

ANWENDUNGSGETRIEBENE
INTEGRATION EINER KAMERA
IN EINEN ROBOTERARM
AM BEISPIEL
EINER SCHACH-APPLIKATION

Fakultät Angewandte Mathematik, Physik und Allgemeinwissenschaften
der technischen Hochschule Nürnberg
Masterstudiengang: Angewandte Mathematik und Physik

Masterarbeit

vorgelegt von

Tim Selig
geboren am 15.06.1997 in Lohr a. Main
Matrikelnummer: 3582116

im Mai 2022

Erstprüfer: Prof. Dr. Oliver Bletz-Siebert
Zweitprüfer: Prof. Dr. Jochen Seufert

Danksagungen

An dieser Stelle möchte ich mich bei all denjenigen bedanken, die mich während der Anfertigung dieser Masterarbeit unterstützt und motiviert haben.

Zuerst gebührt mein Dank Prof. Dr. Oliver Bletz-Siebert und Prof. Dr. Jochen Seufert, die meine Masterarbeit betreut und begutachtet haben. Für die schnelle Beantwortung meiner Fragen, die hilfreichen Anregungen, sowie die konstruktive Kritik bei der Erstellung möchte ich mich herzlich bedanken.

Ein besonderer Dank geht an Herrn Jürgen Schwittek für seine Einführung in die Stereokamera und den Roboterarm sowie die Unterstützung bei der Implementierung einer Client-Server-Architektur.

Zudem möchte ich der Fachhochschule Würzburg-Schweinfurt danken, dass ich die Möglichkeit bekommen habe, mit hochmodernen Hardwarekomponenten arbeiten zu dürfen, durch die diese Arbeit erst ermöglicht wurde.

Außerdem danke ich Frau Nicola Hilkert für das Korrekturlesen meiner Masterarbeit.

Abschließend möchte ich mich bei meinen Eltern bedanken, die mir mein Studium durch ihre Unterstützung ermöglicht haben und stets ein offenes Ohr für mich hatten.

Abstract

Die Intention der vorliegenden Masterarbeit ist die Entwicklung eines interaktiven Schachroboters, welcher nicht nur in der Lage ist, die Schachfiguren zu bewegen, sondern diese auch mithilfe einer Kamera in Echtzeit zu erkennen. Hierfür wird folgende Forschungsfrage gestellt: „Wie kann durch die Integration einer 3D-fähigen Kamera und eines modernen Roboterarms in eine geeignete Programmierumgebung ein sehender, interaktiver Schachroboter realisiert werden?“

Dabei spielt die Kommunikation zwischen Kamera, Roboter und Benutzer eine große Rolle, welche mithilfe einer Client-Server-Architektur realisiert wurde. Um die Figuren auf dem Schachbrett erkennen zu können, wird eine Stereokamera in Verbindung mit der Software HALCON verwendet. Hiermit ist es möglich, eine von der Kamera photographierte Szene als 3D-Punktwolke darzustellen. Innerhalb dieser Punktwolke kann anschließend nach einem Oberflächenmodell der Schachfiguren gesucht werden. Mithilfe verschiedener Koordinatentransformationen werden daraufhin die Koordinaten der gefunden Instanz in das Weltkoordinatensystem des Roboters übersetzt. Für das Greifen und Bewegen der Figuren wird der Roboterarm LBR iiwa (Leichtbauroboter von KUKA) verwendet, welcher mit der Software KUKA Sunrise.OS arbeitet. Die Programmiersprache Python dient als zentrale Schnittstelle von Kamera und Roboter, liefert das Interface für den Benutzer und ist zuständig für die Verarbeitung der Regeln des Schachspiels.

Inhaltsverzeichnis

Danksagungen	I
Abstract	II
Abbildungsverzeichnis	V
Abkürzungsverzeichnis	VII
1 Einführung	1
1.1 Einleitung	1
1.2 Struktur der Arbeit	2
2 Grundlagen	4
2.1 KUKA LBR iiwa	4
2.1.1 KUKA Komponenten	5
2.1.2 Koordinatensysteme und Frames	6
2.1.3 Bewegungsarten	6
2.1.3.1 Point-to-Point-Bewegung (PTP)	6
2.1.3.2 Linearbewegung (LIN)	7
2.2 Stereokamera IDS Ensenso N35	8
2.3 Client-Server-Modell	10
2.4 Schachnotationen	13
2.5 Koordinatentransformation	15
3 Analyse des aktuellen Stands	17
3.1 ChessRoberta	17
3.2 ChessAI	18
3.3 Zusammenfassung und nächste Schritte	18
4 Verwendete Software und Bibliotheken	20
4.1 Eclipse und PyDev	20
4.2 MVTEC HALCON	21
4.3 NxLib und NxView	27
4.4 Sunrise	28
4.5 Versionen	29
5 Projektaufbau	30
5.1 Grundlegende Idee	30

5.2	Hardware und nichttechnische Komponenten	31
5.3	Software	36
6	Bedienung von ChessRobertaVision	38
6.1	Programme und Dateien	38
6.2	Vorbereitungen	39
6.3	Ausführung	43
6.4	Fehlermeldungen	45
6.5	Spezialfälle	46
6.6	Berechnung der Transformationsmatrix für die Hand-Augen-Kalibrierung	47
7	Implementierung	50
7.1	Client-Server-Architektur	50
7.2	Python	51
7.2.1	ClientClass	51
7.2.2	chessCalculator	53
7.2.3	PyClient	56
7.3	HALCON	59
7.3.1	Client-Programm	60
7.3.2	Hauptprogramm	61
7.3.2.1	Initialisierung	61
7.3.2.2	Kommunikation mit Python	78
7.3.3	Hand-Augen-Kalibrierung (HAK) in HALCON	81
7.4	Sunrise	83
7.4.1	Client	84
7.4.2	ChessProgram	84
7.4.3	Hand-Augen-Kalibrierung (HAK) in Sunrise	90
8	Fazit	92
8.1	Zusammenfassung	92
8.2	Ausblick	93
Literaturverzeichnis		95
Eidesstattliche Erklärung		100

Abbildungsverzeichnis

2.1	Kuka LBR iiwa 7 R800 [1]	4
2.2	Komponenten des KUKA LBR iiwa [2, S. 23]	5
2.3	PTP-Bewegung [2, S. 359]	7
2.4	LIN-Bewegung [2, S. 359]	7
2.5	IDS Ensenso N35 3D-Kamera [3]	8
2.6	Auf die Oberfläche einer Tasse projizierte Hilfsmuster sowie deren zugehörige Tiefeninformationen [4]	9
2.7	Visualisierung eines Client-Server-Modells [5]	10
4.1	HDevelop GUI bestehend aus Grafikfenster, Variablenfenster, Operatorfenster und Programmfenster	21
4.2	3D-Objektmodell einer Szene bestehend aus einer 3D-Punktwolke	22
4.3	Oberflächenbasiertes 3D-Matching in einer 3D-Szene	22
4.4	Mit <i>find_shape_model</i> gefundene 2D-Bilder der Schachfiguren visualisiert in der HDevelop GUI	23
4.5	Pixelkoordinatensystem von HALCON. Das Kreuz zeigt den Pixel in der rechten unteren Bildecke an, dessen Zentrum die Koordinaten (6, 5) besitzt [6].	27
4.6	NxView Interface	28
5.1	2-Backen-Parallelgreifer für die Mensch-Roboter-Kollaboration [7, S. 6]	31
5.2	Adaptiv-Greiffinger von Festo [8]	32
5.3	Verwendete Schachfiguren am Beispiel eines weißen Bauers	32
5.4	Musterbilder aller zwölf Schachfiguren	33
5.5	Eine weiße Promotionsfigur aus vier verschiedenen Blickwinkeln	34
5.6	Anfangsstellung einer Partie Schach auf einem Schachbrett von oben	35
5.7	Anfangsstellung einer Partie Schach auf einem Schachbrett von der Seite	35
5.8	Musterbild des Schachbrettes	35
5.9	Architektur des Projekts	36
6.1	Aufbau der nichttechnischen Komponenten von ChessRobertaVision	40
6.2	Ethernet-Einstellungen	41
6.3	Verbindung von Kamera und HALCON mit NxView aktivieren	41
7.1	Beispiel einer Schachposition mit dazugehöriger Figurenstellung und dem mit <i>boardConfig_to_matrix</i> resultierenden numpy-Arrays	53
7.2	Zwei aufeinanderfolgende Schachpositionen und die dazugehörige Liste der Unterschiede, die mit der Funktion <i>fens_to_difference_list</i> , generiert wurde	54

7.3	Musterbild eines leeren Schachbrettes zur Erstellung der formbasierten Modelle aller wichtigen Schachbrettmarkierungen	63
7.4	Musterbilder zur Erstellung der ersten neun formbasierten Modelle am Beispiel des schwarzen Läufers	65
7.5	Beispiel einer <i>winner_list</i> der Schachanfangsposition	67
7.6	Visualisierung der Berechnung des Feldes einer Figur am Beispiel des Bauern auf E2	68
7.7	2D-Bild einer Szene	69
7.8	Bilder einer Szene mit Mustermaske mit der linken bzw. rechten Kamera der Stereokamera	70
7.9	Überlagerungsbild bestehend aus x-, y- und z-Koordinaten einer Szene	70
7.10	Einzelbilder der x-, y- und z-Koordinaten einer 3D-Szene	71
7.11	Veranschaulichung des Koordinatenübergangs mithilfe der <i>Transformations-Matrix_2D_3D</i> zwischen 2D-Bild und z-Komponenten des 3D-Bildes	71
7.12	Bild der z-Komponenten des 3D-Bildes innerhalb einer ROI mit rot markierten Figuren, dessen z-Koordinaten in einem bestimmten Intervall liegen	73
7.13	Visualisierung des ersten Schrittes des Sortieralgorithmus	74
7.14	Durch eine quadratische ROI reduziertes 3D-Objektmodell eines Bereiches auf dem Schachbrett	75
7.15	Durch eine quadratische ROI reduziertes 3D-Objektmodell eines Bereiches auf dem Schachbrett ohne Hintergrund	76
7.16	Gefundene Instanz des Oberflächenmodells eines Würfels visualisiert in dem 3D-Objektmodell einer Szene	77
7.17	Würfelnetz der Promotionsfigur	89

Abkürzungsverzeichnis

FEN	Forsyth–Edwards–Notation
LBR	Leichtbauroboter
iiwa	Intelligent industrial work assistant
MRK	Mensch–Roboter–Kollaboration
TCP	Tool Center Point
IDE	Integrated Development Environment
PTP	Point-to-Point
LIN	Linearbewegung
LINREL	Lineare Relativbewegung
CMOS	Complementary metal–oxide–semiconductor
API	Application Program Interface
CAD	Computer Aided Design
IP	Internet Protocol
PLY	Polygon File Format
ROI	Region of Interest
HAK	Hand–Augen–Kalibrierung
KI	Künstliche Intelligenz

1 Einführung

1.1 Einleitung

Die Bildverarbeitung wurde in den vergangenen Jahren für unzählige industrielle und nicht-industrielle Anwendungen immer bedeutender und hat sich zu einer vielseitigen und leistungsfähigen Technologie in verschiedensten Branchen entwickelt. Hierzu zählen allgemeine Anwendungen wie die Gesichtserkennung, die bereits in jedem Mobiltelefon integriert ist, aber auch die Erkennung von Fehlern oder Verunreinigungen in komplexen Strukturen für industrielle Zwecke. Insbesondere in der digitalen Transformation der Produktion ist die Bildverarbeitung nicht mehr wegzudenken. Durch den „Sehsinn“ der Maschinen können wichtige sensorische Aufgaben wie z. B. das Identifizieren, Verifizieren und Kontrollieren erledigt werden. Zudem sind im industriellen Bereich Roboter in vielen Automatisierungsprozessen unverzichtbar, um einen sicheren und effizienten Arbeitsablauf mit gleichbleibender Qualität zu gewährleisten. Hierbei kommen häufig sogenannte Pick-and-Place-Anwendungen zum Einsatz, bei der Objekte vom Roboter aufgenommen und diese in bestimmter Position und Orientierung an einer anderen Stelle platziert werden.

Das Projekt namens „ChessRobertaVision“ hat die Intention die Komponenten aus Robotik und Bildverarbeitung zu kombinieren, um eine intelligente Pick-and-Place-Anwendung zu realisieren, deren Methoden auf verschiedene industrielle Applikationen übertragbar sind. Da das Brettspiel Schach seit Jahrhunderten von Menschen jeden Alters auf der ganzen Welt gespielt wird und durch das Greifen und Ablegen unterschiedlicher Figuren optimale Voraussetzungen bereitgestellt werden, ist das Strategiespiel ein geeignetes Beispiel zur Veranschaulichung des Vorhabens. Zusammengefasst ist das Ziel der vorliegenden Arbeit, mit einem interaktiven und „sehenden“ Roboter Schach spielen zu können.

Für das Greifen und Bewegen der Figuren ist der sensitive Roboterarm LBR iiwa 7 R800 von KUKA zuständig, der mit der Systemsoftware Sunrise.OS arbeitet. Die Erkennung von Schachbrett und Figuren übernimmt die 3D-fähige Kamera IDS Ensenso N35 in Verbindung mit der Bildverarbeitungssoftware HALCON von MVTEC. Als zentrale Schnittstelle zwischen

Kamera und Roboter wird ein externer Computer mit der Programmiersprache Python verwendet, welcher zusätzlich das Interface für den Benutzer bereitstellt und die Regeln des Schachspiels verarbeitet. Des Weiteren wird eine Client-Server-Architektur benötigt, damit die drei essentiellen Software-Komponenten, bestehend aus Sunrise, HALCON und Python, miteinander kommunizieren können.

Im eigentlichen Spiel führen Roboter und Benutzer alternierend Schachzüge auf dem realen Brett aus. Während der Nutzer am Zug ist, photographiert die Kamera die Szene in kurzen Abständen, bis eine neue Stellung der Figuren erkannt wird und der Zug anhand der gegebenen Bilder mit Python berechnet werden kann. Für die Generierung des Roboterzugs wird die Open-Source-Schachengine Stockfish verwendet. Die Koordinaten der zu bewegenden Figuren werden mit HALCON berechnet und daraufhin mit speziellen Transformationsmatrizen in das Weltkoordinatensystem des Roboters transformiert. Abschließend wird die gegebene Koordinatenfolge mit einer Bewegungssoftware in Sunrise abgearbeitet und der Zug mit dem Roboterarm auf dem Brett ausgeführt.

In dieser Arbeit wird aus Gründen der besseren Lesbarkeit das generische Maskulinum verwendet. Weibliche und anderweitige Geschlechteridentitäten sind dabei ausdrücklich eingeschlossen.

