Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Patrick Zuther

Integration einer sensorgeführten Robotersteuerung

Fakultät Technik und Informatik
Department Fahrzeugbau und Flugzeugbau

Faculty of Engineering and Computer Science
Department of Automotive and Aeronautical Engineering

Patrick Zuther

Integration einer sensorgeführten Robotersteuerung

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung

im Studiengang Mechatronik
am Department Fahrzeugtechnik und Flugzeugbau
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing Jochen Maaß
Zweitgutachter: Prof. Dr.-Ing Jörg Dahlkemper

Abgegeben am 21.08.2019

Patrick Zuther

Thema der Abschlussarbeit

Integration einer sensorgeführten Robotersteuerung

Stichworte

Industrieroboter, Bildverarbeitung, OpenCV, Mensch-Roboter-Kollaboration, UDP, Kalman Filter, Robot Sensor Interface, Qt

Kurzzusammenfassung

Im Zuge des Forschungsprojektes „MeriTec“ der HAW Hamburg gilt es ein System zu entwickeln, mit dessen Hilfe eine sensorgeführte Roboterbewegung auf Basis der optischen Triangulation mit zwei Kameras ermöglicht wird. Auf diese Weise soll die Grundlage für das ferngesteuerte Schweißen geschaffen werden. Diese Thesis behandelt dabei die Theorie der benötigten Triangulation, bei der für die Lösung des Korrespondenzproblems der stereoskopischen Messung, die in OpenCV integrierte ArUco-Softwarebibliothek verwendet wird. Außerdem werden Kalmanfilter zur Interpolation von Messwerten und ein Robot Sensor Interface in eine in Echtzeit kommunizierende Anwendung integriert, die vollständig von einer im QT-Framework entwickelten Benutzeroberfläche gesteuert wird.

Patrick Zuther

Title of the paper

Integration of a sensor-guided robot control system

Keywords

Industrial Robots, Image Processing, OpenCV, Human-Robot-Collaboration, UDP, Kalman Filter, Robot Sensor Interface, Qt

Abstract

In the course oft he research project „MeriTec“ of HAW Hamburg, a system hast o be developed which enablees a sensor-guided robot movement based on optical triangulation with two cameras. In thris way, the basis for remote-controlled welding ist o be created. This thesis deals with the theory oft he required triangulation in which the ArUco software library integrated in OpenCV is used to solve the correspondence problem of stereoscopic measurements. In addition, Kalman filters for interpolating measured values and a Robot Sensor Interface will be integrated into a real-time communicating application fully controlled by a user interface developed in the QT framework.

Inhaltsverzeichnis

1 Einleitung 12

1.1 Problemstellung 13

1.2 Aktueller Stand 13

2 Arbeitsumfeld 15

2.1 Technischer Aufbau 15

2.2 Entwicklungsumgebung und Bildverarbeitungsbibliothek 16

2.3 Kamerasystem 17

2.4 Kuka Agilus KR6 R900 sixx 19

2.5 KUKA KRC4 Compact 21

2.6 Kuka Robot Sensor Interface (RSI) 22

3 Grundlagen 24

3.1 Stereoskopie 24

3.2 Kameraparameter 25

3.2.1 Intrinsische Kameraparameter 25

3.2.2 Extrinsische Kameraparameter 26

3.2.3 Verzeichnung 27

3.2.4 Kamerakalibration 28

3.3 Markererkennung 30

3.3.1 Marker 30

3.3.2 Detektion 31

3.4 Herleitung der Triangulation eines Punktes im Raum bei gegebener Korrespondenz eines stereoskopischen Bildpaares 33

3.4.1 Allgemeine Beschreibung 33
3.4.2 Gleichungsparameter 37
3.5 Berechnung einer Markerpose aus 3 Punkten 38
3.6 Kalmanfilterung 42
4 Anforderungsanalyse 45
5 Kommunikation der Systemkomponenten 46
5.1 Sensorsystem 46
5.2 Kommunikation mit der Robotersteuerung 48
5.2.1 Netzwerkkonfiguration 48
5.2.2 Testserver-Kontrolle 49
5.3 XML-Konfiguration 50
5.3.1 RSI-Kontext 50
5.3.2 Aufbau der Konfigurationsdatei 51
6 Praktische Umsetzung 55
6.1 Programmablauf 55
6.1.1 Die Klasse „MainWindow“ 57
6.1.2 Die Klasse „PoseEstimation“ 58
6.1.3 Die Klasse „ImgTask“ 60
6.1.4 Die Klasse „UdpServer“ 61
6.1.5 Die Klasse „Viz_Visualization“ 62
6.2 KRL Programm 63
6.3 Inbetriebnahme 64
7 Durchführung und Validierung 66
7.1 Messaufbau 66
7.2 Messmittel 67

7.3 Objekterkennung 68
7.3.1 Messrauschen 68
7.3.2 Translatorische Messauswertung 69
7.3.3 Rotatorische Messauswertung 73
7.4 Kalmanfilter 75
7.5 Performance 77
7.5.1 Bildaufnahme/ Bildverarbeitung 77
7.5.2 Kommunikation mit der Robotersteuerung 78
8 Zusammenfassung 79
8.1 Fazit 79
8.2 Ausblick 80
9 Anhang 84

Abbildungsverzeichnis

Abbildung 1: Zielsetzung von MRK-Systemen 12
Abbildung 2.1: Stereoskopischer Messaufbau 15
Abbildung 2.2: Testaufbau Roboter 16
Abbildung 2.3: Qt und OpenCV 17
Abbildung 2.4: Robotersystem-Grundbestandteile 19
Abbildung 2.5: Arbeitsraum KuKa Agiblus KR6 R900 sixx 20
Abbildung 2.6: KRC4-Compact 21
Abbildung 2.7: Philosophie Robot Sensor Interface 22
Abbildung 2.8: Schematischer Aufbau eines RSI-Kontextes 23
Abbildung 2.9: RSI-Monitor 24
Abbildung 3.1: Schematische Darstellung der intrinsischen Kameraparameter 26
Abbildung 3.2: Extrinsische Kameraparameter 27
Abbildung 3.3: Gitter, kissenförmig und tonnenförmig verzeichnet 27
Abbildung 3.4: Stereoskopisches Bildpaar 28
Abbildung 3.5: Variationen von Aufnahmen für die Kamerakalibrierung 28
Abbildung 3.6: ArUco 30
Abbildung 3.7: ArUco 31
Abbildung 3.8: Ablauf der Markerdetektion 32
Abbildung 3.9: Beispielhaftes Koordinatensystem auf dem ArUco-Marker 39
Abbildung 3.10: Nichtkommutativität von Rotationen 40
Abbildung 3.11: Rechthand-Koordinatensystem 41
Abbildung 3.12: Die Grundidee des Kalman-Filters: Vorhersage der Trajektorie von Bewegungsgleichungen und Korrektur der Trajektorie mit jeder neuen Messung .. 42
Abbildung 3.13: Berechnungssequenz des Kalman-Filters mit logischer zeitlicher Zuordnung der Zustände s_k sowie realer zeitlicher Zuordnung der Berechnungsvorgänge 43
Abbildung 3.14: Berechnungsschritte des Kalman-Filters 44
Abbildung 5.1: Konzept der Messwertinterpolation mittels Kalman Filter 46
Abbildung 5.2: Netzwerkkonfiguration – IP Adresse 48
Abbildung 5.3: Kuka Testserverapplikation 49

Abbildung 5.4: Beispielhafter Aufbau des RSI-Kontext 51
Abbildung 5.5: Schema der XML-Konfigurationsdatei 51
Abbildung 5.6: Beispiel XML- Struktur <CONFIG 52
Abbildung 5.7: Beispiel XML-Struktur <SEND 52
Abbildung 5.8: Beispiel versendetes XML-Dokument der Robotersteuerung 53
Abbildung 5.9: Beispiel XML-Struktur <RECEIVE 54
Abbildung 5.10: Beispiel versendetes XML-Dokument des Sensorsystems 54
Abbildung 6.1: Sequenzdiagramm der Prädiktion und Systemkommunikation 55
Abbildung 6.2: Sequenzdiagramm zum Korrigieren der Objektpose im
Kalman Filter 56
Abbildung 6.3: UML-Klassendiagramm der Klasse „MainWindow“ 57
Abbildung 6.4: UML-Klassendiagramm der Klasse PoseEstimation 58
Abbildung 6.5: UML-Diagramm der Klasse „imgTask“ 60
Abbildung 6.6: UML-Diagramm der Klasse „UdpServer“ 61
Abbildung 6.7: UML-Klassendiagramm der Klasse „Viz_Visualization 62
Abbildung 6.8: KRL Programm für Demonstration Zuther 63
Abbildung 6.9: GUI der Anwendung 64
Abbildung 7.1: Untersuchter Arbeitsraum 66
Abbildung 7.2: Untersuchte Markervarianten 67
Abbildung 7.3: Winkelbeziehungen des Prüfkörpers 67
Abbildung 7.4: Messrauschen entlang der X-Achse 69
Abbildung 7.5: Ausgangslage der translatorischen Messwertaufnahme 69
Abbildung 7.6: Frontale Translation des Prüfmarkers 70
Abbildung 7.7: Rotation um horizontale Achse 71
Abbildung 7.8: Rotation um vertikale Achse 72
Abbildung 7.9: Rotation um horizontale Achse 73
Abbildung 7.10: Rotation entlang der Objekttiefe 73
Abbildung 7.11: Rotation um vertikale Achse 74
Abbildung 7.12: Gegenüberstellung der Werte Des Kalman Filter
mit den Messwerten 76
Abbildung 7.13: Fehlverhalten des Kalman-Filters (Detailansicht 76
Abbildung 7.14: FPS des umgesetzten Systems 77

Abbildung 7.15: UDP Antwortzeit des Messsystems bei vollem Funktionsumfang mit RSI $4ms$ Konfiguration 78
Abbildung 8.1: Demonstration der entwickelten Anwendung 79
Abbildung 8.2: Schweißgeräte mit möglicher Markerlage 80

Tabellenverzeichnis

Tabelle 2.1 Daten Kamera 18
Tabelle 2.2 Daten Objektiv 18
Tabelle 6: Inbetriebnahme der sensorgeführten Robotersteuerung 65
Tabelle 7.1: Analyse des Messrauschens der Markererkennung 69
Tabelle 7.2- Analyse der translatorischen Messgenauigkeit 72
Tabelle 7.3: Winkelmessungen um horizontale Achse 73
Tabelle 7.4: Winkelmessungen um Achse entlang der Objekttiefe 74
Tabelle 7.5: Winkelmessungen um vertikale Achse 74
Tabelle 7.6: durchschnittliche Differenz 74

Abkürzungsverzeichnis

API	Application Programming Interface
FPS	Frames per Second
GUI	Graphical User Interface
I/O	Input/Output
IP	Internet Protocol
KLI	Kuka Line Interface
KRC4-C	KRC4-Compact
KRL	Kuka Robot Language
MRK-System	Mensch-Roboterkooperation, auch Mensch-Roboter-Kollaboration
NAT	Network Address Translation
PC	Personal Computer
ROI	Region of Interest
RSI	Robot Sensor Interface
TCP	Transmission Control Protocol
TIS	The Imaging Source
UDP	User Datagram Protocol
VB	Visual Basic
XML	Extensible Markup Language

1 Einleitung

Der Industrieroboter ist seit Jahren fester Bestandteil von Produktionsanlagen und zählt seither als Treiber der Automatisierung in der Serienfertigung. [Müller et al., 2019, S.1f.] Als Ansatz, den Industrieroboter auch für die Fertigung von Kleinserien mit komplexen und flexiblen Prozessen nutzen zu können, werden in dem Themengebiet der Mensch-Roboterkooperation, oder auch Mensch-Roboter-Kollaboration (MRK), Anwendungen entwickelt, die den Austausch von Informationen und Handlungen zwischen Mensch und Roboter zur Ausführung einer Aufgabe verfolgen. Die Kombination der individuellen Stärken von Mensch und Roboter sind dabei das Ziel einer Mensch-Roboter-Kooperation, um durch die Einbringung spezifischer Fähigkeiten eine flexible und optimale Arbeitsteilung zu realisieren. [Shen,2015] (vgl. Abbildung 1).

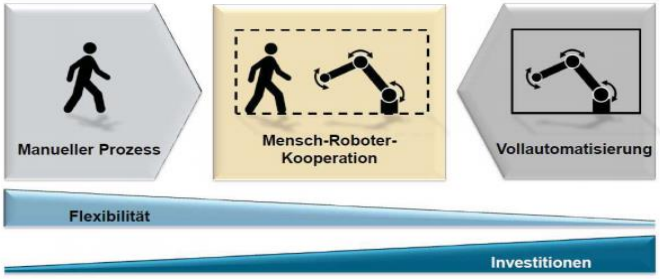


Abbildung 1: Zielsetzung von MRK-Systemen [Schröter, 2018]

Beispielsweise gelingt es Robotern mittlere bis schwere Lasten unabhängig von der Tageszeit und mit einer hohen Wiederholungsrate zu bewegen. Außerdem können sie gefährliche oder giftige Gegenstände handhaben und erledigen ihre Aufgaben mit sehr großer Präzision. [Blume, 2014] Der Mensch hingegen ist flexibel bei der Erledigung seiner Arbeit. Insbesondere bei Überraschungen und Fehlern im Prozessablauf kann der Mensch seine Erfahrung und seine Problemlösungskompetenz einbringen, um das System wieder in einen funktionsfähigen Zustand zu überführen. In der vorliegenden Arbeit wird in diesem Sinne ein MRK-System entwickelt, das einen Roboter der Firma KUKA AG auf der Informationsgrundlage eines stereoskopischen Kameramesssystems analog zu den Bewegungen eines durch den Menschen bewegten Markerobjektes verfährt.

1.1 Problemstellung

Im Rahmen des an der HAW Hamburg entstandenen Forschungsprojektes für die Mensch-Roboter-Kollaboration in der Schweißtechnik (MeriTec) soll ein Verfahren zum ferngesteuerten Schweißen entwickelt werden. Hierfür soll ein System integriert werden, das die Handbewegung des Schweißers unabhängig von der Lichtschwankung, Rauchentwicklung und Spritzbildung eindeutig identifizieren kann. Es wird der Ansatz eines optischen Triangulationsverfahrens unter Einsatz eines Markers verfolgt. Das Verfahren soll evaluiert und ggf. durch eine inverse Lösung ersetzt werden.
Die identifizierten Handbewegungen sollen der Robotersteuerung geeignet übergeben werden. Dazu soll ein Sensor-Interface in Betrieb genommen, und in ein entsprechendes Roboterprogramm integriert werden. Ziel ist die Evaluierung, ob die Handbewegungen des Schweißers eindeutig und mit hinreichender Genauigkeit umgesetzt werden. Konkret sind dies die folgenden Arbeitspakete:

- Die bestehende Robotersteuerung um das Sensor-Interface erweitern
- Entwurf und Umsetzung eines Konzepts für eine Echtzeit-Datenverarbeitungs-Anwendung auf dem PC
- Die Software des Sensorsystems evaluieren und ggf. inverses Sensorprinzip umsetzen
- Integration eines Filter- und Begrenzungsmechanismus für die Eingabe-Bewegungen in die Applikation

1.2 Aktueller Stand

Das von Menschen in Echtzeit gesteuerte Verfahren von Robotern bricht die Barriere der unflexiblen und mit *„Know-how“* belasteten Programmierung von Trajektorien für die Robotersteuerung hin zu einer spielend intuitiven Möglichkeit das Wissen von Facharbeitern dynamisch in den Prozess einfließen zu lassen. Martin Edberg und Peter Nyman von der TU Chalmers [Edberg,Nyman, 2010] haben beispielsweise ein System entwickelt, welches die Echtzeit-Kontrolle eines KuKa KR5 sixx R650 über einen *„Multi-Touch“* Bildschirm ermöglicht.

Dieser ist in der Lage verschiedene Arten von Fingergesten zu interpretieren und daraufhin den Roboter durch eine am TCP angebrachte Kamera aus Sicht des Roboters zu verfahren. Umgesetzt wurde die Ansteuerung über das Kuka Robot Sensor Interface (RSI) Technologiepaket, welches auch zur Realisierung der in dieser Arbeit entwickelten Anwendung zu verwenden ist. Nach Angaben von Nyman und Edberg war der verwendete Rechner, auf dem die Applikation für die Auswertung des *„Multi-Touch“* Bildschirms ausgeführt wurde, nicht in der Lage, die von Kuka geforderte Zykluszeit stabil einzuhalten. Daraufhin wurde beschlossen die zeitkritische Kommunikation auf einen Zwischenrechner auszulagern, welcher lediglich zur Weiterleitung der aktuellen Systemdaten in beide Richtungen dient.

Das Kuka RSI wird ebenfalls in der Habilitationsschrift von Dr.-Ing. Alexander Winkler an der TU Chemnitz zur Entwicklung sensorgeführter Roboterbewegungen verwendet. [Winkler, 2016]

Er untersucht diverse positionsbasierte Regelungsansätze, welche innerhalb des RSI umgesetzt werden können, und zeigt mit praktischen Experimenten, wie z.B. dem Inversen Pendel, wie Regelungsketten im RSI erstellt werden können. Die Dissertation von Michael Visentin der Unversität Augsburg stellt ausführlich Konzepte zur Echtzeitentwicklung von Anwendungen dar und integriert das KUKA Robot Sensor Interface zusammen mit dem Stäubli uniVAL Interface in eine eigene Roboter API, um Roboter beider Hersteller synchron zu verfahren.

Die Dissertation von Daniel Modraw der TU Münschen [Mordrow, 2008] befasst sich mit der echtzeitfähigen aktiven Stereoskopie und liefert dabei eine sehr ausführliche Übersicht über die verschiedenen Varianten der 3D-Messtechnik. In dem Unterkapitel zur passiven Stereoskopie hebt er die Herausforderung des Korrespondenzproblems hervor und stellt zur Lösung die Verwendung von Markern oder strukturierten Lichtquellen in Aussicht. Das Erkennen von eindeutigen Markern wurde von [Gerrido-Jurado et al., 2016] und [j.Romero-Ramirez, Munoz-Salinas, Medina-Carnicer, 2018] in der frei zugänglichen Softwarebibliothek „ArUco“ umgesetzt und auch in die weit verbreitete Bildverarbeitungsbiliothek OpenCV integriert und wird als Ansatz für die Entwicklung der Software dieser Arbeit verwendet.

2 Arbeitsumfeld

2.1 Technischer Aufbau

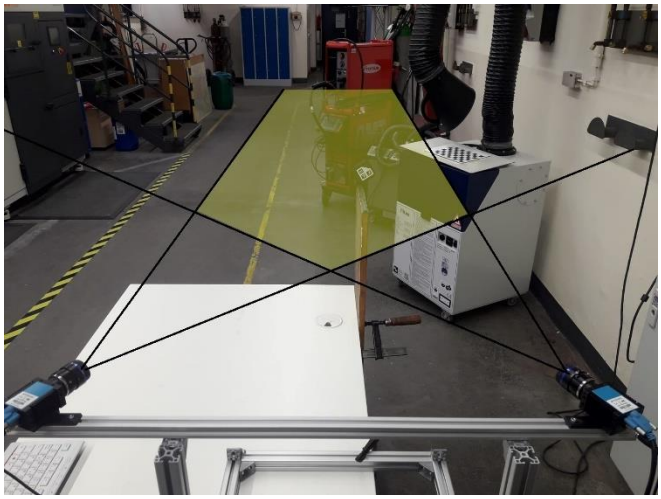
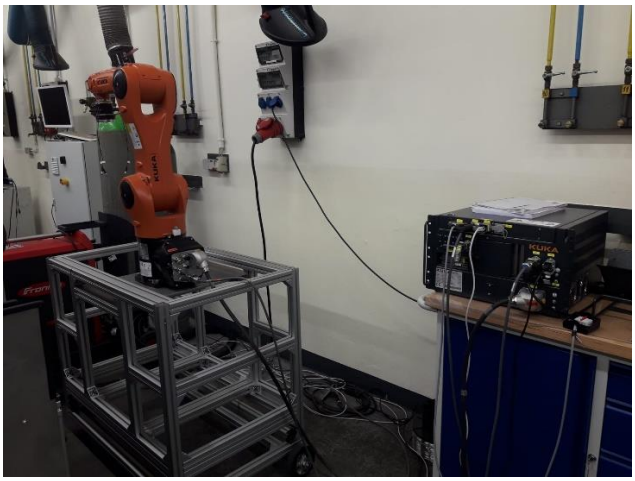


Abbildung 2.1: Stereoskopischer Messaufbau

Abbildung 2.1 zeigt den vorhandenen stereoskopischen Messaufbau, mit dem die Objekterkennung gelöst werden soll. In den unteren beiden Ecken befinden sich die um jeweils $23{,}6°$ zueinander verdrehten Kameras. Diese sind $70cm$ entlang des zu sehenden Alu-Profil versetzt, was zu Folge hat, dass sich die optischen Achsen in einem Abstand von $80cm$ treffen. Der relevante Messbereich, in dem ein Objekt von beiden Kameras erfasst wird, ist gelb eingefärbt.



(a) Mobiler KUKA KR6 R900 sixx mit KRC4-Compact Steuerung

(b) Ausrichtung des Roboters zum Messsystem

Abbildung 2.2: Testaufbau Roboter

In Abbildung 2.2(a) wird der zu verwendende KUKA Agilus KR6 R900 sixx (linke Bildhälfte) und die dazugehörige Robotersteuerung KRC4-Compact (rechte Bildhälfte) gezeigt. Der Roboter befindet sich zur Zeit der Entwicklung auf einem mobilen Alu-Profil, für den Rahmen dieser Arbeit wird der Roboter jedoch als stationäres System angenommen. Abbildung 2.2(b) zeigt den Roboter in seiner Ausgangsstellung für das in Kapitel 1.1 geforderte Demonstrationsszenario. Die Orientierung des Roboters zum Messystem wurde so dabei so gewählt, dass bei erkannter Änderung der Objektpose durch das Sensorsystem die Roboterbewegung für den menschlichen Bediener gut ersichtlich ist.

2.2 Entwicklungsumgebung und Bildverarbeitungsbibliothek

QT

Zum Programmieren dieser Arbeit wird das auf C++ basierende Framework Qt verwendet. Es ist eine objektorientierte und plattformübergreifende Entwicklungsumgebung, die unter anderem das Erstellen von umfangreichen Benutzeroberflächen in seinem Funktionsumfang

integriert hat. Ein grundlegender Mechanismus aller Qt-Programme sind *Signale* und *Slots*. Diese ermöglichen eine Kommunikation zwischen Objekten, ohne dass diese sich gegenseitig kennen.

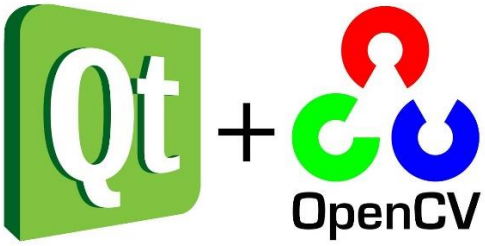


Abbildung 2.3: Qt und OpenCV

OpenCV

OpenCV (Open Source Computer Vision) ist eine freie und mit dem Fokus auf Echtzeitanwendungen entwickelte Bildverarbeitungsbibliothek mit mehr als 2500 Algorithmen. Darüber hinaus unterstützt OpenCV diverse Betriebssysteme und stellt Schnittstellen für die Nutzung verschiedener Programmiersprachen, darunter Python, C und C++, bereit. [Willowgarage] In dieser Arbeit wird OpenCV in Version 4.0 verwendet, außerdem werden die über Github verfügbaren Erweiterungen „ArUco“ (Bibliothek optimiert für Markerdetektionen) und „Viz“ (Bibliothek optimiert für Grafik Visualisierungen) in die Standartversion integriert. [GitHub OpenCV Contrib]

2.3 Kamerasystem

Für die Aufnahme der stereoskopischen Bilder werden zwei Kameras der Firma „The Imaging Source“ (TIS) vom Typ DMK 33GP200e verwendet. Die Spezifikationen können aus Tabelle 2.1 entnommen werden. Es handelt sich dabei um eine monochrome Kamera, was bedeutet, dass ausschließlich Graustufeninformationen aufgenommen werden.

Auflösung horizontal/vertikal	1900 Pixel x 1200 Pixel
Pixelgröße horizontal/vertikal	4,8 μm x 4,8 μm
Bildwiederholrate	50 fps
Objektivanschluss	C-Mount
Shutter	Global
Schnittstelle	GigE

Tabelle 2.1: Daten Kamera

Als Objektiv wird auf jeder Kamera ein Weitwinkelobjektiv vom Typ „TCL 1216 5MP“, ebenfalls von der Firma „The Imaging Source“, verwendet. Die entsprechenden Spezifikationen können aus Tabelle 2.2 entnommen werden.

