  ;FOLD BASISTECH INI
    BAS (#INITMOV,0 )
  ;ENDFOLD (BASISTECH INI)
  ;FOLD USER INI
    ;Make your modifications here

  ;ENDFOLD (USER INI)
;ENDFOLD (INI)
;FOLD Define callback
INTERRUPT DECL 89 WHEN $FLAG[998]==TRUE DO GET_DATA()
INTERRUPT ON 89
;ENDFOLD

RET=EKI_Init("XmlCallBack")
RET=EKI_Open("XmlCallBack")

;FOLD Write data to connection
; Write frame to <LastPos X="" Y="" Z="" A="" B="" C="" />
RET=EKI_SetFrame("XmlCallBack","Robot/Data/LastPos", TOOL_DATA[1])
; Write real to <ActPos X="" />
RET=EKI_SetReal("XmlCallBack","Robot/Data/ActPos/@X", $POS_ACT.X)
; Write int to <Status></Status>
RET=EKI_SetInt("XmlCallBack","Robot/Status", 12345678)
; Write string to <Mode></Mode>
RET=EKI_SetString("XmlCallBack","Robot/Mode","ConnectSensor")
; Write bool to <LightOn></LightOn>
RET=EKI_SetBool("XmlCallBack","Robot/RobotLamp/GrenLamp/LightOn",true)
;ENDFOLD
RET = EKI_Send("XmlCallBack","Robot")

;wait until data read
WAIT FOR $FLAG[1]

RET=EKI_Close("XmlCallBack")
RET=EKI_Clear("XmlCallBack")
END

DEF GET_DATA()
;FOLD Declaration
INT i
DECL EKI_STATUS RET
CHAR valueChar[256]
```

```
INT valueInt
REAL valueReal
BOOL valueBOOL
FRAME valueFrame
;ENDFOLD
;FOLD Initialize sample data
 FOR i=(1) TO (256)
  valueChar[i]=0
 ENDFOR
 valueInt=0
 valueReal=0.0
 valueFrame={X 0.0,Y 0.0,A 0.0,B 0.0,C 0.0}
 valueBOOL=FALSE
;ENDFOLD
;FOLD Get received sensor data
; Get string in <Message>Example message</Message>
RET=EKI_GetString("XmlCallBack","Sensor/Message",valueChar[])
; Get real value in <Current X="4645.2" />
RET=EKI_GetReal("XmlCallBack","Sensor/Positions/Current/@X",valueReal)
; Get int value in <Nmb>8</Nmb>
RET=EKI_GetInt("XmlCallBack","Sensor/Nmb",valueInt)
; Get bool value in textnode <IsActive>1</IsActive>
RET=EKI_GetBool("XmlCallBack","Sensor/Status/IsActive" ,valueBOOL)
; Get bool value in attribute <Show error="0" />
RET=EKI_GetBool("XmlCallBack","Sensor/Show/@error" ,valueBOOL)
; Get frame in <xyzabc X="210.3" Y="825.3" Z="234.3" A="84.2" B="12.3" C="43.5" />
RET=EKI_GetFrame("XmlCallBack","Sensor/Read/xyzabc",valueFrame)
;ENDFOLD
;FOLD Signal read
$FLAG[998]=FALSE
$FLAG[1]=TRUE

END
```

3 KUKA-Kinect: PTP-Verfolgung

3.1 EKI-Server KRL-Programm

```
&ACCESS RVP
&REL 12
&PARAM SensorITMASK = *
&PARAM TEMPLATE = C:\KRC\Roboter\Template\vorgabe
&PARAM DISKPATH = KRC:\R1\Program

DEF KukaKinectGreifer( )
;FOLD Deklaration der Variablen
DECL EKI_STATUS RET
FRAME valueFrame
DECL INT Greifer
;ENDFOLD

;FOLD INI
  ;FOLD BASISTECH INI
    BAS (#INITMOV,0 )
  ;ENDFOLD (BASISTECH INI)
;FOLD SPOTTECH INI
USERSPOT(#INIT)
;ENDFOLD (SPOTTECH INI)
;FOLD GRIPPERTECH INI
USER_GRP(0,DUMMY,DUMMY,GDEFAULT)
;ENDFOLD (GRIPPERTECH INI)

  ;FOLD Initialisierung der Variablen
    valueFrame={X 0.0,Y 0.0,A 0.0,B 0.0,C 0.0}
    Greifer=0;
  ;ENDFOLD (Initialisierung der Variablen)
;ENDFOLD (INI)

; Home-Position anfahren
;FOLD PTP HOME Vel=100 % DEFAULT;%{PE}%R
8.3.34,%MKUKATPBASIS,%CMOVE,%VPTP,%P 1:PTP, 2:HOME, 3:, 5:100, 7:DEFAULT
$BWDSTART=FALSE
PDAT_ACT=PDEFAULT
FDAT_ACT=FHOME
BAS(#PTP_PARAMS,100)
$H_POS=XHOME
PTP XHOME
```

```
;ENDFOLD

; EKI-Kanal "KukaKinectGreifer" initialisieren und öffnen
RET=EKI_Init("KukaKinectGreifer")
RET=EKI_Open("KukaKinectGreifer")

; Auf Verbindung zum MATLAB-Client warten
wait for $FLAG[1]==TRUE

REPEAT
; Systemvariablen auslesen und aktuelle Raumkoordinaten
; des TCP in EKI-Zwischenspeicher laden
  RET=EKI_SetReal("KukaKinectGreifer","Robot/Data/ActPos/@X",$POS_ACT.X)
  RET=EKI_SetReal("KukaKinectGreifer","Robot/Data/ActPos/@Y",$POS_ACT.Y)
  RET=EKI_SetReal("KukaKinectGreifer","Robot/Data/ActPos/@Z",$POS_ACT.Z)
  RET=EKI_SetReal("KukaKinectGreifer","Robot/Data/ActPos/@A",$POS_ACT.A)
  RET=EKI_SetReal("KukaKinectGreifer","Robot/Data/ActPos/@B",$POS_ACT.B)
  RET=EKI_SetReal("KukaKinectGreifer","Robot/Data/ActPos/@C",$POS_ACT.C)
; Inhalt des EKI-Speichers an den MATLAB-Client senden
  RET=EKI_Send("KukaKinectGreifer","Robot")

; Auf den Erhalt des XML-Elementes "Greifer" warten
; und in KRL-Variable "Greifer" laden
  wait for $FLAG[997]==TRUE
  RET=EKI_GetInt("KukaKinectGreifer","Sensor/Read/Greifer", Greifer)

  ; Wenn Greifer==1 dann öffne den Greifer
  IF Greifer==1 THEN
    ;FOLD OUT 1 '' State=FALSE CONT;%{PE}%R 8.3.34,%MKUKATPBASIS,%COUT,%VOUTX,%P
2:1, 3:, 5:FALSE, 6:CONTINUE
    CONTINUE
    $OUT[1]=TRUE
    ;ENDFOLD
    ;FOLD OUT 2 '' State=TRUE CONT;%{PE}%R 8.3.34,%MKUKATPBASIS,%COUT,%VOUTX,%P
2:2, 3:, 5:TRUE, 6:CONTINUE
    CONTINUE
    $OUT[2]=FALSE
    ;ENDFOLD
  ENDIF

  ; Wenn Greifer==0 dann schließe den Greifer
  IF Greifer==0 THEN
    ;FOLD OUT 1 '' State=TRUE CONT;%{PE}%R 8.3.34,%MKUKATPBASIS,%COUT,%VOUTX,%P
2:1, 3:, 5:FALSE, 6:CONTINUE
    CONTINUE
    $OUT[1]=FALSE
    ;ENDFOLD
    ;FOLD OUT 2 '' State=FALSE CONT;%{PE}%R 8.3.34,%MKUKATPBASIS,%COUT,%VOUTX,%P
```