1.2 Struktur der Arbeit

Die Struktur der Arbeit ist folgendermaßen aufgebaut:

- In **Kap. 2** werden die Grundlagen in Bezug auf den Roboterarm, die Kamera und das Client-Server-Modell vorgestellt, die für das Verständnis der Arbeit benötigt werden. Des Weiteren werden die Notationen des Schachspiels erläutert und ein mathematisches Verfahren zur Berechnung von Transformationsmatrizen erklärt.
- In **Kap. 3** wird der aktuelle Stand des Schachprogramms vor dem Start der vorliegenden Arbeit analysiert. Anhand dieser Analyse werden die Schwächen des Projektes aufgezeigt und zusammengefasst, wie die nächsten Schritte das Programm in Bezug auf die Benutzerfreundlichkeit verbessern.
- Anschließend werden in **Kap. 4** alle verwendeten Softwares und Bibliotheken erläutert, die für die Umsetzung der Arbeit nötig waren.

- **Kap. 5** beinhaltet den spezifischen Aufbau des Projektes. Hierbei wird zunächst auf die grundlegende Idee zur Realisierung eines sehenden Schachroboters eingegangen. Daraufhin werden alle Hardware- und Software-Komponenten ausführlich beschrieben und die Architektur dargestellt.
- In **Kap. 6** wird dem Leser eine Bedienungsanleitung präsentiert, die alle Informationen zur Anwendung des Projektes bereitstellt. Alle nötigen Vorbereitungen sowie Fehlermeldungen oder bestimmte Spezialfälle werden in diesem Kapitel detailliert beschrieben.
- **Kap. 7** geht ausführlich auf die Implementierung der Client-Server-Architektur und der Hauptprogramme ein. Hier wird der vollständige Vorgang vom Bild des Schachbrettes bis hin zur Ausführung des Zuges mit dem Roboter deutlich.
- Abschließend liefert **Kap. 8** eine Zusammenfassung der Arbeit sowie einen Ausblick auf zukünftige Verbesserungen und Erweiterungen des Schachroboters.

2 Grundlagen

2.1 KUKA LBR iiwa

Diese Arbeit wurde mithilfe des KUKA LBR (Leichtbauroboter) iiwa (intelligent industrial work assistant) 7 R800 durchgeführt (s. Abb. 2.1). Der Roboter ist MRK-fähig (Mensch-Roboter-Kollaboration), wodurch die Fähigkeit des Menschen mit der Effizienz und Präzision einer Maschine kombiniert wird [9].

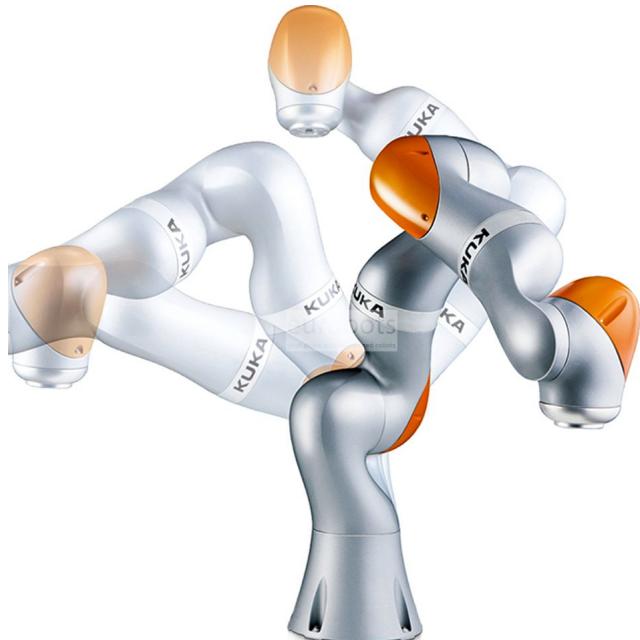


Abbildung 2.1 Kuka LBR iiwa 7 R800 [1]

Hiermit entwickelte KUKA den ersten in Serie gefertigten, sensitiven Roboter und eröffnet damit unzählige neue Möglichkeiten im Bereich der Automatisierung.

Er kann in enger Zusammenarbeit mit dem Menschen hochsensible Aufgaben lösen. Die Einsatzbereiche reichen von Montage- oder Klebeprozesse in der industriellen Fertigung bis hin zu Applikationen im Medizin- oder Servicesektor [10].

2.1.1 KUKA Komponenten

KUKA Sunrise.OS ist die Systemsoftware für den KUKA LBR iiwa und stellt alle Funktionen zum Betrieb von Leichtbaurobotern zur Verfügung [11]. KUKA Sunrise.Workbench ist das Werkzeug zur Inbetriebnahme einer Station und zur Entwicklung von Roboterapplikationen. Das SmartPAD wird hauptsächlich für Aufgaben benötigt, die aus praktischen oder sicherheitstechnischen Gründen nicht mit dem KUKA Sunrise.Workbench ausgeführt werden können. Es werden z. B. Achsen justiert, Werkzeuge vermessen oder Punkte gespeichert [2, S. 23].

Die nachfolgende Abbildung veranschaulicht die einzelnen Komponenten des KUKA LBR iiwa sowie deren Vernetzung.

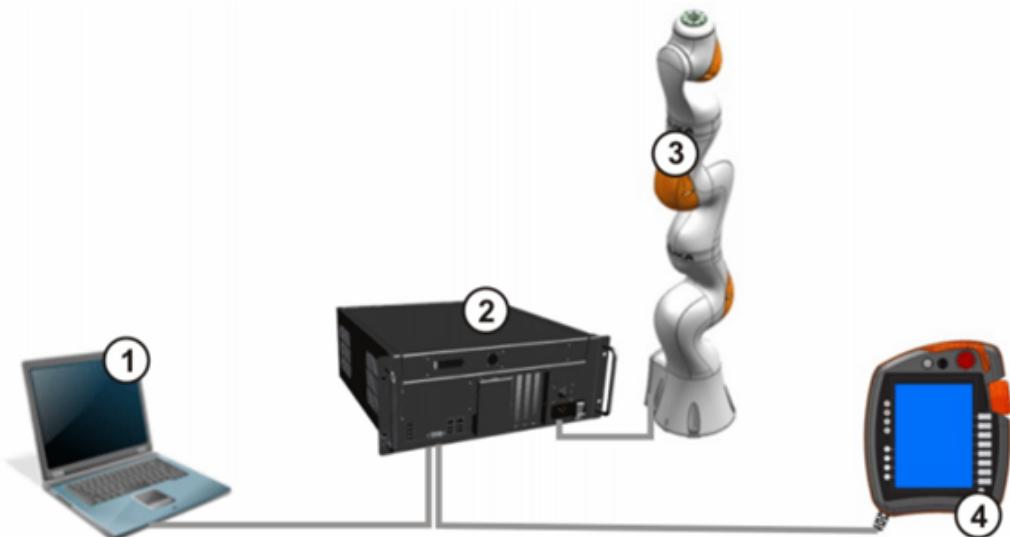


Abbildung 2.2 Komponenten des KUKA LBR iiwa [2, S. 23]

1. Entwicklungsrechner mit dem Werkzeug KUKA Sunrise.Workbench
2. Robotersteuerung KUKA Sunrise Cabinet
3. Roboterarm
4. Bedienhandgerät KUKA SmartPAD

Der Roboterarm besteht dabei aus sieben Gelenken, einem Medien-Flansch und einem Aufsatz zum Greifen. Der Medien-Flansch ist eine universelle Schnittstelle, wodurch elektrische und pneumatische Komponenten am Roboter angeschlossen und über das Roboterprogramm konfiguriert werden können [12, S. 7].

2.1.2 Koordinatensysteme und Frames

Der Roboter hat Zugriff auf verschiedene Bezugskoordinatensysteme. Hierzu gehören das Weltkoordinatensystem im „Fuß“ des Roboters, das Flansch-Koordinatensystem im Medien-Flansch sowie das TCP-Koordinatensystem im Tool Center Point (TCP). Der TCP kann als Arbeitspunkt eines Werkzeuges beschrieben werden, wobei auch mehrere Arbeitspunkte definiert werden können [2, S. 21]. Für diese Arbeit wurde die Spitze des Greifers als TCP initialisiert.

Die Darstellung der Koordinaten erfolgt mithilfe von Frames. Es handelt sich dabei um Koordinatensysteme, welche durch ihre Position und Orientierung bezüglich eines Referenzsystems beschrieben werden [2, S. 19/20]. Hier werden die meisten Frames bzgl. des Weltkoordinatensystems angegeben. Folgende sechs Koordinaten werden zur Beschreibung eines Frames benötigt:

- Translationskoordinaten:
 - x-Komponente
 - y-Komponente
 - z-Komponente
- Rotationskoordinaten:
 - AlphaRad-Komponente: Drehung um die z-Achse
 - BetaRad-Komponente: Drehung um die y-Achse
 - GammaRad-Komponente: Drehung um die x-Achse

2.1.3 Bewegungsarten

Der Roboter ist in der Lage, verschiedene Bewegungsarten auszuführen, dazu gehören unter anderem die Point-to-Point-Bewegung (PTP), die Linearbewegung (LIN), die Circular-Bewegung (CIRC) und die Polynomialbewegung (SPL) [2, S. 358]. In der vorliegenden Arbeit wird jedoch ausschließlich mit PTP und LIN gearbeitet.

2.1.3.1 Point-to-Point-Bewegung (PTP)

Bei einer PTP-Bewegung führt der Roboter den TCP entlang der schnellsten Bahn zum Zielpunkt. In der Regel ist die schnellste Bahn nicht automatisch die kürzeste, also keine

Gerade. Geschwungene Bahnen können schneller ausgeführt werden, da sich die Roboterachsen zeitgleich und rotatorisch bewegen. In Abb. 2.3 ist eine mögliche Bewegung zwischen zwei Punkten visualisiert. Dabei kann der exakte Verlauf der Bewegung zwar nicht vorgesehen werden, aber dieser ist immer gleich, solange die Rahmenbedingungen nicht verändert werden [2, S. 358].

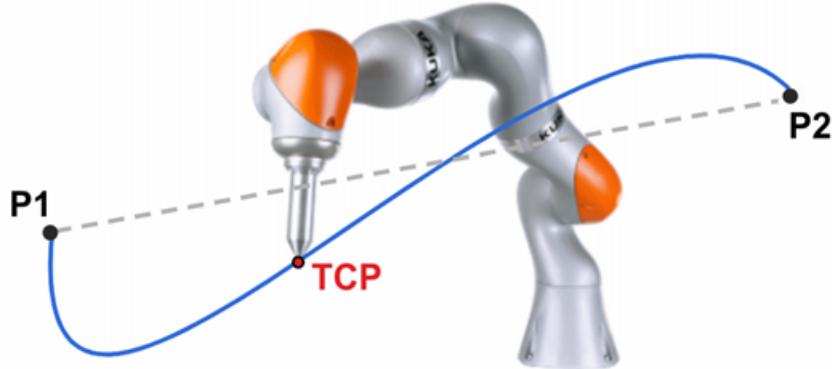


Abbildung 2.3 PTP-Bewegung [2, S. 359]

2.1.3.2 Linearbewegung (LIN)

Bei der LIN-Bewegung führt der Roboter den TCP mit einer definierten Geschwindigkeit entlang einer Geraden im Raum zum Zielpunkt. Dabei wird die Roboterkonfiguration der Zielpose nicht berücksichtigt [2, S. 359]. In Abb. 2.4 ist eine LIN-Bewegung zwischen zwei Punkten dargestellt.

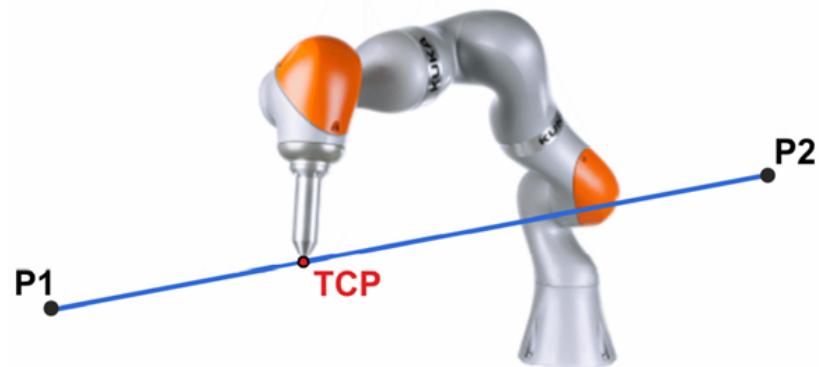


Abbildung 2.4 LIN-Bewegung [2, S. 359]

Bei der Programmierung einer LIN-Bewegung wird außerdem zwischen einer normalen LIN-Bewegung und einer linearen Relativbewegung (LINREL) unterschieden. Hierbei werden die Koordinaten des Zielpunkts relativ zur Zielposition der Vorgängerbewegung angegeben [2, S. 404]. Das hat den Vorteil, keine Koordinaten berechnen zu müssen, wenn sich der Roboter beispielsweise 10 cm in positive oder negative z-Richtung bewegen soll.

2.2 Stereokamera IDS Ensenso N35

Für diese Arbeit wurde die Stereokamera Ensenso N35 von IDS verwendet, welche in Abb. 2.5 visualisiert ist.



Abbildung 2.5 IDS Ensenso N35 3D-Kamera [3]

Die Kamera besteht aus einem kompakten Aluminiumgehäuse mit zwei monochromatischen „komplementären-Metall-Oxid-Halbleiter-Sensoren“, kurz CMOS-Sensoren, und einem Projektor. Die CMOS-Sensoren haben eine Auflösung von 1280×1024 Pixel, 6 mm Brennweite und arbeiten mit einem Global Shutter, um scharfe Bilder von sich bewegenden Objekten zu ermöglichen. Der Projektor verwendet blaues Licht mit einer Wellenlänge von 465 nm [3] und erzeugt mit Hilfe einer sogenannten „Mustermaske“ eine kontrastreiche Textur auf der Objektoberfläche, wie in Abb. 2.6 dargestellt. Die projizierte Textur ergänzt die schwache oder nicht vorhandene Oberflächenstruktur des Objekts. Das Ergebnis ist eine detailliertere Disparitätskarte und eine vollständigere und homogenere Tiefeninformation der Szene [4].