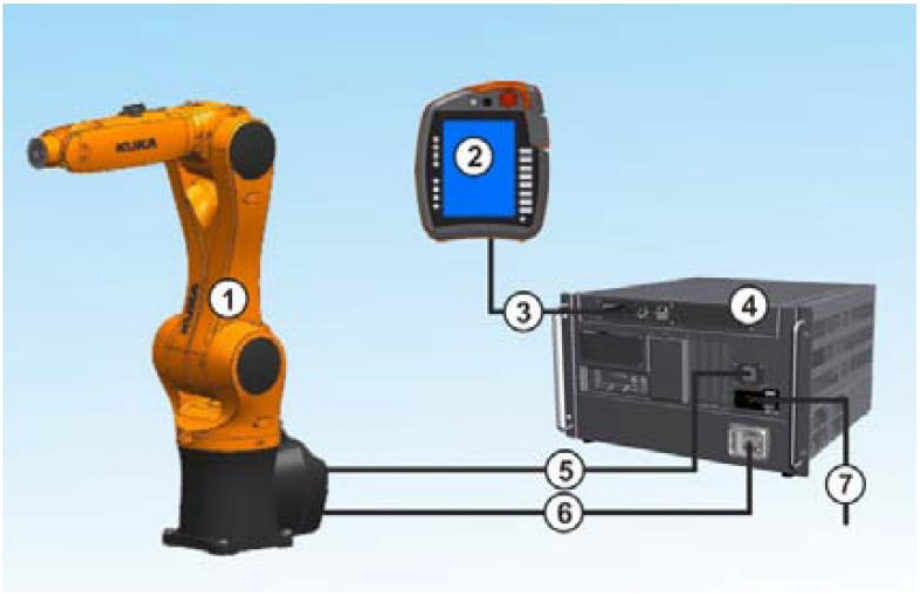
Format	2/3
Mount	C
Brennweite	12 mm
Blendenzahl	1,6 - 16
Minimale Objektdistanz	0,1 m

Tabelle 2.2: Daten Objektiv

Für das Erfassen von Bildern der Kamera wurde die vom Hersteller entwickelte IC Imaging Control SDK (Software Development Kit) verwendet. Das Tool ermöglicht unter Windows die Einstellung sämtlicher Kameraparameter und unterstützt eine Verwendung unter C++, C# und VB. Des Weiteren stellt TIS Tools und Support für weitere Betriebssysteme und Programmiersprachen auf Github [GitHub OpenCV_Contrib] bereit.

2.4 Kuka Agilus KR6 R900 sixx

In dieser Arbeit wird der KUKA Agilus KR6 R900 sixx zusammen mit der Robotersteuerungseinheit „KRC4-Compact“ (KRC4-C) eingesetzt. Die Agilus-Modelle repräsentieren die Kleinroboterserie von KUKA. Diese sind als Fünf- und Sechsachsmodelle erhältlich und besitzen eine Traglastfähigkeit von bis zu 10kg. [KUKA Roboter GmbH, 2015] Die Kombination aus hoher Positionswiederholgenauigkeit von $\pm 0.03\mathrm{mm}$, integrierter Energiezuführung und diverser Einbaulagen des Manipulators, machen die Agilus-Serie flexibel und leistungsstark. Das Robotersystem des KUKA Agilus umfasst einen Manipulator, die KRC4-Steuerungseinheit, Verbindungsleitungen, Werkzeuge und weitere Ausrüstungsteile. Alle Systemkomponenten sind in Abbildung 2.3 dargestellt und gekennzeichnet.



1. Manipulator
2. Programmiergerät smartPad
3. Verbindungsleitung smartPad
4. Robotersteuerung
5. Datenleitung
6. Motorleitung
7. Externe Ethernetverbindung

Abbildung 2.4: Robotersystem-Grundbestandteile

Der Manipulator stellt den Bewegungsapparat des Robotersystems dar. Eine Besonderheit ist, dass dieser selbst Raumkoordinaten anfahren kann, die sich nah am Roboterfuß befinden. In Abbildung 2.4 wird der Arbeitsraum des Manipulators dargestellt. Alle Maßangaben sind in Millimeter gültig. Aus den Angaben zum Arbeitsraum lassen sich kinematische Gleichungen bzw. Modelle herleiten. Diese können z.B. für die Bewegungssimulation und die Trajektorienplanung genutzt werden.

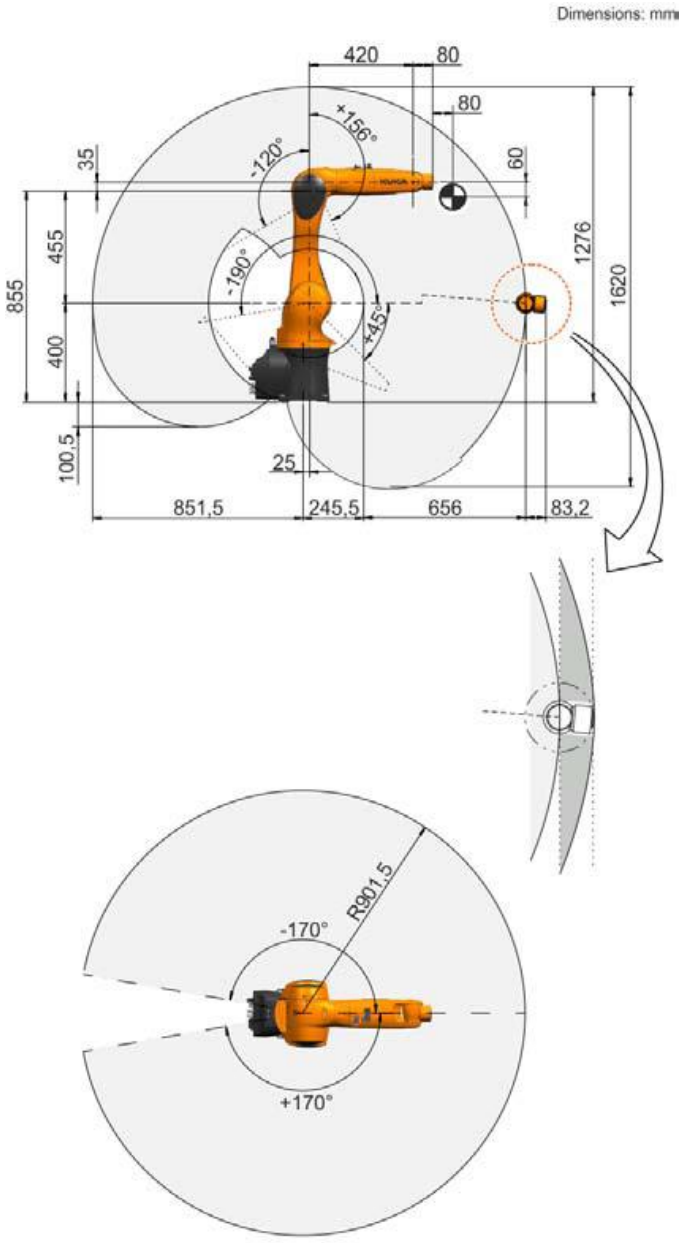


Abbildung 2.5: Arbeitsraum KUKA Agiblus KR6 R900 sixx

2.5 KUKA KRC4 Compact

Das zentrale Element des Robotersystems ist die KRC4-Robotersteuerung, welche in verschiedenen Variationen erhältlich ist. [KUKA Roboter GmbH, 2014] Bei der in dieser Arbeit verwendeten Robotersteuerung handelt es sich um die KRC4-Compact, welche für Roboter mit bis zu sechs Achsen und geringe Traglasten ausgelegt ist. Abbildung 2.5 kennzeichnet die möglichen Schnittstellen der KRC4-C.

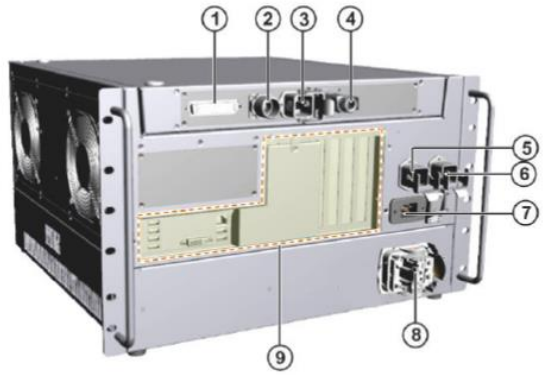


Abbildung 2.6: KRC4-Compact

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-Schnittstelle

In dieser Arbeit steuert das entwickelte Sensorsystem den Roboter über die X66-Ethernet-Schnittstelle. Die dafür benötigte Konfigurationsbeschreibung findet sich in Kapitel 5.2. Für weitere Informationen bzgl. der KRC4-C, z.B. den technischen Spezifikationen oder der grundsätzlichen Installation, wird auf die Bedienungsanleitung von KUKA verwiesen. 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.

2.6 Kuka Robot Sensor Interface (RSI)

In der Grundversion der KUKA-Steuerungssoftware stehen in der Programmiersprache Kuka Robot Language (KRL) Befehle für die Punkt zu Punkt- Linear- und Kreisinterpolation zur Verfügung. Das optionale Technologiepaket Robot Sensor Interface (RSI) hingegen erweitert den Zugriff auf die Robotertrajektorie durch den Zugriff auf die Sollwerte der Lageregelung. RSI wurde speziell für die Realisierung sensorgeführter Roboterbewegungen konzipiert und operiert seit der KRC4 je nach Konfiguration im festen Zyklustakt von $4ms$ oder $12ms$. [KUKA Roboter GmbH, 2016]

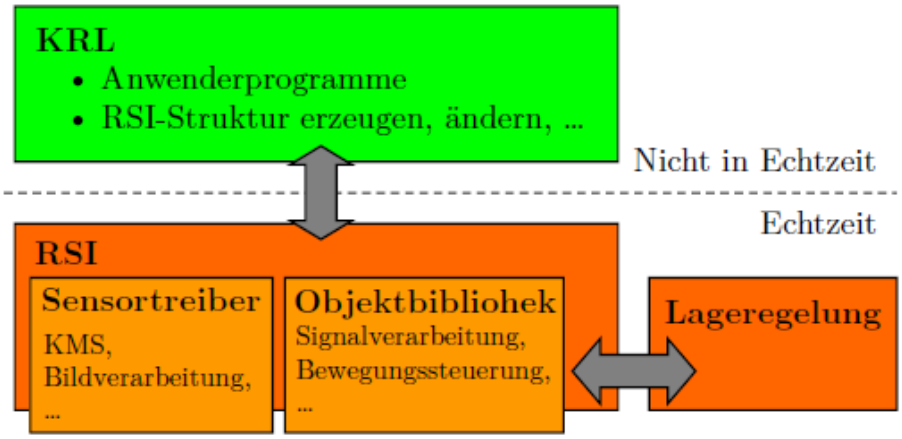


Abbildung 2.7: Philosophie Robot Sensor Interface [Robotik-Buch-Regelung,53]

Um das RSI zu verwenden, muss in dem angewählten Roboterprogramm zunächst die RSI-Struktur erzeugt werden, damit die Ausführung von echtzeitfähigen Programmen ermöglicht wird. (vgl. Abbildung 2.6) Innerhalb des RSI-Kontextes, welches den Inhalt der RSI Funktionen zusammenfasst, ist es darüber hinaus möglich, verschiedene RSI-Objekte zu erzeugen, mit denen diverse Reglerstrukturen realisierbar sind. [Winkler, 2016, S.53]

Die Programmierung des RSI-Kontextes erfolgt dabei über die von Kuka beigelieferte Software „RSI-VisualShell“. Diese bietet eine grafische Oberfläche und kann dem Erstellen von Signalflussdiagrammen in der bekannten MATLAB Erweiterung „Simulink“ nachempfunden werden. Abbildung 2.7 veranschaulicht diesen Zusammenhang. Es wird der schematische Aufbau und die Verknüpfung von RSI-Objekten gezeigt.

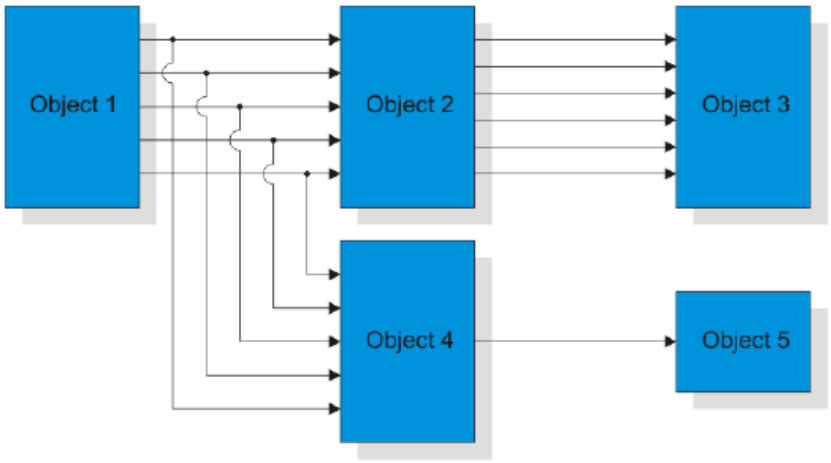


Abbildung 2.8: Schematischer Aufbau eines RSI-Kontextes

Für eine Beschreibung der möglichen RSI-Objeke, die zur Verfügung stehen, wird an dieser Stelle auf die offizielle Dokumentation [KUKA Roboter GmbH, 2016] verwiesen. Dennoch wird im Folgenden ein grober Überblick zwischen den Arten der RSI-Objekte präsentiert.

Zunächst gibt es Objekte, die **Eingangssignale** für eine Robotersteuerung liefern. Dies können Gelenkwinkel, kartesische Koordinaten, Kraft-/Momentsensoren, Motorströme, Generatoren für Testsignale sein, oder auch Variablen, die vom Anwenderprogramm in den RSI-Kontext übertragen werden.

Des Weiteren werden **signalverarbeitende** Objekte wie z.B. einfache mathematische Funktionen und Signalvergleiche, aber auch zeitkontinuierliche und zeitdiskrete lineare Übertragungsglieder zur Verfügung gestellt.

Außerdem gibt es Objekte, deren Aufgabe das Übernehmen von **Stelleingriffen** ist. Das können beispielweise die Aufschaltung von Positionskorrekturen, das Steuern von Ausgängen der Robotersteuerung, oder auch das Setzen von Signalen und Variablen im ausgeführten KRL Programm sein.

Ist ein gewünschter RSI-Kontext erstellt, so generiert das „RSI-VisualShell“ die benötigten Konfigurationsdateien, welche in einem speziellen Verzeichnis auf der Robotersteuerung zu hinterlegen sind. (vgl. Kapitel 5.2.2)

Der Datentransfer des RSI mit einem externen Sensorsystem kann dabei sowohl über Ethernet unter Verwendung des UDP-Protokolls, als auch über das IO-System der Robotersteuerung erfolgen. UDP bedeutet „User Datagramm Protocol“ und beschreibt einen Standard zur

verbindungslosen Kommunikation zwischen Teilnehmern innerhalb eines Netzwerkes. Verbindungslos bedeutet hierbei ungesichert, da nicht garantiert werden kann, dass gesendete Pakete zuverlässig ankommen, oder in der Reihenfolge ankommen, in der sie gesendet wurden. Vorteile bietet das Protokoll jedoch in seiner Kommunikationsgeschwindigkeit, da es nicht zu sogenannten „Handshake“ Situationen kommt, in denen sich die Teilnehmer der sicheren Übertragung ihrer Daten vergewissern.

Das Datenformat XML, ein Standard zur Erstellung maschinen- und menschenlesbarer Dokumente in Form einer vorgegebenen Baumstruktur, bildet den Inhalt der UDP-Nachrichten. Der Aufbau einer XML-Datei für die Kommunikation mit der Robotersteuerung, als auch dessen Konfiguration wird in Kapitel 5.2 behandelt.

Zusätzlich zur direkten Lageregelung stellt das RSI mit dem RSI-Monitor eine Möglichkeit dar, sich aktuelle Mess- und Zustandsgrößen darstellen zu lassen und für spätere Auswertungen abzuspeichern

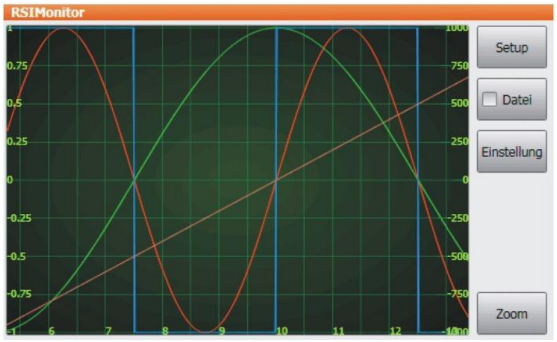


Abbildung 2.9: RSI-Monitor

3 Grundlagen

3.1 Stereoskopie

Die Stereoskopie ist Bestandteil der 3D-Messtechnik und beschreibt die Fähigkeit des räumlichen Sehens. Als Grundlage dient der geometrische Ansatz der Triangulation, bei dem die Tiefenberechnung eines Objektes aus verschiedenen Positionen durchgeführt wird. Für technische Verfahren wird dabei zwischen passiven und aktiven Stereo-Verfahren unterschieden.

Zu den passiven Stereo-Verfahren gehören diejenigen Messmethoden, die auf einem bildbasierenden Ansatz unter Verwendung von Kameras aufbauen und sich in ihrer Grundkonzeption in gewisser Weise am menschlichen Sehen und Erkennen orientieren. Der Begriff „passiv“ bezieht sich also auf reine Auswertung von Sensor-Informationen, die ein Abbild der Umgebung ohne zusätzlich eingebrachte Informationen darstellen. Zur dreidimensionalen Rekonstruktion der Szene müssen in den verschiedenen Kamerabildern Korrespondenzen gefunden werden. Dies sind Punkte, die den gleichen Ort auf einem Objekt entsprechen.

Im Gegensatz dazu bezeichnet die aktive Stereoskopie diejenigen Verfahren, die zwar auf dem Stereo-Prinzip basieren, dieses aber durch eine strukturierte Lichtquelle als aktive Komponente unterstützen. Diese aktive Komponente bringt dabei Informationen in eine zu vermessende Szene, die eine Tiefenberechnung erleichtert. [Mordrow, 2008] beschreibt solche Systeme in ihren Arbeiten.

Das in dieser Arbeit verwendet stereoskopische Messsystem zur Bestimmung der Lage und Orientierung eines Objektes verfolgt den Ansatz einer passiven Stereoskopie, jedoch löst es das Korrespondenzproblem mit der Verwendung einer robusten und eindeutigen Markererkennung. In den folgenden Kapiteln werden die theoretischen Grundlagen beschrieben, um ein solches Messystem zu realisieren.

3.2 Kameraparameter

3.2.1 Intrinsische Kameraparameter

Intrinsische Kameraparameter beschreiben die innere Geometrie einer Kamera und sind in Abbildung 3.1 schematisch dargestellt. Zu sehen sind das Koordinatensystem des Kamerazentrums im Ursprung O und der sich im Raum befindliche Punkt P_W, welcher auf den Kamerasensor durch den Punkt P_B abgebildet wird.

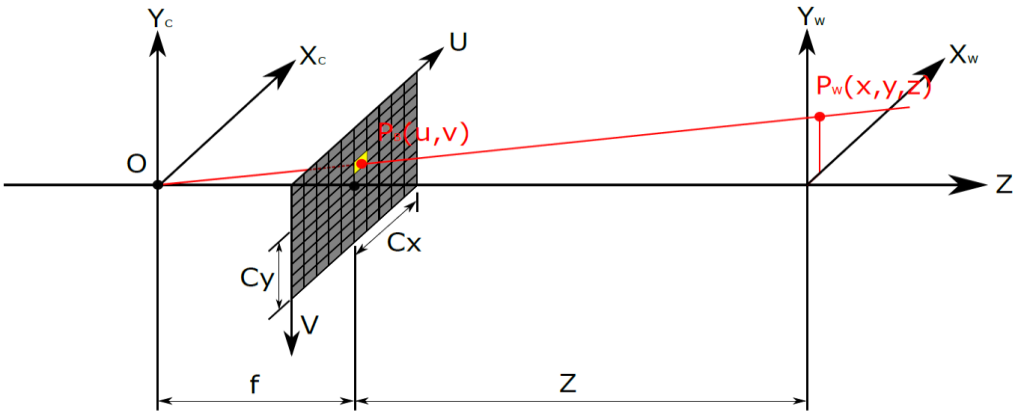


Abbildung 3.1: Schematische Darstellung der intrinsischen Kameraparameter [Zander,17]

Die Bildweite f definiert dabei den Abstand zwischen der Sensorfläche und dem Kamerazentrum, welches sich auf der Höhe des Sensormittelpunktes befindet. Die Abstände C_X und C_Y definieren den Abstand zwischen dem Ursprung des Sensors und dem Kameraursprung O . Wichtig zu erwähnen ist, dass sich vor allem bei nichtindustriellen Kameras die Bildweite f in x- und y-Richtung unterscheiden kann.

3.2.2 Extrinsische Kameraparameter

Die extrinsischen Parameter beschreiben die Lage und Orientierung eines betrachteten Kamerakoordinatensystems gegenüber einem Referenzkoordinatensystems. Im Falle der stereoskopischen Kameratechnik beschreiben sie die Transformation $T: \in \mathbb{R}^{4x4}$ der einen Kamera in das Koordinatensystem der anderen Kamera. Somit gilt nach Abbildung 3.2:

$$^{A}O_r = {}^{A}T_B \cdot {}^{B}O_r \tag{3.1}$$

Oder auch

$$^{B}O_l = {}^{B}T_A \cdot {}^{A}O_l \tag{3.2}$$

Wobei:

$$T = \begin{pmatrix} R & \vec{T} \\ 0 & 1 \end{pmatrix} mit\ \vec{T}: \in \mathbb{R}^3\ und\ R: \in \mathbb{R}^{3x3} \tag{3.3}$$

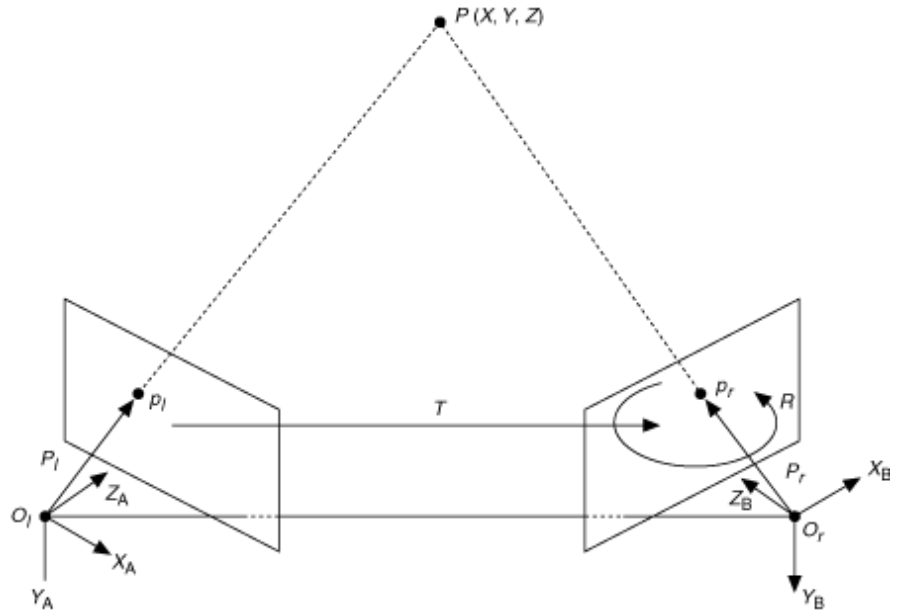


Abbildung 3.2: Extrinsische Kameraparameter

3.2.3 Verzeichnung

Der Abbildungsmaßstab von Linsen ist abhängig vom Abstand der Strahlen zur optischen Achse, d.h. der Abbildungsmaßstab für Punkte am Rand kann größer oder kleiner sein, wodurch das Bild verzerrt wird. Geraden in einem Bild werden daher zum Mittelpunkt hin- bzw. weggekrümmt, wodurch die sogenannte Kissen- bzw. Tonnenverzeichnung entsteht. (vgl. Abbildung 3.3)

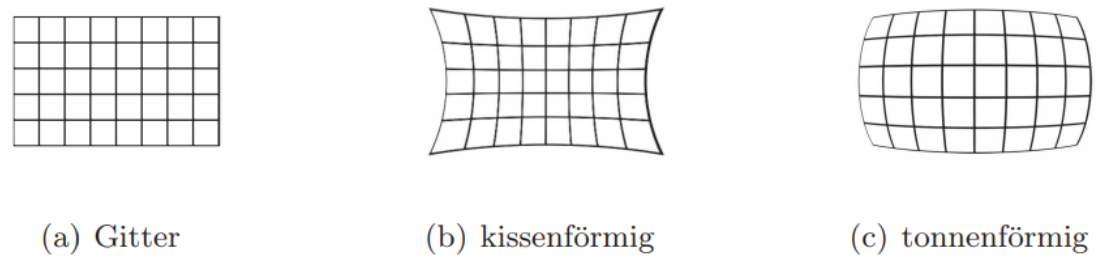
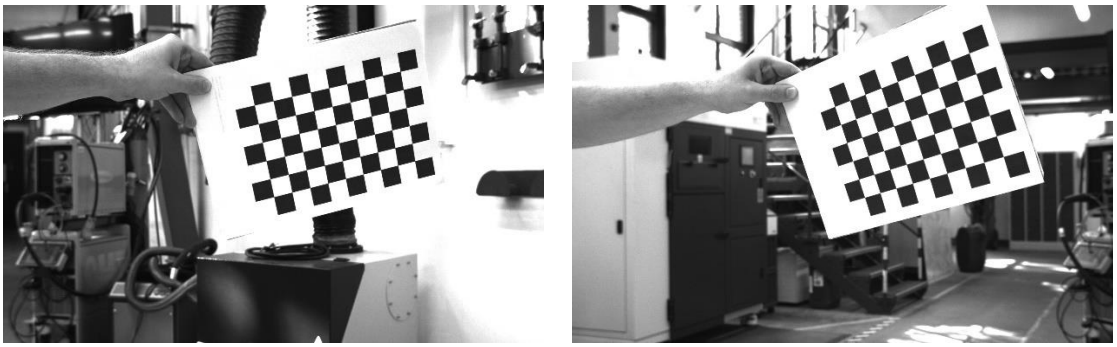


Abbildung 3.3: Gitter, kissenförmig und tonnenförmig verzeichnet [Graf, 2007, S.30]

3.2.4 Kamerakalibration

Den Prozess, die in Kapitel 3.1.1, 3.1.2 und 3.1.3 genannten Parameter zu bestimmen, wird Kamerakalibration genannt und basiert auf der Erkennung eines Vordefinierten Kalibriermusters. Diese Arbeit verwendet dafür ein Schachbrettmuster mit 10x7 Quadraten, welche eine Kantenlänge von 26,2mm besitzen. (vgl. Abbildung 3.4)



(a) Linke Kamera (b) Rechte Kamera

Abbildung 3.4: Stereoskopisches Bildpaar

In Theorie sind zur Bestimmung der relevanten Parameter nur zwei Aufnahmen nötig [Kaehler, Bradski, 2017, S.665f], OpenCV empfiehlt jedoch mindestens 10 Bilder zu verwenden, in denen das Kalibriermuster in möglichst unterschiedlichen Einstellungen platziert ist. (vgl. Abbildung 3.5)

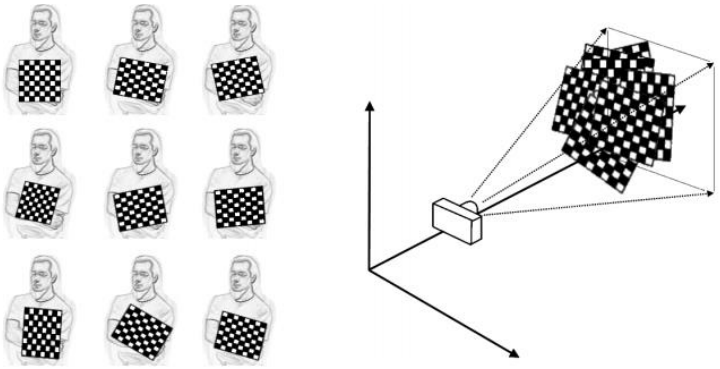


Abbildung 3.5: Variationen von Aufnahmen für die Kamerakalibrierung

[Kaehler, Bradski, 2017, S.653]

OpenCV bietet zum Kalibrieren von Kameras die Funktionen *cv::calibrateCamera()* und *cv::stereoCalibrate()*. Erstere kann sowohl die intrinsischen Kameraparameter, als auch die Verzeichnung der Linse einer einzelnen Kamera bestimmen, wohingegen die Stereokalibration die Verzeichnung der Linse, die intrinsischen Kameraparameter beider Kameras, und dessen extrinsische Parameter R und $\vec{T}$ bestimmen kann.