```
2:2, 3:, 5:TRUE, 6:CONTINUE
    CONTINUE
    $OUT[2]=TRUE
    ;ENDFOLD
  ENDIF

  ; Auf den Erhalt des XML-ELementes "RechteHandFrame" warten
  ; und in KRL Variable "valueFrame" laden
  wait for $FLAG[998]==TRUE
  RET=EKI_GetFrame("KukaKinectGreifer","Sensor/Read/RechteHandFrame",valueFrame)

; Erhaltene Raumkoordinate über eine PTP-Bewegung anfahren
PTP valueFrame

  ; Flags wieder explizit auf FALSE setzen
  $FLAG[998]=FALSE
  $FLAG[997]=FALSE

  ; EKI-Zwischenspeicher entleeren und alle
  ; XML-Elemente von "Sensor" löschen
  RET=EKI_ClearBuffer("KukaKinectGreifer","Sensor")

UNTIL $FLAG[1]==FALSE

; wenn FLAG[1]==FALSE, dann ist die Verbindung zum Client unterbrochen
; EKI-Kanal "KukaKinectGreifer" schließen und löschen
RET=EKI_Close("KukaKinectGreifer")
RET=EKI_Clear("KukaKinectGreifer")
END
```

3.2 EKI-Server XML-Konfiguration

```
<ETHERNETKRL>
  <CONFIGURATION>
    <EXTERNAL>
      <TYPE>Client</TYPE>
    </EXTERNAL>
    <INTERNAL>
      <ALIVE Set_Flag="1"/>              // FLAG[1] zur zyklischen Verbindungsprüfung
      <IP>100.100.100.100</IP>           // IP-Adresse des MATLAB-Clients bzw. PC
      <PORT>54600</PORT>                 // Port des MATLAB-Client bzw. PC
      <PROTOCOL>TCP</PROTOCOL>           // Übertragungsprotokoll
    </INTERNAL>
  </CONFIGURATION>
  <RECEIVE>
    <XML>
      <ELEMENT Tag="Sensor/Read/RechteHandFrame" Type="FRAME" Set_Flag="998"/>
      <ELEMENT Tag="Sensor/Read/Greifer" Type="INT" Set_Flag="997"/>
    </XML>
  </RECEIVE>
  <SEND>
    <XML>
      <ELEMENT Tag="Robot/Data/ActPos/@X"/>
      <ELEMENT Tag="Robot/Data/ActPos/@Y"/>
      <ELEMENT Tag="Robot/Data/ActPos/@Z"/>
      <ELEMENT Tag="Robot/Data/ActPos/@A"/>
      <ELEMENT Tag="Robot/Data/ActPos/@B"/>
      <ELEMENT Tag="Robot/Data/ActPos/@C"/>
    </XML>
  </SEND>
</ETHERNETKRL
```

3.3 MATLAB-Client Programm

```
addpath('Mex');
clear all
close all

% Festgelegte XML-Struktur. Erfasste Raumkoordinaten der Kinect werden
% zwischen diese XML-Struktur hinterlegt und anschließend versendet
a = ('<Sensor><Read><RechteHandFrame X="');
b = ('" Y="');
c = ('" Z="');
d = ('" A="-6.6" B="86.8" C="-6.4" /><Greifer>');
e = ('</Greifer></Read></Sensor>');

% Initialiserung des Kinect-Objekts
% Verfügbare Bildquellen: 'color', 'depth', 'infrared', 'body_index',
% 'body', 'face' und 'HDface'
k2 = Kin2('color','depth','body');

% Knopfdruck erfassen. Bei der Eingabe eines 'q' wird das Programm beendet.
set(gcf,'keypress','k=get(gcf,''currentchar'');'); % listen keypress
k=[];
disp('Zum Beenden "q" drücken')

% Initialisierung eines TCP/IP-Clients. Diesem werden die IP-Adresse und
% der Port des EKI-Servers übergeben
t = tcpclient('100.100.100.100',54600);

% Initialisierung der Variable 'Greifer'
Greifer = 0;

% Endlosschleie läuft so lange, bis "q" gedrückt wird
while true
    % Es wird versucht, eventuell empfangene XML-Datenpakete zu lesen
    try
        %Daten des TCPIP_Clients werden ausgelesen und in 'r' gespeichert
        r = read(t);
    end

    % Wenn 'r' nicht leer ist
    if ~isempty(r)
        % Ausgabe des empfangen XML-Datenpaketes und Filterung der
        % beinhalteten Werte

        % 'r' wird von ASCII in Char konvertiert und in 'get_data'
        % gespeichert
        get_data=(char(r))

        % X-Koordinate wird herausgefiltert und gespeichert
        X_Value_Pos = (strfind(get_data, 'X'))+3;
        X_Value = str2double(get_data(X_Value_Pos));

        % Y-Koordinate wird herausgefiltert und gespeichert
        Y_Value_Pos = strfind(get_data, 'Y')+3;
        Y_Value = str2double(get_data(Y_Value_Pos));

        % Z-Koordinate wird herausgefiltert und gespeichert
        Z_Value_Pos = strfind(get_data, 'Z')+3;
        Z_Value = str2double(get_data(Z_Value_Pos));

        % Orientierung-A wird herausgefiltert und gespeichert
```

```
        A_Value_Pos = strfind(get_data, 'A')+3;
        A_Value = str2double(get_data(A_Value_Pos(2)));

        % Orientierung-B wird herausgefiltert und gespeichert
        B_Value_Pos = strfind(get_data, 'B')+3;
        B_Value = str2double(get_data(B_Value_Pos));

        % Orientierung-C wird herausgefiltert und gespeichert
        C_Value_Pos = strfind(get_data, 'C')+3;
        C_Value = str2double(get_data(C_Value_Pos));
    end

    % Kinect Frames abrufen und zwischenspeichern
    validData = k2.updateData;

    if validData
        % Folgende Beschreibung stammt aus dem Add-In von Juan R. Terven:
        % Get 3D bodies joints
        % Input parameter can be 'Quat' or 'Euler' for the joints
        % orientations.
        % getBodies returns a structure array.
        % The structure array (bodies) contains 6 bodies at most
        % Each body has:
        % -Position: 3x25 matrix containing the x,y,z of the 25 joints in
        %   camera space coordinates
        % - Orientation:
        %   If input parameter is 'Quat': 4x25 matrix containing the
        %   orientation of each joint in [x; y; z, w]
        %   If input parameter is 'Euler': 3x25 matrix containing the
        %   orientation of each joint in [Pitch; Yaw; Roll]
        % -TrackingState: state of each joint. These can be:
        %   NotTracked=0, Inferred=1, or Tracked=2
        % -LeftHandState: state of the left hand
        % -RightHandState: state of the right hand
        [bodies, fcp, timeStamp] = k2.getBodies('Quat');

        % Anzahl der detektierten Körper
        numBodies = size(bodies,2);

        % Wenn mehr als 0 Personen detektiert werden
        if numBodies > 0
            % Wenn die linke Hand von Person 1 OFFEN ist
            if ((bodies(1).LeftHandState) == 2)
                disp('Right Hand Position')
                disp(bodies(1).Position(:,k2.JointType_HandRight));
                disp('Left Hand State: OPEN');

                % Erfasse die Raumposition der rechten Hand von Person 1
         RightHandPosition = bodies(1).Position(:,k2.JointType_HandRight);

                % Empfohlener Arbeitsraum
                % (Gilt nur im Mechanik Labor der HAW-Hamburg):
                % X = 450 - 920; Xstandart = 750
                % Y = -500 bis 500
                % Z = 300 - 870  Zstandart = 500