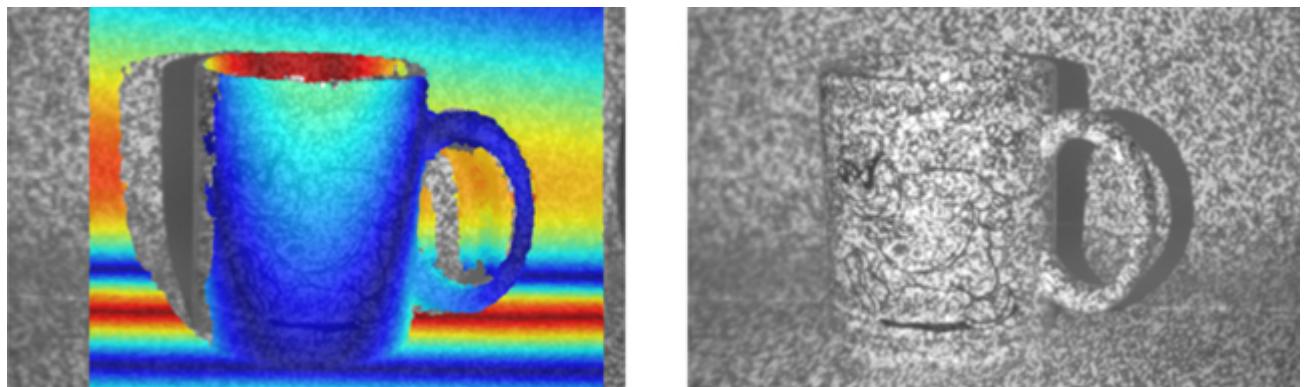


Abbildung 2.6 Auf die Oberfläche einer Tasse projizierte Hilfsmuster sowie deren zugehörige Tiefeninformationen [4]

Das Modell besitzt zudem eine integrierte FlexView-Technologie, womit der Detailgrad statischer Szenen verbessert werden kann. Durch ein mechanisches System mit einem piezoelektrischen Aktuator kann die Position der Mustermaske in den Projektionsstrahlen in kleinen Schritten verschoben werden, mit dem Ergebnis einer variierenden Textur auf der Objektoberfläche. Die Aufnahme mehrerer Bildpaare mit unterschiedlichen Texturen der gleichen Objektszene erzeugt sehr viel mehr Bildpunkte, wodurch der Matching-Algorithmus deutlich verbesserte Disparitätskarten berechnet [4], so dass die Kamera besonders für die 3D-Erfassung von unbewegten Objekten und für Arbeitsabstände von bis zu 3 m geeignet ist. Des Weiteren sind die N35 Stereokameras vorkalibriert und verfügen über eine MVtec HALCON-Schnittstelle (s. Kap. 4.2) sowie eine objektorientierte API [3]. Zusätzliche produktsspezifische Daten können unter <https://en.ids-imaging.com/ensenso-selector.html> nachgelesen werden, nachdem das Modell N35 mit blauem Licht ausgewählt wird.

Nachfolgend sind noch einige Anwendungsmöglichkeiten für die Stereokamera IDS Ensenso N35 aufgelistet [3]:

- 3D-Objekterkennung sowie die Klassifizierung und Lokalisierung des Objektes,
- 3D-Objekt-Rekonstruktion,
- Roboteranwendungen, z. B. Auswahl eines Objektes in einem Behälter,
- Logistik-Automatisierung, z. B. (De-)Palettierung,
- Fabrik-Automatisierung und
- Automatische Lagersysteme.

2.3 Client-Server-Modell

Das Client-Server-Modell ist ein Architekturkonzept, um Dienste und Aufgaben innerhalb eines Netzwerkes zu verteilen. Hierbei werden die Dienste von den Servern bereitgestellt und können von beliebig vielen Clients verwendet werden [13]. Alternative Bezeichnungen für das Client-Server-Modell sind Client-Server-Konzept, -Architektur, -System oder -Prinzip [14].

Für die folgenden Definitionen von Server und Client ist es wichtig zu verstehen, dass es sich bei den Begriffen nicht um die Hardware, sondern um bestimmte Rollen handelt [13]. Der physische Rechner, auf dem ein oder mehrere Server-Programme laufen, wird als Host bezeichnet [15].

Server: In der Informatik ist ein Server ein Dienstleister, der innerhalb eines Computersystems in der Lage ist, Daten oder Ressourcen zur Verfügung zu stellen. Das System kann dabei aus einzelnen Computern, aber auch aus einem Netzwerk mehrerer Computer bestehen, wie in Abb. 2.7 dargestellt [15]. Konkret nimmt der Server Anfragen von Clients entgegen, verarbeitet sie und sendet die entsprechende Antwort an die Clients zurück [13].

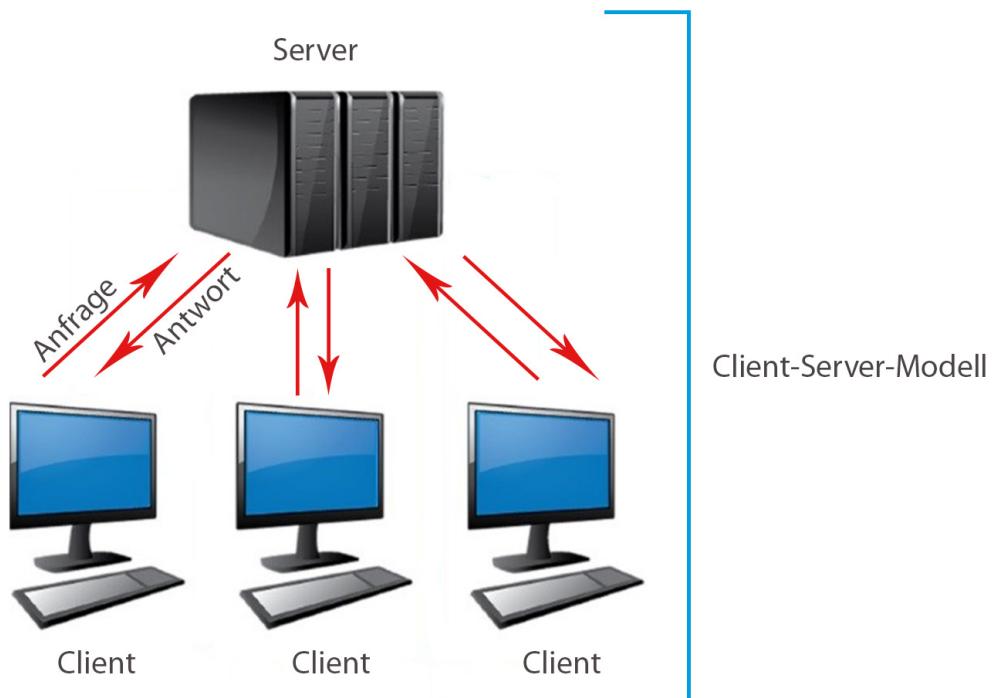


Abbildung 2.7 Visualisierung eines Client-Server-Modells [5]

Client: Ein Client ist eine Anwendung, die innerhalb eines Netzwerkes verschiedene Dienste eines Servers in Anspruch nehmen kann [15]. Außerdem liefert der Client eine Schnittstelle zum Benutzer, indem er dessen angefragte Dienste an den Server sendet. Anschließend können die Daten, die auf dem Server verarbeitet wurden, auf dem Endgerät des Nutzers abgelesen werden [14]. Die Initiative für die Kommunikation geht dabei stets vom Client aus. Er ist dafür zuständig die Anfrage zu formulieren, sie entweder lokal oder über ein Netzwerk zu senden und die erhaltene Antwort zu interpretieren [13]. In der Praxis wird ein Client häufig als Clientanwendung oder Clientprogramm bezeichnet [14].

Bei einem Client-Server-Modell wird der Datenaustausch von Server und Client über bestimmte Übertragungsprotokolle ermöglicht. Damit der Client dauerhaft in der Lage ist auf den Server zuzugreifen, um dessen Dienste beliebig nutzen zu können, befinden sich die Server während einer Kommunikation dauerhaft im Betrieb. Beispiele für bekannte Client-Server-Modelle sind nachfolgend aufgelistet [14]:

- **Webserver:**

Zunächst sendet der Client seine IP-Adresse an den Webserver und fordert diesen auf, eine bestimmte Website zu öffnen. Daraufhin wartet der Client auf die Antwort des Servers, mit der er die Möglichkeit hat, auf die Website zuzugreifen.

- **Druck-Server:**

Über einen Client wird ein Druckauftrag an den Druck-Server gesendet, welcher dann an den Drucker weitergeleitet wird.

- **E-Mail-Server:**

Um E-Mails verwalten und speichern zu können, stellt ein E-Mail-Server jedem Benutzer ein Postfach zur Verfügung, auf das über ein E-Mail-Client zugegriffen werden kann.

Weitere wichtige Begriffe zum Verständnis von Client-Server-Modellen sind IP-Adressen, Ports, Portnummern und Sockets, welche anschließend erklärt werden.

IP-Adresse: Eine internet protocol (IP)-Adresse ist eine Identifikationsnummer, welche mit einem bestimmten Computer oder Computernetzwerk verbunden ist. Mithilfe einer Internetverbindung ermöglicht die IP-Adresse das Senden und Empfangen von Informationen [16].

Port und Portnummer: Ein Port ist ein virtueller Endpunkt einer Netzwerkverbindung, welcher zum Austausch von Informationen dient [17]. Computer können mithilfe von Ports zwischen verschiedenen Diensten und Arten des Datenverkehrs unterscheiden. Beispielsweise sind E-Mails mit einem anderen Port verbunden als Websites, wobei jedem Port eine individuelle Portnummer zugewiesen wird und einige Ports bereits für bestimmte Protokolle reserviert sind, beispielsweise werden alle Hypertext Transfer Protocol (HTTP)-Nachrichten an Port 80 gesendet. Ähnlich wie IP-Adressen das Senden an bzw. von bestimmten Geräten ermöglichen, sind Portnummern in der Lage bestimmte Dienste oder Anwendungen innerhalb eines Gerätes anzubieten [18].

Socket: Ein Socket ist ein Endpunkt einer zweiseitigen Kommunikationsverbindung zwischen zwei im Netz laufenden Programmen. Ein Endpunkt bezeichnet hierbei die Kombination aus IP-Adresse und Portnummer. Da jede TCP-Verbindung durch ihre beiden Endpunkte eindeutig identifizierbar ist, können mehrere Verbindungen zwischen Host und Server hergestellt werden [19].

Abschließend werden zwei alternative Modelle sowie einige Vor- und Nachteile eines Client-Server-Modells aufgelistet:

Alternative Modelle:

Zur Verteilung von Aufgaben und Diensten existieren noch weitere Architekturmodelle, wie das Peer-to-Peer-Modell oder das Master-Slave-Modell [13]:

- **Peer-to-Peer-Modell:**

Im Gegensatz zum Client-Server-Modell ist ein sogenannter „Peer“ im Peer-to-Peer-Modell zeitgleich ein Server und ein Client. Hierbei gibt es auch keine Hierarchie, stattdessen sind die Peers untereinander gleichberechtigt.

- **Master-Slave-Modell:**

In einem Master-Slave-Modell muss der „Master“ den „Slaves“ das Recht geben, auf gemeinsame Ressourcen zuzugreifen. Im Vergleich zum Client-Server-Modell kann die Initiative für eine Anfrage nicht von einen Slave ausgehen, stattdessen muss er auf die Berechtigung zur Kommunikation vom Master warten.

Vorteile von Client-Server-Modellen ([13], [14]):

- Zentrale Verwaltung:

Es können alle Ressourcen der Anwender, wie z. B. eine Datenbank, zentral verwaltet werden, da der Server im Zentrum des Netzwerks steht.

- Hohe Zugriffssicherheit:

Es besteht eine hohe Sicherheit und leichte Zugriffskontrolle, da sich die Clients bei jedem Zugriff auf den Server authentifizieren müssen.

- Erweiterbares Netzwerk:

Es können beliebig viele Clients gelöscht bzw. hinzugefügt werden, ohne die Leistung des Netzwerkes zu beeinträchtigen.

- Standortunabhängigkeit:

Mithilfe der zentralen Datenspeicherung werden flexible Einsatzmöglichkeiten und eine standortunabhängige Nutzung ermöglicht.

Nachteile von Client-Server-Modellen ([13], [14]):

- Ausfall des Servers:

Der Ausfall eines Servers bedeutet in den meisten Fällen einen Totalausfall des Netzwerks, da das gesamte Netzwerk um den Server herum aufgebaut ist und dem Client der Zugriff auf Daten und Dienste des Servers verwehrt bleibt. Um diesem Problem entgegenzuwirken, ist es möglich, mehrere Server einzusetzen (Stichwort: Fall-back-Lösung).

- Überlastung des Servers:

Für jede Verbindung zwischen Client und Server gibt es eine Bandbreite, die zur Verfügung steht und mit steigender Anzahl an Verbindungen sinkt. Deshalb kann es zu langen Wartezeiten kommen, wenn verschiedene Clients gleichzeitig mehrere Anfragen an den Server senden.

- Hohe Anschaffungskosten:

Der Erwerb leistungsstarker Server ist in der Regel mit hohen Kosten verbunden.

2.4 Schachnotationen

Dieses Kapitel geht nicht auf das Regelwerk des Schachspiels ein, sondern liefert lediglich einen Einblick in die Notationen der Schachzüge und -positionen, welche für den internen

Code benötigt werden.

Jede Schachfigur wird von einem Buchstaben repräsentiert, welcher vom Anfangsbuchstaben der englischen Bezeichnungen abgeleitet wird:

- Bauer = P, p (*engl.*: Pawn)
- Turm = R, r (*engl.*: Rook)
- Springer = N, n (*engl.*: Knight)
- Läufer = B, b (*engl.*: Bishop)
- Dame = Q, q (*engl.*: Queen)
- König = K, k (*engl.*: King)

Hierbei kennzeichnen großgeschriebene Buchstaben die weißen Figuren und kleingeschriebene Buchstaben die schwarzen Figuren.

Für die Züge wurde in diesem Projekt eine algebraische Notation gewählt, wobei das Start- und Zielfeld angegeben wird (z. B. E2E4, D5E4, ...). Hierbei ist darauf zu achten, dass auch spezielle Züge, wie das Schlagen einer Figur oder die Rochade, auf diese Weise angegeben werden. Im Falle der Rochade wird das Start- und Zielfeld des Königs angegeben (z. B. E1G1). Die einzigen Ausnahmen bilden Bauernumwandlungen, denn für diese Züge wird, unabhängig von der Farbe, zusätzlich ein Kleinbuchstabe angehängt, welcher die Umwandlungsfigur kennzeichnet (z. B. B7B8q).

Die Position des Spielfelds wird in der Forsyth-Edwards-Notation (FEN) angegeben. Eine FEN besteht aus sechs Teilen, die mit einem Leerzeichen voneinander getrennt sind [20].

1. Figurenstellung:

Die Stellung der Figuren wird zeilenweise von der achten bis zur ersten Zeile ermittelt, welche jeweils mit einem Schrägstrich „/“ voneinander getrennt sind. Jede Zeile untersucht dabei die Felder von A bis H. Eine Figur wird mit ihrem zugehörigen Buchstaben (s. o.) gekennzeichnet, und aufeinanderfolgende leere Felder werden mit einer Dezimalzahl dargestellt, die die Anzahl der leeren Felder angibt. Daraus ergibt sich folgende Form für die Anfangsposition eines Schachspiels:

„rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR“

2. Spieler am Zug:

Hier wird der Spieler angegeben, der in dieser Stellung am Zug ist. Für Weiß wird der Buchstabe „w“ und für Schwarz der Buchstabe „b“ verwendet.