Die Ausgabe der Stereokalibration von OpenCV hat zur Folge, dass R und $\vec{T}$ gemäß Gleichung 3.4 ausgegeben werden.

$$^{B}O_r = {}^{B}T_A \cdot {}^{A}O_r \tag{3.4}$$

Für den Ansatz der in dieser Arbeit verwendeten Triangulation, ist jedoch eine Form gemäß Gleichung 3.1 gefordert. Die Umrechnung erfolgt über das Multiplizieren der Inversen der Transformationsmatrix T. Es ergibt sich:

$$^{B}T_A^{-1} \cdot {}^{B}O_r = {}^{B}T_A^{-1} \cdot {}^{B}T_A \cdot {}^{A}O_r \tag{3.5}$$

$$^{A}T_B \cdot {}^{B}O_r = {}^{A}O_r \tag{3.6}$$

$$^{A}O_r = {}^{A}T_B \cdot {}^{B}O_r \tag{3.7}$$

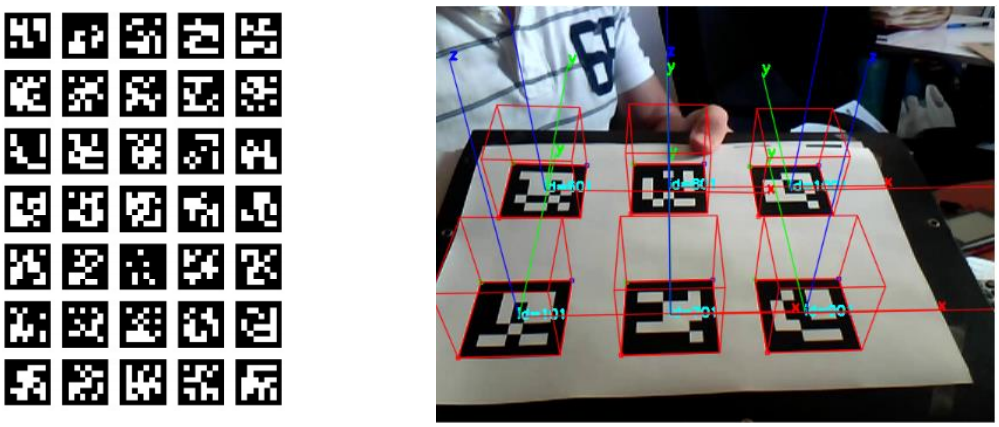
Die Transformationsmatrix $^{A}T_B$ enthält die geforderten Werte, um das Koordinatensystem der rechten Kamera innerhalb des Koordinatensystems der linken Kamera zu beschreiben (vgl. Kapitel 3.2.2).

Für den Prozess der Stereokalibration ist bei der Bildaufnahme zusätzlich darauf zu achten, dass das Kalibriermuster stets in beiden Kamerabildern vollständig zu erkennen ist. Für eine zufriedenstellende Kalibrierung hat sich eine Anzahl aus 40 Bildpaaren als ausreichend herausgestellt.

3.3 Markererkennung

Diese Arbeit verwendet zur Lösung des in Kapitel 3.1 angesprochenen Korrespondenzproblems der stereoskopischen Bildpaare die in OpenCV integrierbare Softwarebibliothek „ArUco“. [Munoz-Salinas]

ArUco ist eine OpenSource-Bibliothek zur Erkennung von quadratischen Referenzmarken in Bildern. Zusätzlich bietet sie neben der Möglichkeit der Kamerakalibration unter Verwendung eines ArUco-Brettes auch die Bestimmung der Pose. (siehe Abbildung 3.6).

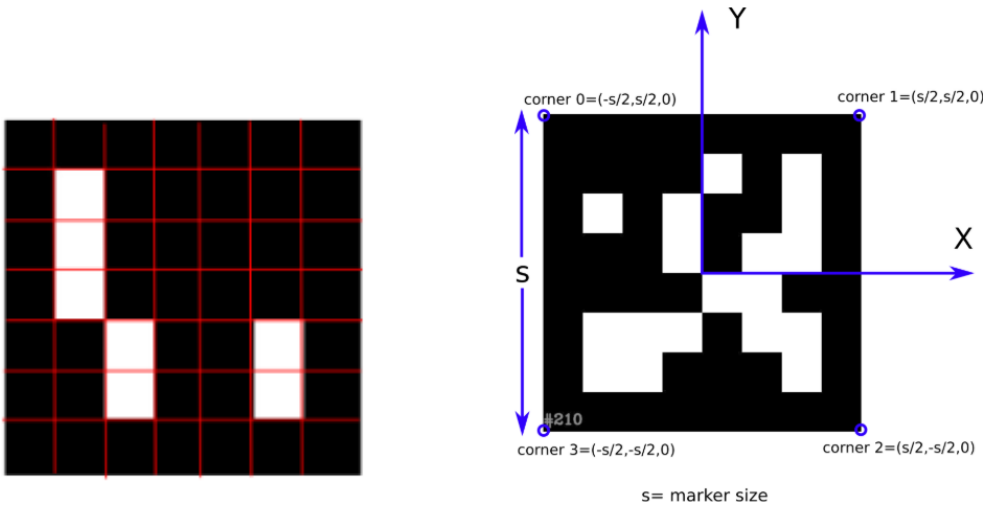


(a) ArUco-Brett zur Kamerakalibration (b) Erkennung der Markerpose

Abbildung 3.6: ArUco [Munoz-Salinas]

3.3.1 Marker

Die Marker bestehen aus einem äußeren schwarzen Rand und einem inneren Bereich, der als ein binäres Muster kodiert ist. (vgl. Abbildung 3.7 (a)) Das binäre Muster ist eindeutig und identifiziert jeden Marker. Je nachdem, welche Marker verwendet werden, gibt es Marker mit mehr oder weniger Bits. Je mehr Bits, desto mehr Marker können existieren und desto geringer die Wahrscheinlichkeit einer Verwechslung. Jedoch bedeuten mehr Bits auch, dass für eine korrekte Erkennung eine höhere Auflösung erforderlich ist. Intern ist jeder Marker als ein Vektor mit 4 Punkten gespeichert, aus denen dann, z.B. im Markerzentrum, die Pose berechnet werden kann. (vgl. Abbildung 3.7 (b))



(a) Marker-Kodierung (b) Marker-Aufbau

Abbildung 3.7: ArUco [Munoz-Salinas]

3.3.2 Detektion

Die Erkennung der ArUco-Marker folgt einem festen Schema. Zunächst wird, wie in Abbildung 3.8 (1) zu sehen, eine Kantendetektion durchgeführt, um die relevanten Marker von der Umgebung zu segmentieren. Diese basiert auf der Anwendung eines adaptiven Grenzwertes zur Binarisierung der Bildszene zusammen mit einer anschließenden Konturextraktion. [OpenCV ArUco] Als nächstes werden die gefundenen Konturen, die sich nicht zu einem Quadrat approximieren lassen, zu groß- oder klein oder auch zu nah aneinander sind, herausgefiltert (vgl. Abbildung 3.8)

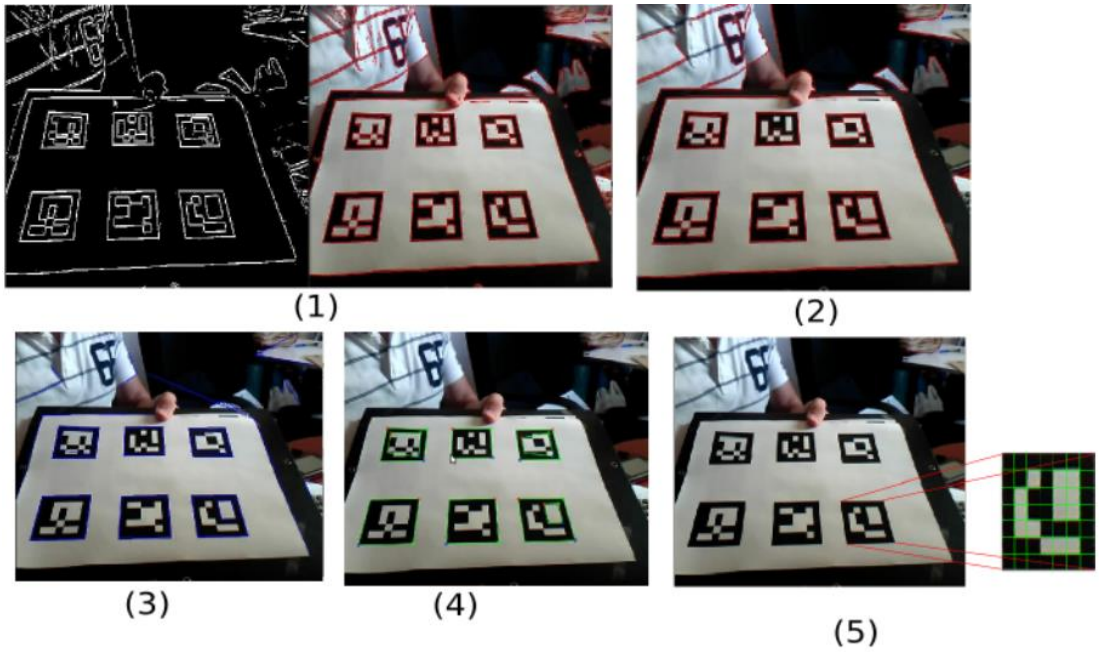


Abbildung 3.8: Ablauf der Markerdetektion [Munoz-Salinas]

Nach der Erkennung der möglichen Markerkandidaten ist es notwendig festzustellen, ob es sich tatsächlich um Marker handelt, indem ihre innere Kodifizierung analysiert wird. Dieser Schritt beginnt mit dem Extrahieren der Marker-Bits der einzelnen Marker. Dazu wird zunächst eine perspektivische Transformation durchgeführt, um den Marker in seiner kanonischen Form zu erhalten. Dann wird das kanonische Bild mit Otsu geschwächt, um weiße und schwarze Bits zu trennen. Das Bild wird, je nach Marker- und Randgröße, in verschiedene Zellen unterteilt, und die Anzahl der schwarzen oder weißen Pixel auf jeder Zelle werden gezählt, um festzustellen, ob es sich um ein weißes oder ein schwarzes Bit handelt. Schließlich werden die Bits analysiert, um festzustellen, ob der Marker zur spezifischen Wahl an vorgegebenen Markern gehört.

3.4 Herleitung der Triangulation eines Punktes im Raum bei gegebener Korrespondenz eines stereoskopischen Bildpaares

Dieser Abschnitt befasst sich mit der Herleitung der Triangulation eines Punktes im Raum. Hierbei wird zunächst die mathematische Grundlage hergeleitet und im Anschluss die konkrete Parametrisierung beschrieben.

3.4.1 Allgemeine Beschreibung

Für die Triangulation wird für jede Kamera eine Geradengleichung $\vec{g}$ angenommen, bei dem der Ortsvektor $\vec{o}$ im Kamerazentrum und der Richtungsvektor $\vec{p}$ durch dem auf dem Kamerasensor detektierten Pixel beschrieben wird. (vgl. Abbildung 3.2)
Somit ergeben sich folgende Geradengleichungen.

$$\overrightarrow{g_l} = \overrightarrow{o_l} + \lambda \cdot \overrightarrow{p_l} = \begin{pmatrix} a_1 \\ b_1 \\ c_1 \end{pmatrix} + \lambda \cdot \begin{pmatrix} D_1 \\ E_1 \\ F_1 \end{pmatrix} \tag{3.8}$$

$$\overrightarrow{g_r} = \overrightarrow{o_r} + \sigma \cdot \overrightarrow{p_r} = \begin{pmatrix} a_2 \\ b_2 \\ c_2 \end{pmatrix} + \sigma \cdot \begin{pmatrix} D_2 \\ E_2 \\ F_2 \end{pmatrix} \tag{3.9}$$

Um den in beiden Kameras entdeckten Korrespondenzpunkt P_W zu bestimmen, muss der Schnittpunkt von den aufgestellten Geradengleichungen ermittelt werden. Da es in der realen Messtechnik jedoch zu Ungenauigkeiten bei der Detektion kommen kann, muss mathematisch davon ausgegangen werden, dass die Geraden windschief zueinander verlaufen, sich also die Bedingung $\overrightarrow{g_l} = \overrightarrow{g_r}$ nie erfüllen wird. Daher wird der Ansatz verfolgt, zunächst den kürzesten Verbindungsvektor $\overrightarrow{F_{gl}F_{gr}}$ zwischen den Geradengleichungen $\overrightarrow{g_l}$ und $\overrightarrow{g_r}$ zu bestimmen, für den, da er orthogonal zu den jeweiligen Geraden verlaufen muss, gilt:

$$\overrightarrow{F_{gl}F_{gr}} \cdot \overrightarrow{p_l} = 0 \tag{3.10}$$

$$\overrightarrow{F_{gl}F_{gr}} \cdot \overrightarrow{p_r} = 0 \tag{3.11}$$

Das Gleichungssystem aus Gleichung 3.10 und 3.11 führt zu einer Lösung für die Variablen λ und σ der Geraden $\overrightarrow{g_l}$ und $\overrightarrow{g_r}$. Der Punkt, der sich dann auf der Hälfte des Verbindungsvektors $\overrightarrow{F_{gl}F_{gr}}$ befindet, ergibt sich in dieser Arbeit dann zu dem entsprechenden Korrespondenzpunkt P.

Es folgt die ausformulierte Herleitung der beschriebenen Zusammenhänge.

1. Erstellen des Verbindungsvektors $\overrightarrow{F_{gl}F_{gr}}$:

$$\overrightarrow{F_{gl}F_{gr}} = \overrightarrow{g_r} - \overrightarrow{g_l} = \begin{pmatrix} a_2 + \sigma \cdot D_2 \\ b_2 + \sigma \cdot E_2 \\ c_2 + \sigma \cdot F_2 \end{pmatrix} - \begin{pmatrix} a_1 + \lambda \cdot D_1 \\ b_1 + \lambda \cdot E_1 \\ c_1 + \lambda \cdot F_1 \end{pmatrix} \tag{3.12}$$

2. Aus den Orthogonalitätsbedingungen entwickelt sich das zu lösende Gleichungssystem zu:

$$\overrightarrow{F_{gl}F_{gr}} \cdot \overrightarrow{p_l} = \begin{pmatrix} a_2 + \sigma \cdot D_2 - a_1 - \lambda \cdot D_1 \\ b_2 + \sigma \cdot E_2 - b_1 - \lambda \cdot E_1 \\ c_2 + \sigma \cdot F_2 - c_1 - \lambda \cdot F_1 \end{pmatrix} \cdot \begin{pmatrix} D_1 \\ E_1 \\ F_1 \end{pmatrix} = 0 \tag{3.13}$$

$$\overrightarrow{F_{gl}F_{gr}} \cdot \overrightarrow{p_r} = \begin{pmatrix} a_2 + \sigma \cdot D_2 - a_1 - \lambda \cdot D_1 \\ b_2 + \sigma \cdot E_2 - b_1 - \lambda \cdot E_1 \\ c_2 + \sigma \cdot F_2 - c_1 - \lambda \cdot F_1 \end{pmatrix} \cdot \begin{pmatrix} D_2 \\ E_2 \\ F_2 \end{pmatrix} = 0 \tag{3.14}$$

3. Durch das Ausmultiplizieren erhält man folgende Gleichungen:

$$\begin{aligned} \overrightarrow{F_{gl}F_{gr}} \cdot \overrightarrow{p_l} = \sigma(D_2D_1 + E_2E_1 + F_2F_1) + \lambda(-D_1D_1 - E_1E_1 - F_1F_1) \\ + D_1(a_2 - a_1) + E_1(b_2 - b_1) + F_1(c_2 - c_1) = 0 \end{aligned} \tag{3.15}$$

$$\begin{aligned} \overrightarrow{F_{gl}F_{gr}} \cdot \overrightarrow{p_r} = \sigma(D_2D_2 + E_2E_2 + F_2F_2) + \lambda(-D_1D_2 - E_1E_2 - F_1F_2) \\ + D_2(a_2 - a_1) + E_2(b_2 - b_1) + F_2(c_2 - c_1) = 0 \end{aligned} \tag{3.16}$$

4. Substituieren mit:

$$\alpha = (D_2D_1 + E_2E_1 + F_2F_1) \tag{3.17}$$

$$\beta = (-D_1D_1 - E_1E_1 - F_1F_1) \tag{3.18}$$

$$\gamma = (D_2D_2 + E_2E_2 + F_2F_2) \tag{3.19}$$

$$\delta = (-D_1D_2 - E_1E_2 - F_1F_2) \tag{3.20}$$

$$T_1 = D_1(a_2 - a_1) + E_1(b_2 - b_1) + F_1(c_2 - c_1) \tag{3.21}$$

$$T_2 = D_2(a_2 - a_1) + E_2(b_2 - b_1) + F_2(c_2 - c_1) \tag{3.22}$$

Resultiert in folgendes Gleichungssystem:

$$\overrightarrow{F_{gl}F_{gr}} \cdot \overrightarrow{p_l} = \sigma \cdot \alpha + \lambda \cdot \beta + T_1 = 0 \tag{3.23}$$

$$\overrightarrow{F_{gl}F_{gr}} \cdot \overrightarrow{p_r} = \sigma \cdot \gamma + \lambda \cdot \delta + T_2 = 0 \tag{3.24}$$

5. Zum Lösen des nun gegebenen Gleichungssystems erfolgt in dieser Arbeit eine Umwandlung in Matrixschreibweise:

$$\begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix} \cdot \begin{pmatrix} \sigma \\ \lambda \end{pmatrix} = \begin{pmatrix} -T_1 \\ -T_2 \end{pmatrix} \tag{3.25}$$

Die Umstellung des Gleichungssystems nach $\begin{pmatrix} \sigma \\ \lambda \end{pmatrix}$ ergibt:

$$\begin{pmatrix} \sigma \\ \lambda \end{pmatrix} = \begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix}^{-1} \cdot \begin{pmatrix} -T_1 \\ -T_2 \end{pmatrix} \tag{3.26}$$

$$\begin{pmatrix} \sigma \\ \lambda \end{pmatrix} = \frac{1}{\alpha \cdot \delta - \beta \cdot \gamma} \cdot \begin{pmatrix} \delta & -\beta \\ -\gamma & \alpha \end{pmatrix} \cdot \begin{pmatrix} -T_1 \\ -T_2 \end{pmatrix} \tag{3.27}$$

6. Substituiert man nun $L = \frac{1}{\alpha * \delta - \beta * \gamma}$ und stellt σ und λ auf, so erhält man folgende Gleichungen:

$$\sigma = L \cdot (\delta \cdot (-T_1) + (-\beta) \cdot (-T_2)) \tag{3.28}$$

$$\lambda = L \cdot ((-\gamma) \cdot (-T_1) + \alpha \cdot (-T_2)) \tag{3.29}$$

7. Nach Gleichung 3.30 und 3.31 können nun die Weltpunkte der Geraden $\overrightarrow{g_l}$ und $\overrightarrow{g_r}$ berechnet werden, um den gesuchten Korrespondenzpunkt P_W zu bestimmen:

$$\overrightarrow{g_l} = \begin{pmatrix} a_1 \\ b_1 \\ c_1 \end{pmatrix} + \lambda \cdot \begin{pmatrix} D_1 \\ E_1 \\ F_1 \end{pmatrix} = \begin{pmatrix} Gl_x \\ Gl_y \\ Gl_z \end{pmatrix} \tag{3.30}$$

$$\overrightarrow{g_r} = \begin{pmatrix} a_2 \\ b_2 \\ c_2 \end{pmatrix} + \sigma \cdot \begin{pmatrix} D_2 \\ E_2 \\ F_2 \end{pmatrix} = \begin{pmatrix} Gr_x \\ Gr_y \\ Gr_z \end{pmatrix} \tag{3.31}$$

$$P_W = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \overrightarrow{g_l} - \frac{\overrightarrow{g_l} - \overrightarrow{g_r}}{2} \tag{3.32}$$

3.4.2 Gleichungsparameter

Dieser Abschnitt beschreibt, wie die Parameter der Gleichungen 3.8 und 3.9 zustande kommen.

Diese Arbeit verfolgt den Ansatz, dass die Triangulation im Koordinatensystem der linken Kamera erfolgt. Demnach gilt für den Ortsvektor $\overrightarrow{o_l}$ der Gerade $\overrightarrow{g_l}$:

$$\overrightarrow{o_l} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \tag{3.33}$$

Der Richtungsvektor $\overrightarrow{p_l}$ der Gerade $\overrightarrow{g_l}$ ergibt sich aus den in Kapitel 3.2.1 beschriebenen intrinsischen Kameraparametern. Aus der Betrachtung des Koordinatensystems der linken Kamera befindet sich der auf dem Kamerasensor detektierte Punkt $P_B = \begin{pmatrix} u \\ v \end{pmatrix}$ (vgl. Abbildung 3.1) auf der Entfernung f. Zusätzlich muss die ermittelte Verzeichnung der Linse aus den detektierten Pixeln u und v herausgerechnet werden. Damit ergibt sich für $P'_B = \begin{pmatrix} u' \\ v' \end{pmatrix}$. Zuletzt sind die P'_B Pixel noch mit den kameraspezifischen Versatzwerten C_X und C_Y zu beaufschlagen. Somit ergibt sich für $\overrightarrow{g_l}$:

$$\overrightarrow{g_l} = \overrightarrow{o_l} + \lambda \cdot \overrightarrow{p_l} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + \lambda \cdot \begin{pmatrix} u'_l - C_{X_l} \\ v'_l - C_{Y_l} \\ f_l \end{pmatrix} \tag{3.34}$$

Fast analog zu der Beschreibung von $\overrightarrow{g_l}$ ergibt sich $\overrightarrow{g_r}$. Der Unterschied liegt in der Verrechnung der in Kapitel 3.2.1 beschriebenen Transformation. Somit definiert sich $\overrightarrow{g_r}$ als:

$$\overrightarrow{g_r} = \overrightarrow{o_r} + \sigma \cdot \overrightarrow{p_r} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} + \vec{T} + \sigma \cdot R \cdot \begin{pmatrix} u'_r - C_{X_r} \\ v'_r - C_{Y_r} \\ f_r \end{pmatrix} \tag{3.35}$$

3.5 Berechnung einer Markerpose aus 3 Punkten

Die Pose ist die Kombination von Position und Orientierung im dreidimensionalen Raum.
Die Position eines Punktes im Raum in Relation zu einem kartesischen Koordinatensystem definiert sich demnach durch die Abstände entlang den Koordinatenrichtungen X, Y und Z. Spannt man in diesem Punkt ein zweites kartesisches Koordinatensystem aus, so definiert sich die Orientierung durch den Winkelversatz seiner Koordinatenachsen in Bezug zu den entsprechenden Achsen des Basiskoordinatensystems. Angegeben kann die Pose beispielsweise in Homogener Form, wie in Gleichung 3.3, oder auch in Vektorschreibweise. Hierfür muss die Rotationsmatix R jedoch noch in entsprechende Euler-Winkel umgerechnet werden.

$$\text{Pose mit Euler-Winkel: } P = \begin{pmatrix} \vec{T} \\ \vec{R}_{Euler} \end{pmatrix} mit\ \vec{T} = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} und\ R_{Euler} = \begin{pmatrix} A \\ B \\ C \end{pmatrix} \qquad (3.36)$$

Homogene Beschreibungen der Pose sind vor allem durch ihre kompakte Notation als lineare Abbildung und der erleichterten Programmierung von Umrechnungen zwischen Koordinatensystemen für 3-dimensionale Anwendungen geeignet. [Siciliano, Khatib, 2008, s.13]
Durch die Detektion von mindestens 3 Eckpunkten eines ArUco-Markers im Raum (vgl. Kapitel 3.3) ist es möglich seine Pose zu bestimmen.
Wird zum Beispiel die in Abbildung 3.9 markierte Ecke „3“ als Basis der zu bestimmenden Pose festgelegt, so ergibt sich der entsprechende Translationsvektor als Raumkoordinate der entsprechenden Markerecke und es gilt $\vec{T} = P_{Wcorner3}$.
Um die Berechnung der Orientierung des Koordinatensystems auf dem Marker durchzuführen, ist eine gewünschte Ausrichtung des Koordinatensystems mit dem Ursprung in der Markerecke „3“ im Vorhinein festzulegen. In den folgenden Schritten wird gezeigt, wie ein Koordinatensystem gemäß Abbildung 3.9 erstellt wird.