                % Konstante X-Koordinate des TCPs
                x_dbl = 800.0;
                x_str = num2str(x_dbl);

                % Regulierung der Y-Koordinate an den Arbeitsraum
                y_dbl = 0+800*RightHandPosition(1);
                if y_dbl <= (-500.0)
```

```
                y_dbl = -500.0;
            end
            if y_dbl >= 500.0
                y_dbl = 500.0;
            end
            y_str = num2str(y_dbl);

            % Regulierung der Z-Koordinate an den Arbeitsraum
            z_dbl = 500+1000*RightHandPosition(2);
            if z_dbl <= 350.0
                z_dbl = 350.0;
            end
            if z_dbl >= 800.0
                z_dbl = 800.0;
            end
            z_str= num2str(z_dbl);

            % Wenn die rechte Hand von Person 1 GEÖFFNET ist, soll der
            % Greifer ebenfalls ÖFFNEN
            if (bodies(1).RightHandState == 2)
                Greifer_num = 1;
                Greifer = num2str(Greifer_num);
            % Ist die rechte Hand von Person 1 GESCHLOSSEN, soll der
            % Greifer ebenfalls SCHLIESSEN
            elseif (bodies(1).RightHandState == 3)
                Greifer_num = 0;
                Greifer = num2str(Greifer_num);
            end

            % Erfasste Raumkoordinaten in die feste XML-Struktur
            % hinterlegen und nach uint8 konvertieren
            senddata = uint8([a x_str b y_str c z_str d Greifer e]);
            % XML-Datenpaket über TCP/IP an den EKI-Server senden
            write(t, senddata);
            char(senddata) % Ausgabe des gesendeten XML-Datenpakets
        end

        % Wenn Status der linken Hand unbekannt ist:
        if ((bodies(1).LeftHandState) == 0)
            disp('Left Hand State: UNKNOWN');
        end

        % Wenn linke Hand nicht erfasst werden konnte:
        if ((bodies(1).LeftHandState) == 1)
            disp('Left Hand State: NOT TRACKED');
        end

        % Wenn linke Hand geschlossen ist:
        if ((bodies(1).LeftHandState) == 3)
            disp('Left Hand State: CLOSED');
        end

        % Wenn die linke Hand ein LASSO-Zeichen bildet:
        if ((bodies(1).LeftHandState) == 4)
            disp('Left Hand State: LASSO');
        end
    end
        % Wenn 'q' gedrückt wurde, wird die Schleife verlassen
    if ~isempty(k)
        if strcmp(k,'q'); break; end
    end
end
pause(0.6)  % Wartezeit
```

```
end

% Kinect-Objekt löschen
k2.delete;

close all;
```

4 KUKA-Kinect: SPLINE-Bewegung

4.1 EKI-Server KRL-Programm

```
&ACCESS RVP
&REL 13
&PARAM SensorITMASK = *
&PARAM TEMPLATE = C:\KRC\Roboter\Template\vorgabe
&PARAM DISKPATH = KRC:\R1\Program

DEF KukaKinectSpline( )
;FOLD Declaration
DECL EKI_STATUS RET
FRAME valueFrame
INT laufvar

;Spline-Block Variablen
BOOL spline_go
DECL FRAME zwischenspeicher
DECL FRAME spline_punkt[10] ;Spline-Segmente

;ENDFOLD
;FOLD INI
  ;FOLD BASISTECH INI
    BAS (#INITMOV,0 )
  ;ENDFOLD (BASISTECH INI)
;FOLD SPOTTECH INI
USERSPOT(#INIT)
;ENDFOLD (SPOTTECH INI)
;FOLD GRIPPERTECH INI
USER_GRP(0,DUMMY,DUMMY,GDEFAULT)
;ENDFOLD (GRIPPERTECH INI)
  ;FOLD USER INI

    valueFrame={X 0.0,Y 0.0,A 0.0,B 0.0,C 0.0}
    zwischenspeicher={X 0.0,Y 0.0,A 0.0,B 0.0,C 0.0}

    ;Spline-Array zum Zwischenspeichern initialisieren
    spline_go=FALSE
    spline_punkt[1]={X 0.0,Y 0.0,A 0.0,B 0.0,C 0.0}
    spline_punkt[2]={X 0.0,Y 0.0,A 0.0,B 0.0,C 0.0}
    spline_punkt[3]={X 0.0,Y 0.0,A 0.0,B 0.0,C 0.0}
    spline_punkt[4]={X 0.0,Y 0.0,A 0.0,B 0.0,C 0.0}
    spline_punkt[5]={X 0.0,Y 0.0,A 0.0,B 0.0,C 0.0}
    spline_punkt[6]={X 0.0,Y 0.0,A 0.0,B 0.0,C 0.0}
    spline_punkt[7]={X 0.0,Y 0.0,A 0.0,B 0.0,C 0.0}
    spline_punkt[8]={X 0.0,Y 0.0,A 0.0,B 0.0,C 0.0}
    spline_punkt[9]={X 0.0,Y 0.0,A 0.0,B 0.0,C 0.0}
```

```
    spline_punkt[10]={X 0.0,Y 0.0,A 0.0,B 0.0,C 0.0}

  ;ENDFOLD (USER INI)
;ENDFOLD (INI)

; Home-Position anfahren
;FOLD PTP HOME Vel=100 % DEFAULT;%{PE}%R 8.3.34,%MKUKATPBASIS,%CMOVE,%VPTP,%P 1:PTP,
2:HOME, 3:, 5:100, 7:DEFAULT
$BWDSTART=FALSE
PDAT_ACT=PDEFAULT
FDAT_ACT=FHOME
BAS(#PTP_PARAMS,100)
$H_POS=XHOME
PTP XHOME
;ENDFOLD

; EKI-Kanal "KukaKinectSpline" initialisieren und öffnen
RET=EKI_Init("KukaKinectSpline")
RET=EKI_Open("KukaKinectSpline")

; Auf Verbindung zum MATLAB-Client warten
wait for $FLAG[1]==TRUE

; Endlosschleife solange Verbindung besteht (FLAG[1]==TRUE)
REPEAT
  ; Systemvariablen auslesen und aktuelle Raumkoordinaten
  ; des TCP in EKI-Zwischenspeicher laden
  RET=EKI_SetReal("KukaKinectSpline","Robot/Data/ActPos/@X",$POS_ACT.X)
  RET=EKI_SetReal("KukaKinectSpline","Robot/Data/ActPos/@Y",$POS_ACT.Y)
  RET=EKI_SetReal("KukaKinectSpline","Robot/Data/ActPos/@Z",$POS_ACT.Z)
  RET=EKI_SetReal("KukaKinectSpline","Robot/Data/ActPos/@A",$POS_ACT.A)
  RET=EKI_SetReal("KukaKinectSpline","Robot/Data/ActPos/@B",$POS_ACT.B)
  RET=EKI_SetReal("KukaKinectSpline","Robot/Data/ActPos/@C",$POS_ACT.C)
  ; Inhalt des EKI-Speichers an den MATLAB-Client senden
  RET=EKI_Send("KukaKinectSpline","Robot")

; Auf den Erhalt des XML-Elementes "RechteHandFrame" warten
; und in KRL-Variable "valueFrame" laden
  wait for $FLAG[998]==TRUE
  RET=EKI_GetFrame("KukaKinectSpline","Sensor/Read/RechteHandFrame",valueFrame)

  ; Position der rechten Hand anfahren
  PTP valueFrame

  ; Flags explizit auf FALSE setzen
  $FLAG[998]=FALSE
  $FLAG[997]=FALSE

  ; EKI-Zwischenspeicher löschen
  RET=EKI_ClearBuffer("KukaKinectSpline","Sensor")

  ; Wenn der Startbefehl für die Aufnahme von
  ; Spline-Segmenten gegeben wurde, dann Schleife ausführen
```

```
  REPEAT
  ; Auf den Erhalt der XML-Elemente warten!
  wait for $FLAG[998]==TRUE
  wait for $FLAG[997]==TRUE

  ; Auf den Erhalt des XML-Elementes "spline" warten
  ; und in KRL-Variable "spline_go" laden
  RET=EKI_GetBool("KukaKinectSpline","Sensor/Status/spline",spline_go)
  $FLAG[997]=FALSE

  IF spline_go==FALSE THEN
    RET=EKI_ClearBuffer("KukaKinectSpline","Sensor")
  ENDIF

  UNTIL spline_go==TRUE

  ; Spline Punkte aufnehmen und abspeichern!