3. Rochaderechte:

Die Rochaderechte geben die noch erlaubten Rochaden an ($K \hat{=}$ kurze Rochade für Weiß, $Q \hat{=}$ lange Rochade für Weiß, $k \hat{=}$ kurze Rochade für Schwarz, $q \hat{=}$ lange Rochade für Schwarz). Wenn keine der Rochaden mehr möglich ist, wird das Symbol „-“ verwendet.

4. Möglicher En-passant-Schlag:

Wenn im letzten Zug ein Bauer zwei Felder vorgerückt ist, wird hier das übersprungene Feld angegeben, unabhängig davon, ob ein En-passant-Schlag auf diesem Feld möglich ist, ansonsten wird das Symbol „-“ verwendet.

5. Anzahl hintereinander gespielter Halbzüge:

Dieser Teil gibt die Anzahl der Halbzüge seit dem letzten Bauernzug oder Schlagen einer Figur an. Das wird für die 75-Züge-Regel benötigt.

6. Nummer des nächsten Zuges:

Beginnend mit 1 erhöht sich dieser Wert nach jedem Zug von Schwarz um 1, womit die Länge des Spiels angegeben wird.

Der vollständige FEN-String für die Startposition eines Schachspiels sieht dann folgendermaßen aus: „rnbqkbnr/pppppppp/8/8/8/8/PPPPPPPP/RNBQKBNR w KQkq - 0 1“

Zum besseren Verständnis sind anschließend drei Züge nacheinander ausgeführt worden, woraus sich die zugehörigen FENs ergeben:

- E2E4: rnbqkbnr/pppppppp/8/8/4P3/8/PPPP1PPP/RNBQKBNR w KQkq e3 0 1
- E7E5: rnbqkbnr/ppp1ppp/8/4p3/4P3/8/PPPP1PPP/RNBQKBNR w KQkq e6 0 2
- B1C3: rnbqkbnr/ppp1ppp/8/4p3/4P3/2N5/PPPP1PPP/R1BQKBNR w KQkq - 1 2

2.5 Koordinatentransformation

Koordinatentransformationen werden angewendet, wenn eine Aufgabe in einem anderen Koordinatensystem leichter zu lösen ist.

Definition:

Koordinatentransformationen sind Abbildungen

$$H : \mathbb{R}^n \rightarrow \mathbb{R}^n, x \mapsto Hx + a,$$

die durch Multiplikationen eines Vektors $x \in \mathbb{R}^n$ mit einer Matrix $H \in \mathbb{R}^{n \times n}$ und einer Translation um den Verschiebungsvektor $a \in \mathbb{R}^n$ gegeben sind [21].

Zu den elementaren Koordinatentransformationen gehören beispielsweise die Drehung, Skalierung, Scherung oder Translation [22].

Ein wichtiges Konzept im Zusammenhang mit Transformationsaufgaben ist die Einführung von homogenen Koordinaten. Dabei wird dem Vektor $x \in \mathbb{R}^n$ eine weitere, virtuelle Koordinate hinzugefügt, um eine geschlossene Bearbeitung der Aufgaben zu ermöglichen. So können die Translationen mithilfe von Matrixmultiplikation, anstelle von Additionen, dargestellt werden [23]. Die homogenen Koordinaten stehen mit den kartesischen Koordinaten in folgendem Zusammenhang (in 3D):

$$\vec{x}_{\text{kartesisch}} = \begin{pmatrix} x/\lambda \\ y/\lambda \\ z/\lambda \end{pmatrix} \quad \rightarrow \quad \vec{x}_{\text{homogen}} = \begin{pmatrix} x \\ y \\ z \\ \lambda \end{pmatrix} \quad \text{für } \lambda \in \mathbb{R} \setminus \{0\}$$

Im Spezialfall $\lambda = 1$ stimmen die ersten drei Komponenten der Darstellungen eines Punktes überein [24].

Optimierungsverfahren:

In der praktischen Anwendung kommt es häufig vor, dass einige Punkte als Koordinaten bzgl. unterschiedlicher Koordinatensysteme gegeben sind. In diesem Fall kann mithilfe eines Optimierungsverfahrens die Transformationsmatrix H näherungsweise bestimmt werden:

Es gilt $n, m \in \mathbb{N}$. Seien m korrespondierende Punkte $P_i, Q_i \in \mathbb{R}^n$ für $i = 1, \dots, m$ gegeben. Gesucht ist eine homogene Matrix $H \in \mathbb{R}^{(n+1) \times (n+1)}$, welche die Abstände zwischen den transformierten Eingabepunkten P_i und den Punkten Q_i minimiert, wie in der folgenden Optimierungsaufgabe beschrieben. (Hierbei werden P_i und Q_i als homogenen Koordinaten für $\lambda = 1$ verwendet) [25]:

$$\arg \min_H \sum_{i=1}^m \left\| \begin{pmatrix} Q_i \\ 1 \end{pmatrix} - H \begin{pmatrix} P_i \\ 1 \end{pmatrix} \right\|_2^2$$

3 Analyse des aktuellen Stands

Der Projekttitel der vorliegenden Arbeit lautet „ChessRobertaVision“. In diesem Kapitel werden die Vorgängerprojekte der Arbeit namens „ChessRoberta“ und „ChessAI“ beschrieben, um die Verbesserung der Benutzerfreundlichkeit von „ChessRobertaVision“ zu verdeutlichen. Der Quellcode der Projekte wurde auf GitHub veröffentlicht und kann unter folgenden Links nachgelesen werden:

- ChessRoberta: <https://github.com/FHWS-FANG/ChessRoberta>
- ChessAI: <https://github.com/TimSelig/ChessAI>
- ChessRobertaVision: <https://github.com/FHWS-FANG/ChessRobertaVision>

3.1 ChessRoberta

ChessRoberta ist eine Software für einen Schachroboter, der vom Autor entwickelt wurde, wobei im Unterschied zu vorliegender Arbeit, lediglich ein Roboterarm und ein externer Computer ohne Kamera verwendet werden. Das Programm ist auf einer Client-Server-Architektur aufgebaut, um verschiedene Komponenten miteinander kommunizieren zu lassen. Als Server agiert hierbei der KUKA LBR iiwa 7 R800 (s. Kap. 2.1) und als Client ein externer Computer, welcher außerdem die Benutzeroberfläche beinhaltet. Das gesamte Projekt verwendet dabei die Programmiersprache Java.

Da der Roboter aufgrund einer fehlenden Kamera das Schachbrett und die Figuren nicht erkennt, werden für die Initialisierung die Koordinaten der beiden Felder A1 und H8 mithilfe des Roboters eingelesen. Die restlichen Felder können daraufhin über die beiden Koordinaten berechnet werden. Diese Methode hat den Nachteil, dass der Roboter die Figuren stets in der Mitte eines Feldes greift. Aus sicherheitstechnischen Gründen und zur Vermeidung von Kollisionen übernimmt der Roboter die Züge des Menschen mit, sodass die Figuren immer wieder exakt mittig auf die Felder gesetzt werden.

Die Vorbereitung eines konkreten Spiels sieht dann folgendermaßen aus:

1. Aufbau des Schachbrettes.
2. Initialisierung der Felder A1 und H8.
3. Aufbau der Spielfiguren.
4. Starten des Server- und Client-Programms.
5. Eingabe von weiß bzw. schwarz in den Client, um die gewünschte Spielfarbe festzulegen.

Daraufhin kann das eigentliche Spiel starten. Während eines Spiels werden nun, abhängig von der gewählten Farbe, abwechselnd entweder Züge vom Benutzer eingegeben und vom Roboter ausgeführt oder vom Computer berechnet und ebenfalls vom Roboter ausgeführt, bis das Spiel zu Ende ist.

3.2 ChessAI

ChessAI ist eine Software für eine trainierbare Schach-KI (künstliche Intelligenz), die vom Autor in Zusammenarbeit mit Herrn Stefan Schramm entwickelt wurde. Indem die KI Schachspiele gegen sich selbst spielt, werden mithilfe von Reinforcement-Learning-Methoden die Hyperparameter in einem Faltungsnetz iterativ angepasst.

3.3 Zusammenfassung und nächste Schritte

Dieses Kapitel geht auf die Vorgängerprojekte ChessRoberta und ChessAI ein. Hierbei wird der größte Schwachpunkt von ChessRoberta deutlich, welcher einen Anlass zur Weiterentwicklung liefert. Es handelt sich dabei um die Tatsache, dass der Spieler nicht die Möglichkeit hat, die Figuren selbstständig zu bewegen, wodurch der Spielspaß erheblich verringert wird. Mithilfe einer 3D-fähigen Kamera sowie einer geeigneten Bildverarbeitungssoftware soll im Projekt ChessRobertaVision der Roboter in der Lage sein, das Schachbrett und die Figuren zu „sehen“ und deren Koordinaten direkt anzufahren. Somit relativiert sich der mögliche, menschliche Fehleinfluss, die Figur nicht perfekt mittig zu platzieren und der Spieler ist in der Lage, die Züge selbstständig auszuführen. Des Weiteren muss der Nutzer nur noch zu Beginn mit dem externen Client kommunizieren, um die Farbe und Stärke des Computers festzulegen. Während des Spiels soll die Kommunikation voll automatisiert ablaufen, sodass

der Spieler, wie bei einem realen Schachspiel gegen einen Menschen, abwechselnd seinen Zug ausführt und dann auf den Zug des Roboters wartet. Ein weiterer Vorteil von ChessRober-taVision besteht darin, dass die Felder A1 und H8 nicht mehr initialisiert werden müssen, da diese nun selbstständig vom Roboter erkannt werden können.

4 Verwendete Software und Bibliotheken

Die verschiedenen Software-Tools zur Umsetzung der im vorherigen Kapitel beschriebenen Ziele sollen nun erläutert werden. Die spezifischen Versionen der einzelnen Programme sind am Ende des Kapitels aufgelistet.

4.1 Eclipse und PyDev

Eclipse ist eine integrierte Entwicklungsumgebung (IDE) für eine Vielzahl von Programmiersprachen [26]. Mithilfe des Zusatzmoduls PyDev kann Eclipse als Python-IDE verwendet werden. Es bietet fortschrittliche Techniken zur Code-Vervollständigung, Code-Analyse und zum Debuggen [27].

Nachfolgend sind die wichtigsten in Python verwendeten Bibliotheken aufgelistet und beschrieben, wofür sie verwendet werden:

- **python-chess**

Die Software python-chess ist eine Schachbibliothek für Python, welche in der Lage ist, Züge zu generieren, zu überprüfen und auf einem virtuellen Brett auszuführen [28]. Die Klasse *Board* zeichnet dabei das reale Spiel auf, indem jeweils der aktuelle FEN-String abgespeichert wird. Außerdem kann überprüft werden, ob ein gegebener Zug in einer bestimmten Position legal ist und diese Züge können auf den aktuellen FEN-String angewendet werden. Des Weiteren liefert die Klasse Funktionen zur Überprüfung des Spielendes, sei es Matt, Patt, ungenügendes Material, die 75-Züge-Regel oder Stellungswiederholung.

- **stockfish**

Die Bibliothek umfasst die Open-Source-Schachengine Stockfish und liefert eine einfache Integration in Python [29]. In dieser Arbeit wird Stockfish verwendet, um die Züge des Roboters zu generieren. Die Stärke der Engine kann ebenfalls eingestellt werden, um dem Nutzer mehr Individualität zu gewährleisten.

- **numpy**

Diese Bibliothek wird für eine einfache Handhabung von mehrdimensionalen Arrays verwendet. In Python werden in der Regel Listen verwendet, die als Arrays dienen. Da die Verarbeitung der Listen jedoch sehr langsam ist, stellt numpy ein Array-Objekt bereit, welches bis zu 50 Mal schneller ist als herkömmliche Python-Listen [30].

- **socket**

Dieses Modul stellt verschiedene Objekte, Konstanten und Funktionen zur Verfügung, um vollwertige Netzwerkanwendungen, einschließlich Client- und Serverprogrammen, zu erstellen [31].

4.2 MV Tec HALCON

HALCON von MV Tec ist eine umfassende Standardsoftware, die weltweit im Bereich der industriellen Bildverarbeitung (Machine Vision) zum Einsatz kommt [32]. Sie wird in der IDE HDevelop ausgeführt (s. Abb. 4.1), wodurch eine schnelle, effektive Entwicklung im Bereich der industriellen sowie der medizinischen Bildverarbeitung und -analyse möglich ist [33].

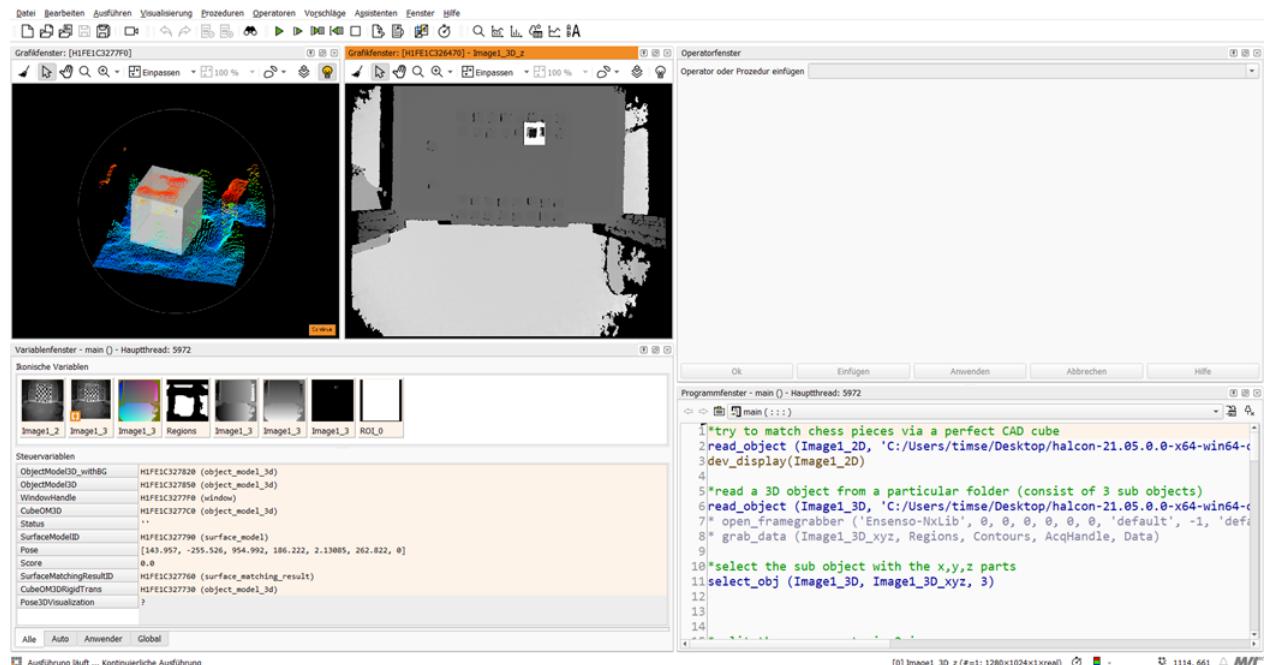


Abbildung 4.1 HDevelop GUI bestehend aus Grafikfenster, Variablenfenster, Operatorfenster und Programmfenster

Des Weiteren wird HALCON in allen Industriezweigen verwendet und bietet beispielsweise Methoden zur Blob-Analyse, verschiedene Matching-Verfahren, morphologische Operatoren, Klassifikationsalgorithmen, Deep-Learning-Verfahren sowie Methoden zur Vermessung von 2D- bzw. 3D-Objekten ([32], [34]). Durch eine Vielzahl von Schnittstellen zu gängigen Industriekameras und Frame Grabbern, kann HALCON weitestgehend hardwareunabhängig angewandt werden [33].