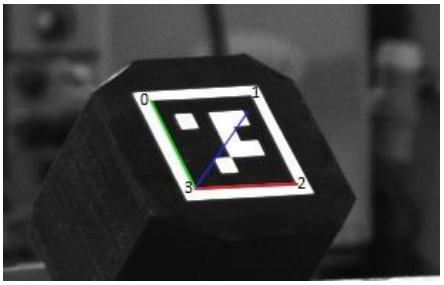


Abbildung 3.9: Beispielhaftes Koordinatensystem auf dem ArUco-Marker

Zunächst werden die Richtungsvektoren der X- und Y-Achse aufgestellt:

$$\vec{x} = P_{Wcorner2} - P_{Wcorner3} \tag{3.37}$$

$$\vec{y} = P_{Wcorner0} - P_{Wcorner3} \tag{3.38}$$

Diese Richtungsvektoren bilden gemeinsam eine Ebene, zu der die bislang fehlende Z-Achse orthogonal verläuft. Die Richtung der Z-Achse lässt sich demnach durch das Kreuzprodukt der X- und Y-Achse bestimmen:

$$\vec{z} = \vec{x} \times \vec{y} \tag{3.39}$$

Aus den berechneten Richtungsvektoren lässt sich bislang noch keine Rotationsmatrix zusammenstellen, da noch nicht sichergestellt ist, dass alle Vektoren orthogonal zueinander verlaufen. Für die Kompensation der Fehllage wird in dieser Arbeit allein die Lage der Y-Achse durch erneute Kreuzproduktbildung zwischen der zuvor berechneten Z-Achse und der X-Achse korrigiert.

Die resultierenden Richtungsvektoren der jeweiligen Achsen müssen normiert werden, und können dann in Form der Gleichung 3.41 zu einer Rotationsmatrix zusammengefügt werden.

$$\vec{x} = \frac{\vec{x}}{\|\vec{x}\|}, \vec{y} = \frac{\vec{y}}{\|\vec{y}\|}, \vec{z} = \frac{\vec{z}}{\|\vec{z}\|} \tag{3.40}$$

$$R = \begin{pmatrix} r_{xx} & r_{yx} & r_{zx} \\ r_{xy} & r_{yy} & r_{zy} \\ r_{xz} & r_{yz} & r_{zz} \end{pmatrix} \tag{3.41}$$

Aus den somit ermittelten Informationen kann eine Beschreibung der Pose in Homogenen Koordinaten bereits erfolgen (vgl. Gleichung 3, s.X), wohingegen für die Bestimmung der Pose über Euler-Winkel noch die konkreten Winkel zu bestimmen sind.

Es muss festgelegt sein, nach welcher Konvention die Berechnung erfolgen soll, da aufeinanderfolgende Rotationen um die jeweiligen Achsen nicht kommutativ sind und zu verschiedenen Endstellungen führen, wie in Abbildung 3.10 dargestellt.

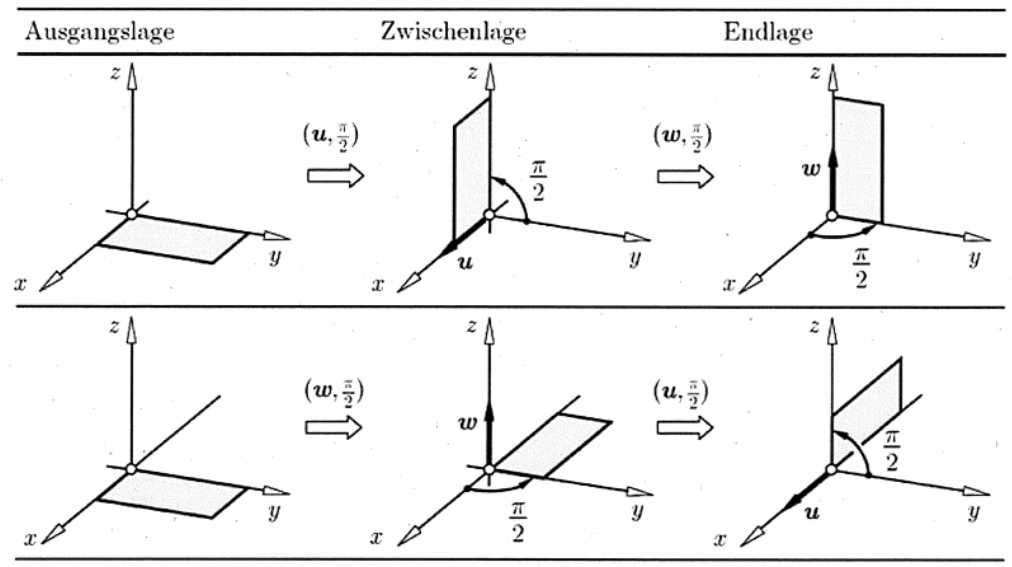


Abbildung 3.10: Nichtkommutativität von Rotationen [Woernle, 2011,S. 62]

Für die Verwendung der Kuka RSI Funktionalität zum Ansteuern des Roboters über Kartesische Koordinaten, ist für die Angabe der Rotation entweder eine Drehreihenfolge nach z-y‘-x‘‘ um die mitrotierende-, oder eine x-y‘-z‘‘ Drehfolge um raumfeste Achsen zu verwenden. Diese Arbeit hat die Drehkonvention nach z-y‘-x‘‘ realisiert, was also bedeutet, dass sich die Orientierung des untersuchten Objekts zuerst entlang der Z-Achse, dann um die Y-Achse und letztlich um die X-Achse verdreht wobei die in Abbildung 3.11 geltende Drehrichtung für Rechthandsysteme gilt.

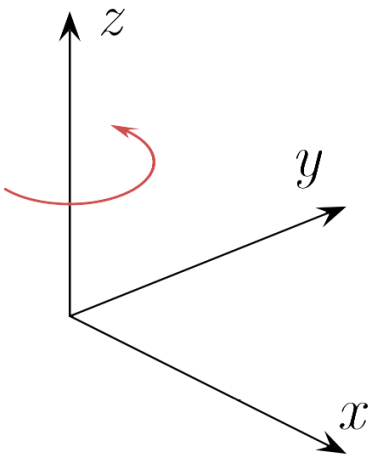


Abbildung 3.11: Rechthand-Koordinatensystem

Gemäß der z-y'-x'' Konvention ergibt sich folgende Gesamtrotationsmatrix:

$$R = R_z(A) \cdot R_y(B) \cdot R_x(C) \tag{3.42}$$

$$R = \begin{pmatrix} \cos A & -\sin A & 0 \\ \sin A & \cos A & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos B & 0 & \sin B \\ 0 & 1 & 0 \\ -\sin B & 0 & \cos B \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos C & -\sin C \\ 0 & \sin C & \cos C \end{pmatrix} \tag{3.43}$$

$$R = \begin{pmatrix} \cos B \cos A & \sin C \sin B \cos A - \cos C \sin A & \cos C \sin B \cos A + \sin C \sin A \\ \cos B \sin A & \sin C \sin B \sin A + \cos C \cos A & \cos C \sin B \sin A - \sin C \cos A \\ -\sin B & \sin C \cos B & \cos C \cos B \end{pmatrix} \tag{3.44}$$

Die entsprechenden Euler-Winkel können dann durch die folgenden Gleichungen gelöst werden. [Siciliano, Khatib, 2008, S.13]

$$R = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} \tag{3.45}$$

$$B = Atan2\left(-r_{31}, \sqrt{{r_{11}}^2 + {r_{21}}^2}\right) \tag{3.46}$$

$$A = Atan2\left(\frac{r_{21}}{\cos B}, \frac{r_{11}}{\cos B}\right) \tag{3.47}$$

$$C = Atan2\left(\frac{r_{32}}{\cos B}, \frac{r_{33}}{\cos B}\right) \tag{3.48}$$

Ist $B = \pm\frac{\pi}{2}$, treten Singularitäten, sogenannte Gimbal Locks, auf, die sich darin äußern, dass es in diesen Fällen für A und C unendlich viele Lösungen gibt. In diesem Falle der bereits erwähnten Singularitäten dagegen sind folgende Formeln zweckmäßig [Carig, S.41f.]:

Wenn $B = \frac{\pi}{2}$:

$$A = 0 \tag{3.49}$$

$$C = Atan2(r_{12}, r_{22}) \tag{3.50}$$

Wenn $B = -\frac{\pi}{2}$:

$$A = 0 \tag{3.51}$$

$$C = -Atan2(r_{12}, r_{22}) \tag{3.52}$$

3.6 Kalmanfilterung

Das Kalman-Filter ist ein Filter, welches den Zustand eines dynamischen Systems anhand von Messungen optimal abschätzt. Die Grundidee ist, dass wegen ungenauer Messungen der wahre Systemzustand immer unbekannt ist und daher abgeschätzt werden muss. Das Kalman-Filter macht sich bei der Schätzung des Zustandes zunutze, dass die Erwartungswerte sowohl für das System, als auch für das Messrauschen bekannt sind und liefert unter Berücksichtigung dieses Rauschens einen optimalen Schätzwert für den Systemzustand (vgl. Abbildung 3.12)

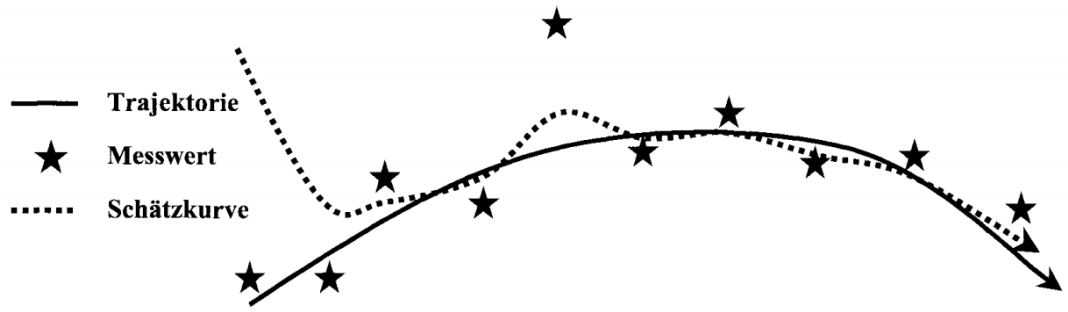


Abbildung 3.12: Die Grundidee des Kalman-Filters: Vorhersage der Trajektorie von Bewegungsgleichungen und Korrektur der Trajektorie mit jeder neuen Messung. [Nischwitz et al., 2011]

Entwickelt wurde das Kalman-Filter in 1960 vom gleichnamigen Dr. Rudolf Emil Kalman in seiner Veröffentlichung „A New Approach to Linear Filtering and Prediction Problems“. Die Arbeit hatte einen bedeutsamen Einfluss auf das Feld der linearen Filterung und hat durch seine Effizienz in der Schätzung von dynamischen Zuständen ein großes Feld von Anwendungen in der Welt der Technik erobern können. [A. McGee, F. Schmidt, 1985] Namentlich wurde z.B. das Kalman-Filter als mathematische Methode der Apollo-Mondlandung verwendet, um dort im Hauptprogramm das Navigationssystem zu steuern.

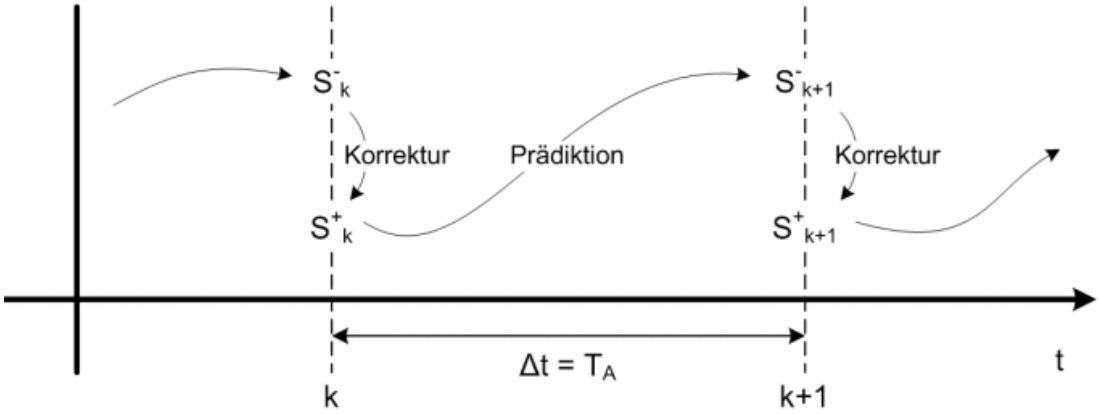


Abbildung 3.13 Berechnungssequenz des Kalman-Filters mit logischer zeitlicher Zuordnung der Zustände s_k sowie realer zeitlicher Zuordnung der Berechnungsvorgänge. [Peters, 2009, s.41]

Ausgehend von einem bekannten Startzustand s_0 führt das Kalman-Filter abwechselnd die Arbeitsschritte Prädiktion und Korrektur durch. Abbildung 3.13 dient der Veranschaulichung der Transitionen und Zustände, auf die sich der mathematische Ablauf in Abbildung 3.14 bezieht. Die **Prädikation** berechnet auf Basis des ausgelegten Systemmodells und dem aktuellen Zustand s_k den voraussichtlichen Folgezustand s_{k+1}^-. (das “–” bedeutet hier a *priori*). Die **Korrektur** wird bei einer Messung des Systemzustandes durchgeführt und wandelt den s_{k+1}^- *priori* – Zustand in einen s_{k+1}^+ post*priori* – Zustand. Wegen dieser rekursiven Arbeitsweise muss das Kalman-Filter stets nur den aktuellen Systemzustand speichern. Daraus ergibt sich ein geringer Speicherbedarf sowie geringer Rechenaufwand, was den Kalman-Filter besonders für den Einsatz in Echtzeitsystemen eignet.

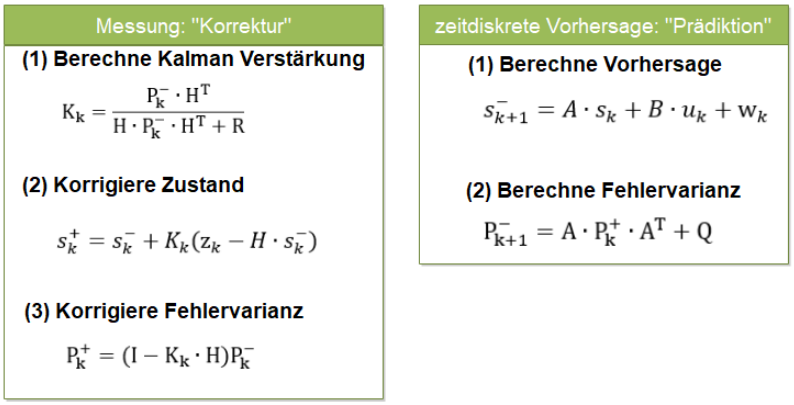


Abbildung 3.14: Berechnungsschritte des Kalman-Filters

Im Allgemeinen kann ein zeitdiskretes dynamisches System durch seine Systemgleichung und seine Messgleichung beschrieben werden. Die Systemgleichung beschreibt den Zustandsübergang des Systemzustandes s_k zum Zeitpunkt k auf den Systemzustand s_{k+1}^+ zum Zeitpunkt $k+1$. Die Systemmatrix A bildet den Systemzustand auf den Folgezustand ab, ohne dabei äußere Einflüsse u_k, wie etwa Regeleingriffe, zu berücksichtigen. Solche äußeren Einflüsse werden durch die Eingabematrix B auf den Systemzustand abgebildet. Die Zufallsvariable w_k repräsentiert das Systemrauschen. Daraus ergibt sich die Systemgleichung, welche auch den Prädiktionsschritt (1) (vgl. Abbildung 3.14) des Kalman-Filters beschreibt. Der Prädikationsschritt (2) ist anschließend dafür verantwortlich, die Systeminterne Schätzfehlervarianz P_k zu berechnen. Die Schätzfehlervarianz ist Bestandteil der Berechnung der Korrektur und kann durch das Prozessrauschen Q eingestellt werden.

Für die Korrektur wird in Schritt (1) zunächst die Kalman-Verstärkung berechnet. Diese verarbeitet die aktuelle Schätzfehlervarianz des Systems in Abhängigkeit der Messwertmatrix H und der Messfehlerkovarianz R. im zweiten Schritt wird der *priori* Zustand s_k^- um die Kalman-verstärkte Differenz der aktuellen Messung z_k und der Messprädiktion korrigiert. Die aktuelle Messung z_k ergibt sich dabei aus:

$$z_k = H \cdot s_k + v_k \tag{4.1}$$

Wobei v_k für das Messrauschen ist.

Im letzten Schritt wird die Schätzfehlervarianz des Systems neu berechnet.

4 Anforderungsanalyse

Die Robotersteuerung erwartet vom Sensorsystem eine Rückmeldung im Zyklustakt von $4ms$ oder $12ms$. Dies hat zur Folge, dass das zu realisierende Sensorsystem in der Lage sein muss, innerhalb dieser Zykluszeit zu antworten. Bei maximaler Auflösung können die verwendeten Kameras jedoch nur eine Bildwiederholrate von 50 FPS (Frames Per Second) erreichen. Demnach kann das Sensorsystem im besten Fall alle $20ms$ einen neuen Messwert bereitstellen, sollte die Berechnung der Pose des zu entdeckenden Objektes vor Eintreffen des nächsten Bildes abgeschlossen sein.

Für die Minimierung der Eingabeverzerrung, von Änderung der Objektpose zu Roboterbewegung, muss also zum einen ein Messwertinterpolationsalgorithmus erarbeitet werden, der unabhängig vom Bildverarbeitungsprozess in Echtzeit berechenbar und abrufbar ist. Zum anderen muss die Bildverarbeitung, neben der Realisierung einer hinreichend genauen Messung, in der Lage sein, 50 Bilder pro Sekunde zu analysieren.

Der Begriff „Genauigkeit“ umfasst dabei den relativen Messwert der erfassten Objektpose um ein von der Kamera lokalisiertes Referenzkoordinatensystem und nicht die absolute Genauigkeit der Objektpose zum Kamerakoordinatensystem.

Das Ziel dieser Arbeit liegt in der Realisierung einer echtzeitfähigen Trajektorievorgabe für den Roboter. Daher müssen auch verschiedene Sicherheitsmaßnahmen wie die Begrenzung des Arbeitsraumes, oder das Abschalten der Roboterbewegung bei nicht-Detektion der Markerpose über einen gewissen Sicherheitszeitraum, und bei mangelnder Kommunikationsgeschwindigkeit vom Sensorsystem, realisiert werden.

5 Kommunikation der Systemkomponenten

Dieses Kapitel beschäftigt sich mit der Kommunikation zwischen den Systemkomponenten und deren Realisierungskonzepte. Dabei wird zunächst ein Konzept für die Realisierung der Kommunikation des Sensorsystems auf Basis der Anforderungen der Kuka-Sensorzykluszeit vorgestellt. Im Anschluss wird erklärt, welche Einstellungen für die Konfiguration der RSI-Schnittstelle in der Robotersteuerung notwendig sind.

5.1 Sensorsystem

Die Herausforderung der Echtzeitlageregelung des Robotermanipulators liegt in der Bereitstellung von akkuraten Korrekturwerten, da bekannt ist, dass das Sensorsystem nicht in der Lage ist, eigenständig innerhalb der geforderten Zykluszeit der Robotersteuerung eine neue Messung zu liefern. Im Rahmen dieser Arbeit wird daher das in Kapitel 3.6 vorgestellte Verfahren der Kalmanfilterung auf die Herausforderung der Interpolation von Messwerten erprobt, indem die Vorhersage der Objektpose auf die Anfrage der Robotersteuerung und die Korrektur auf die Berechnung eines neuen Bildpaares erfolgt. Abbildung 5.1 visualisiert die hardwarebedingten zeitlichen Randbedingungen mit den dazugehörigen Berechnungsschritten des Kalman Filter.

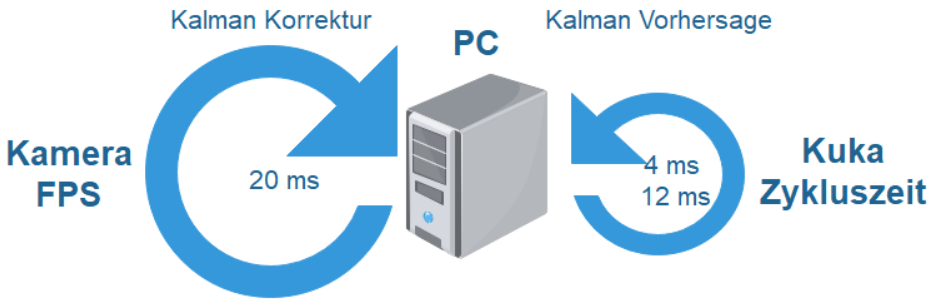


Abbildung 5.1: Konzept der Messwertinterpolation mittels Kalman Filter

Das Kalmanmodell ist hierbei in Zusammenarbeit mit dem Herrn Prof. Jochen Maaß entstanden, indem für eine Reduzierung der Berechnungszeit jeder der sechs Freiheitsgrade der Objektpose in einem eigenen Kalman Filter berechnet werden soll. Bei der Berechnung der Orientierung in Euler-Winkel kann es jedoch für den Kalman Filter zu drastischen Messwertänderungen bei einer nur minimal veränderten Orientierung aufgrund von Vorzeichenwechseln wie Beispielsweise $\pm 180°$ kommen. Der Kalman-Filter würde dies als eine Winkeländerung von $360°$ auffassen und den entsprechenden Freiheitsgrad nachziehen, was in der Realität fatale Auswirkungen auf die Roboterlage zur Folge hätte. Aus diesem Grund werden, statt der Euler-Winkel selbst, ihre Real- und Imaginäranteile in wiederum einzelnen Kalman-Filtern berechnet, und im Anschluss wieder in Euler-Winkel transformiert. Es ergeben sich somit 9 zu implementierende Kalman-Filter (drei für Translation, 6 für Rotation), welche in eindimensionaler Formulierung jeweilig den Kalmanzustand s (Gl. 5.1) mit der Position x und der Geschwindigkeit x besitzen.

$$s = \begin{bmatrix} x \\ \dot{x} \end{bmatrix} = \begin{matrix} Position \\ Geschwindigkeit \end{matrix} \tag{5.1}$$

Die regelungstechnische Auslegung soll nicht Inhalt dieser Arbeit sein, es ist jedoch eine ausreichend funktionierende Prozesskette des Konzeptes zu realisieren, deren Optimierung Bestandteil weiterer Arbeiten sein kann. Als solches werden die Transformationsmatrix A (Gl. 5.2), die Messmatrix H (Gl. 5.3), die Prozesskovarianzmatrix Q (Gl. 5.4) und die Messrauschkovarianzmatrix R (Gl. 5.5) vorgegeben.

$$A = \begin{pmatrix} 1 & dT \\ 0 & 1 \end{pmatrix} \tag{5.2}$$

$$H = \begin{pmatrix} 1 & 0 \end{pmatrix} \tag{5.3}$$

$$Q = \begin{pmatrix} 0 & 0 \\ 0 & Qvalue \end{pmatrix} \tag{5.4}$$

$$R = Rvalue \tag{5.5}$$

Die Funktionsweise des Kalman-Filters ist vor allem von dem Verhältnis der Mess- und Prozesskovarianzmatrix, also den Parametern $Qvalue$ und $Rvalue$, abhängig. Je höher das Verhältnis, desto schneller reagiert das Filter auf die Änderung der Messwerte und andersrum.