  IF spline_go==TRUE THEN
    FOR laufvar=1 TO 10
      wait for $FLAG[998]==TRUE
      RET=EKI_GetFrame("KukaKinectSpline","Sensor/Read/RechteHandFrame",zwischenspeicher)
      spline_punkt[laufvar]=zwischenspeicher
      $FLAG[998]=FALSE
      RET=EKI_ClearBuffer("KukaKinectSpline","Sensor")
    ENDFOR

    ; Spline fahren!
    SPLINE
      SPL spline_punkt[1]
      SPL spline_punkt[2]
      SPL spline_punkt[3]
      SPL spline_punkt[4]
      SPL spline_punkt[5]
      SPL spline_punkt[6]
      SPL spline_punkt[7]
      SPL spline_punkt[8]
      SPL spline_punkt[9]
      SPL spline_punkt[10]
    ENDSPLINE
  ENDIF

UNTIL $FLAG[1]==FALSE
; Schleife ausführen solange Verbindung zum MATLAB-Client vorhanden,
; bzw. FLAG[1] == TRUE ist. Bei Verbindungsabbrung: FLAG[1]==FALSE

; wenn FLAG[1]==FALSE, dann ist die Verbindung zum Client unterbrochen
; EKI-Kanal "KukaKinectGreifer" schließen und löschen
RET=EKI_Close("KukaKinectSpline")
RET=EKI_Clear("KukaKinectSpline")

END
```

4.2 EKI-Server XML-Konfiguration

```
<ETHERNETKRL>
  <CONFIGURATION>
    <EXTERNAL>
      <TYPE>Client</TYPE>
    </EXTERNAL>
    <INTERNAL>
      <ALIVE Set_Flag="1"/>              // FLAG[1] zur zyklischen Verbindungsprüfung
      <IP>100.100.100.100</IP>           // IP-Adresse des MATLAB-Clients bzw. PC
      <PORT>54600</PORT>                 // Port des MATLAB-Client bzw. PC
      <PROTOCOL>TCP</PROTOCOL>           // Übertragungsprotokoll
    </INTERNAL>
  </CONFIGURATION>
  <RECEIVE>
    <XML>
      <ELEMENT Tag="Sensor/Read/RechteHandFrame" Type="FRAME" Set_Flag="998"/>
      <ELEMENT Tag="Sensor/Status/spline" Type="BOOL" Set_Flag="997"/>
    </XML>
  </RECEIVE>
  <SEND>
    <XML>
      <ELEMENT Tag="Robot/Data/ActPos/@X"/>
      <ELEMENT Tag="Robot/Data/ActPos/@Y"/>
      <ELEMENT Tag="Robot/Data/ActPos/@Z"/>
      <ELEMENT Tag="Robot/Data/ActPos/@A"/>
      <ELEMENT Tag="Robot/Data/ActPos/@B"/>
      <ELEMENT Tag="Robot/Data/ActPos/@C"/>
    </XML>
  </SEND>
</ETHERNETKRL>
```

4.3 MATLAB-Client Programm

```
addpath('Mex');
clear all
close all

% Festgelegte XML-Struktur. Erfasste Raumkoordinaten der Kinect werden
% zwischen diese XML-Struktur hinterlegt und anschließend versendet
a = ('<Sensor><Read><RechteHandFrame X="');
b = ('" Y="');
c = ('" Z="');
d = ('" A="-6.6" B="86.8" C="-6.4" /></Read><Status><spline>');
e = ('</spline></Status></Sensor>');

% Initialiserung des Kinect-Objekts
% Verfügbare Bildquellen: 'color', 'depth', 'infrared', 'body_index',
% 'body', 'face' und 'HDface'
k2 = Kin2('color','depth','body');

% Knopfdruck erfassen. Bei der Eingabe eines 'q' wird das Programm beendet.
set(gcf,'keypress','k=get(gcf,''currentchar'');'); % listen keypress
k=[];
disp('Zum Beenden "q" drücken')

% Initialisierung eines TCP/IP-Clients. Diesem werden die IP-Adresse und
% der Port des EKI-Servers übergeben
t = tcpclient('100.100.100.100',54600);

% Endlosschleie läuft so lange, bis "q" gedrückt wird
while true
    % Es wird versucht, eventuell empfangene XML-Datenpakete zu lesen
    try
        %Daten des TCPIP_Clients werden ausgelesen und in 'r' gespeichert
        r = read(t);
    end

    % Wenn 'r' nicht leer ist
    if ~isempty(r)
        % Ausgabe des empfangen XML-Datenpaketes und Filterung der
        % beinhalteten Werte

        % 'r' wird von ASCII in Char konvertiert und in 'get_data'
        % gespeichert
        get_data=(char(r))

        % X-Koordinate wird herausgefiltert und gespeichert
        X_Value_Pos = (strfind(get_data, 'X'))+3;
        X_Value = str2double(get_data(X_Value_Pos));

        % Y-Koordinate wird herausgefiltert und gespeichert
        Y_Value_Pos = strfind(get_data, 'Y')+3;
        Y_Value = str2double(get_data(Y_Value_Pos));

        % Z-Koordinate wird herausgefiltert und gespeichert
        Z_Value_Pos = strfind(get_data, 'Z')+3;
        Z_Value = str2double(get_data(Z_Value_Pos));

        % Orientierung-A wird herausgefiltert und gespeichert
```

```
        A_Value_Pos = strfind(get_data, 'A')+3;
        A_Value = str2double(get_data(A_Value_Pos(2)));

        % Orientierung-B wird herausgefiltert und gespeichert
        B_Value_Pos = strfind(get_data, 'B')+3;
        B_Value = str2double(get_data(B_Value_Pos));

        % Orientierung-C wird herausgefiltert und gespeichert
        C_Value_Pos = strfind(get_data, 'C')+3;
        C_Value = str2double(get_data(C_Value_Pos));
    end

    % Kinect Frames abrufen und zwischenspeichern
    validData = k2.updateData;

    if validData
        % Folgende Beschreibung stammt aus dem Add-In von Juan R. Terven:
        % Get 3D bodies joints
        % Input parameter can be 'Quat' or 'Euler' for the joints
        % orientations.
        % getBodies returns a structure array.
        % The structure array (bodies) contains 6 bodies at most
        % Each body has:
        % -Position: 3x25 matrix containing the x,y,z of the 25 joints in
        %   camera space coordinates
        % - Orientation:
        %   If input parameter is 'Quat': 4x25 matrix containing the
        %   orientation of each joint in [x; y; z, w]
        %   If input parameter is 'Euler': 3x25 matrix containing the
        %   orientation of each joint in [Pitch; Yaw; Roll]
        % -TrackingState: state of each joint. These can be:
        %   NotTracked=0, Inferred=1, or Tracked=2
        % -LeftHandState: state of the left hand
        % -RightHandState: state of the right hand
        [bodies, fcp, timeStamp] = k2.getBodies('Quat');