Die vorliegende Masterarbeit ist ein spezieller Fall einer Pick-and-Place-Anwendung. Wie die exakte Pose, also die 3D-Position und Orientierung, eines Objekts ermittelt werden kann, ist eine der häufigsten Fragestellungen in solchen Anwendungen. Hierfür bietet HALCON verschiedene Lösungen, wobei die in dieser Arbeit verwendeten Lösungen nachfolgend kurz erläutert werden.

Zunächst wird HALCON ein Modell des Objektes, z. B. als Computer Aided Design (CAD)-Modell zur Verfügung gestellt. Für die Suche werden hier zwei verschiedene Ansätze verwendet. Beim formbasierten Matching wird die Form des Modells in 2D-Bildern gesucht, beim oberflächenbasierten 3D-Matching wird das Modell in einer 3D-Szene gesucht. Eine 3D-Szene ist hierbei eine Menge von 3D-Punkten, die als 3D-Objektmodell vorliegen (s. Abb. 4.2) und direkt aus den beiden Bildern der Stereokamera erzeugt werden können [35, S. 10]. Für Abb. 4.3 hat HALCON in dieser 3D-Szene alle Würfel einer bestimmten Größe gesucht und angezeigt.

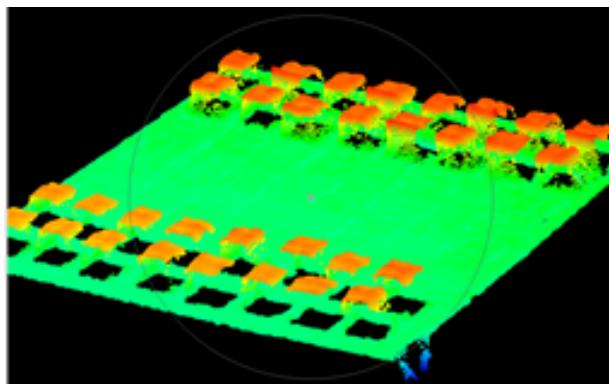


Abbildung 4.2 3D-Objektmodell einer Szene bestehend aus einer 3D-Punktwolke

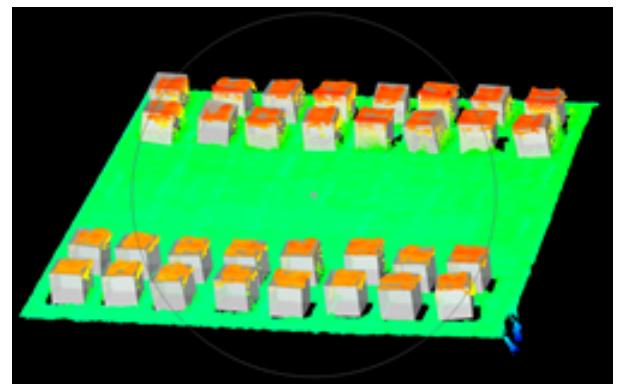


Abbildung 4.3 Oberflächenbasiertes 3D-Matching in einer 3D-Szene

Für die Anwendung dieser Verfahren sind interne HALCON-Funktionen zur Generierung und Suche der Modelle notwendig, welche anschließend beschrieben werden:

- **create_shape_model:**

Die Funktion *create_shape_model* bereitet ein formbasiertes Modell für das Matching vor. Über die zahlreichen Parameter der Funktion kann das Modell in mehreren Rotationen abgespeichert und festgelegt werden, welche Grauwertkontraste die Punkte des Modells besitzen müssen, ebenso kann eine Metrik definiert werden, unter welchen Bedingungen das Muster im Bild erkannt wird und vieles mehr [36]. In diesem Projekt werden alle Parameter der Funktion mit den Standardwerten belegt und ausschließlich 2D-Modelle verwendet.

- **find_shape_model:**

Die Funktion *find_shape_model* ist für die Suche der besten Matches eines formbasierten Modells in einem Bild zuständig. In Abb. 4.4 werden alle gefundenen Matches mit einer roten Umrandung visualisiert.

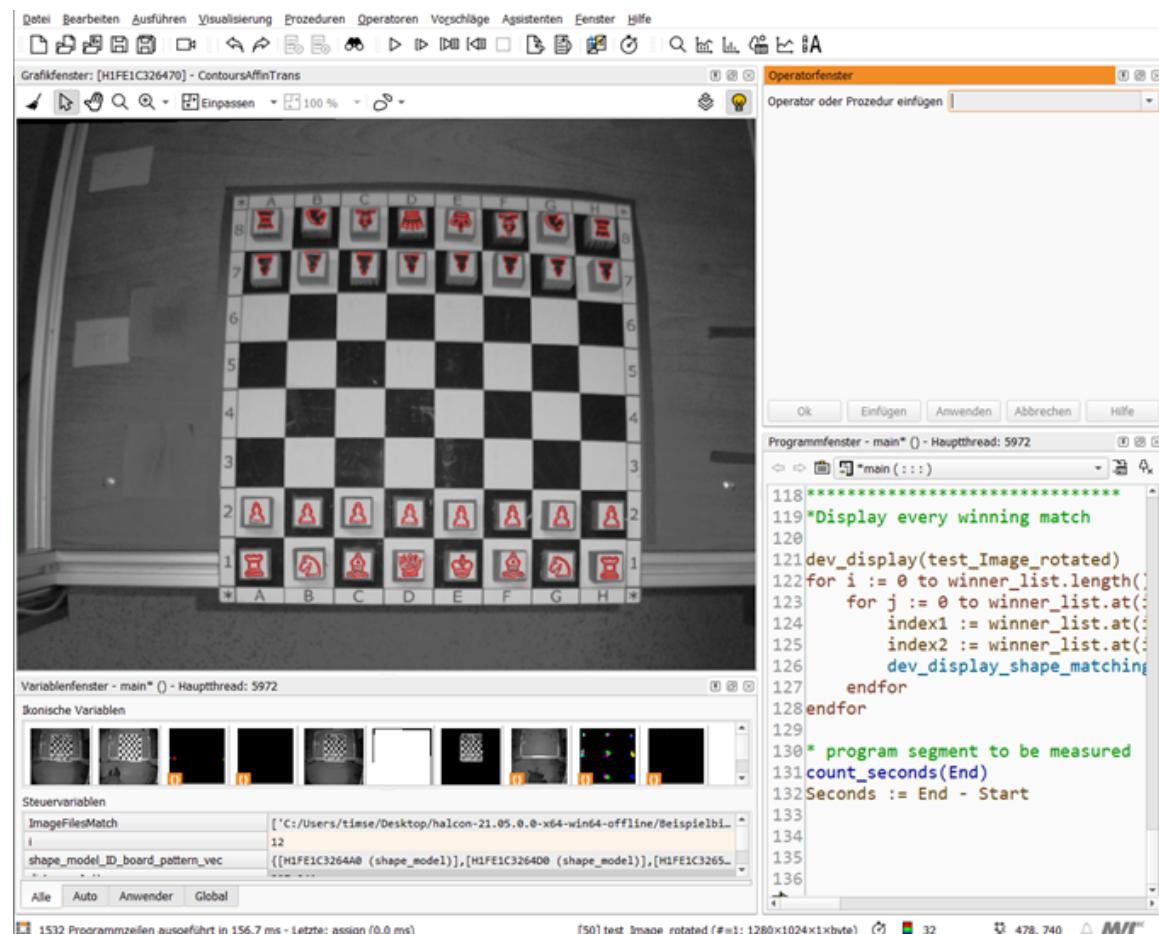


Abbildung 4.4 Mit *find_shape_model* gefundene 2D-Bilder der Schachfiguren visualisiert in der HDevelop GUI

Die Funktion liefert die Position und Rotation der gefundenen Instanzen in den Ausgabeparametern *Row*, *Column* und *Angle*, außerdem wird eine Bewertung zwischen 0 und 1 der gefundenen Instanzen in *Score* zurückgegeben. Die wichtigsten Eingabeparameter zur Anpassung der Funktion sind nachfolgend aufgelistet [37]:

- *AngleStart* und *AngleExtent* legen fest, in welchem Winkelbereich das Modell gesucht wird.
- *MinScore* ist eine untere Schranke für die Bewertung, die ein potentieller Match mindestens besitzen muss, um als Instanz des Modells im Bild angesehen zu werden.
- *NumMatches* legt die maximale Anzahl an gefundenen Instanzen des Modells im Bild fest.
- *Greediness* gibt mit einem Wert zwischen 0 und 1 an, wie „gierig“ das Modell gesucht werden soll. Für den Wert 0 wird das Modell besser gefunden, aber die Suche ist sehr zeitaufwendig, bei einem Wert von 1 ist die Suche viel schneller, es kann jedoch vorkommen, dass das Modell nicht gefunden wird. Der Standardwert von *Greediness* ist 0.9, welcher auch für diese Arbeit gute Ergebnisse lieferte.

- **`create_surface_model`:**

Die Funktion *create_surface_model* erstellt aus einem 3D-Objektmodell ein Modell für oberflächenbasiertes Matching. Für das 3D-Objektmodell können beispielsweise Punkte und ein Polygongitter aus einer CAD-Datei verwendet werden. Zur Erstellung des oberflächenbasierten Modells wird die Oberfläche des 3D-Objektmodells mit einem bestimmten Abstand abgetastet. Hierbei sollte unbedingt auf Ausreißerpunkte geachtet werden, welche das Ergebnis des Matchings stark beeinflussen können. Es existieren einige Eingabeparameter zur Anpassung der Funktion, die den internen Algorithmus beeinflussen können. Anschließend werden alle Parameter, die in dieser Arbeit nicht den Standardwert verwenden, beschrieben [38]:

- *RelSamplingDistance*:

RelSamplingDistance gibt die Abtastdistanz relativ zum Durchmesser des Objektes an. Wird *RelSamplingDistance* beispielsweise auf 0.15 gesetzt und der Durchmesser des 3D-Objektmodells ist 3 cm, dann haben die Punkte der abgetasteten Oberfläche einen Abstand von jeweils $30 \text{ mm} \cdot 0.15 = 4.5 \text{ mm}$. Es ist zu beachten, dass eine größere Anzahl an Punkten im oberflächenbasierten Objektmodell zu längeren Zeiten beim Suchen führt.

- `train_3d_edges`:

Mithilfe des booleschen Parameters `train_3d_edges` können zusätzlich zu den 3D-Punkten auch 3D-Kanten für das oberflächenbasierte Matching verwendet werden, um die Ausrichtung des Modells zu verbessern. Das kann insbesondere dann sehr hilfreich sein, wenn das 3D-Objektmodell größere, planare Seiten aufweist, um zu verhindern, dass sie in eine Hintergrundebene geraten oder falsch rotiert gefunden werden. Hierfür muss das Modell eine Vermaschung aufweisen, d. h. Dreiecks- oder Polygongitter enthalten. Es ist außerdem zu beachten, dass diese Art der Kantenunterstützung das Training deutlich verlängert.

- `train_view_based`:

Der boolesche Parameter `train_view_based` sorgt dafür, dass die Berechnung der Bewertung ansichtsbasiert erfolgt. Das kann insbesondere für Szenen hilfreich sein, in denen Objekte nur zu einem kleinen Teil sichtbar sind. Zur Bestimmung eines aussagekräftigeren Wertes wird hierbei die Bewertung als Verhältnis zwischen der Anzahl der Punkte, die dem Modell zugeordnet werden, und der Anzahl an Modellpunkten, die aus einer bestimmten Ansicht sichtbar sind, berechnet. Das 3D-Objektmodell muss ebenfalls eine Vermaschung aufweisen.

- **`find_surface_model`:**

Die Funktion `find_surface_model` sucht in einer 3D-Szene die besten Matches des oberflächenbasierten Modells und gibt deren Posen sowie die dazugehörigen Bewertungen zurück. Eine 3D-Szene bezeichnet hierbei das 3D-Objektmodell einer Szene. Die Punkte und Normalen dieser 3D-Szene werden zur Suche des oberflächenbasierten Modells verwendet und müssen beispielsweise Punkte und ein 2D-Mapping enthalten [39]. In dieser Arbeit werden die 3D-Szenen ausschließlich mit der internen HALCON-Funktion `xyz_to_object_model_3d` erstellt, welche ein Bildtripel aus x-, y- und z-Koordinaten von 3D-Punkten in ein 3D-Objektmodell transformiert. Das erstellte Modell enthält, zusätzlich zu den 3D-Koordinaten der Punkte, ein 2D-Mapping, welches für jeden der 3D-Punkte dessen ursprüngliche Bildkoordinaten, bestehend aus Zeile und Spalte, enthält [40]. Nachfolgend werden alle Parameter von `find_surface_model` beschrieben, welche nicht mit dem jeweiligen Standardwert belegt sind [39]:

- `RelSamplingDistance`:

Wie in der oben beschriebenen Funktion `create_surface_model` gibt der Parameter `RelSamplingDistance` die Abtastdistanz der Szene relativ zum Durchmesser des oberflächenbasierten Modells an.

- KeyPointFraction:

Mit dem Parameter *KeyPointFraction* kann bestimmt werden, wie viele der abgetasteten Szenenpunkte als Schlüsselpunkte ausgewählt werden. Schlüsselpunkte sind ein wichtiger Aspekt für den Suchalgorithmus, da für jeden ausgewählten Schlüsselpunkt die optimale Pose des oberflächenbasierten Modells berechnet wird, wobei angenommen wird, dass der Schlüsselpunkt auf der Oberfläche des Objektes liegt. Daraufhin wird aus allen Posen, die mithilfe der Schlüsselpunkte generiert wurden, die Pose mit der höchsten Bewertung ausgewählt und als Näherung in den kommenden Schritten des Suchalgorithmus verwendet. Wie der Name *KeyPointFraction* vermuten lässt, wird hiermit der Anteil an abgetasteten Szenenpunkten, welche als Schlüsselpunkte verwendet werden, angegeben. Je höher der Wert gewählt wird, desto robustere Ergebnisse liefert der Algorithmus, was jedoch gleichzeitig zu einem langsameren Matching führt.

- MinScore:

Der Parameter *MinScore* wird zur Filterung der Ergebnisse verwendet, indem die Funktion *find_surface_model* nur Instanzen zurückgibt, deren Bewertung größer als der Wert von *MinScore* ist. Für einen Wert von 0 werden alle gefundenen Instanzen zurückgegeben.