5.2 Kommunikation mit der Robotersteuerung

5.2.1 Netzwerkkonfiguration

Um eine Verbindung des Sensorsystems mit der Robotersteuerung aufzubauen, muss zunächst auf der Robotersteuerung eine entsprechende Ethernet-Schnittstelle eingerichtet werden[KUKA Roboter GmbH, 2016, S.29f] Hierfür muss in den Netzwerkkonfigurationen der Robotersteuerung (vgl. Abbildung 5.2) ein neues Interface hinzugefügt werden. Für das RSI notwendig ist die Wahl des Adresstypen „Gemischte IP-Adresse“, wodurch das Netzwerk für einen fast beliebigen Adressraum und der Verwendung von UDP als Kommunikationsprotokoll konfiguriert werden kann. Der IP-Adressbereich „192.168.0.x“ als auch die, falls vorhanden, anderen KLI-Subnetze dürfen hierbei nicht verwendet werden.

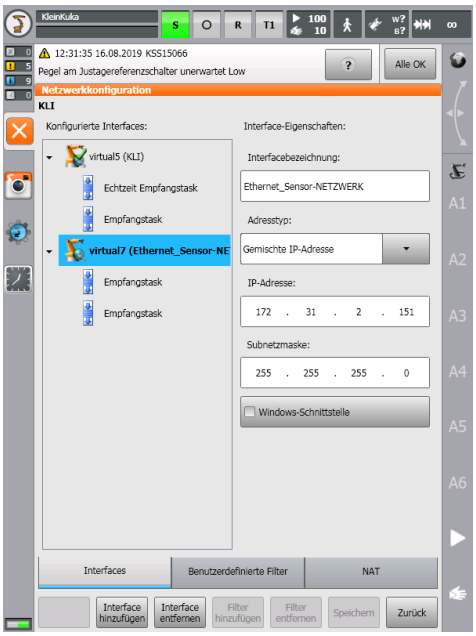


Abbildung 5.2: Netzwerkkonfiguration – IP Adresse

Des Weiteren ist sicherzustellen, dass ein entsprechender Port der Robotersteuerung für die Kommunikation über UDP eingerichtet ist. Die dafür notwendigen Einstellungen sind ebenfalls in den Netzwerkkonfigurationen in dem Unterpunkt „NAT“ konfigurierbar.

5.2.2 Testserver-Kontrolle

Um eine erfolgreiche Konfigurierung der Robotersteuerung zu testen, empfiehlt es sich die Testserverapplikation zusammen mit der Ethernet Beispielkonfiguration zu verwenden, welche vom RSI-Technologiepaket mitgeliefert werden. Als Voraussetzung für den Aufbau einer erfolgreichen Kommunikation, muss der jeweilige externe PC mit der in den Konfigurationsdateien hinterlegten Netzwerkeinstellungen eingerichtet und die entsprechenden Beispielkonfigurationen der Robotersteuerung im Verzeichnis *C:\KRC\ROBOTER\Config\User\Common\SensorInterface* abgelegt sein.
Abbildung 5.3 zeigt die Testserveranwendung im laufenden Betrieb, eine detaillierte Beschreibung der einzelnen Schaltflächen erfolgt in [KUKA Roboter GmbH, 2016, S.97f.]

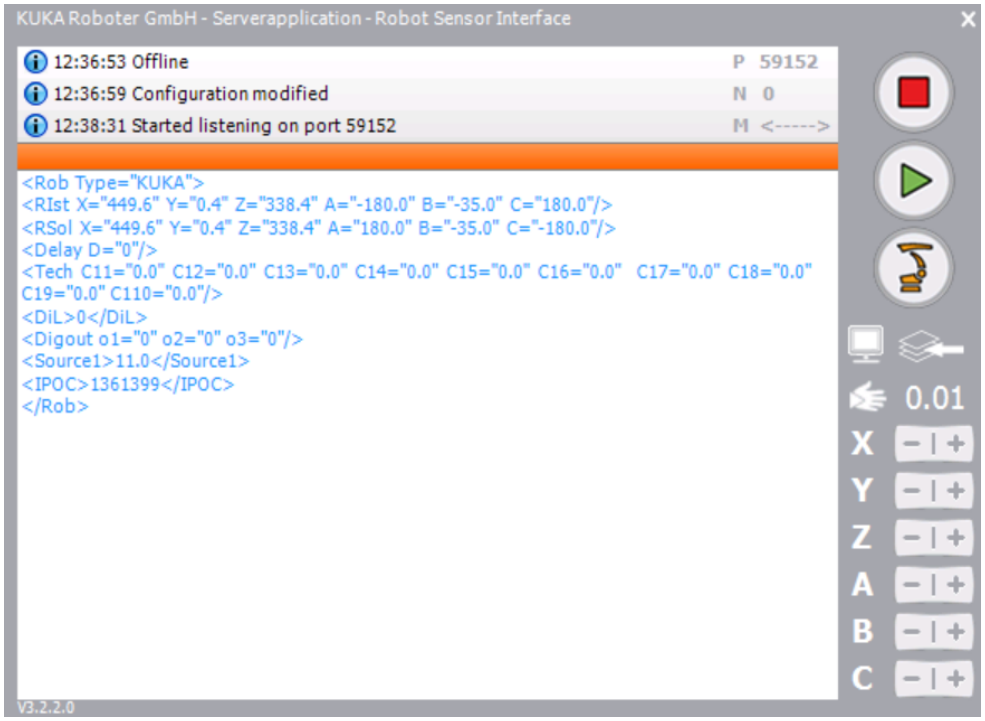


Abbildung 5.3: KUKA Testserverapplikation

5.3 XML-Konfiguration

Um den Aufbau der XML-Struktur in Kapitel 5.3.2 zur Kommunikation zwischen der Robotersteuerung und dem Sensorsystem zu verstehen, wird im Folgenden zunächst der Aufbau des RSI-Kontextes dargestellt.

5.3.1 RSI-Kontext

Abbildung 5.4 zeigt einen beispielhaften Aufbau eines RSI-Kontextes, der im RSI-VisualShell Programm erstellt wurde. Zu sehen sind verschiedene Arten von Objekten, die nach Ihrer Funktionalität zusammengefasst sind. Gruppe (1) sind Funktionsblöcke, die interne Variablen der Robotersteuerung auslesen und an gezielte Eingänge des mit (2) markierten Ethernetobjektes weitergeben. Das Ethernet Objekt ist für die Realisierung der Kommunikation verantwortlich. Die Eingänge können verwendet werden, um bestimmte Variablen oder Messgrößen and das entsprechende Sensorsystem zu versenden. Die Ausgänge definieren die Schnittstellen, an die das Sensorsystem seine Daten abgibt. Der darauffolgende optionale Datenfluss zu den Objekten (3) verursacht eine Messwertfilterung, bevor mit Objekt (4) eine kartesische Lageregelung durchgeführt wird. Die Objekte (5) repräsentieren Möglichkeiten, um den internen Programmablauf zu beeinflussen, indem eine Auswahl an Systemvariablen bearbeitet werden kann. Die Objekte innerhalb der Markierung (6) zeigen, wie der RSI-Monitor zur Visualisierung und Datenspeicherung des Systemzustandes angesteuert werden kann. Und das Objekt (7) dient zur Festlegung erlaubter Maximalwerte der Lageregelung.

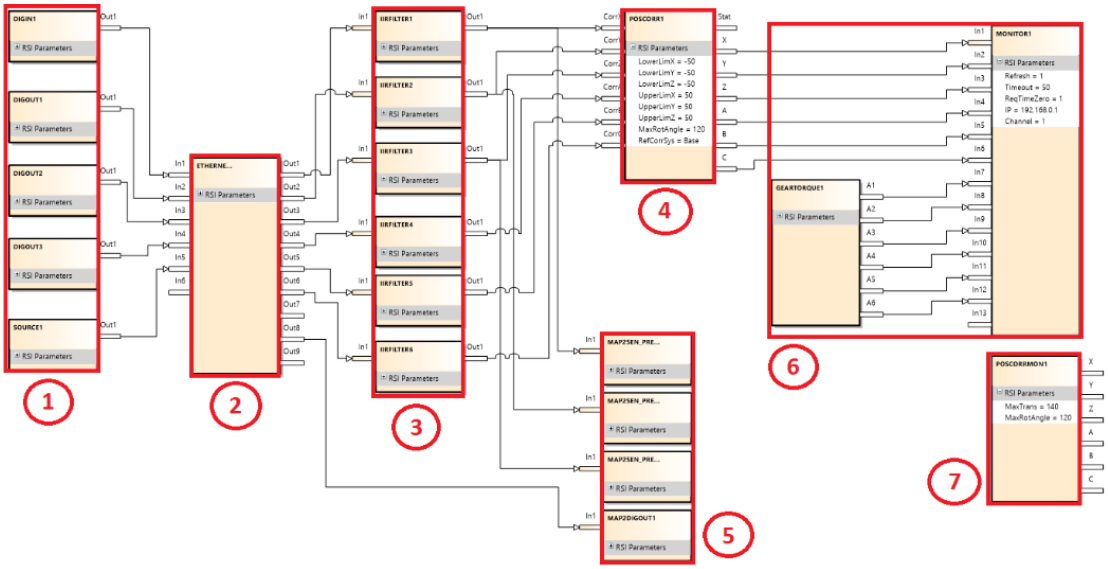


Abbildung 5.4: Beispielhafter Aufbau des RSI-Kontext

5.3.2 Aufbau der Konfigurationsdatei

Wie in Kapitel 2.6 bereits erwähnt, erfolgt der Transfer von Daten über die UDP Verbindung in einem festen XML-Format. Dieses muss vor der Inbetriebnahme und je nach gefordertem Umfang konfiguriert werden. Abbildung 5.5 zeigt das Schema der Konfigurationsdatei, welche, genauso wie der RSI-Kontext, in der Robotersteuerung hinterlegt sein muss.

```
<ROOT>
 <CONFIG></CONFIG>
 <SEND>
  <ELEMENTS></ELEMENTS>
 </SEND>
 <RECEIVE>
  <ELEMENTS></ELEMENTS>
 </RECEIVE>
</ROOT>
```

Abbildung 5.5: Schema der XML-Konfigurationsdatei

Im Folgenden wird ein Überblick auf die Bestandteile der XML-Konfiguration gegeben. Eine ausführliche Dokumentation findet sich in [KUKA Roboter GmbH, 2016, S.47f.]

<CONFIG>

Der Abschnitt <CONFIG> dient zur Konfiguration der Verbindungsparameter zwischen dem Sensorsystem und der Schnittstelle der Robotersteuerung. Hierbei ist der Robotersteuerung mitzuteilen, welche IP-Adresse das Sensorsystem besitzt und über welchen Port dieses zu erreichen ist. Darüber hinaus muss eine Kennung der Datenpakete des Sensorsystems über den Unterpunkt <SENTYPE> und die Art der Kommunikationsrichtung definiert werden. Die Kennung repräsentiert den Namen der Anwendung und ist frei wählbar, muss aber in jeder Nachricht zusätzlich beigefügt sein. Die Kommunikationsrichtung definiert, ob die Robotersteuerung das Empfangen von Daten erwartet, währenddessen es selbst im RSI Zyklustakt Daten versendet. Abbildung 5.6 zeigt eine beispielhafte Konfiguration.

```
- <CONFIG>
    <IP_NUMBER>172.31.2.150</IP_NUMBER>
      <!-- IP-number of the external socket -->
    <PORT>59152</PORT>
      <!-- Port-number of the external socket -->
    <SENTYPE>ImFree</SENTYPE>
      <!-- The name of your system send in <Sen Type="" > -->
    <ONLYSEND>FALSE</ONLYSEND>
      <!-- TRUE means the client don't expect answers. Do not send anything to robot -->
  </CONFIG>
```

Abbildung 5.6: Beispiel XML- Struktur <CONFIG>

<SEND>

Im <SEND> Abschnitt der XML-Konfigurationsdatei werden die von der Robotersteuerung gesendeten Elemente definiert. Hierbei können interne Parameter des Roboters, als auch frei bestimmbare Parameter innerhalb des RSI-Kontextes übertragen werden. Abbildung 5.7 zeigt die XML-Struktur der Konfigurationsdatei in Attribut-Schreibweise, und Abbildung 5.8 das daraus versendete XML-Dokument. Neben der Attribut-Schreibweise, ist ebenfalls eine Element-Schreibweise für die Konfiguration zulässig. [KUKA Roboter GmbH, 2016, S.51f.]

```
- <SEND>
  - <ELEMENTS>
      <ELEMENT INDX="INTERNAL" TYPE="DOUBLE" TAG="DEF_RIst"/>
      <ELEMENT INDX="INTERNAL" TYPE="DOUBLE" TAG="DEF_RSol"/>
      <ELEMENT INDX="INTERNAL" TYPE="LONG" TAG="DEF_Delay"/>
      <ELEMENT INDX="INTERNAL" TYPE="DOUBLE" TAG="DEF_Tech.C1"/>
      <ELEMENT INDX="1" TYPE="LONG" TAG="DiL"/>
      <ELEMENT INDX="2" TYPE="BOOL" TAG="Digout.o1"/>
      <ELEMENT INDX="3" TYPE="BOOL" TAG="Digout.o2"/>
      <ELEMENT INDX="4" TYPE="BOOL" TAG="Digout.o3"/>
      <ELEMENT INDX="5" TYPE="DOUBLE" TAG="Source1"/>
    </ELEMENTS>
  </SEND>
```

Abbildung 5.7: Beispiel XML-Struktur <SEND>

Die XML-Struktur zeigt, dass jedem Element ein Index zugeschrieben wird. „INTERNAL“ steht hierbei für systeminterne Variablen, welche unabhängig vom erstellten RSI-Kontext ausgelesen werden können. Im Gegensatz dazu entspricht ein nummerierter INDEX dem im RSI-Kontext am Ethernet anliegenden Input. Die Objekte aus Abbildung 5.4 (1) können hiermit der XML-Datei hinzugefügt werden. Nach dem Index ist der Typ des jeweiligen Elements zu definieren. RSI unterstütz die Verwendung von „Bool“, String“,“Long“ und „Double“. Der TAG legt den Namen des Elementes fest und ist frei wählbar, mit Ausnahme der mit „DEF_“ gekennzeichneten Systemvariablen.

```
<Rob Type="KUKA">
        <RIst X="0.0" Y="0.0" Z="0.0" A="0.0" B="0.0" C="0.0"/>
        <RSol X="0.0" Y="0.0" Z="0.0" A="0.0" B="0.0" C="0.0"/>
        <Delay D="0"/>
        <Tech C11="0.0" C12="0.0" C13="0.0" C14="0.0" C15="0.0" C16="0.0"  C17="0.0" C18="0.0" C19="0.0" C110="0.0"/>
        <DiL>0</DiL>
        <Digout o1="0" o2="0" o3="0"/>
        <Source1>-28.4</Source1>
        <IPOC>13424104</IPOC>
</Rob>
```

Abbildung 5.8: Beispiel versendetes XML-Dokument der Robotersteuerung

Man beachte, die IPOC Nummer, welche automatisch von der Robotersteuerung als letztes Element generiert wird. Sie ist eine Sicherheitsmaßnahme für das ungesicherte UDP Protokoll, mit dem die XML-Datenpakete versendet werden. Versendet die Robotersteuerung eine Nachricht, so muss das Sensorsystem innerhalb der RSI-Zykluszeit, sollte eine Antwort erwartet werden, mit einer Nachricht Antworten, in der die neuen Vorgaben für die Robotersteuerung, zusammen mit derselben IPOC Referenz, enthalten sind.

<RECEIVE>

Im Abschnitt <RECEIVE> werden die von der Robotersteuerung zu erwartenden XML-Elemente und deren Struktur definiert. Das RSI kann die empfangenen XML-Datenpakete anhand der Strukturdefinitionen einlesen und entsprechend des RSI-Kontextes den entsprechenden Variablen zuweisen. Dieser Prozess wird auch als „Parsen“ bezeichnet. Empfangene XML-Datenpakete, die nicht mit der festgelegten Datenstruktur konform sind, können nicht gelesen bzw. ausgewertet werden.

```
- <RECEIVE>
  - <ELEMENTS>
      <ELEMENT INDX="INTERNAL" TYPE="STRING" TAG="DEF_EStr"/>
      <ELEMENT INDX="INTERNAL" TYPE="DOUBLE" TAG="DEF_Tech.T2" HOLDON="0"/>
      <ELEMENT INDX="1" TYPE="DOUBLE" TAG="RKorr.X" HOLDON="1"/>
      <ELEMENT INDX="2" TYPE="DOUBLE" TAG="RKorr.Y" HOLDON="1"/>
      <ELEMENT INDX="3" TYPE="DOUBLE" TAG="RKorr.Z" HOLDON="1"/>
      <ELEMENT INDX="4" TYPE="DOUBLE" TAG="RKorr.A" HOLDON="1"/>
      <ELEMENT INDX="5" TYPE="DOUBLE" TAG="RKorr.B" HOLDON="1"/>
      <ELEMENT INDX="6" TYPE="DOUBLE" TAG="RKorr.C" HOLDON="1"/>
      <ELEMENT INDX="8" TYPE="LONG" TAG="DiO" HOLDON="1"/>
    </ELEMENTS>
  </RECEIVE>
```

Abbildung 5.9: Beispiel XML-Struktur <RECEIVE>

Abbildung 5.9 zeigt die <RECEIVE> XML-Struktur, welche analog zu der von <SEND> aufgebaut ist. Der Unterschied besteht lediglich in der Erweiterung des „HOLDON“ Parameters. Dieser dient, wenn mit „1“ parametrisiert, zur Sicherheit der Echtzeitkommunikation dafür, dass bei nicht eingetroffenem Sensorupdate innerhalb der konfigurierten Zykluszeit der letzte am Ethernetausgang anliegende Messwert wiederverwendet wird. Ist „HOLDON“ mit „0“ parametrisiert und es kommt zu einem fehlenden Sensortelegramm, wird ein Error die aktive Roboterbewegung stoppen und Das Roboterprogramm muss neu gestartet werden.

```
<Sen Type="ImFree">
        <Estr>Message from Sensorsystem</Estr>
        <Tech  T24="0.0" T27="0.0" T25="0.0" T22="0.0" T28="0.0" T20="0.0" T21="0.0" T23="0.0" T26="0.0" T29="0.0"/>
        <RKorr  Z="0.0000" X="0.0000" B="0.0000" A="0.0000" Y="0.0000" C="0.0000"/>
        <DiO>125</DiO>
        <IPOC>13424104</IPOC>
</Sen>
```

Abbildung 5.10: Beispiel versendetes XML-Dokument des Sensorsystems

6 Praktische Umsetzung

6.1 Programmablauf

Um die inneren Abläufe der umgesetzten Applikation zu veranschaulichen, werden in Abbildung 6.1 und 6.2 die Prozessabläufe zum Korrigieren und Vorhersagen der Objektpose in vereinfachten Sequenzdiagrammen dargestellt.

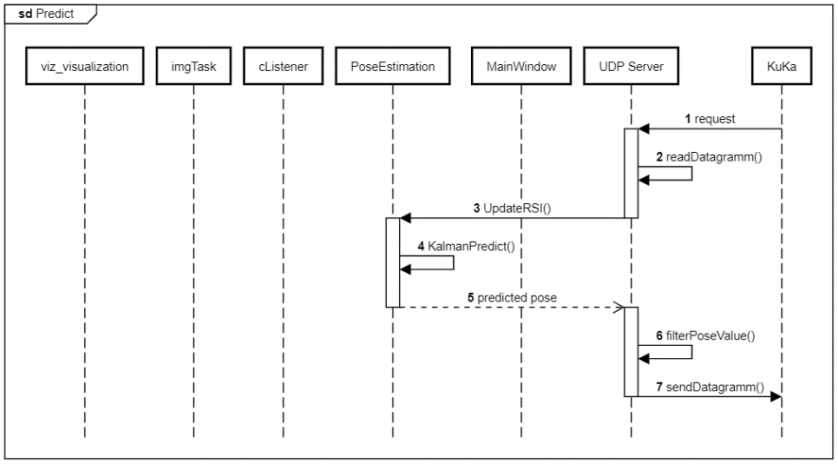


Abbildung 6.1: Sequenzdiagramm der Prädiktion und Systemkommunikation

Abbildung 6.1 zeigt den vereinfachten Ablauf der Prädiktion. Eingeleitet wird der Prozess durch das Eingangstelegramm der Robotersteuerung. Daraufhin fordert der UDP Server einen neuen Messwert an. Aus Sicherheitsgründen erfolgt in letzter Instanz noch eine Kontrolle (vgl. Signal 6 in Abbildung 6.1), ob sich die Korrekturwerte innerhalb eines vorgeschriebenen Arbeitsraumes befinden. Anschließend werden die ermittelten Daten an die Robotersteuerung gesendet.

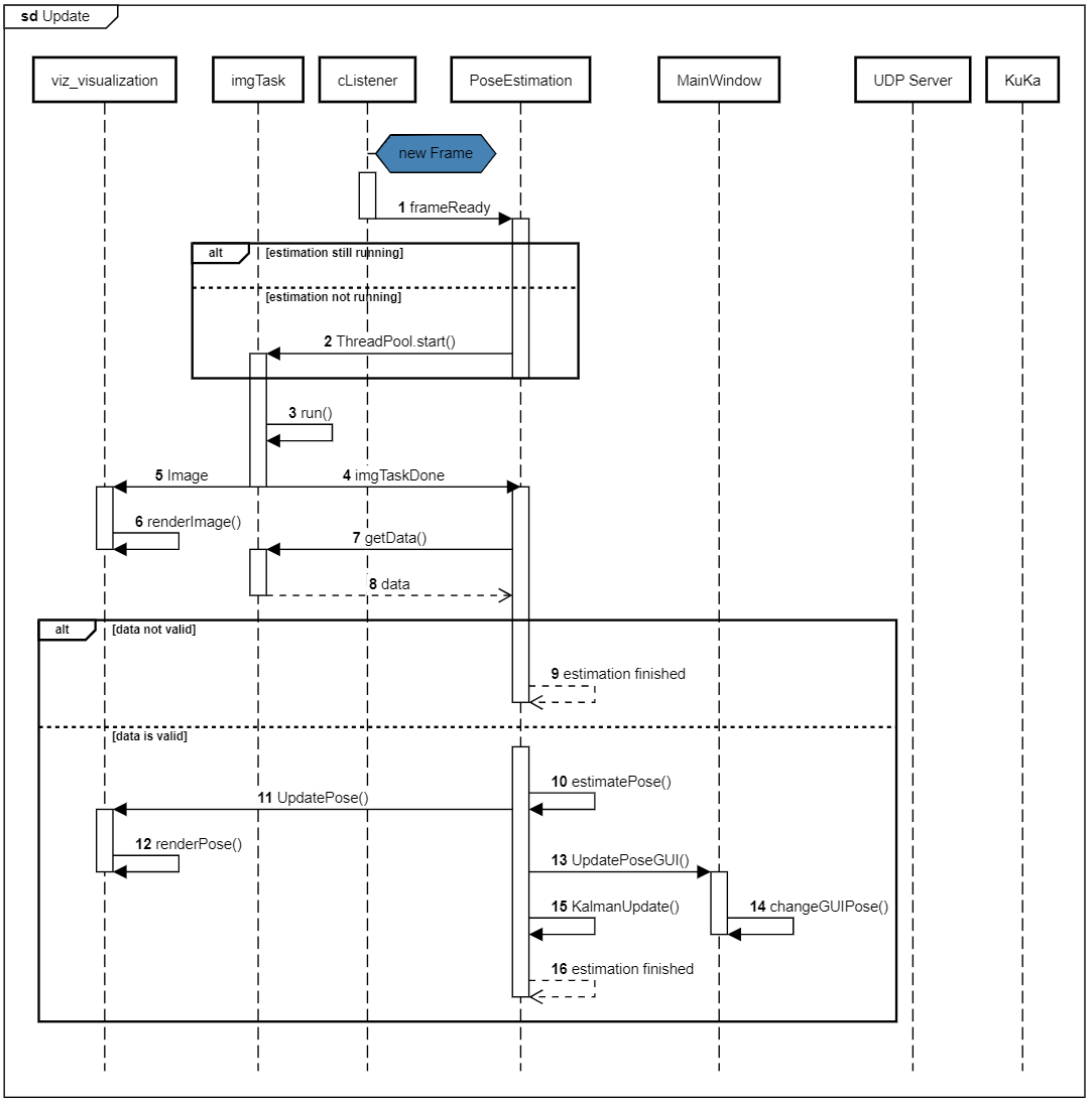


Abbildung 6.2: Sequenzdiagramm zum Korrigieren der Objektpose im Kalman Filter

Abbildung 6.2 zeigt den vereinfachten Ablauf der Korrektur des Kalman Filters. Zu erkennen ist dabei die modulare Struktur und die zentrale Klasse „PoseEstimation“, die alle anderen Klassen miteinander verbindet. Eingeleitet wird der Prozess durch die Aufnahme eines neuen Bildes, welches von der Klasse „cListener“ überwacht wird. Eine Optimierung der Laufzeit erfolgt durch die Arbeitsverteilung auf mehrere Threads, so übernimmt die Klasse „imgTask“ in Form eines Threadpools die Bildverarbeitung der Kamerabilder und die graphische Visualisierung für den Benutzer erfolgt durch den Thread „viz_visualization“. (vgl. Signal 2-5 in Abbildung 6.2) Diese Struktur erlaubt die Analyse der ermittelten Bildinformationen während

im Hauptthread die Verarbeitung von Bedienereingaben, Berechnung der Pose und die Netzwerkkommunikation durchgeführt wird.