        % Anzahl der detektierten Körper
        numBodies = size(bodies,2);

        % Wenn mehr als 0 Personen detektiert werden
        if numBodies > 0
            % Wenn die linke Hand von Person 1 OFFEN ist
            if ((bodies(1).LeftHandState) == 2)
                disp('Left Hand State: OPEN');
                disp('Right Hand Position.');
                disp(bodies(1).Position(:,k2.JointType_HandRight));

                % Erfasse die Raumposition der rechten Hand von Person 1
          RightHandPosition = bodies(1).Position(:,k2.JointType_HandRight);

                % Empfohlener Arbeitsraum (Gilt nur im Mechanik Labor der
HAW-Hamburg):
                % X = 450 - 920; Xstandart = 750
                % Y = -500 bis 500
                % Z = 300 - 870  Zstandart = 500

                % Konstante X-Koordinate des TCPs
                x_dbl = 800.0;
                x_str = num2str(x_dbl);

                % Regulierung der Y-Koordinate an den Arbeitsraum
```

```
        y_dbl = 0+800*RightHandPosition(1);
        if y_dbl <= (-500.0)
            y_dbl = -500.0;
        end
        if y_dbl >= 500.0
            y_dbl = 500.0;
        end
        y_str = num2str(y_dbl);

        % Regulierung der Z-Koordinate an den Arbeitsraum
        z_dbl = 500+1000*RightHandPosition(2);
        if z_dbl <= 300.0
            z_dbl = 300.0;
        end
        if z_dbl >= 800.0
            z_dbl = 800.0;
        end
        z_str= num2str(z_dbl);

        % Wenn rechte Hand geöffnet ist, soll kein Spline-Segment
        % aufgenommen werden -> spline_bool = spline = FALSE / 0
        spline_nbr = 0;             % FALSE
        spline_bool= num2str(spline_nbr);

        % Erfasste Raumkoordinaten in die feste XML-Struktur
        % hinterlegen und nach uint8 konvertieren
      senddata = uint8([a x_str b y_str c z_str d spline_bool e]);
        % XML-Datenpaket über TCP/IP an den EKI-Server senden
        write(t, senddata);
        char(senddata) % Ausgabe des gesendeten XML-Datenpakets
    end

    % Wenn Status der linken Hand unbekannt ist:
    if ((bodies(1).LeftHandState) == 0)
        disp('Left Hand State: UNKNOWN');
    end

    % Wenn linke Hand nicht erfasst werden konnte:
    if ((bodies(1).LeftHandState) == 1)
        disp('Left Hand State: NOT TRACKED');
    end

    % Wenn linke Hand geschlossen ist:
    if ((bodies(1).LeftHandState) == 3)
        disp('Left Hand State: CLOSED');
    end

    % Wenn die linke Hand ein LASSO-Zeichen bildet, wird mit der
    % Aufnahme von Spline-Segmenten begonnen!
    if ((bodies(1).LeftHandState) == 4)
        disp('Left Hand State: LASSO -> SPLINE GO!');

        % Erfasse die Raumposition der rechten Hand von Person 1
  RightHandPosition = bodies(1).Position(:,k2.JointType_HandRight);

        % Spline-Segment soll übergeben werden!
        spline_nbr = 1;                 % TRUE
        spline_bool=num2str(spline_nbr);

        % Konstante X-Koordinate des TCPs
        x_dbl = 850.0;
```

```
                x_str = num2str(x_dbl);

                % Regulierung der Y-Koordinate an den Arbeitsraum
                y_dbl = 0+400*RightHandPosition(1);
                if y_dbl <= (-500.0)
                    y_dbl = -500.0;
                end
                if y_dbl >= 500.0
                    y_dbl = 500.0;
                end
                y_str = num2str(y_dbl);

                % Regulierung der Z-Koordinate an den Arbeitsraum
                z_dbl = 500+500*RightHandPosition(2);
                if z_dbl <= 300.0
                    z_dbl = 300.0;
                end
                if z_dbl >= 800.0
                    z_dbl = 800.0;
                end
                z_str= num2str(z_dbl);

                % Erfasste Raumkoordinaten in die feste XML-Struktur
                % hinterlegen und nach uint8 konvertieren
               senddata = uint8([a x_str b y_str c z_str d spline_bool e]);
                % Ausgeschriebenes XML Format:
                % <Sensor>
                %   <Read>
                %      <RechteHandFrame X="850.0" Y="Y.Y" Z="Z.Z"
                %                       A="-6.6" B="86.8" C="-6.4" />
                %       <spline>spline_bool</spline>
                %   </Read>
                %</Sensor>

                % XML-Datenpaket über TCP/IP an den EKI-Server senden
                write(t, senddata);
                char(senddata) % Ausgabe des gesendeten XML-Datenpakets
            end
        end
            % Wenn 'q' gedrückt wurde, wird die Schleife verlassen
        if ~isempty(k)
            if strcmp(k,'q'); break; end
        end
    end
    pause(0.5)  % Wartezeit
end

% Kinect-Objekt löschen
k2.delete;

close all;
```

5 KUKA-Kinect: Objektübergabe

5.1 EKI-Server KRL-Programm

```
&ACCESS RVP
&REL 23
&PARAM SensorITMASK = *
&PARAM EDITMASK = *
&PARAM TEMPLATE = C:\KRC\Roboter\Template\vorgabe
DEF KukaKinectUebergabe( )

;FOLD DECLARATION
DECL EKI_STATUS RET
DECL INT Uebergabe
FRAME RechteHandPos
; ENDFOLD DECLARATION

;FOLD INI;%{PE}
  ;FOLD BASISTECH INI
    GLOBAL INTERRUPT DECL 3 WHEN $STOPMESS==TRUE DO IR_STOPM ( )
    INTERRUPT ON 3
    BAS (#INITMOV,0 )
  ;ENDFOLD (BASISTECH INI)
;FOLD SPOTTECH INI
USERSPOT(#INIT)
;ENDFOLD (SPOTTECH INI)
;FOLD GRIPPERTECH INI
USER_GRP(0,DUMMY,DUMMY,GDEFAULT)
;ENDFOLD (GRIPPERTECH INI)

  ;FOLD USER INI
     Uebergabe = 0
     RechteHandPos = {X 0.0,Y 0.0,A 0.0,B 0.0,C 0.0}
  ;ENDFOLD (USER INI)
;ENDFOLD (INI)

; Home-Position anfahren
;FOLD PTP HOME  Vel= 100 % DEFAULT;%{PE}%MKUKATPBASIS,%CMOVE,%VPTP,%P 1:PTP, 2:HOME,
3:, 5:100, 7:DEFAULT
$BWDSTART = FALSE
PDAT_ACT=PDEFAULT
FDAT_ACT=FHOME
BAS (#PTP_PARAMS,100 )
$H_POS=XHOME
PTP  XHOME
;ENDFOLD

; Greifer öffnen
;FOLD WAIT Time=1 sec;%{PE}%R 8.3.34,%MKUKATPBASIS,%CWAIT,%VWAIT,%P 3:1
```

```
WAIT SEC 1
;ENDFOLD
;FOLD OUT 1 '' State=TRUE CONT;%{PE}%R 8.3.34,%MKUKATPBASIS,%COUT,%VOUTX,%P 2:1, 3:,
5:TRUE, 6:CONTINUE
CONTINUE
$OUT[1]=TRUE
;ENDFOLD
;FOLD OUT 2 '' State=FALSE CONT;%{PE}%R 8.3.34,%MKUKATPBASIS,%COUT,%VOUTX,%P 2:2, 3:,
5:FALSE, 6:CONTINUE
CONTINUE
$OUT[2]=FALSE
;ENDFOLD
;FOLD WAIT Time=1 sec;%{PE}%R 8.3.34,%MKUKATPBASIS,%CWAIT,%VWAIT,%P 3:1
WAIT SEC 1
;ENDFOLD