- num_matches:

Der Parameter *num_matches* legt die maximale Anzahl an gefundenen Instanzen fest, die von der Funktion zurückgegeben werden.

- use_3d_edges:

Dieser boolesche Parameter bestimmt, ob die Kantenunterstützung ein- oder ausgeschaltet wird und kann nur verwendet werden, wenn das oberflächenbasierte Modell bei der Erstellung den Parameter *train_3d_edges* aktiviert hat.

- use_view_based:

Der boolesche Parameter *use_view_based* ist dafür zuständig, die ansichtsbasierte Bewertung beim oberflächenbasierten Matching ein- oder auszuschalten. Zur Verwendung dieses Parameters muss *train_view_based* bei Erstellung des oberflächenbasierten Modells aktiviert werden.

Des Weiteren ist das 2D-Koordinatensystem aus HALCON zu erwähnen, welches in Abb. 4.5 visualisiert ist der Nullpunkt befindet sich hierbei in der linken oberen Ecke. Ein Punkt im Koordinatensystem wird über den Aufenthaltsort seines Pixels beschrieben. Da Pixel

diskret sind, ergibt sich ein Koordinatensystem, das nur ganzzahlige Werte verwendet. Ein Pixel ist durch die sogenannten *Row-* und *Column-Koordinaten* definiert, die die Zeilen bzw. Spalten, die der Pixel vom Ursprung entfernt ist, angeben. Der in Abb. 4.5 mit einem „ \times “ gekennzeichnete Pixel erhält somit die Koordinaten (6, 5) [6].

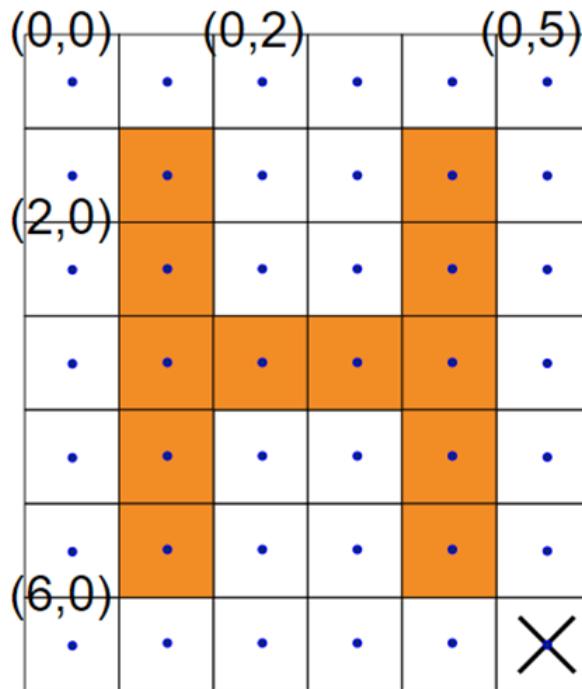


Abbildung 4.5 Pixelkoordinatensystem von HALCON. Das Kreuz zeigt den Pixel in der rechten unteren Bildecke an, dessen Zentrum die Koordinaten (6, 5) besitzt [6].

4.3 NxLib und NxView

NxLib ist eine zentrale Bibliothek des Ensenso Software Development Kit und eine Schnittstelle, um Bilder aufzunehmen und zu verarbeiten [41]. NxView ist ein Beispielprogramm, welches die Hauptfunktionen der NxLib Bibliothek demonstriert. Mit dieser Software ist es möglich, eine oder mehrere Stereo- bzw. Farbkameras zu öffnen, wobei Bild- und Tiefendaten in Echtzeit visualisiert werden können, wie in Abb. 4.6 dargestellt. Verschiedene Parameter wie Belichtungszeit und Tiefenmessbereich lassen sich dabei live anpassen. Über das HALCON Image Acquisition Interface können Ensenso Kameras problemlos über HALCON bzw. HDevelop angesprochen werden [42].

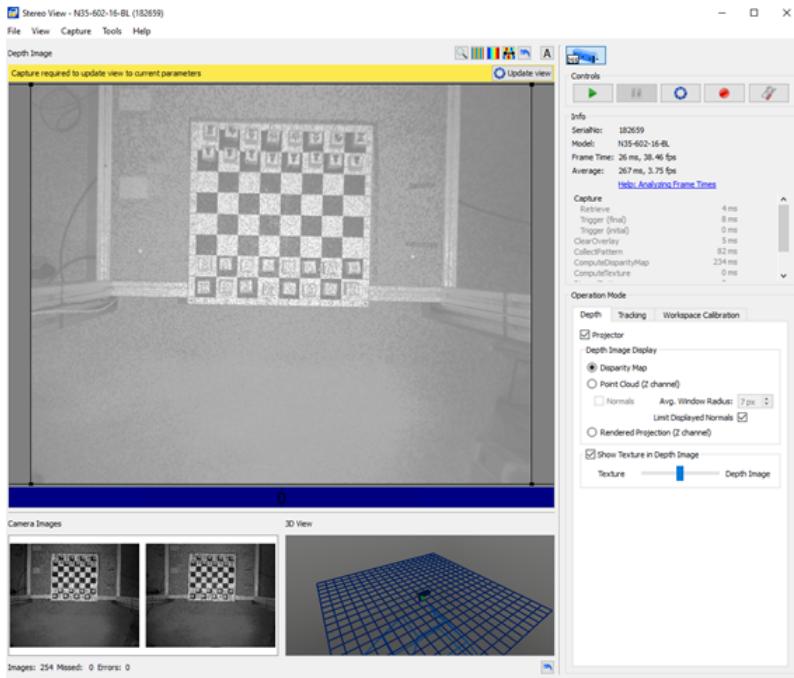


Abbildung 4.6 NxView Interface

4.4 Sunrise

Wie in Kap. 2.1.1 beschrieben, wird das Werkzeug KUKA Sunrise.Workbench zur Entwicklung von Roboterapplikationen für den KUKA LBR iiwa benötigt. Die Entwicklungsumgebung verfügt über folgende Funktionalitäten zur Applikationsentwicklung [43, S. 21]:

- Übertragung von Projekten auf die Robotersteuerung,
- Programmieren von Roboterapplikationen in Java,
- Verwaltung von Projekten und Programmen sowie
- die Synchronisation von Projekten.

In dieser Entwicklungsumgebung werden für das Projekt einige Sunrise-Bibliotheken benötigt. Die Bibliotheken, die relevant für den internen Quellcode sind, werden nachfolgend aufgelistet:

- **com.kuka.roboticsAPI.deviceModel.LBR**

Diese Bibliothek ermöglicht das Zugreifen auf die aktuellen kartesischen Koordinaten im TCP und wird zur Initialisierung des Greifers benötigt.

- **com.kuka.roboticsAPI.geometricModel.Frame**

Hiermit können neue Frames angelegt werden, welche im Programm mehrfach verwendbar sind.

- **com.kuka.roboticsAPI.motionModel.BasicMotions.ptp**

Diese Bibliothek ist für die PTP-Bewegung zuständig.

- **com.kuka.roboticsAPI.motionModel.BasicMotions.linRel**

Diese Bibliothek ist für die LINREL-Bewegung zuständig.

- **com.kuka.grippertoolbox.gripper.zimmer.ZimmerR840**

Mit dieser Bibliothek kann der Aufsatz des Greifers geöffnet und geschlossen werden.

- **com.kuka.roboticsAPI.geometricModel.World**

Diese Bibliothek wird verwendet, um das Weltkoordinatensystem als das zu referenzierende Koordinatensystem während einer LINREL-Bewegung zu definieren.

- **com.kuka.roboticsAPI.applicationModel.IApplicationData**

Mit dem SmartPAD können manuell verschiedene Punkte im Raum als Frames gespeichert werden. Diese Bibliothek ist dafür zuständig, im Sunrise-Programm auf die Koordinaten dieser Frames zuzugreifen.

4.5 Versionen

Nachfolgend sind die spezifischen Versionen der verwendeten Software-Tools aufgelistet:

- Eclipse 4.21.0,
- Python 3.9.10,
- python-chess 1.8.0,
- stockfish 3.22.1,
- HALCON 21.05.0.0,
- HDevelop 21.05.0.0,
- NxLib 3.2.489,
- Sunrise.Workbench 1.16.1.9.

5 Projektaufbau

Dieses Kapitel beschreibt den gesamten Projektaufbau und erklärt, welche Programme für die Umsetzung notwendig sind. Eine ausführliche Erklärung findet sich in Kap. 7. Zunächst liefert das kommende Unterkapitel die grundlegende Idee zur Umsetzung eines sehenden Schachroboters, welche in den darauf folgenden Kapiteln detaillierter ausgeführt wird.

5.1 Grundlegende Idee

Für den Beginn eines Spieles muss der Nutzer einige wichtige Informationen bzgl. der Spielfarbe und der Stärke des Roboters angeben. Daraufhin startet das Spiel zwischen Mensch und Maschine. Wie soll nun ein Zug vom Nutzer und Roboter im Detail ablaufen? Voraussetzung für den Ablauf eines Spiels ist, dass nach jedem Zug der aktuelle FEN-String abgespeichert wird.

1. Die Kamera macht in kurzen Abständen Bilder des Schachbrettes, um auf den Zug des Benutzers zu warten. Aus einem Bild wird zunächst ein neuer FEN-String der aktuellen Position erstellt, daraufhin wird aus dem alten und neuen FEN-String ein Zug berechnet. Wenn es sich dabei um einen legalen Zug handelt, folgt Schritt 2, andernfalls wird entweder eine Fehlermeldung ausgegeben oder obiger Prozess wiederholt, bis ein legaler Zug erkannt wird.
2. Berechnung des nächsten Zuges in der aktuellen Position mithilfe einer Schachengine bzw. Schach-KI.
3. Übersetzung des zuvor berechneten Zuges in eine Koordinatenfolge für die Roboterbewegung.
4. Ausführung der Koordinatenfolge mithilfe des Roboterarms.
5. Wiederholung ab Schritt 1, bis das Spiel zu Ende ist.

5.2 Hardware und nichttechnische Komponenten

Dieses Unterkapitel beschreibt die physischen Komponenten, die für die Umsetzung des Projektes verwendet werden. Hierzu gehören der Roboterarm und die Stereokamera, welche ausführlich in Kap. 2.1 bzw. 2.2 beschrieben werden. Für die Verarbeitung der Schachregeln, das Interface zur Kommunikation mit dem Benutzer und zur Verwendung der Bildverarbeitungssoftware HALCON wird zusätzlich ein externer Computer benötigt. Anschließend werden alle nichttechnischen Komponenten der Arbeit behandelt.

Greiferaufsatz des Roboters:

Der KUKA LBR iiwa ist mit verschiedenen Aufsätzen kompatibel. Das Projekt wurde mithilfe eines 2-Backen-Parallelgreifers für die Mensch-Roboter-Kollaboration bearbeitet, welcher für das zeitbegrenzte Greifen, Handhaben und Halten in geschlossenen Räumen gedacht ist [7, S. 4].

Abb. 5.1 visualisiert den Aufsatz des Roboters sowie dessen Komponenten, die anschließend aufgelistet werden [7, S. 6]:

1. Greiferbacken
2. Gehäuse
3. Elektrischer Anschluss
4. Sicherheitsbacken
5. Greiffinger



Abbildung 5.1 2-Backen-Parallelgreifer für die Mensch-Roboter-Kollaboration [7, S. 6]



Abbildung 5.2 Adaptiv-Greiffinger von Festo [8]

Als Greiffinger werden die Adaptiv-Greiffinger von Festo genutzt (s. Abb. 5.2), welche zusätzlich verstieft worden sind.

Schachfiguren:

Die Schachfiguren, die in dieser Arbeit verwendet werden, bestehen aus $3 \times 3 \times 3$ cm großen Würfeln aus Holz, welche einseitig mit einem Bild beklebt sind (s. Abb. 5.3).



Abbildung 5.3 Verwendete Schachfiguren am Beispiel eines weißen Bauers

In Abb. 5.4 sind die Muster für alle verschiedenen Figuren dargestellt.



Abbildung 5.4 Musterbilder aller zwölf Schachfiguren

Der spezifische Aufbau der Figuren hat gegenüber herkömmlichen Schachfiguren folgende Vorteile:

1. Für die oben beschriebenen Greiffinger des Roboters eignen sich einfache Geometrien wie die eines Würfels, für traditionelle Figuren ist der Aufsatz des Roboters nicht optimal.
2. Im Vorgängerprojekt ChessRoberta (s. Kap. 3.1) wurden diese Figuren bereits verwendet.
3. Die Kamera, welche exakt oberhalb des Spielfelds positioniert ist, kann die 2D-Bilder der Figuren einfacher erkennen und verarbeiten. Um in der Lage zu sein, klassische Schachfiguren zu erkennen, wird eine 3D-Bilderkennung der Figuren benötigt, wofür zunächst 3D-Objektmodelle aller Figuren angelegt werden müssen (vgl. Kap. 4.2). Diese könnten beispielsweise aufwendig und zeitintensiv in CAD-Programmen konstruiert werden, das Hauptproblem liegt jedoch in der optimalen Erkennung der Figuren von oben, woran noch ausgiebiger geforscht werden muss.
4. Die Größe der Würfel wurde in Abhängigkeit des Schachbrettes gewählt (s. u.).

Promotionsfigur:

Wenn ein weißer Bauer die achte Reihe bzw. ein schwarzer Bauer die erste Reihe auf einem Schachbrett erreicht, werden diese in eine andere Figur umgewandelt. Der Bauer hat hierbei die Möglichkeit zu einer Dame, einem Läufer, einem Springer oder einem Turm zu werden. Dieser Vorgang wird als Promotion, Umwandlung oder Bauernumwandlung bezeichnet. In der Theorie hat ein Spieler die Möglichkeit, alle acht Bauern umzuwandeln und sich dabei jeweils für eine der vier oben genannten Figuren zu entscheiden. Daraus resultiert, dass ein Spieler bis zu $8 \cdot 4 = 32$ zusätzliche Figuren benötigt, um jeden möglichen Spielverlauf abzudecken. In der Praxis ist es jedoch extrem selten, dass während eines Spiels mehr als eine Promotion pro Seite durchgeführt wird. Um dem Spieler dennoch die Möglichkeit zu geben, einen Bauern in eine der vier verschiedenen Figuren umzuwandeln, ohne zu viele Holzwürfel zu verwenden, sind die Promotionsfiguren, welche neben dem Brett liegen, vierseitig beklebt, wie in Abb. 5.5 dargestellt.



Abbildung 5.5 Eine weiße Promotionsfigur aus vier verschiedenen Blickwinkeln

In den meisten Fällen wird ein Bauer in eine Dame umgewandelt, da sie die stärkste Figur auf dem Spielfeld ist. Dennoch gibt es einige Fälle in denen eine Promotion in eine der schwächeren Figuren sinnvoll ist, beispielsweise zur Verhinderung einer Pattsituation.