6.1.1 Die Klasse „MainWindow“

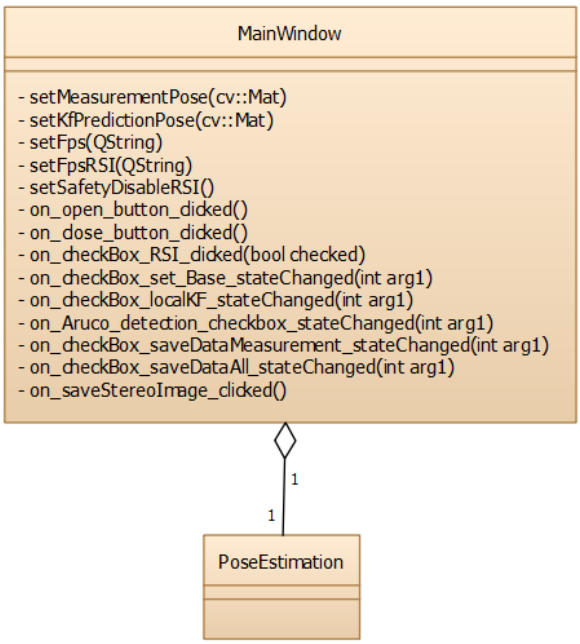


Abbildung 6.3: UML-Klassendiagramm der Klasse „MainWindow“

Die Hauptklasse „MainWindow“ erzeugt eine grafische Benutzeroberfläche (GUI), welche die Schnittstelle zwischen dem menschlichen Bediener und dem internen Systemzustand darstellt. Durch die „set…()“-Funktionen werden die aktuellen Messwerte der Klasse „PoseEstimation“ auf dem GUI angezeigt und durch die „on…()“-Funktionen der vom Benutzer gewünschte Funktionsumfang des Sensorsystems kotrolliert.

6.1.2 Die Klasse „PoseEstimation“

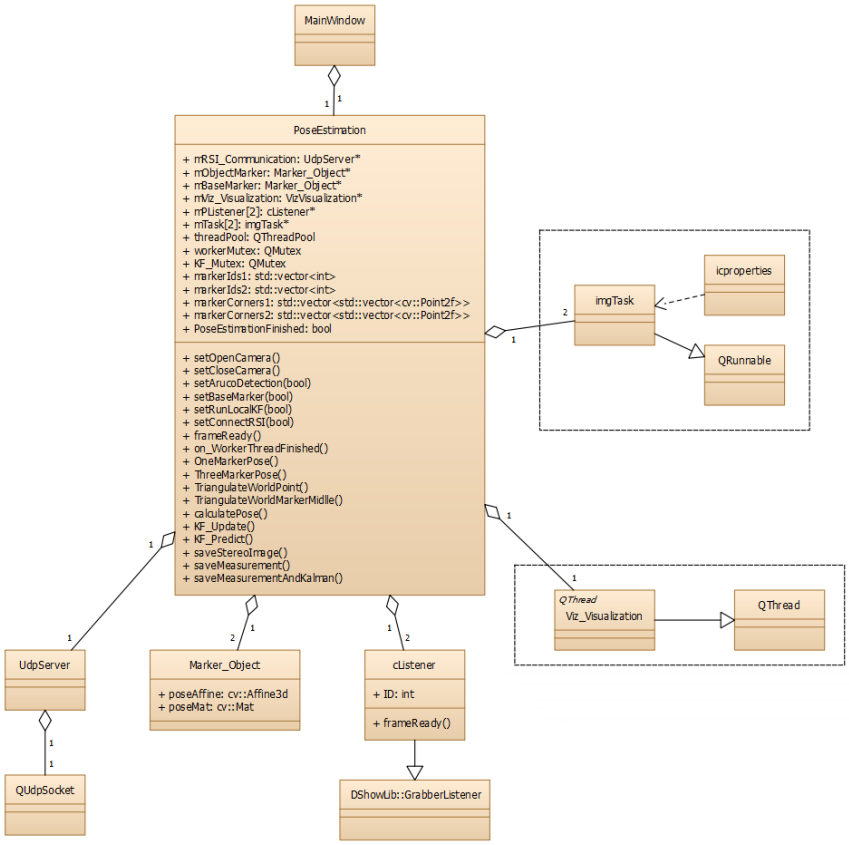


Abbildung 6.4: UML-Klassendiagramm der Klasse PoseEstimation

Die Klasse „PoseEstimation“ verbindet sämtliche Klassen miteinander und steuert den Funktionsumfang der Anwendung. Der Funktionsumfang wird von der Klasse „MainWindow“ vorgegeben und über die „set…()“-Funktionen weitergeleitet. Abbildung 6.4 visualisiert das dazugehörige Klassendiagramm, aus dem die Klassenabhängigkeiten der gesamten Applikation ersichtlich sind. Die Umrandung der Klasse „Viz_Visualization“, als auch der Klasse „imgTask“ kennzeichnen die Abgrenzung zum restlichen Programmablauf in separate Threads. Die Klasse „PoseEstimation“ verwendet die vom Kamerahersteller empfohlenen Objekte „mPListener“ der Klasse „cListener“, um das Event „frameReady()“ zu generieren, sobald die verwendeten Kameras ein neues Bild aufnehmen. Liegt in beiden Kameras ein neues Bild vor, so wird der Threadpool, in dem sich die beiden Objekte „mTask“ befinden gestartet und die

Bildverarbeitung durchgeführt. Ist diese von beiden Threads absolviert führt die Funktion „on_WorkerThreadFinished()“ die Berechnung der Pose durch, indem für die Dauer der Ausführung der „workerMutex“ aktiviert und die Daten der Bildverarbeitung in die „marker…1/2“ Variablen übertragen werden. Die Funktionen „OneMarkerPose()“ und „ThreeMarkerPose()“ berechnen, je nach der Anzahl an gefundenen Markern, mit der Funktion „TriangulateWorldPoint()“ und „TriangulateWorldMarkerMiddle()“ die relevanten Punkte im Raum, wonach die Funktion „calculatePose()“ (vgl. Kapitel 3.5) die Objektpose in affiner und kartesischer Form ermittelt und in dem Objekt „mObjectMarker“ speichert. Die Funktionen „KF_Update()“ und „KF_Predict()“ führen die Berechnungsschritte der Kalman Filter durch und blockieren für die jeweilige Dauer auf die Kalman Parameter den gegenseitigen Zugriff durch das aktivieren des „KF_Mutex“.

Die „save…()“-Funktionen erlauben das Aufnehmen von Messwerten bei aktiver Berechnung.

6.1.3 Die Klasse „ImgTask“

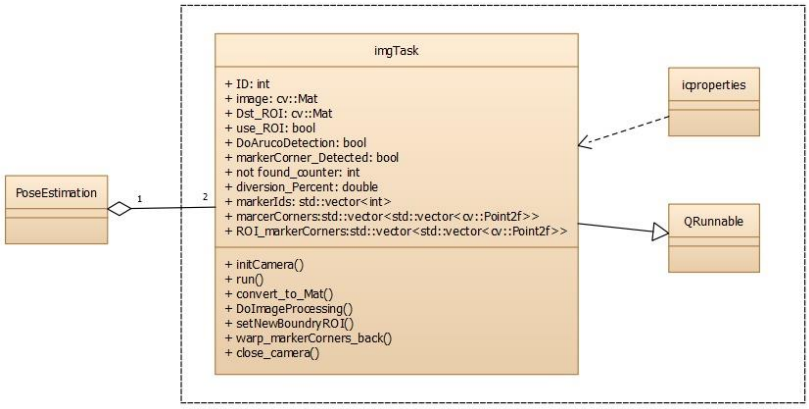


Abbildung 6.5: UML-Diagramm der Klasse „imgTask“

Die Klasse „icproperties“ ermöglicht die Konfiguration der Kameraparameter und wird von der Funktion „initCamera()“ verwendet, welche dem Thread eine ID und eine Kamera zuweist. Die Klasse „imgTask“ wird von der Klasse „PoseEstimation“ durch den Aufruf der Funktion „run()“ dazu aufgefordert, die Bildverarbeitung der beiden Kameras in einem jeweils eigenen Thread durchzuführen, sollte die Variable „DoArucoDetection“ „true“ sein. Dabei werden die Funktionen „convert_to_Mat()“ und „DoImageProcessing()“ ausgeführt, welche das aktuelle Bild der Kamera in das für OpenCV geeignete Format „Mat“ konvertieren und die ArUco-Detektion anwenden. Für eine Optimierung der Laufzeit dient die Funktion „setNewBoundryROI()“, wenn in der vorherigen Bildanalyse bereits Marker gefunden wurden. Es wird dadurch nicht mehr das gesamte Bild, sondern nur noch der Ausschnitt in der Umgebung der letzten Markerposition durchsucht. Die Funktion „warp_markerCorners_back()“ transformiert die aus dem Bildausschnitt ermittelten Markerpositionen zurück auf die realen Pixelkoordinaten des gesamten Bildes.

6.1.4 Die Klasse „UdpServer“

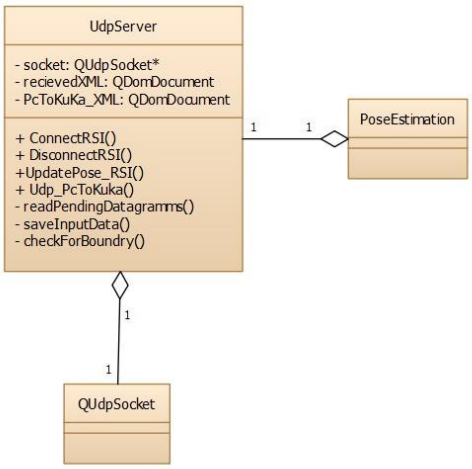


Abbildung 6.6: UML-Diagramm der Klasse „UdpServer“

Die Klasse „UdpServer“ realisiert die UDP Kommuniktationsschnittstelle zwischen der Robotersteuerung und dem Sensorsystem. Die Funktion „ConnectRSI()“ initialisiert einen „socket“, der die Ankunft von Telegrammen der Robotersteuerung erkennt. Daraufhin wird die Funktion „readPendingDatagramm()“ aufgerufen, welche im Wesentlichen zwei Aufgaben erfüllt. Zum einen wird das angekommene Telegramm von der Funktion „saveInputData()“ in die Variable „recievedXML“ geparst und der Inhalt übernommen, zum anderen wird die Klasse „PoseEstimation“ darüber informiert einen neuen Soll-Zustand für die Robotersteuerung zu ermitteln. Die Funktion „UpdatePose_RSI()“ wird von der Klasse „PoseEstimation“ bei neu berechnetem Soll-Zustand aufgerufen und lässt diesen von der Funktion „checkForBoundry()“ auf den zugelassenen Arbeitsraum begrenzen, bevor die Funktion „Udp_PcToKuka()“ ein XML-Dokument mit den neuen soll-Zuständen erzeugt und dieses an die Robotersteuerung sendet.

6.1.5 Die Klasse „Viz_Visualization“

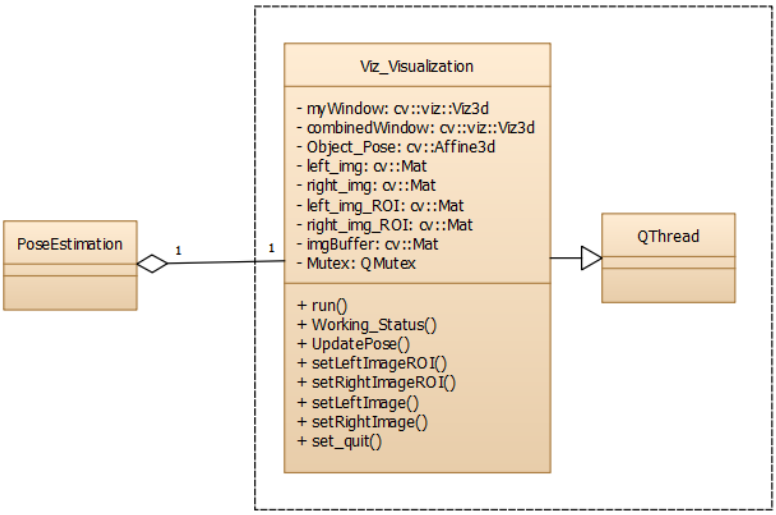


Abbildung 6.7: UML-Klassendiagramm der Klasse „Viz_Visualization

Die Klasse „Viz_Visualization“ ist ein Thread, der die Aufgabe hat die Visualisierung der Kamerabilder und die grafische Abbildung der Objektpose zu übernehmen. Durch die Funktion „UpdatePose()“ wird die aktuell gemessene Objektpose der Klasse „PoseEstimation“ und durch die „set…image()“ Funktionen die aktuellen Kamerabilder der „imgTask“ Klassen übertragen. Die Funktion „run()“ startet den Thread und stoppt ihn erst wieder durch die von der Klasse „PoseEstimation“ angeforderte Funktion „set_quit()“.

6.2 KRL Programm

Das verwendete KRL Programm für die Durchführung der Demonstration einer Sensorgeführten Roboterbewegung ist in Abbildung 6.8 dargestellt.

Zur Realisierung einer Roboterbewegung die identisch zu der Pose des jeweils verwendeten Objektmarkers ist, müssen sich die Bewegungen auf ein Koordinatensystem mit derselben Orientierung beziehen. Im Rahmen dieser Arbeit wird dieses Koordinatensystem von dem Objektmarker definiert und der Robotersteuerung in seiner Ausgangslage vorgegeben. Der Befehl „BAS(#...,…)“ ermöglicht es die eingelernten Komponenten für die Roboterbewegung zu verwenden. Mit der Funktion „PTP“ fährt der Roboter in die gewünschte Ausgangslage. Die Funktion „RSI_CREATE()“ lädt den auf der Robotersteuerung hinterlegten RSI-Kontext, anschließend wird die Signalverarbeitung mit der Funktion „RSI_ON()“ aktiviert. In Abbildung 6.8 wird der Korrekturmodus „#ABSOLUTE“ verwendet, wodurch die Robotersteuerung absolute Werte für die Bewegungskorrektur erwartet. Die Funktion „RSI_MOVECORR()“ schaltet die sensorgeführte Bewegung ein und der Roboter wird nun rein korrekturgesteuert auf Basis der anliegenden Sensordaten verfahren.

```
; ===========================================
;
; RSI Demo: Zuther
; Realtime UDP data exchange
;
; ===========================================

; Declaration of KRL variables
DECL INT ret    ; Return value for RSI commands
DECL INT CONTID ; ContainerID

;FOLD INI
  ;FOLD BASISTECH INI
  BAS (#INITMOV,0 )
  ;ENDFOLD (BASISTECH INI)
  ;FOLD USER INI
        BAS(#BASE,6)
        BAS(#TOOL,3)
  ;ENDFOLD (USER INI)
;ENDFOLD (INI)

; Move to start position

PTP {A1 0, A2 -90, A3 90, A4 0, A5 90, A6 0}

; Create RSI Context
ret = RSI_CREATE("RSI_Ethernet.rsi",CONTID,TRUE)
IF (ret <> RSIOK) THEN
  HALT
ENDIF

; Start RSI execution
;ret = RSI_ON(#RELATIVE)
ret = RSI_ON(#ABSOLUTE)
IF (ret <> RSIOK) THEN
  HALT
ENDIF

; Sensor guided movement
RSI_MOVECORR()

; Turn off RSI
ret = RSI_OFF()
IF (ret <> RSIOK) THEN
  HALT
ENDIF

PTP {A1 0, A2 -90, A3 90, A4 0, A5 90, A6 0}

END
```

Abbildung 6.8: KRL Programmfür Demonstration Zuther

6.3 Inbetriebnahme

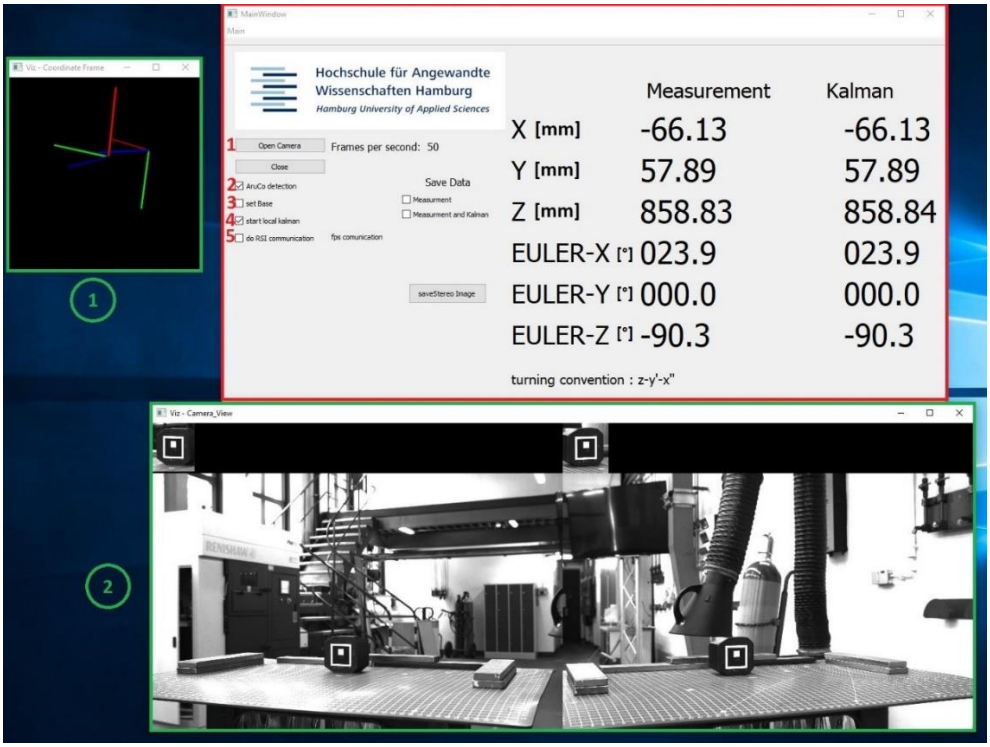


Abbildung 6.9: GUI der Anwendung

Abbildung 6.9 zeigt das für den Benutzer entwickelte GUI. Die in rot umrandete Oberfläche ist das Interface, mit dem zum einen der Programmablauf durch die Klasse „MainWindow“ gesteuert werden kann (linke Hälfte), zum andern die aktuellen Messwerte der Triangulation und des Kalman Filters angezeigt werden (rechte Hälfte). Die grün umrandeten Oberflächen sind die von der Klasse „Viz_Visualization“ erzeugten Visualisierungen. Das Fenster (1) zeigt hierbei die aktuelle Pose des Markers in Referenz zu dem Koordinatensystem der linken Kamera und das Fenster (2) ein live Bild der beiden Kameras zusammen mit Ihren intern durchsuchten „Region of Interest“ (ROI) Ausschnitten. Tabelle 6 auf der folgenden Seite führt die notwendigen Schritte auf, um die sensorgeführte Robotersteuerung zu in Betrieb zu nehmen. Die entsprechenden Schaltflächen zu den einzelnen Durchführungsschritten sind in Abbildung 6.9 rot nummeriert.

Durchführungsschritt	Beschreibung
0. Kamerakalibration	Bei gleichbleibender Anordnung der Kameras muss dieser Prozess nicht wiederholt ausgeführt werden. Es sind durch das Betätigen der Schaltfläche „saveStereo Image“ Bildpaare mit einem entsprechenden Kalibriermuster aufzunehmen. (vgl. Kapitel 3.2.4) In dem Dateipfad der Anwendung werden die Bilder im Ordner „stereoData“ gespeichert. Dieser Ordner ist in den Dateipfad der Anwendung „Stereo-Calib“ zu verschieben und dort mit den entsprechenden Kalibrierparametern auszuführen. Die daraus erzeugten Dateien „Calibration_LEFT“ und „Calibration_RIGHT“ sind dem Dateipfad der Anwendung dieser Arbeit beizufügen.
1. Drücke „open Camera“	Die Kameras werden initialisiert und die aktuellen Bilder in Fenster (2) dargestellt.
2. Platziere das Markerobjekt	Die Orientierung hat der eingelernten Vorgabe des Koordinatensystems der Robotersteuerung zu entsprechen.
3. Aktiviere „ArUco detektion“	Die Kamerabilder werden nach ArUco-Markern durchsucht, bei gefundenem Marker erweitert sich das Fenster (2) und zeigt zusätzlich den Bildausschnitt an, der intern zur Verfolgung der Markerposition verwendet wird. Die Pose des erkannten Markerobjektes wird auf dem GUI ausgegeben und in Fenster (1) visualisiert.
4. Aktiviere „set Base“	Die zuletzt ermittelte Pose wird als Basiskoordinatensystem festgelegt. Die Anzeige der Pose des Objektmarker erfolgt nun in Referenz auf das Basiskoordinatensystem.
5. Aktiviere „start local kalman“	Die Kalman Filter werden nun mit einem $4ms$ Timer berechnet. Dies erlaubt das Einschwingen der Kalman Filter auf den aktuell vorgegebenen Wert und ermöglicht die Beobachtung des Filterverhaltens, ohne dass die Robotersteuerung mit dem Sensorsystem verbunden sein muss.
6. Aktiviere „do RSI communication“	„start local kalman“ wird deaktiviert und die Kommunikation des Sensorsystems mit der Robotersteuerung aktiviert.
7. Starte das KRL Programm auf der Robotersteuerung	Die sensorgeführte Roboterbewegung wird nun durchgeführt. *Hinweis*: Das KRL Programm muss sich im Verzeichnis: *„C:\KRC\ROBOTER\KRC\R1\Program*“ befinden.

Tabelle 6: Inbetriebnahme der sensorgeführten Robotersteuerung

7 Durchführung und Validierung

Diese Arbeit verfolgt nicht das Ziel eine allgemeingültige Schlussfolgerung für die Wahl sämtlicher Prozessparameter zu erlangen, da eine solche Untersuchung den zeitlichen Rahmen übersteigen würde. Darüber hinaus sind diese Parameter stark Anwendungsspezifisch und müssen je nach Einsatzumgebung und geforderten Funktionsumfang, bei geforderten Arbeitsraum, mit einer geforderten Genauigkeit und verfügbarer Hardware individuell optimiert werden. Stattdessen wird die Qualität des entwickelten Systems anhand eines festen Messaufbaus untersucht, dieser wird in Kapitel 7.1 beschrieben. In Kapitel 7.2 wird die Objekterkennung des Messsystems geprüft und das Verhalten der implementierten Kalman Filter in Kapitel 7.3 beschrieben. Kapitel 7.4 beschäftigt sich mit der realisierten Systemleistung indem die FPS.

7.1 Messaufbau

Wie bereits erwähnt, wird aus Zeitgründen nur ein einzelner Messaufbau betrachtet. Dieser ist in Abbildung 7.1 dargestellt. Der untersuchte Arbeitsraum befindet sich $70cm$ von der Basiskennlinie der Kameras entfernt und erstreckt sich auf einer Breite von $50cm$ Tiefe von $30cm$ mit einer Höhe an der Vorderkante von $30cm$ und an der Hinterkante von $40cm$.

Abbildung 7.1: Untersuchter Arbeitsraum

7.2 Messmittel

Um eine Bewertung des Messsystems durchzuführen werden Referenzobjekte mit festen Prüfmaßen verwendet und mit den vom Messsystem ermittelten Prüfgrößen verglichen. Abbildung 7.2 zeigt das Referenzobjekt, welches zur Ermittlung der translatorischen Genauigkeit des Messystems verwendet wird. Es handelt es sich dabei um einen einzelnen ArUco-Marker mit einer Kantenlänge von $11{,}045cm$. Die Kantenlänge entspricht hierbei dem Prüfmaß, auf welches sich die Qualität des Messystems abbilden lässt, indem das reale Maß mit dem digital ermittelten Abstand der Eckpunkte des Markers im Raum verglichen wird.

Abbildung 7.2: Untersuchte Markervarianten

Für die Messung der rotatorischen Bestandteile der Pose ist ein 3D gedruckter Prüfkörper mit festen Winkelbeziehungen konstruiert worden, der die Drehung eines sich auf der Vorderseite befindlichen Markers um verschiedene Winkel ermöglicht. Abbildung 7.3 zeigt dessen entscheidende Winkelbeziehungen. Somit lassen sich feste Winkel innerhalb des gesamten Arbeitsraumes reproduzierbar überprüfen.

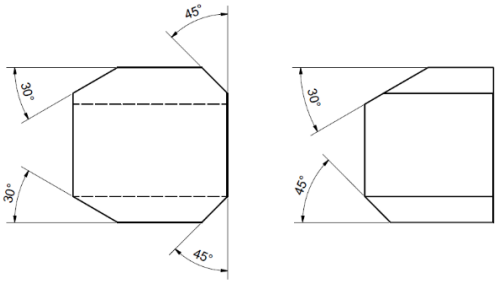


Abbildung 7.3: Winkelbeziehungen des Prüfkörpers

7.3 Objekterkennung

7.3.1 Messrauschen

Für die Aufnahme des statischen Messrauschens wird ein Markerobjekt frontal in der Mitte des Arbeitsraumes platziert. Abbildung 7.4 zeigt auszugsweise eine der aufgenommenen Messkurven. Die weiteren Messkurven befinden sich in Anhang A.