; EKI-Kanal "KukaKinectUebergabe" initialisieren und öffnen
RET=EKI_Init("KukaKinectUebergabe")
RET=EKI_Open("KukaKinectUebergabe")

; Auf Verbindung zum MATLAB-Client warten
wait for $FLAG[1]==TRUE

; Endlosschleife solange Verbindung besteht (FLAG[1]==TRUE)
REPEAT
; Systemvariablen auslesen und aktuelle Raumkoordinaten
; des TCP in EKI-Zwischenspeicher laden
  RET=EKI_SetReal("KukaKinectUebergabe","Robot/Data/ActPos/@X",$POS_ACT.X)
  RET=EKI_SetReal("KukaKinectUebergabe","Robot/Data/ActPos/@Y",$POS_ACT.Y)
  RET=EKI_SetReal("KukaKinectUebergabe","Robot/Data/ActPos/@Z",$POS_ACT.Z)
  RET=EKI_SetReal("KukaKinectUebergabe","Robot/Data/ActPos/@A",$POS_ACT.A)
  RET=EKI_SetReal("KukaKinectUebergabe","Robot/Data/ActPos/@B",$POS_ACT.B)
  RET=EKI_SetReal("KukaKinectUebergabe","Robot/Data/ActPos/@C",$POS_ACT.C)
; Inhalt des EKI-Speichers an den MATLAB-Client senden
  RET=EKI_Send("KukaKinectUebergabe","Robot")

; Auf den Erhalt des XML-Elementes "RechteHandFrame" warten
; und in KRL-Variable "RechteHandFrame" laden
WAIT FOR $FLAG[998]==TRUE
RET=EKI_GetFrame("KukaKinectUebergabe","Sensor/Read/RechteHandFrame", RechteHandPos)
WAIT FOR $FLAG[997]==TRUE
RET=EKI_GetInt("KukaKinectUebergabe","Sensor/Read/Uebergabe", Uebergabe)

; Stab aus dem linken Fach entnehmen
IF Uebergabe==1 THEN

;FOLD SPTP ueberLinks Vel=100 % PDAT1 Tool[1]:tool1 Base[0];%{PE}%R
8.3.34,%MKUKATPBASIS,%CSPLINE,%VSPTP_SB,%P 1:SPTP_SB, 2:ueberLinks, 3:, 5:100, 7:PDAT1
SPTP XueberLinks WITH
$VEL_AXIS[1]= SVEL_JOINT( 100), $TOOL= STOOL2( FueberLinks), $BASE= SBASE( FueberLinks.BASE_
NO),$IPO_MODE= SIPO_MODE( FueberLinks.IPO_FRAME), $LOAD= SLOAD( FueberLinks.TOOL_NO),
$ACC_AXIS[1]= SACC_JOINT( PPDAT1), $GEAR_JERK[1]= SGEAR_JERK( PPDAT1)
;ENDFOLD
```

```
;FOLD SLIN Links Vel=2 m/s CPDAT1 ADAT0 Tool[1]:tool1 Base[0];%{PE}%R
8.3.34,%MKUKATPBASIS,%CSPLINE,%VSLIN_SB,%P 1:SLIN_SB, 2:Links, 3:, 5:2, 7:CPDAT1, 8:ADAT0
SLIN XLinks WITH
$VEL=SVEL_CP( 2, , LCPDAT1), $TOOL=STOOL2( FLinks), $BASE= SBASE( FLinks.BASE_NO),$IPO_MOD
E=SIPO_MODE( FLinks.IPO_FRAME), $LOAD=SLOAD( FLinks.TOOL_NO), $ACC=SACC_CP( LCPDAT1), $
ORI_TYPE=SORI_TYP( LCPDAT1), $JERK=SJERK( LCPDAT1)
;ENDFOLD
;FOLD WAIT Time=1 sec;%{PE}%R 8.3.34,%MKUKATPBASIS,%CWAIT,%VWAIT,%P 3:1
WAIT SEC 1
;ENDFOLD
;FOLD OUT 1 '' State=FALSE CONT;%{PE}%R 8.3.34,%MKUKATPBASIS,%COUT,%VOUTX,%P 2:1, 3:,
5:FALSE, 6:CONTINUE
CONTINUE
$OUT[1]=FALSE
;ENDFOLD
;FOLD OUT 2 '' State=TRUE CONT;%{PE}%R 8.3.34,%MKUKATPBASIS,%COUT,%VOUTX,%P 2:2, 3:,
5:TRUE, 6:CONTINUE
CONTINUE
$OUT[2]=TRUE
;ENDFOLD
;FOLD WAIT Time=1 sec;%{PE}%R 8.3.34,%MKUKATPBASIS,%CWAIT,%VWAIT,%P 3:1
WAIT SEC 1
;ENDFOLD
;FOLD SLIN LinksRaus Vel=2 m/s CPDAT2 ADAT0 Tool[1]:tool1 Base[0];%{PE}%R
8.3.34,%MKUKATPBASIS,%CSPLINE,%VSLIN_SB,%P 1:SLIN_SB, 2:LinksRaus, 3:, 5:2, 7:CPDAT2,
8:ADAT0
SLIN XLinksRaus WITH
$VEL=SVEL_CP( 2, , LCPDAT2), $TOOL=STOOL2( FLinksRaus), $BASE= SBASE( FLinksRaus.BASE_NO),$I
PO_MODE=SIPO_MODE( FLinksRaus.IPO_FRAME), $LOAD=SLOAD( FLinksRaus.TOOL_NO), $ACC=SAC
C_CP( LCPDAT2), $ORI_TYPE=SORI_TYP( LCPDAT2), $JERK=SJERK( LCPDAT2)
;ENDFOLD
;FOLD LIN pflll Vel=2 m/s CPDAT3 Tool[1]:tool1 Base[0];%{PE}%R
8.3.34,%MKUKATPBASIS,%CMOVE,%VLIN,%P 1:LIN, 2:pflll, 3:, 5:2, 7:CPDAT3
$BWDSTART=FALSE
LDAT_ACT=LCPDAT3
FDAT_ACT=Fpflll
BAS(#CP_PARAMS,2)
LIN Xpflll
;ENDFOLD

; Stab der rechten Hand des Anwenders übergeben
PTP RechteHandPos

; 3 Sekunden warten und dann den Greifer öffnen
WAIT SEC 3
CONTINUE
$OUT[1]=TRUE
CONTINUE
$OUT[2]=FALSE

; Flags wieder explizit auf FALSE setzen
$FLAG[998]=FALSE
$FLAG[997]=FALSE
```

```
; Home-Position anfahren
;FOLD PTP HOME Vel=100 % DEFAULT;%{PE}%R 8.3.34,%MKUKATPBASIS,%CMOVE,%VPTP,%P 1:PTP,
2:HOME, 3:, 5:100, 7:DEFAULT
$BWDSTART=FALSE
PDAT_ACT=PDEFAULT
FDAT_ACT=FHOME
BAS(#PTP_PARAMS,100)
$H_POS=XHOME
PTP XHOME
;ENDFOLD
ENDIF