Die Promotionsfigur wird zu Beginn eines Spiels so gedreht, dass die Dame nach oben ausgerichtet ist. Falls der Roboter nun eine der übrigen drei Figuren verwenden will, wird er die Promotionsfigur so lange drehen, bis das gewünschte Bild oben liegt und die Figur anschließend auf die gewünschte Position setzen.

Schachbrett:

In Abb. 5.6 und 5.7 ist das Schachbrett aus zwei unterschiedlichen Perspektiven visualisiert. Die Bilder zeigen das Brett und die Figuren in der Anfangsstellung einer Schachpartie. Die Größe des Schachbrettes muss in abhängig vom Arbeitsraum des Roboters gewählt werden, sodass er in der Lage ist, jeden Punkt auf dem Brett zu erreichen. Aufgrund von Tests wurde ein Schachbrett der Größe 43×43 cm gewählt, wobei ein Feld 5×5 cm entspricht.

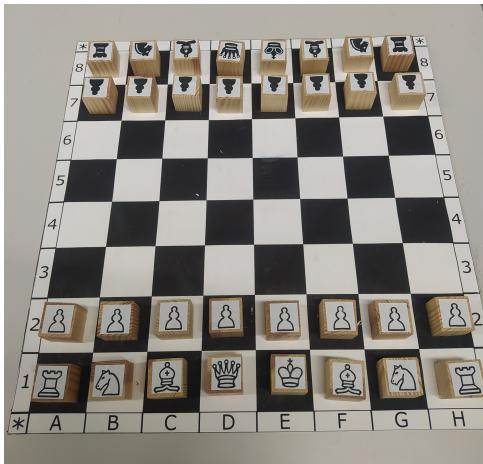


Abbildung 5.6 Anfangsstellung einer Partie Schach auf einem Schachbrett von oben

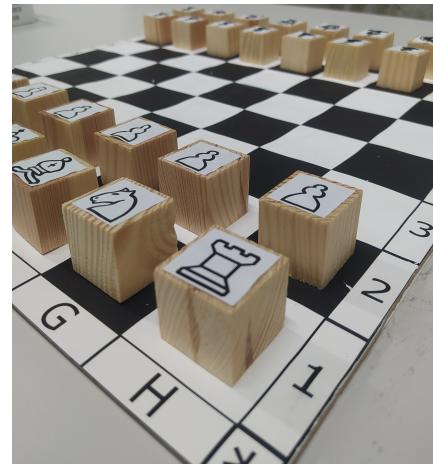


Abbildung 5.7 Anfangsstellung einer Partie Schach auf einem Schachbrett von der Seite

Die Sterne an den Eckpunkten sind eine wichtige Markierung zur Lokalisierung des Brettes mithilfe der Kamera, da sie sich zu einem Polygon verbinden lassen, dessen Inneres dem Aufenthaltsort des Schachbretts entspricht. Zur Berechnung der Koordinaten spezifischer Felder werden zusätzlich die Beschriftungen von „A“, „H“ und „1“ verwendet, die sich auf dem Rand des Brettes befinden (vgl. Abb. 5.8).

Mithilfe der Musterbilder aus Abb. 5.4 und 5.8 sowie den Maßen aus diesem Kapitel kann das verwendete Schachset unverändert nachgebildet werden.

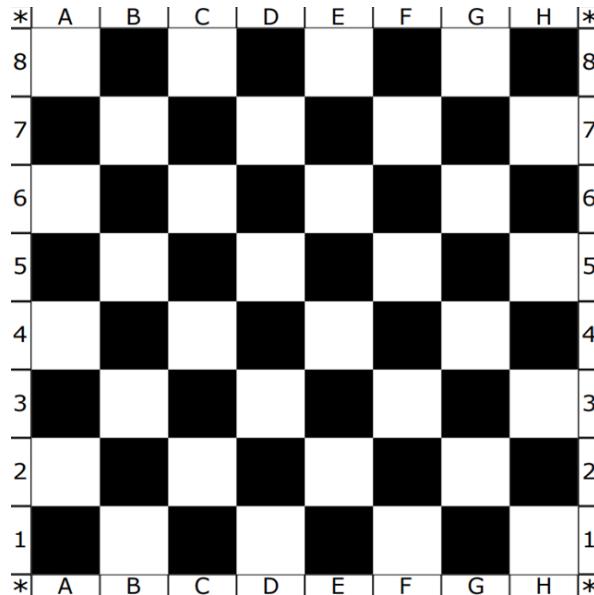


Abbildung 5.8 Musterbild des Schachbrettes

Aus-Feld:

Mit dem sogenannten Aus-Feld ist der Bereich gemeint, in dem alle geschlagenen Figuren abgelegt werden. Die Software ist so programmiert, dass die Figuren stets auf der rechten Seite des Schachbrettes abgeworfen werden, unabhängig davon, welche Farbe der Spieler ausgewählt hat. Weiterhin müssen drei Voraussetzungen für einen optimalen Spielverlauf berücksichtigt werden. Die Ablage

- muss bis zu 30 Figuren fassen,
- darf eine Höhe von 10 cm nicht überschreiten, da der Roboter die Figuren aus etwa dieser Höhe fallen lässt,
- und sollte mindestens 1 cm Abstand zum Brett haben, sodass sie nicht mit einer Figur verwechselt werden kann.

5.3 Software

Abb. 5.9 visualisiert die Architektur des Projektes. Hierbei sind die Verbindungen zwischen den drei technischen Hardwarekomponenten, die im vorherigen Kapitel erwähnt wurden, dargestellt.

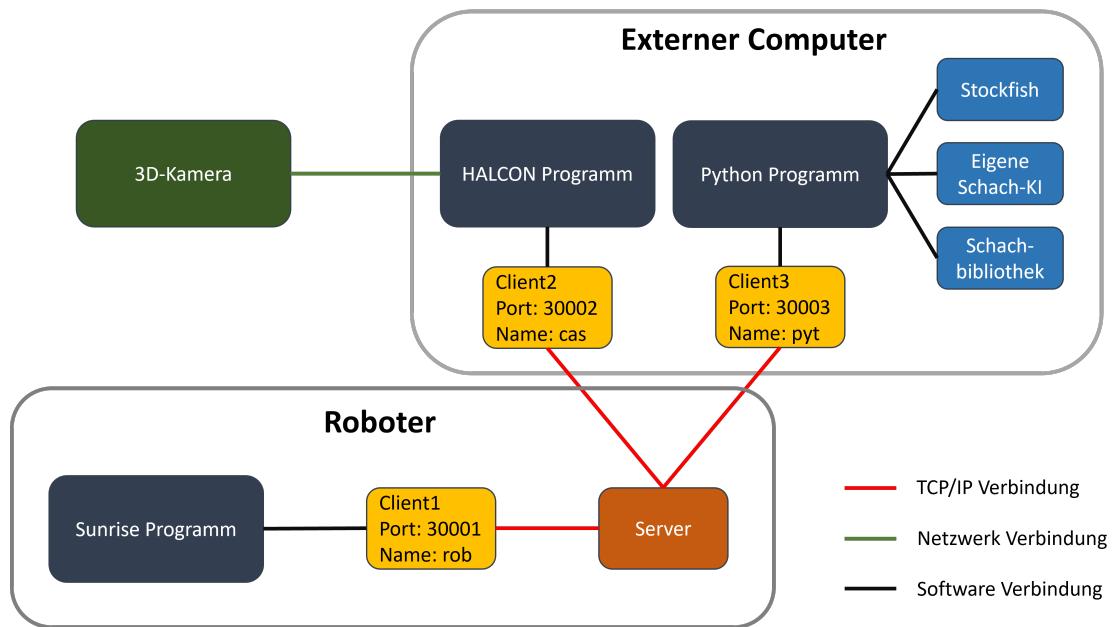


Abbildung 5.9 Architektur des Projekts

Zur Umsetzung des Prozesses in Kap. 5.1 benötigen die einzelnen Programme, die in Abb. 5.9 dargestellt sind, folgende Komponenten:

- **HALCON-Programm:**

- Aufnahme von 2D- und 3D-Bildern.
- Erstellung eines FEN-Strings aus einem 2D-Bild des Schachbrettes.
- Berechnung des Frames einer Figur auf einem bestimmten Feld mittels eines 3D-Bildes.
- Berechnung der Koordinaten eines leeren Feldes mithilfe eines 3D-Bildes.

- **Python-Programm:**

- Interface zur Kommunikation mit dem Benutzer.
- Berechnung eines Zuges aus zwei aufeinanderfolgenden FEN-Strings.
- Überprüfung der Legalität eines Zuges in einer bestimmten Position.
- Berechnung eines Zuges in einer bestimmten Position mithilfe einer Schachengine bzw. Schach-KI.
- Erstellung einer Bewegungsfolge aus einem beliebigen Zug. Eine Bewegungsfolge enthält eventuell mehrere Paare aus Start- und Zielfeld.

- **Sunrise-Programm:**

Ausführung von Bewegungsabläufen inklusive Greifen mithilfe einer Bewegungsfolge bestehend aus Frames und Koordinaten.

Zur Kommunikation der verschiedenen Komponenten und Programme wird ein Client-Server-Modell angewendet. Der Server läuft auf der Robotersteuerung, und die drei Hauptprogramme (Python, HALCON und Sunrise) fungieren als Clients. Wie in Abb. 5.9 dargestellt, verwendet jeder der Clients einen eigenen Port und Namen, um die Möglichkeit zu haben, Nachrichten an alle oder nur an spezifische Clients zu senden.

6 Bedienung von ChessRobertaVision

Dieses Kapitel ist eine Bedienungsanleitung für das Projekt ChessRobertaVision, beinhaltet den Link zu allen notwendigen Programmen und Dateien, geht auf die nötigen Vorbereitungen ein und erklärt, wie das Projekt tatsächlich ausgeführt wird. Darüber hinaus wird beschrieben, wie mit bestimmten Fehlermeldungen umgegangen werden sollte, und es werden Spezialfälle aufgezählt, die beachtet werden müssen, um einen reibungsfreien Spielverlauf zu gewährleisten.

6.1 Programme und Dateien

Alle Programme und Dateien, die für das Projekt notwendig sind, wurden auf GitHub veröffentlicht und können unter <https://github.com/FHWS-FANG/ChessRobertaVision> heruntergeladen werden.

- **ChessProgram_Python:**

Python-Programm.

- **ChessProgram_Roberta:**

Sunrise-Projekt.

- **ChessProgram_Halcon:**

HALCON-Hauptprogramm.

- **HandEyeCalibration_Halcon:**

HALCON-Programm für die Hand-Augen-Kalibrierung (HAK).

- **Camera_to_Robot_TM:**

Eine Textdatei, die die Transformationsmatrix der HAK enthält. Die Anleitung zur Berechnung einer neuen Transformationsmatrix zwischen Kamera- und Roboterkoordinatensystem kann in Kap. 6.6 nachgelesen werden.

- **MatchingImages:**

Dieser Ordner enthält alle Matching-Bilder, die für das HALCON-Hauptprogramm benötigt werden.

- MatchingPieces:

Ordner mit 216 Bildern der Schachfiguren (je 18 Bilder für zwölf verschiedene Figuren).

- MatchingBoardPattern.hobj:

Bilddatei des Schachbrettes, welches als Muster für die Markierungen (A, H, 1, *) dient.

- ObjectModelCube.ply:

Polygon File Format (PLY)-Datei eines Würfels der Größe $3 \times 3 \times 3$ cm.

6.2 Vorbereitungen

Nachfolgend werden alle notwendigen Vorbereitungen aufgelistet, um das Schachprogramm auf einem beliebigen Computer auszuführen.

- Anfangs müssen alle Programme und Dateien aus Kap. 6.1 auf einen externen Computer heruntergeladen werden.
- Danach werden das Schachbrett und die -figuren aufgebaut. In Abb. 6.1 ist ein Musterbeispiel für den Spielaufbau dargestellt. Es muss darauf geachtet werden, dass der Roboter in der Lage ist, jede Figur auf dem Schachbrett erreichen zu können. Aus diesem Grund wurde ein quadratischer Bereich für das Brett markiert, in dem jede Bewegung vom Roboter ausgeführt werden kann, für einen reibungsfreien Ablauf empfiehlt es sich also, dass Brett innerhalb der Markierung abzulegen, dabei müssen beide Promotionsfiguren links neben dem Brett platziert werden. Die Promotionsfigur des Roboters muss auf einer 3 cm hohen Ablage, um 180° verdreht zum Benutzer und parallel zum Tisch, abgelegt werden, damit der Roboter die Drehbewegung der Figur besser ausführen kann, außerdem sollte eine Ablage für die geschlagenen Figuren 1 cm rechts neben dem Brett aufgestellt werden.
- Im nächsten Schritt muss gewährleistet werden, dass die drei Hardwarekomponenten, bestehend aus Roboter, Stereokamera und externem Computer, miteinander kommunizieren können. Alle Geräte werden dafür mit je einem Netzwerkkabel über einen

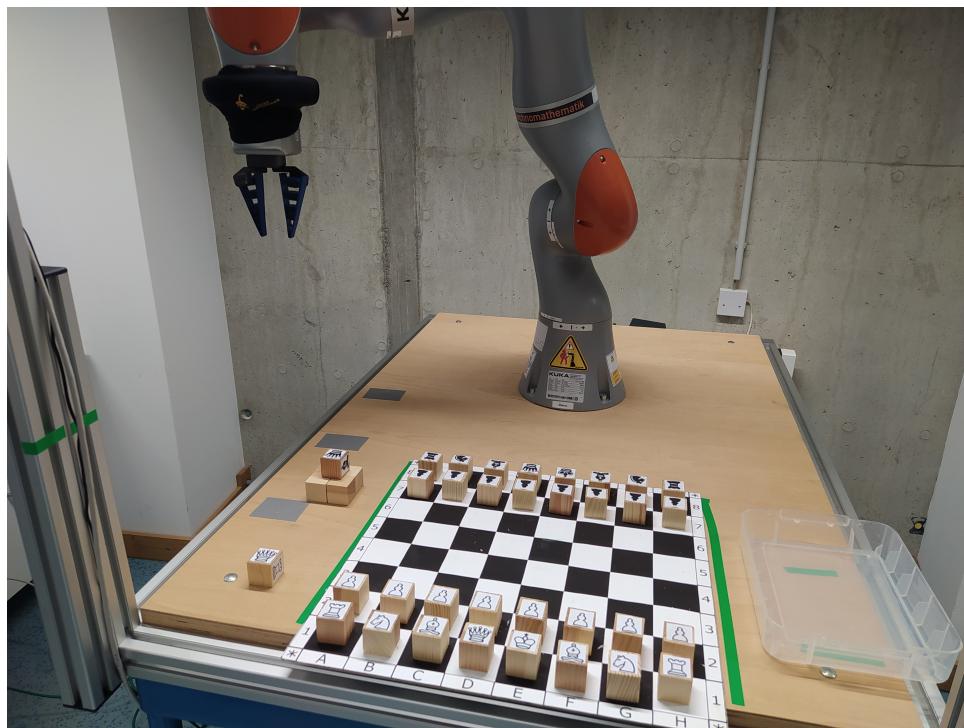


Abbildung 6.1 Aufbau der nichttechnischen Komponenten von ChessRobertaVision

Netzwerk-Hub verbunden.