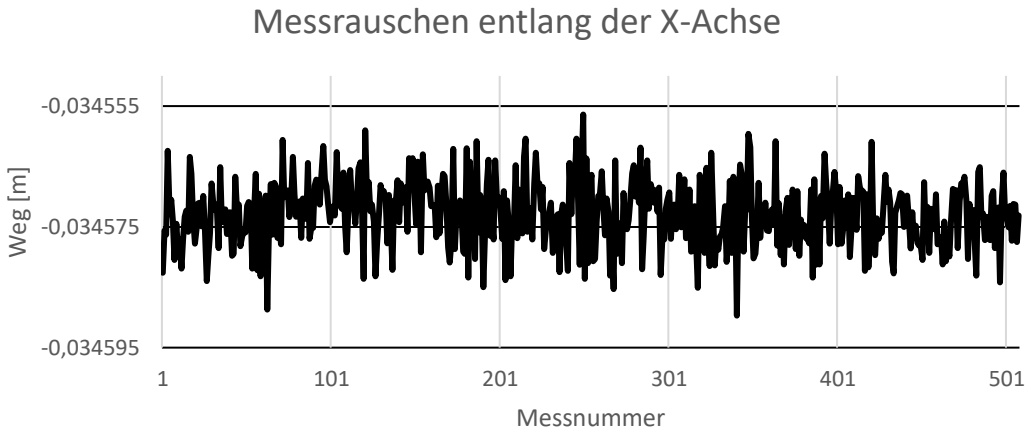


Abbildung 7.4: Messrauschen entlang der X-Achse

Tabelle 7.1 enthält die ermittelten Standartabweichungen der Markerpose. Die translatorischen Bestandteile sind blau gekennzeichnet und zeigen, dass die Streuung der X- und Y-Achse einander sehr ähnlich sind und dass die Streuung entlang der Z-Achse (Bildtiefe) die höchste Abweichung von $0{,}014mm$ aufweist. Die rotatorischen Anteile der Objektpose sind rot gekennzeichnet und weisen eine maximale Standartabweichung von $0{,}035°$ auf.

	Standartabweichung
X in m	5,3985E-06
Y in m	7,6924E-06
Z in m	1,4094E-05
X in °	3,2391E-02
Y in °	3,5250E-02
Z in °	1,6895E-02

Tabelle 7.1: Analyse des Messrauschens der Markererkennung

7.3.2 Translatorische Messauswertung

Für die Bewertung der translatorischen Messwertgenauigkeit, wird der in Kapitel 7.2 beschriebene Marker verwendet. Zu Beginn jeder Messung nimmt der Marker eine vergleichbare Position ein, welche im Folgenden als Ausgangslage bezeichnet wird. (vgl. Abbildung 7.4)

Abbildung 7.5: Ausgangslage der translatorischen Messwertaufnahme

Es wurden verschiedene Messungen durchgeführt, die je unterschiedliche Aspekte untersuchen. Zunächst wird die in Abbildung 7.4 zu sehende Orientierung zu den Kameras beibehalten und dann translatorisch der untersuchte Arbeitsbereich händisch bewegt. (vgl. Abbildung 7.5) Das Ziel ist eine Aussage über die Messgenauigkeit im Raum bei einer frontalen Markerausrichtung. Weitere Messungen beinhalten den Fokus auf rotatorische Einflüsse für die Messgenauigkeit. Hierbei wird der Marker aus seiner Ausgangslage um seine horizontale- und vertikale Achse verdreht.

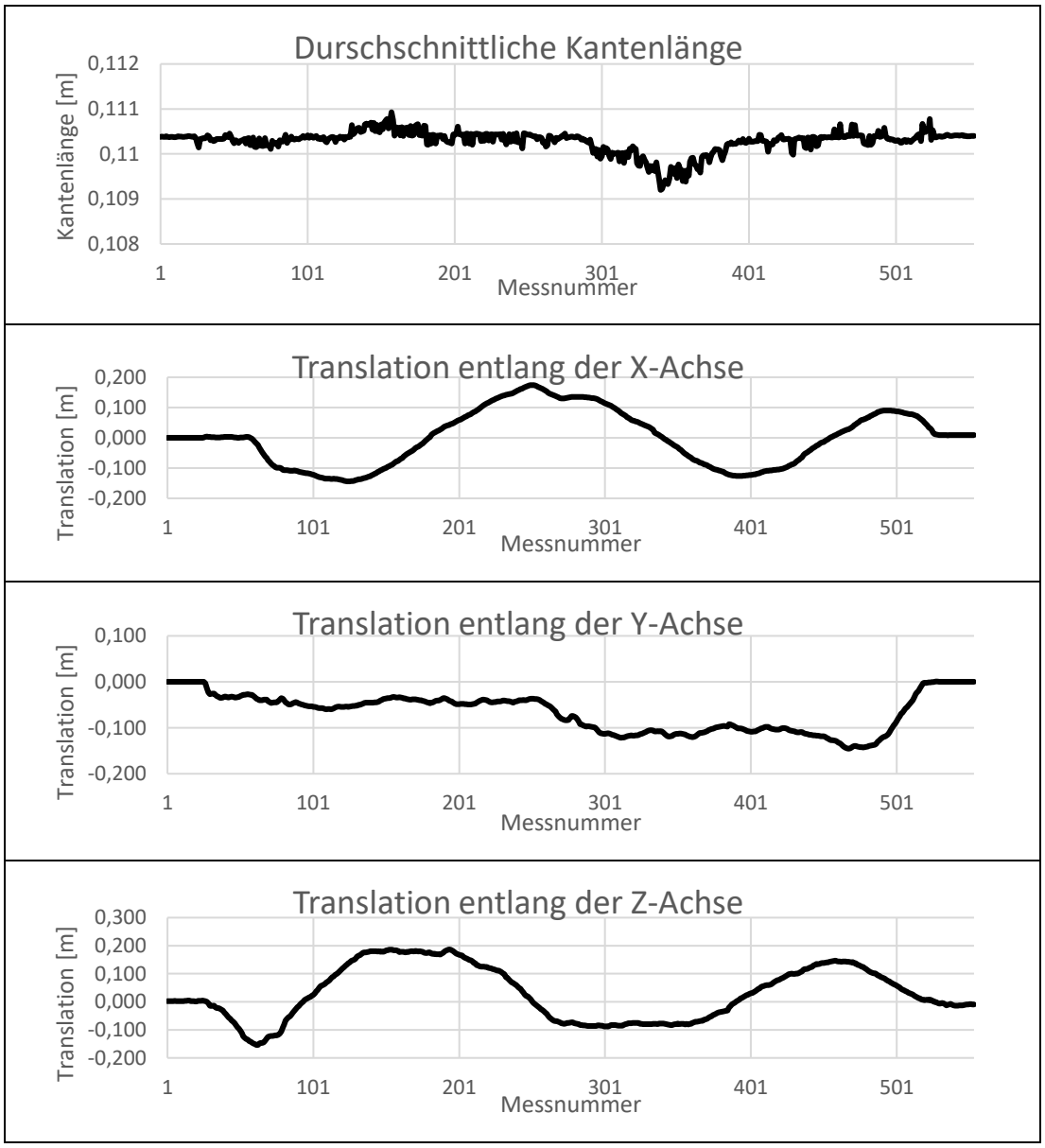


Abbildung 7.6: Frontale Translation des Prüfmarkers

Abbildung 7.5 stellt die durchschnittliche Kantenlänge des Objektmarkers den von der Ausgangslage verfahrenen Raumachsen gegenüber. Da der Marker um 90° verdreht ist (siehe Abbildung 7.4) ergibt sich in der Ausgangslage die Markereigene X-Achse als horizontale Achse, die Y-Achse als vertikale Achse mit einer positiven Ausrichtung in Richtung des Bodens und die Z-Achse der Raumtiefe, die mit steigender Entfernung positiv orientiert ist. Der Koordinatenursprung des Markers befindet sich in der in der oberen linken Ecke, daher ist bei

den Translationen im Arbeitsraum entlang der X-und Y-Achse zu beachten, dass eine zusätzliche Distanz, entsprechend der ungefähren Prüfmaßlänge von $11cm$, vermessen wurde, aufgrund der Abstände der Markereckpunkte zueinander.

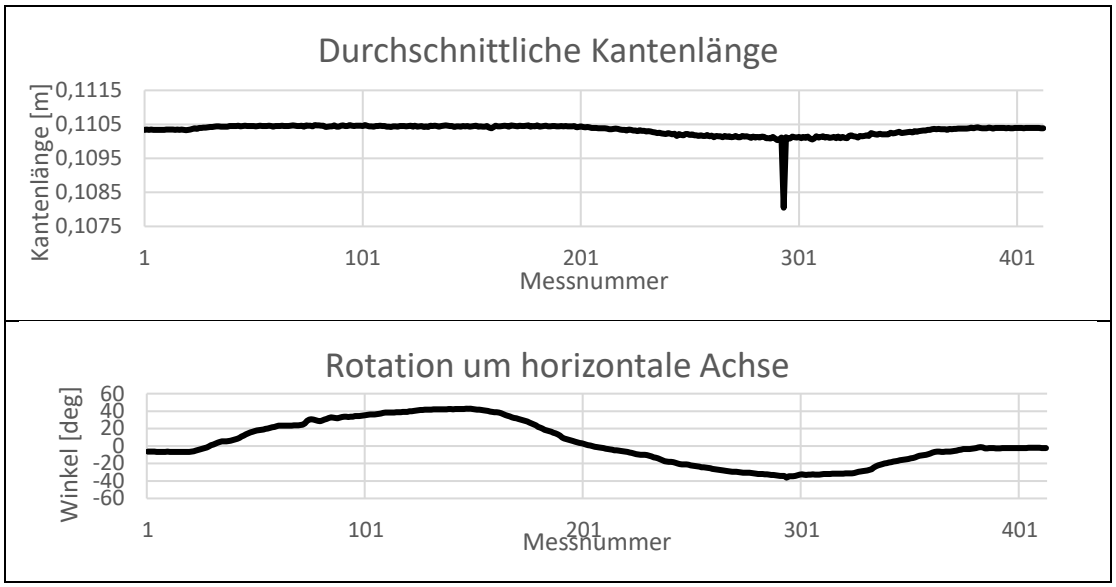


Abbildung 7.7: Rotation um horizontale Achse

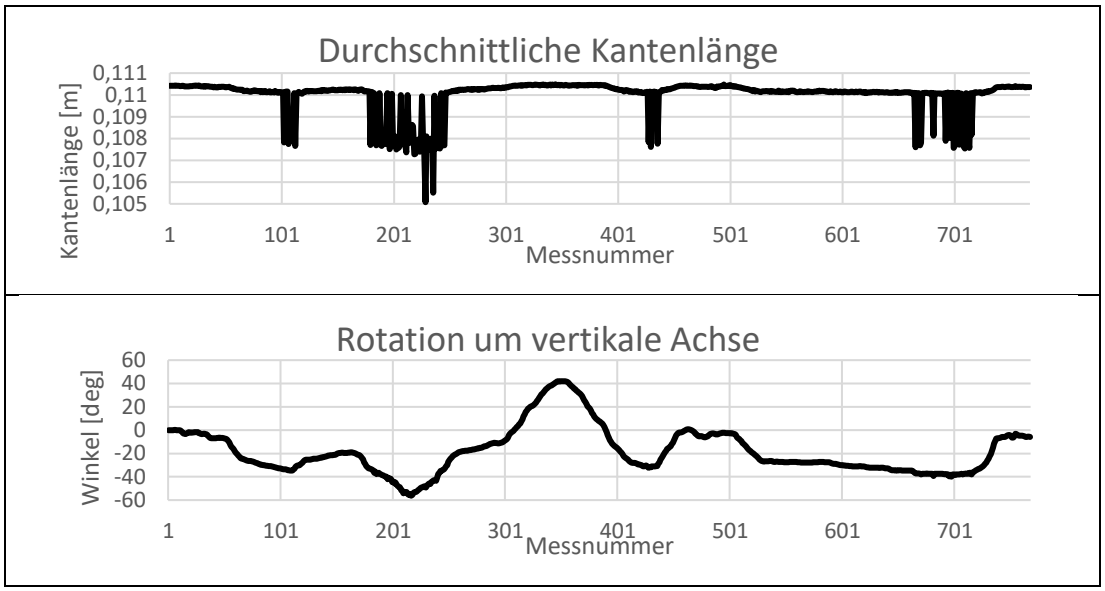


Abbildung 7.8: Rotation um vertikale Achse

	Mittelwert in m	Standartabweichung in m
Translation - Frontal	1,10282E-01	2,48E-04
Rotation - horizontale Achse	1,10335E-01	1,6749E-04
Rotation -Vertikale Achse	1,09982E-01	7,9891E-04

Tabelle 7.2: Analyse der translatorischen Messgenauigkeit

In Tabelle 7.2 sind die Mittelwerte und die dazugehörigen Standartabweichungen aus den Messungen in Abbildung 7.5-7.7 aufgeführt. Die Messwerttabelle legt hierbei nah, dass die Rotation um die Vertikale Achse das höchste Streuungsmaß besitzt. Betrachtet man die dazugehörige Abbildung 7.7 ist jedoch zu erkennen, dass dieser Wert stark von Messwertsprüngen beeinflusst wird, die ab einem Drehwinkel von $-30°$ auftreten. Das Auftreten der Messwertsprünge kann mehrere Ursachen haben. Zum einen kann es sein, dass der Ausgangswinkel des ArUco Markers zu dem Kamerasystem keine frontale Ausrichtung hatte und die gemessenen Winkel somit mit einem Offset zur realen Winkelmessung des Kamerasystems belastet sind. Zum anderen könnte eine ungleichmäßige Raumbeleuchtung für die Änderung der Markererkennungsqualität verantwortlich sein. In Abbildung 2.2 (b) ist zu erkennen, dass die Deckenbeleuchtung nicht zentral entlang der Arbeitsfläche verläuft und es hierdurch bei der Rotation um die vertikale Raumachse zu einer richtungsabhängigen Lichtintensität kommt. Für zukünftige Evaluierungsverfahren sollte daher dem Kamerasystem eine eigene Beleuchtung installiert werden, um orts- und zeitunabhängige Lichtqualität zu gewährleisten. Darüber hinaus bietet der Algorithmus zum Erkennen der ArUco Marker die Möglichkeit, eine Optimierung für das integrierte „adaptive Thresholding“ durchzuführen, in dieser Arbeit werden bislang jedoch die OpenCV Standartwerte verwendet.

Aus den Messreihen ergibt sich insgesamt ein durchschnittlicher Mittelwert von $11{,}019cm$, bezogen auf das Prüfmaß von $11{,}045cm$ ergibt das eine durchschnittliche Messabweichung von $0{,}026cm$. Diese Abweichung ist, neben den bereits erwähnten Licht- Offset- und Detektionsbedingungen, auch auf einen Fehler der Kalibration zurückzuführen.

7.3.3 Rotatorische Messauswertung

Für die rotatorische Messauswertung wird ein ArUco Marker auf die Vorderseite des in Kapitel 7.2.2 beschriebenen Prüfkörpers geklebt und von der Ausgangslage heraus mit jeweils spezifischen Winkeln um die einzelnen Koordinatenachsen verdreht.

Winkelmessung um die horizontale Achse

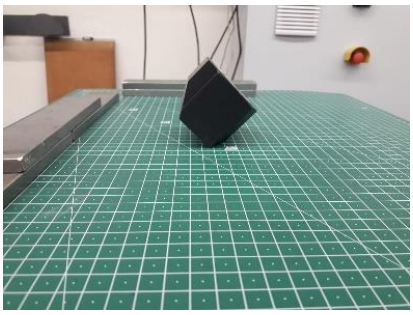


Abbildung 7.9: Rotation um die horizontale Achse

Sollwert S in °	Messung 1 in °	Messung 2 in °	Messung 3 in °	Messung 4 in °	Messung 5 in °	Mittelwert M in °	|S-M| in °
30	29.5	29.6	29.6	29.3	29.7	29,54	0,46
45	45.2	45.5	45.5	45.4	45.5	45,42	0,42

Tabelle 7.3: Winkelmessungen um die horizontale Achse

Winkelmessung um Achse entlang der Objekttiefe

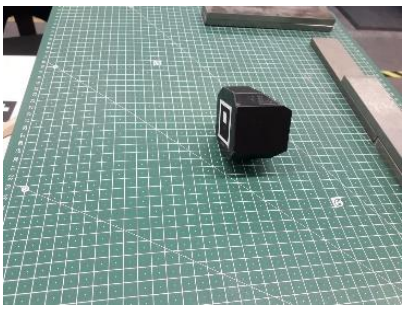


Abbildung 7.10: Rotation entlang der Objekttiefe

Sollwert S in °	Messung 1 in °	Messung 2 in °	Messung 3 in °	Messung 4 in °	Messung 5 in °	Mittelwert M in °	|S-M| in °
45	45.4	45.3	45.5	45.5	45.3	45,4	0,4
60	60.1	60.3	60.3	60.1	59.8	60,12	0,12
−45	-44.6	-44.5	-44.9	-44.4	-44.5	-44,58	0,58
−60	-60.4	-60.6	-60.5	-60.6	-60.6	60,52	0,52

Tabelle 7.4: Winkelmessungen um Achse entlang der Objekttiefe

Winkelmessung um vertikale Achse

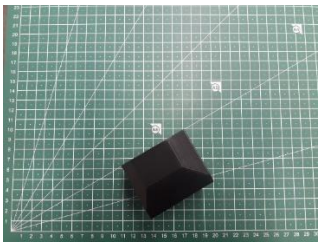


Abbildung 7.11: Rotation um vertikale Achse

Sollwert S in °	Messung 1 in °	Messung 2 in °	Messung 3 in °	Messung 4 in °	Messung 5 in °	Mittelwert M in °	|S-M| in °
15	14.8	14.9	15.0	15.1	15.2	15.0	0
30	26	26.7	26.5	26.9	29.6	27,14	2,86
45	40.6	40.0	40.6	40.4	40.6	40,44	4,56
−15	-15.5	-15.6	-15.0	-14.9	-14.6	15,12	0,12
−30	-30.1	-29.9	-30.5	-30	-29.7	30.04	0,04
−45	-42.2	-44.5	-42,7	-42,5	-42,3	42,84	2,16

Tabelle 7.5: Winkelmessungen um vertikale Achse

	Horizontale Achse in °	Vertikale Achse in °	Achse entlang der Objekttiefe in °
Durchschnitt	0,44	1,948	0,405

Tabelle 7.6: durchschnittliche Differenz der rotatorischen Messung

Tabelle 7.6 fasst die Ergebnisse aus den jeweilig achsenspezifischen Messwerttabellen zusammen und zeigt, dass sich der Durchschnittwert der Differenz der berechneten Mittelwerte zu ihren entsprechenden Soll-Wertvorgaben bei der horizontalen Achse, als auch der Achse entlang der Objekttiefe, zu einer Genauigkeit von unter einem halben Grad ergibt. Bei der vertikalen Achse berechnet sich hierbei ein Wert von ungefähr zwei Grad. Die Messwerttabelle 7.5 zeigt jedoch starke richtungsabhängige Messunterschiede auf. Dieses Verhalten wurde bereits im vorherigen Kapitel 7.3.2 beobachtet und kann durch Abbildung 7.7 verdeutlicht werden. Es ist zu beachten, dass die vertikale Achse des in Kapitel 7.3.2 verwendeten Markers entgegen der Richtung des in diesem Kapitel verwendeten Markers verläuft.

7.4 Kalmanfilter

Wie unter Kapitel 5.1 bereits erwähnt, hat diese Thesis nicht die Aufgabe der regelungstechnischen Auslegung der Kalman Filter, muss aber einen grundsätzlichen Nachweis der Implementierung erbringen um im Rahmen dieser Arbeit eine sensorgeführte Roboteransteuerung und Bewegung zu realisieren, die als Grundlage in folgenden Arbeiten optimiert werden kann. Das Verhalten der händisch eingestellten Kalman Filter ist dem Anhang B beigefügt. Exemplarisch werden in Abbildung 7.11 die Werte der Vorhersage der Kalman Filterung mit den dazugehörigen Kamerasystemmesswerten aufgetragen. Zu erkennen sind zunächst glättende Eigenschaften der Filterung auf rauschende Messwerte und ein überschwingen bei starker Änderung der Winkelrichtung. Der Kalman Filter dient in dieser Arbeit zur Interpolation der Messwerte und führt bei der Kommunikation mit der Robotersteuerung die Vorhersage 5-mal häufiger durch, als das Messsystem den Systemzustand korrigiert. Mit den aktuell eingestellten Parametern weisen die Kalman Filter zum einen sprunghafte anpassungsschritte der Lage aus (vgl. Abbildung 7.11 zwischen den Messnummern 1251 und 1501) und zum andern passt sich die Geschwindigkeit nicht schnell genug auf die Änderung der Messwerte an. Gezeigt wird dies in Abbildung 7.12 , welche den Messbereich aus Abbildung 7.11 um die Messung 2300 vergrößert. Es ist ein deutliches Fehlverhalten der Vorhersage zu erkennen. In der entwickelten Demonstration ist daher, um

eine ruhige Bewegung zu ermöglichen, ein zusätzlicher Tiefpassfilter implementiert. (vgl. Kapitel 5.3.1)

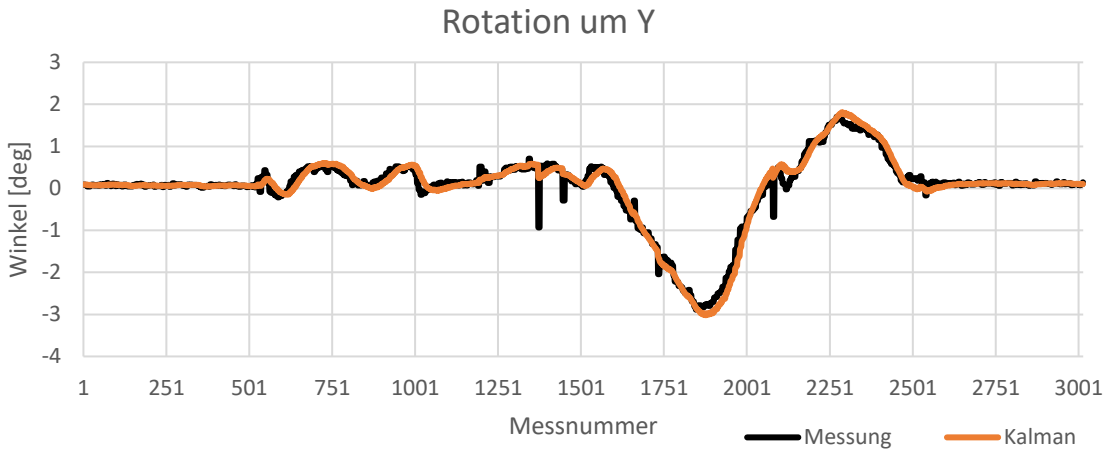


Abbildung 7.12: Gegenüberstellung der Werte Des Kalman Filter mit den Messwerten

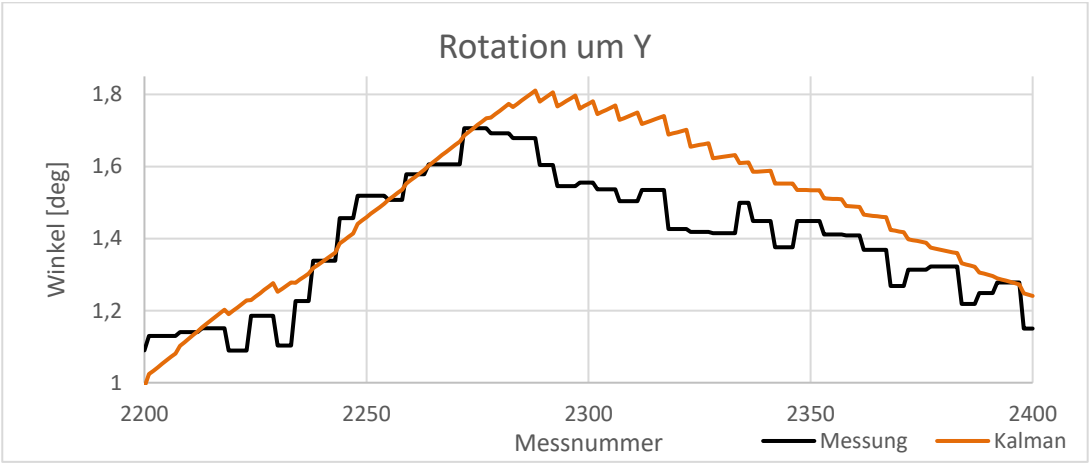


Abbildung 7.13: Fehlverhalten des Kalman-Filters (Detailansicht)

7.5 Performance

7.5.1 Bildaufnahme/ Bildverarbeitung

Um in der Lage zu sein die Kalman Filter optimal zu korrigieren, ist neben der Anforderung auf die Genauigkeit des Messystems ausschlaggebend mit welcher Geschwindigkeit dieser Prozess wiederholt wird. Die maximale Begrenzung entspricht in dieser Arbeit den 50 FPS der verwendeten Kameras bei voller Auflösung. Um den gesamten Bildbereich innerhalb dieser Zeit analysieren zu können, wird im Falle einer Markererkennung der Suchraum auf die Bildgröße des Markers mit aufaddiertem Offset begrenzt. Abbildung 7.12 zeigt wie viele Bilder pro Sekunde vom Messsystem analysiert werden, während der Marker vom hinteren Ende des festgelegten Arbeitsraumes (vgl. Abbildung 7.1) bis an die vordergrenze verschoben wird. Die Messung ergibt, dass über den Arbeitsraum hinweg eine Bildwiederholrate von 49,7 FPS erreicht wird. Die Reaktionszeit des Systems beträgt dementsprechend $20{,}11ms$.