; Stab aus dem mittleren Fach entnehmen
IF Uebergabe==2 THEN

;FOLD PTP ueberMitte Vel=100 % PDAT2 Tool[1]:tool1 Base[0];%{PE}%R
8.3.34,%MKUKATPBASIS,%CMOVE,%VPTP,%P 1:PTP, 2:ueberMitte, 3:, 5:100, 7:PDAT2
$BWDSTART=FALSE
PDAT_ACT=PPDAT2
FDAT_ACT=FueberMitte
BAS(#PTP_PARAMS,100)
PTP XueberMitte
;ENDFOLD
;FOLD SLIN Mitte Vel=2 m/s CPDAT4 ADAT0 Tool[1]:tool1 Base[0];%{PE}%R
8.3.34,%MKUKATPBASIS,%CSPLINE,%VSLIN_SB,%P 1:SLIN_SB, 2:Mitte, 3:, 5:2, 7:CPDAT4, 8:ADAT0
SLIN XMitte WITH
$VEL=SVEL_CP( 2, , LCPDAT4), $TOOL=STOOL2( FMitte), $BASE= SBASE( FMitte.BASE_NO),$IPO_MO
DE=SIPO_MODE( FMitte.IPO_FRAME), $LOAD=SLOAD( FMitte.TOOL_NO), $ACC=SACC_CP( LCPDAT4),
 $ORI_TYPE=SORI_TYP( LCPDAT4), $JERK=SJERK( LCPDAT4)
;ENDFOLD
;FOLD WAIT Time=1 sec;%{PE}%R 8.3.34,%MKUKATPBASIS,%CWAIT,%VWAIT,%P 3:1
WAIT SEC 1
;ENDFOLD
;FOLD OUT 1 '' State=FALSE CONT;%{PE}%R 8.3.34,%MKUKATPBASIS,%COUT,%VOUTX,%P 2:1, 3:,
5:FALSE, 6:CONTINUE
CONTINUE
$OUT[1]=FALSE
;ENDFOLD
;FOLD OUT 2 '' State=TRUE CONT;%{PE}%R 8.3.34,%MKUKATPBASIS,%COUT,%VOUTX,%P 2:2, 3:,
5:TRUE, 6:CONTINUE
CONTINUE
$OUT[2]=TRUE
;ENDFOLD
;FOLD WAIT Time=1 sec;%{PE}%R 8.3.34,%MKUKATPBASIS,%CWAIT,%VWAIT,%P 3:1
WAIT SEC 1
;ENDFOLD
;FOLD SLIN MitteRaus Vel=2 m/s CPDAT5 ADAT0 Tool[1]:tool1 Base[0];%{PE}%R
8.3.34,%MKUKATPBASIS,%CSPLINE,%VSLIN_SB,%P 1:SLIN_SB, 2:MitteRaus, 3:, 5:2, 7:CPDAT5,
8:ADAT0
SLIN XMitteRaus WITH
$VEL=SVEL_CP( 2, , LCPDAT5), $TOOL=STOOL2( FMitteRaus), $BASE= SBASE( FMitteRaus.BASE_NO),$
IPO_MODE=SIPO_MODE( FMitteRaus.IPO_FRAME), $LOAD=SLOAD( FMitteRaus.TOOL_NO), $ACC=SA
```

```
CC_CP( LCPDAT5), $ORI_TYPE=SORI_TYP( LCPDAT5), $JERK=SJERK( LCPDAT5)
;ENDFOLD
;FOLD PTP MitteHoch Vel=100 % PDAT3 Tool[1]:tool1 Base[0];%{PE}%R
8.3.34,%MKUKATPBASIS,%CMOVE,%VPTP,%P 1:PTP, 2:MitteHoch, 3:, 5:100, 7:PDAT3
$BWDSTART=FALSE
PDAT_ACT=PPDAT3
FDAT_ACT=FMitteHoch
BAS(#PTP_PARAMS,100)
PTP XMitteHoch
;ENDFOLD

; Stab der rechten Hand des Anwenders übergeben
PTP RechteHandPos

; 3 Sekunden warten und dann den Greifer öffnen
WAIT SEC 3
CONTINUE
$OUT[1]=TRUE
CONTINUE
$OUT[2]=FALSE
ENDIF

; Flags wieder explizit auf FALSE setzen
$FLAG[998]=FALSE
$FLAG[997]=FALSE

; Home-Position anfahren
;FOLD PTP HOME Vel=100 % DEFAULT;%{PE}%R 8.3.34,%MKUKATPBASIS,%CMOVE,%VPTP,%P 1:PTP,
2:HOME, 3:, 5:100, 7:DEFAULT
$BWDSTART=FALSE
PDAT_ACT=PDEFAULT
FDAT_ACT=FHOME
BAS(#PTP_PARAMS,100)
$H_POS=XHOME
PTP XHOME
;ENDFOLD

; Endlosschleife wird ausführt, solange Verbindung
; zum Matlab-Client vorhanden ist
UNTIL $FLAG[1]==FALSE

; wenn FLAG[1]==FALSE, dann ist die Verbindung zum Client unterbrochen
; EKI-Kanal "KukaKinectGreifer" schließen und löschen
RET=EKI_Close("KukaKinectUebergabe")
RET=EKI_Clear("KukaKinectUebergabe")

END
```

5.2 EKI-Server XML-Konfiguration

```
<ETHERNETKRL>
  <CONFIGURATION>
    <EXTERNAL>
      <TYPE>Client</TYPE>
    </EXTERNAL>
    <INTERNAL>
      <ALIVE Set_Flag="1"/>              // FLAG[1] zur zyklischen Verbindungsprüfung
      <IP>100.100.100.100</IP>           // IP-Adresse des MATLAB-Clients bzw. PC
      <PORT>54600</PORT>                 // Port des MATLAB-Client bzw. PC
      <PROTOCOL>TCP</PROTOCOL>           // Übertragungsprotokoll
    </INTERNAL>
  </CONFIGURATION>
  <RECEIVE>
    <XML>
      <ELEMENT Tag="Sensor/Read/RechteHandFrame" Type="FRAME" Set_Flag="998"/>
      <ELEMENT Tag="Sensor/Read/Uebergabe" Type="INT" Set_Flag="997"/>
    </XML>
  </RECEIVE>
  <SEND>
    <XML>

      <ELEMENT Tag="Robot/Data/ActPos/@X"/>
      <ELEMENT Tag="Robot/Data/ActPos/@Y"/>
      <ELEMENT Tag="Robot/Data/ActPos/@Z"/>
      <ELEMENT Tag="Robot/Data/ActPos/@A"/>
      <ELEMENT Tag="Robot/Data/ActPos/@B"/>
      <ELEMENT Tag="Robot/Data/ActPos/@C"/>
    </XML>
  </SEND>
</ETHERNETKRL>
```

5.3 MATLAB-Client Programm

```
addpath('Mex');
clear all
close all

% Festgelegte XML-Struktur. Erfasste Raumkoordinaten der Kinect werden
% zwischen diese XML-Struktur hinterlegt und anschließend versendet
a = ('<Sensor><Read><RechteHandFrame X="');
b = ('" Y="');
c = ('" Z="');
d = ('" A="-6.6" B="86.8" C="-6.4" /><Uebergabe>');
e = ('</Uebergabe></Read></Sensor>');

% Initialiserung des Kinect-Objekts
% Verfügbare Bildquellen: 'color', 'depth', 'infrared', 'body_index',
% 'body', 'face' und 'HDface'
k2 = Kin2('color','depth','body');

% Knopfdruck erfassen. Bei der Eingabe eines 'q' wird das Programm beendet.
set(gcf,'keypress','k=get(gcf,''currentchar'');'); % listen keypress
k=[];
disp('Zum Beenden "q" drücken')

% Initialisierung eines TCP/IP-Clients. Diesem werden die IP-Adresse und
% der Port des EKI-Servers übergeben
t = tcpclient('100.100.100.100',54600);

% Endlosschleie läuft so lange, bis "q" gedrückt wird
while true
    % Es wird versucht, eventuell empfangene XML-Datenpakete zu lesen
    try
        %Daten des TCPIP_Clients werden ausgelesen und in 'r' gespeichert
        r = read(t);
    end

    % Wenn 'r' nicht leer ist
    if ~isempty(r)
        % Ausgabe des empfangen XML-Datenpaketes und Filterung der
        % beinhalteten Werte

        % 'r' wird von ASCII in Char konvertiert und in 'get_data'
        % gespeichert
        get_data=(char(r))

        % X-Koordinate wird herausgefiltert und gespeichert
        X_Value_Pos = (strfind(get_data, 'X'))+3;
        X_Value = str2double(get_data(X_Value_Pos));

        % Y-Koordinate wird herausgefiltert und gespeichert
        Y_Value_Pos = strfind(get_data, 'Y')+3;
        Y_Value = str2double(get_data(Y_Value_Pos));

        % Z-Koordinate wird herausgefiltert und gespeichert
        Z_Value_Pos = strfind(get_data, 'Z')+3;
        Z_Value = str2double(get_data(Z_Value_Pos));

        % Orientierung-A wird herausgefiltert und gespeichert
```

```
        A_Value_Pos = strfind(get_data, 'A')+3;
        A_Value = str2double(get_data(A_Value_Pos(2)));

        % Orientierung-B wird herausgefiltert und gespeichert
        B_Value_Pos = strfind(get_data, 'B')+3;
        B_Value = str2double(get_data(B_Value_Pos));

        % Orientierung-C wird herausgefiltert und gespeichert
        C_Value_Pos = strfind(get_data, 'C')+3;
        C_Value = str2double(get_data(C_Value_Pos));
    end

    % Kinect Frames abrufen und zwischenspeichern
    validData = k2.updateData;

    if validData
        % Folgende Beschreibung stammt aus dem Add-In von Juan R. Terven:
        % Get 3D bodies joints
        % Input parameter can be 'Quat' or 'Euler' for the joints
        % orientations.
        % getBodies returns a structure array.
        % The structure array (bodies) contains 6 bodies at most
        % Each body has:
        % -Position: 3x25 matrix containing the x,y,z of the 25 joints in
        %   camera space coordinates
        % - Orientation:
        %   If input parameter is 'Quat': 4x25 matrix containing the
        %   orientation of each joint in [x; y; z, w]
        %   If input parameter is 'Euler': 3x25 matrix containing the
        %   orientation of each joint in [Pitch; Yaw; Roll]
        % -TrackingState: state of each joint. These can be:
        %   NotTracked=0, Inferred=1, or Tracked=2
        % -LeftHandState: state of the left hand
        % -RightHandState: state of the right hand
        [bodies, fcp, timeStamp] = k2.getBodies('Quat');

        % Anzahl der detektierten Körper
        numBodies = size(bodies,2);

        % Wenn mehr als 0 Personen detektiert werden
        if numBodies > 0
            % Wenn Status der linken Hand unbekannt ist:
            if ((bodies(1).LeftHandState) == 0)
                disp('Left Hand State: UNKNOWN');
            end

            % Wenn linke Hand nicht erfasst werden konnte:
            if ((bodies(1).LeftHandState) == 1)
                disp('Left Hand State: NOT TRACKED');
            end

            % Wenn die linke Hand offen ist
            if ((bodies(1).LeftHandState) == 2)
                disp('Left Hand State: OPEN');
            end

            % Wenn linke Hand geschlossen ist:
            if ((bodies(1).LeftHandState) == 3)
                disp('Left Hand State: CLOSED');
            end
```

```
        % Wenn die linke Hand ein Lasso-Zeichen zeigt
        if ((bodies(1).LeftHandState) == 4)
            disp('Left Hand State: LASSO');

            %Raumkoordinaten der rechtsseitigen Hand
    RightHandPosition = bodies(1).Position(:,k2.JointType_HandRight);

            % Auswahl des Fachs
            block_nbr = 1;          % Rechte Seite des Hanoi Turms
            block_str = num2str(block_nbr); % Typumwandlung in String

            % Empfohlener Arbeitsraum:
            % (Gilt nur im Mechanik Labor der HAW-Hamburg):
            % X = 450 - 920; Xstandart = 750
            % Y = -500 bis 500
            % Z = 300 - 870  Zstandart = 500

            % Konstante X-Koordinate des TCPs
            x_dbl = 900.0;
            x_str = num2str(x_dbl);

            % Regulierung der Y-Koordinate an den Arbeitsraum
            y_dbl = 0+800*RightHandPosition(1);
            if y_dbl <= (-500.0)
                y_dbl = -500.0;
            end
            if y_dbl >= 500.0
                y_dbl = 500.0;
            end
            y_str = num2str(y_dbl);

            % Regulierung der Z-Koordinate an den Arbeitsraum
            z_dbl = 500+1000*RightHandPosition(2);
            if z_dbl <= 300.0
                z_dbl = 300.0;
            end
            if z_dbl >= 800.0
                z_dbl = 800.0;
            end
            z_str= num2str(z_dbl);

            % Gesendete XML-Information:
            % <Sensor>
            %   <Read>
            %       <RightHandFrame X="..." Y="..." Z="..." A="..."
            %       B="..." C="..." />
            %       <Uebergabe>INT_ZAHL</Uebergabe>
            %   </Read>
            % </Sensor>

            % Erfasste Raumkoordinaten in die feste XML-Struktur
            % hinterlegen und nach uint8 konvertieren
            senddata = uint8([a x_str b y_str c z_str d block_str e]);
            % XML-Datenpaket über TCP/IP an den EKI-Server senden
            write(t, senddata);
            char(senddata)% Ausgabe des gesendeten XML-Datenpakets
        end

        if ((bodies(1).RightHandState) == 4)
            disp('Right Hand State: LASSO')
```

```
                %Raumkoordinaten der rechtsseitigen Hand
            RightHandPosition = bodies(1).Position(:,k2.JointType_HandRight);

                % Auswahl des Objektes
                block_nbr = 2;                      % Mitte des Hanoi Turms
                block_str = num2str(block_nbr);  % Typumwandlung in String

                % Empfohlener Arbeitsraum:
                % (Gilt nur im Mechanik Labor der HAW-Hamburg):
                % X = 450 - 920; Xstandart = 750
                % Y = -500 bis 500
                % Z = 300 - 870  Zstandart = 500

                % Konstante X-Koordinate des TCPs
                x_dbl = 900.0;
                x_str = num2str(x_dbl);

                % Regulierung der Y-Koordinate an den Arbeitsraum
                y_dbl = 0+800*RightHandPosition(1);
                if y_dbl <= (-500.0)
                    y_dbl = -500.0;
                end
                if y_dbl >= 500.0
                    y_dbl = 500.0;
                end
                y_str = num2str(y_dbl);

                % Regulierung der Z-Koordinate an den Arbeitsraum
                z_dbl = 500+1000*RightHandPosition(2);
                if z_dbl <= 300.0
                    z_dbl = 300.0;
                end
                if z_dbl >= 800.0
                    z_dbl = 800.0;
                end
                z_str= num2str(z_dbl);

                % Gesendete XML-Information:
                % <Sensor>
                %   <Read>
                %       <RightHandFrame X="..." Y="..." Z="..." A="..."
                %       B="..." C="..." />
                %       <Uebergabe>INT_ZAHL</Uebergabe>
                %   </Read>
                % </Sensor>

                senddata = uint8([a x_str b y_str c z_str d block_str e]);
                write(t, senddata);
                char(senddata)
            end
        end
        % Wenn 'q' gedrückt wurde, wird die Schleife verlassen
        if ~isempty(k)
            if strcmp(k,'q'); break; end
        end
    end
    pause(1)  % Wartezeit bzw. Abtastfrequenz
end


% Kinect-Objekt löschen
```

```
k2.delete;

close all;
```