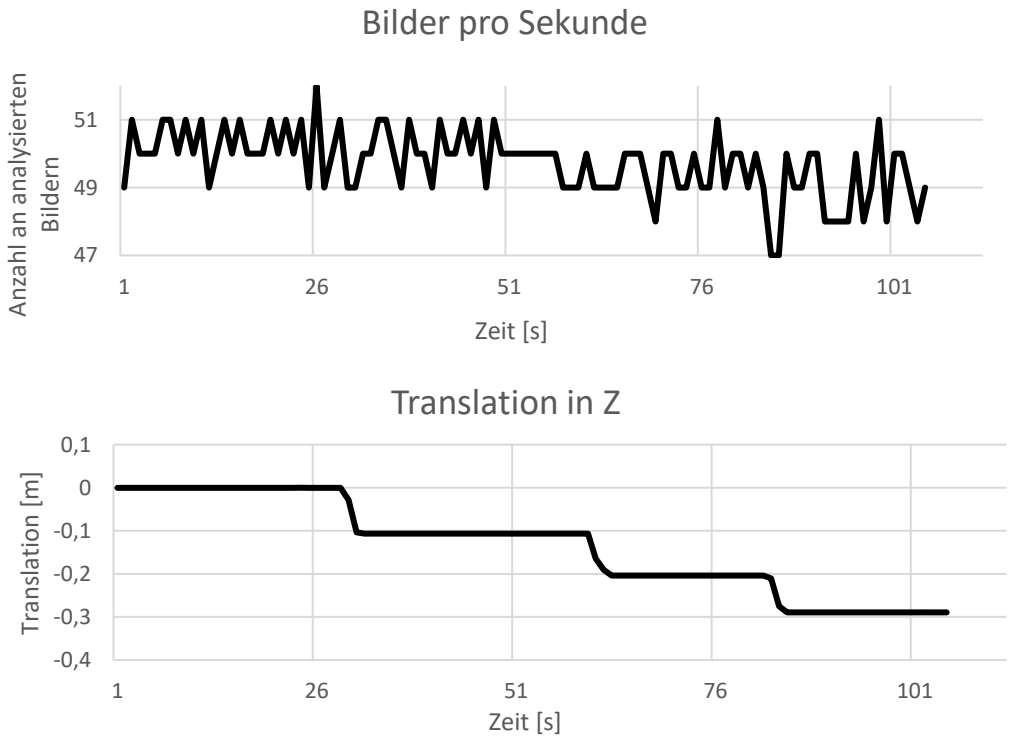


Abbildung 7.14: FPS des umgesetzten Systems

7.5.2 Kommunikation mit der Robotersteuerung

Die stabile Kommunikation mit der Robotersteuerung in einem Zyklustakt von $4ms$ oder $12ms$ bei voll umgesetzten Funktionsumfang ist Voraussetzung um eine sensorgeführte Roboterbewegung unter Verwendung des RSI-Technologiepakets zu ermöglichen. Abbildung 7.13 zeigt die Antwortzeit des umgesetzten Systems bei aktiver Sensorkorrektur des Roboters durch die vom Kamerasystem aufgenommene Markerbewegung.

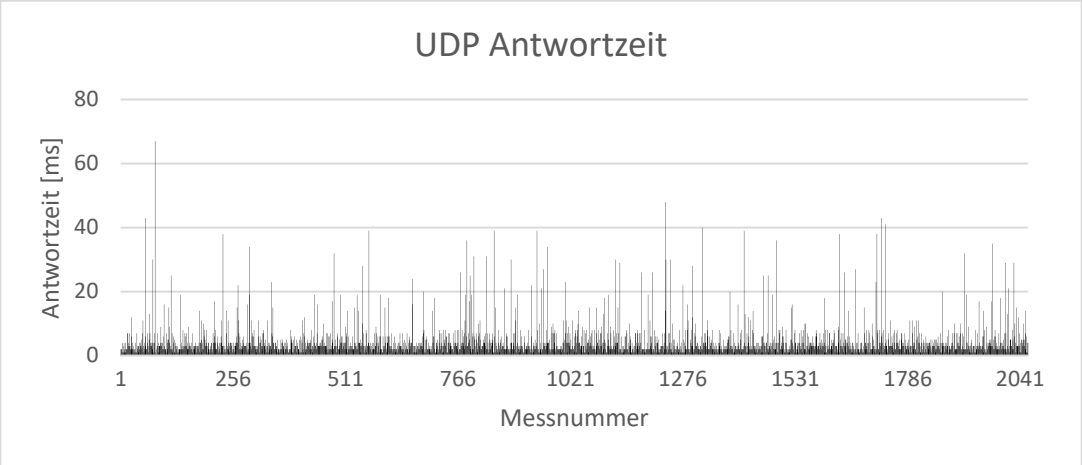


Abbildung 7.15: UDP Antwortzeit des Messsystems bei vollem Funktionsumfang mit RSI $4ms$ Konfiguration

Die Messung ergibt, dass das Messsystem grundsätzlich in der Lage ist die Kommunikation mit der Robotersteuerung innerhalb der maximalen Geschwindigkeit von $4ms$ umzusetzen. Zu erkennen ist jedoch auch, dass Antwortzeiten existieren, die diesen Zeitrahmen nicht erfüllen. Durch die Konfiguration des „Holdon“ Parameters in der Konfiguration (vgl. Kapitel 5.3.2) können die nichteintreffenden Telegramme jedoch auf der Robotersteuerung überbrückt werden. Die Option einen weiteren Rechner als eine unabhängig von den Prozessen des Messsystems verlässliche Kommunikationsinstanz zu realisieren wie in [Edberg, Nyman, 2010] wäre vor allem für den in folgearbeiten steigenden Funktionsumfang sinnvoll.

8 Zusammenfassung

8.1 Fazit

Innerhalb des Bearbeitungszeitraums konnte ein schnelles und visualisiertes System zur sensorgeführten Robotersteuerung entwickelt werden. Der Datenaustausch zwischen der KRC4-Robotersteuerung und dem Arbeitsrechner konnte realisiert und die Robotersteuerung mit dem RSI-Technologiepaket konfiguriert werden. Das Sensorsystem ist in dem Qt-Framework in C++ programmiert und verwendet für die Realisierung der Bildverarbeitung die Softwarebiltiothek OpenCV. Das Korrespondenzproblem der optischen Triangulation wird durch die Verwendung von Markerobjekten gelöst und darauf aufbauend ein hinreichend genaues und auf Echtzeit optimiertes Messystem hergeleitet und zusammen mit einem Filterkonzept für die Interpolation von Messwerten implementiert werden. Sicherheitsmaßnahmen, die implementiert wurden, umfassen auf Seite der Robotersteuerung innerhalb des RSI-Kontext eine Abgrenzung sowohl des maximal zulässigen Arbeitsraumes, als auch maximal zulässigen Rotation. Darüber hinaus kann der Arbeitsraum auch über das Sensorsystem beschränkt werden.

Abbildung 8.1: Demonstration der entwickelten Anwendung

Abbildung 8.1 zeigt die funktionierende Prozesskette, in der die Pose des Roboters die Pose des Objektmarkers verfolgt, welcher frei bewegt werden kann.

8.2 Ausblick

Das in dieser Arbeit umgesetzte System repräsentiert die Grundsfunktionalität um das ferngesteuerte Schweißen innerhalb des MeriTec Forschungsprojektes zu ermöglichen. Für folgende Arbeiten steht somit nun ein performantes Messystem zur Verfügung, um eine anwendungsspezifische Entwicklung voranzutreiben. Abbildung 8.2 zeigt beispielsweise zwei Lösungen, um die Bewegung eines Schweißgerätes messen zu können. In Abbildung 8.2 (a) handelt es sich um den Schweißgriff der Firma Soldamatic, die Lösungen zum simulierten Schweißen anbieten. Die Platzierung von Markern um den gesamten Schweißkopf herum erlaubt eine Richtungsunabhängige Bedienung und die Option, schlecht zum Messystem orientierte Marker zu ignorieren zu können. Abbildung 8.2 (b) zeigt eine Variation, die bereits jetzt mit dem Vorhandenen Funktionsumfang verwendet werden könnte.

(a) Soldamatic (b) Eigene Markerplazierung

Abbildung 8.2: Schweißgeräte mit möglicher Markerlage

Um eine ausgeprägtere, reproduzierbare, den Prozess vereinfachende und zeiteffizientere Evaluierung des Messystems in Zukunft vorzunehmen, wird empfohlen, ein Referenzobjekt , das an den TCP des Roboters befestigt werden kann, zusammen mit einer Kalibriertrajektorie für den Roboter zu entwickeln.

Literatur

[A. McGee, F. Schmidt]

A.McGee, Leonard; F. Schmdt, Standly : Discovery oft he Kalman Filter as a Practical Tool for Aerospace and Industry, 1985, URL: https://ntrs.nasa.gov/search.jsp?R=19860003843-Zugriffsdatum: 2019-01.08

[Blume, 2014]

Blume, Jürgen: Methoden und Anwendungen zur intuitiven Mensch-Roboter-Interaktion, Dissertation, TU München, 2014

[Edberg,Nyman, 2010]

Edberg, Martin; Nyman, Peter: Implementing Multi-Touch Screen for Real-time Control of a Robotic cell, Masterthesis, Chalmers Universität of Technology, 2010

[Gerrido-Jurado et al., 2016]

Garrido-Jurado, Sergio et al.: SGeneration of fiducial marker dictionaries using Mixed Integer Linear Programming, 2015 In:Pattern Recognition, Band 51, Seite 481-491, 2016, DOI:10.1016/j.patcog.2015.09.023

[GitHub TIS]

The Imaging Source: URL: https://github.com/TheImagingSource-Zugriffsdatum: 2019-05.21

[GitHub OpenCV Contrib]

OpenCV: opencv_contriv, URL: https://github.com/opencv/opencv_contrib-Zugriffsdatum: 2019-05.21

[Graf, 2007]

Graf, Simone: Kamerakalibrierung mit radialer Verzeichnung- die radiale essentielle Matrix, Dissertation, Universität Passau, 2007

[j.Romero-Ramirez,Munoz-Salinas, Medina-Carnicer, 2018]

J.Romero-Ramirez, Francisco; Munoz-Salinas, Rafael; Medina-Carnicer, Rafael: Speeded up detection of squared fiducial markers, In: Image and Vision Computing, Band 76, Seite 38-47, 2018, DOI:10.1016/j.imavis.2018.05.004

[Kaehler, Bradski, 2017]

Kaehler, Adrian; Bradski, Gray: Learning OpenCV3-Computer Vision in C++ with the OpenCV Library, Sebastapol (O'Reilly),2017, ISBN 978-1-491-93799-0

[KUKA Roboter GmbH, 2014]

KUKA Roboter GmbH(Hrsg.):KRC4 compact: Bedienanleitung, Augsburg, 2014

[KUKA Roboter GmbH, 2015]

KUKA Roboter GmbH(Hrsg.):KR AGILUS sixx Mit W- und C-Variante, Augsburg, 2015

[KUKA Roboter GmbH, 2016]

KUKA Roboter GmbH(Hrsg.):KUKA.RobotSensorInterface 3.3, Augsburg, 2016

[Mordrow, 2008]

Modrow, Daniel: Echtzeitfähige aktive Stereoskopie für technische und biometrische Anwendungen, Dissertation, TU München, 2008

[Munoz-Salinas]

Munoz-Salinas, Rafael: ArUco-An efficient library for detection of planat markers and camera pose estimation,
URL:https://docs.google.com/document/d/1QU9KoBtjSM2kF6ITOjQ76xqL7H0TEtXriJX5kwi9Kgc/edit#

[Müller et al., 2019]

Müller, Rainer et al.: Handbuch Mensch-Roboter-Kollaboration, München (Carl Hanser), Januar 2019, ISBN 978-3-446-45016-5

[Nischwitz et al., 2011]

Nischwitz, Alfred et al.: Computergrafik und Bildverarbeitung, Dritte Auflage, Band2 Bildverarbeitung, Berlin (Springer), 2011

[OpenCV ArUco]

OpenCV Dokumentation: Detection of ArUco Markers,
URL: https://docs.opencv.org/3.1.0/d5/dae/tutorial_aruco_detection.html - Zugriffsdatum: 2019-05.21

[Peters, 2009]

Peter, Falko: FPGA-basierte Bildverarbeitung zur Fahrspurerkennung eines autonomen Fahrzeugs, Bachslorarbeit, HAW Hamburg, 2009

[Schröter, 2018]

Schröter, Daniel: Entwicklung einer Methodik zur Planung von Arbeitssystemen in Mensch-Roboter-Kooperation In: Stuttgarter Beiträge zur Produktionsforschung, Band 81, Suttgart, Frauenhofer IPS (Hrsg.), ISBN 978-3-8396-1329-0

[Shen,2015]

Shen, Yi: System für die Mensch-Roboter-Koexistenz in der Fließmontage, Dissertation, TU München, 2015

[Siciliano, Khatib, 2008]

Siciliano, Bruno; Khatib, Oussama:Handbook of Robotics, Berlin(Springer), 2008, ISBN: 978-3-540-23957-4

[Willowgarage]

Willowgarage: OpenCV- URL
http://www.willowgarage.com/pages/software/opencv
-Zugriffsdatum: 2019-06.12

[Winkler, 2016]

Winkler, Alexander: Sensorgeführte Bewegungen stationärer Roboter, Dissertation, TU Chemnitz, 2016

[Woernle, 2011]

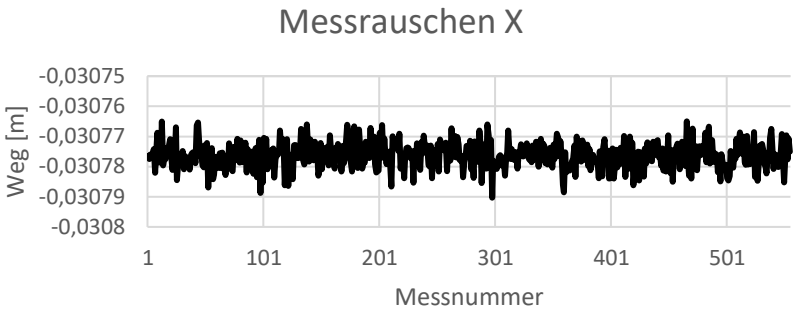
Woernle, Christoph: Mehrkörpersysteme- Eine Einführung in die Kinematik und Dynamik von Systemen starrer Körper, Berlin(Springer), 2011, DOI 10.1007/978-3-642-15982-4

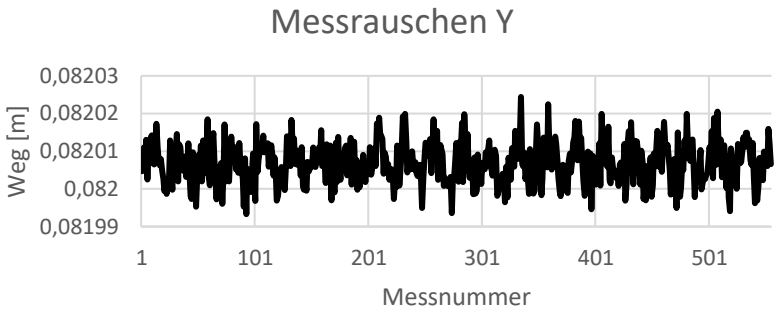
[Zander, 2017]

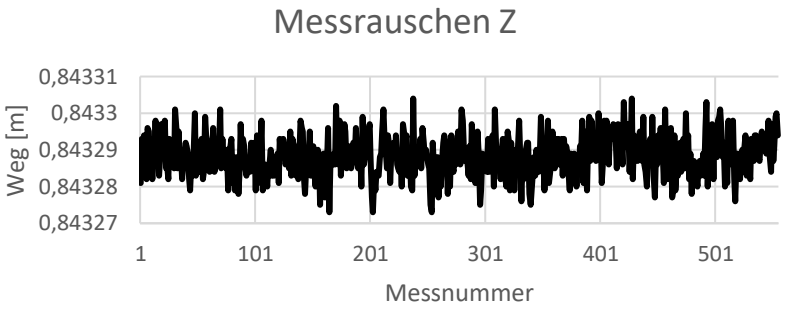
Zander, Mike Steven: Tiefendaten unterstützte Bildauswertung für die Realisierung einer Mensch-Roboter-Kollaboration, Masterthesis, HAW Hamburg, 2017

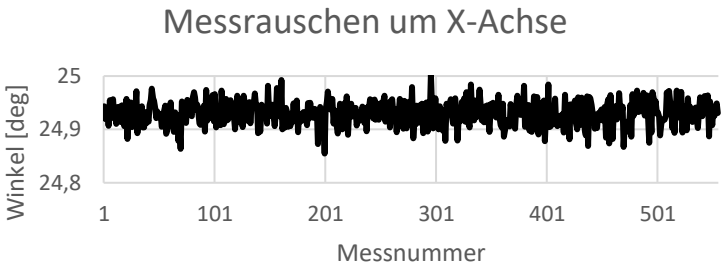
9 Anhang

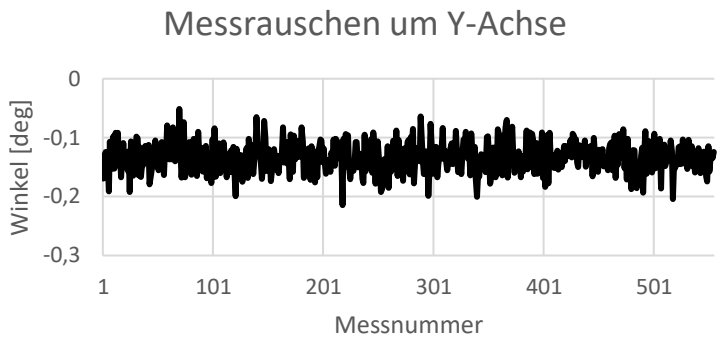
Anhang A

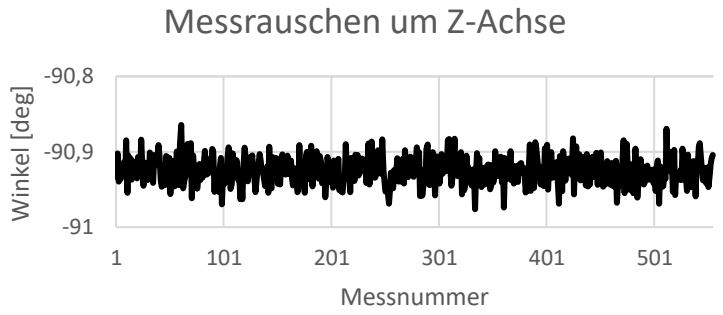




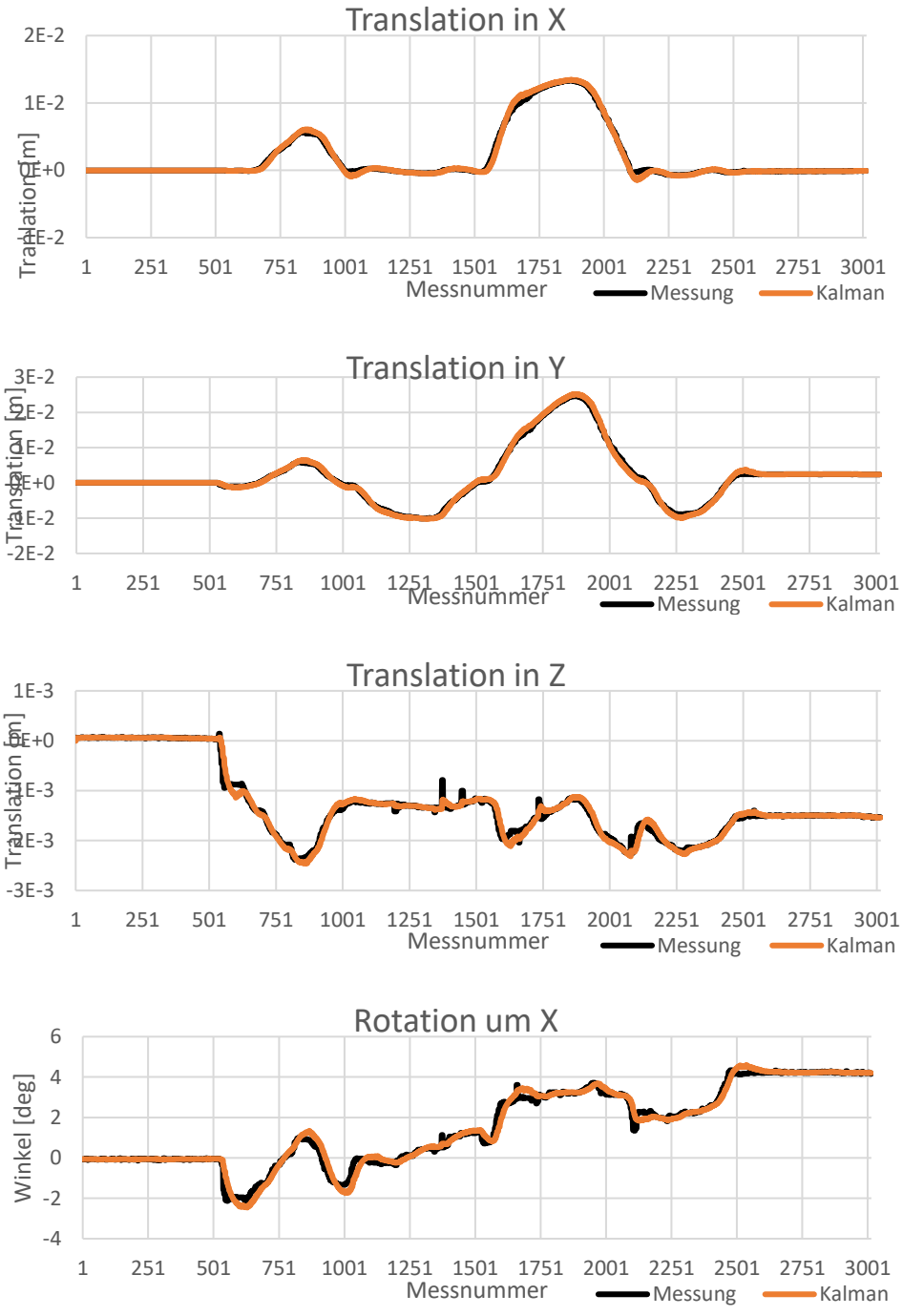


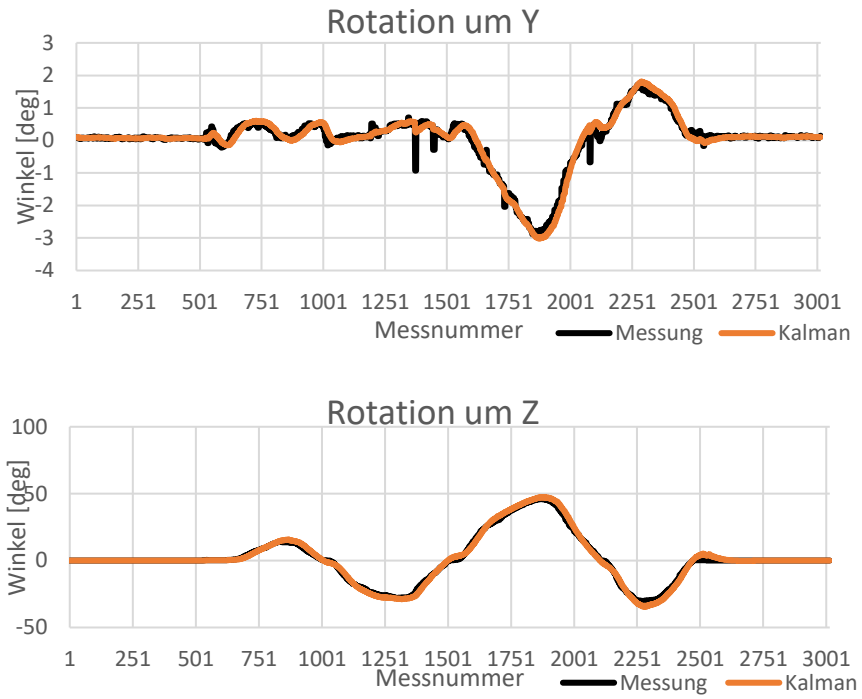

Messrauschen um X-Achse
Winkel [deg]
25
24,9
24,8
1
101
201
301
401
501
Messnummer


Messrauschen um Y-Achse
Winkel [deg]
0
-0,1
-0,2
-0,3
1
101
201
301
401
501
Messnummer


Messrauschen um Z-Achse
Winkel [deg]
-90,8
-90,9
-91
1
101
201
301
401
501
Messnummer

Anhang B


Translation in X
2E-2
1E-2
0E+0
-1E-2
Translation [m]
1
251
501
751
1001
1251
1501
1751
2001
2251
2501
2751
3001
Messnummer
Messung
Kalman
Translation in Y
3E-2
2E-2
1E-2
0E+0
-1E-2
-2E-2
Translation [m]
Messnummer
Messung
Kalman
Translation in Z
1E-3
0E+0
-1E-3
-2E-3
-3E-3
Translation [m]
Messnummer
Messung
Kalman
Rotation um X
6
4
2
0
-2
-4
Winkel [deg]
Messnummer
Messung
Kalman



Anhang C - CD

Inhalt:

-PDF-Datei der Arbeit

-Qt- Projekt remoteRobotControl

-Qt Projekt stereo-Calibration

-Messwerttabellen

-RSI- Konfigurationsdateien und das Demo Programm

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, den _____ _____