Zusätzlich müssen die Ethernet-Einstellungen für den externen Computer angepasst werden. Eine mögliche Vorgehensweise für Microsoft Windows wird nachfolgend beschrieben:

- Netzwerk- und Freigabeecenter öffnen,
 - Ethernet-Einstellungen öffnen,
 - „Internetprotokoll, Version 4 (TCP/IPv4)“ auswählen und auf Eigenschaften klicken (vgl. Abb. 6.2),
 - „Folgende IP-Adresse verwenden:“ auswählen,
 - die IP-Adresse 192.168.70.2 mit der Subnetzmaske 255.255.255.0 eingeben (vgl. Abb. 6.2),
 - und „OK“ klicken.
- Anschließend muss der Zugriff auf die Stereokamera aktiviert werden. Hierfür wird die NxView-Software geöffnet, in der die Kamera mit dem Status „Invalid IP Address“ angezeigt wird (linkes Bild in Abb. 6.3). Um die korrekte IP-Adresse zu beziehen, muss folgendermaßen vorgegangen werden (vgl. mittleres Bild in Abb. 6.3):

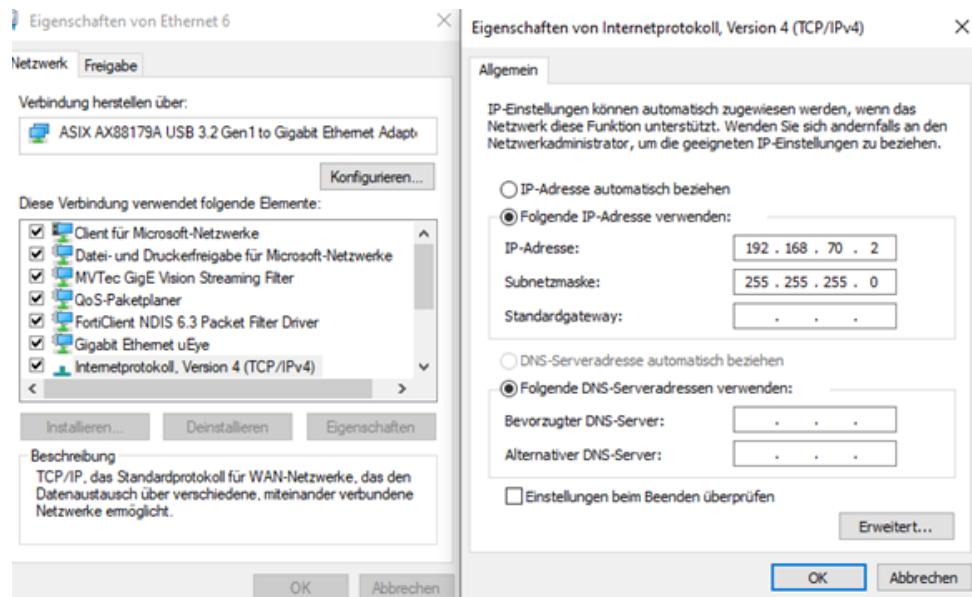


Abbildung 6.2 Ethernet-Einstellungen

- Kamera aussuchen,
- „IP Config“ auf der rechten Seite auswählen,
- Haken in „Use AutoIP“ setzen,
- und „Apply Changes“ anklicken.

Die Kamera nimmt den Status „Available“ an, und HALCON kann jetzt darauf zugreifen (rechtes Bild in Abb. 6.3)

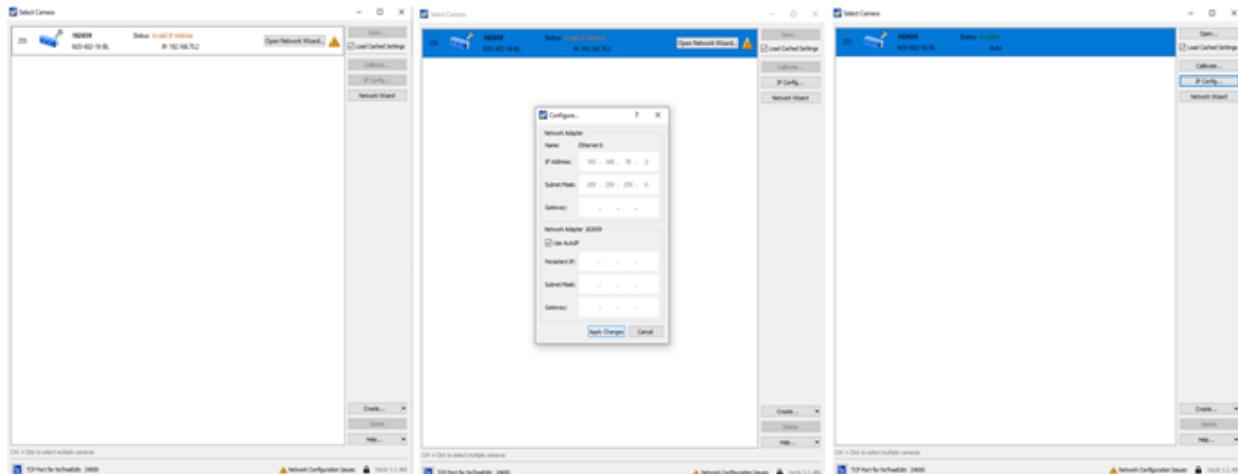


Abbildung 6.3 Verbindung von Kamera und HALCON mit NxView aktivieren

- Wenn sich der Aufbau des Roboters und der Kamera verändert hat, muss eine neue Transformationsmatrix für die Hand-Augen-Kalibrierung (HAK) berechnet werden (s. Kap. 6.6).
- Im nächsten Schritt werden die drei Programme geöffnet.
 - Das Python-Programm kann beispielsweise mit der IDE Eclipse geöffnet werden. Das auszuführende Programm hat den Namen *PyClient.py*.
 - Nach dem Öffnen von HALCON kann das Hauptprogramm *ChessProgram_Halcon* in die Software geladen werden.
 - Ähnlich wird das Werkzeug Sunrise.Workbench aufgerufen und das Sunrise-Projekt *ChessProgram_Roberta* in der Software geöffnet.
- Falls keine Startposition im Roboter gespeichert ist, muss eine neue angelegt werden. Der Roboter wird zu diesen Zweck manuell zu einem beliebigen Punkt im Raum gefahren, welcher sich nicht zwischen Kamera und Schachbrett befindet, die Rotation des Greifers ist dabei irrelevant. Mithilfe des SmartPADs kann unter Frames die aktuelle Position in der Variable *StartPos* abgespeichert werden. Daraufhin muss das Sunrise-Projekt synchronisiert werden, damit das Programm die korrekten Koordinaten verwendet, wofür lediglich links in der Symbolleiste der Button „Projekt synchronisieren“ und „Übertrage auf lokales Projekt“ ausgewählt werden muss.
- Anschließend muss das Sunrise-Projekt gegebenenfalls auf die Steuerung übertragen werden, indem der Button „Projekt synchronisieren“ mit der Option „Übertrage auf Steuerung“ ausgewählt wird.
- Zudem wird der Modus „AUT“ für den Roboter benötigt, um das Programm automatisch auszuführen. Dafür muss der Schlüssel über dem Bildschirm des SmartPADs gedreht und die Funktion „AUT“ ausgewählt werden.
- Abschließend sollte überprüft werden, wo die Dateien und Bilder gespeichert wurden, da HALCON auf diese zugreift. Zu Beginn des HALCON-Hauptprogramms wird der Pfad zu dem Hauptordner, welcher alle Programme und Dateien aus Kap. 6.1 enthält, definiert, dieser Pfad muss gegebenenfalls geändert werden.
- Im Python-Programm *PyClient.py* wird in Zeile 27 der Pfad zur Stockfish-Engine angegeben, welcher geändert werden muss, falls das Programm von einem neuen externen Computer gestartet wird.

6.3 Ausführung

Nachdem alle Vorbereitungen abgeschlossen sind, kann das Schachspiel beginnen. Nachfolgend ist der Ablauf stichpunktartig beschrieben:

- **Das Starten der drei Programme:**

Die dargestellte Reihenfolge muss eingehalten werden, um zu garantieren, dass eine Client-Server-Kommunikation zwischen den drei Komponenten aufgebaut werden kann, die Programme sollten unmittelbar hintereinander ausgeführt werden.

1. **Sunrise:** Zunächst muss auf dem SmartPAD *ChessProgram* ausgewählt werden. Durch Drücken des „Play Buttons“ auf der linken Seite des Bildschirmes kann das Programm gestartet werden.
2. **HALCON:** Entweder in der Symbolleiste den Button „Ausführen“ auswählen oder F5 drücken.
3. **Python:** Das Python-Programm wird gestartet, indem in der Symbolleiste der Button „Run“ ausgewählt wird.

- **Warten auf Initialisierung:**

In diesem Schritt befindet sich HALCON in der Initialisierung und versucht die Schachanfangsposition auf dem Brett zu erkennen. Währenddessen kann HALCON verschiedene Fehlermeldungen an Python senden, die daraufhin in der Konsole ausgegeben und vom Benutzer eingesehen werden können. Eine Liste aller möglichen Fehlermeldungen sowie die dazugehörige Vorgehensweise kann in Kap. 6.4 nachgelesen werden.

- **Kommunikation mit Python:**

Nach Beendigung der Initialisierung muss der Benutzer über die Konsole drei Variablen definieren, die Einfluss auf den Spielverlauf haben.

1. Zunächst wird nach der Stärke der Stockfish-Engine gefragt. Hierbei ist es möglich einen Elo-Wert zwischen 0 und 3000 festzulegen. (Ein Elo-Wert gibt die Stärke eines Schachspielers an)
2. Anschließend kann festgelegt werden, ob der Roboter in der Lage sein soll, das Spiel aufzugeben. Falls diese Frage mit „y“ beantwortet wird, bewertet der Roboter nach jeden Zug die Position und gibt das Spiel auf, falls er einen bestimmten Grenzwert unter- bzw. überschreitet.
3. Letztlich muss der Benutzer auswählen, ob er mit den weißen oder schwarzen Figuren spielen möchte.

- **Das Spiel:**

Abhängig von der gewählten Farbe eröffnet der Benutzer oder der Roboter das Spiel. Im Folgenden wird beschrieben, was zu tun ist, wenn der Benutzer bzw. der Roboter am Zug ist. Beide Punkte werden abwechselnd wiederholt bis das Spiel vorbei ist.

- Benutzer am Zug: Auf dem realen Schachbrett muss ein legaler Zug ausgeführt werden. Falls ein illegaler Zug ausgeführt wird, wird in der Python-Konsole „The following move was detected and is not legal: *move*“ ausgegeben, wobei *move* der erkannte Zug ist.
- Roboter am Zug: Hierbei muss auf den Zug des Roboters gewartet werden. Wenn der Roboter zurück in die Startposition gefahren ist, ist wieder der Benutzer am Zug.

- **Spielende:**

Für das Ende des Spiels gibt es mehrere Möglichkeiten:

- Benutzer gibt auf:
Hierfür müssen, während der Benutzer am Zug ist, beide Könige gedreht werden, sodass sie nicht mehr von der Kamera erkannt werden können.
- Roboter gibt auf:
Für den Fall, dass die Funktion aktiviert wurde.
- Schachmatt für Schwarz oder Weiß.
- Patt für Schwarz oder Weiß.
- Unentschieden durch ungenügendes Material.
- Unentschieden durch die 75-Züge-Regel.
- Unentschieden durch fünfmalige Stellungswiederholung.

Der genaue Ausgang wird am Ende des Spiels in der Konsole ausgegeben.

Um wieder ein neues Spiel zu spielen, ist es nötig die Figuren aufzubauen und das Projekt erneut mit der Steuerung zu synchronisieren. Anschließend müssen wieder die drei Programme gestartet werden und das Spiel kann beginnen.

6.4 Fehlermeldungen

Dieses Kapitel listet alle Fehlermeldungen auf, die vom Autor hart codiert wurden und während der Initialisierung von HALCON in der Konsole von Python ausgegeben werden können und liefert mögliche Lösungsvorschläge. In den meisten Fällen werden lediglich ein paar Versuche benötigt, bis das Schachbrett korrekt gefunden wird, eine minimale Verschiebung des Brettes kann bereits in vielen Fällen das Problem lösen. Falls dennoch die gleiche Fehlermeldung mehrmals über zehn Sekunden auftritt, sollte eine der folgenden Strategien angewandt werden, um das Problem zu lösen, im schlimmsten Fall muss das Programm neu gestartet werden:

- „**There have been ... '1'/'A'/'H's found on the chessboard. It has to be either 1 or 2. Please try again!**“

In jedem der drei Fälle sollte das HALCON-Programm zunächst gestoppt werden. Dann wird ein Breakpoint in Zeile 164 gesetzt und das Programm ab Zeile 150 erneut gestartet. Jetzt können im Grafikfenster die gefundenen Schachbrettmarkierungen von A, H und 1 begutachtet werden. Hierbei lässt sich meist erkennen, was das Problem ist. Beispiele:

- Ein verdrehtes A wird in einer 4 gefunden:
Die 4 kann auf dem realen Schachbrett während der Initialisierung verdeckt werden.
- Eine 1 wird in einer Figur erkannt:
Eine minimale Bewegung der Figur kann die Erkennung verändern.
- Die Markierung wird nirgendwo gefunden:
Die bestimmte Markierung sollte entstaubt und darauf geachtet werden, dass kein Schatten die Markierung verdeckt.

- „**Halcon found not enough/too many pieces on the chessboard. Please try again!**“

In diesem Fall wurden während der 2D-Erkennung weniger bzw. mehr als 32 Figuren erkannt, und das HALCON-Programm kann ebenfalls gestoppt werden. Ein Breakpoint wird in Zeile 278 gesetzt und das Programm ab Zeile 150 ausgeführt. Daraufhin wird im Grafikfenster die aktuelle Position mit den gefundenen Figuren in rot dargestellt. Falls eine Figur nicht rot umrandet ist, kann durch minimale Bewegung der Figur die Erkennung verändert werden. Falls sich eine zusätzliche rote Umrandung einer Figur

auf dem Brett befindet, kann in den meisten Fällen durch Verschiebung der Figuren an den spezifischen Stellen sowie durch Entstaubung das Problem behoben werden.

- „**The corner stars of the chessboard are not found correctly. Please try again!**“

Diese Fehlermeldung gibt an, dass die Sterne nicht korrekt gefunden wurden. In diesem Fall kann das HALCON-Programm gestoppt, ein Breakpoint in Zeile 227 gesetzt und das Programm ab Zeile 150 gestartet werden. Die gefundenen Sterne werden jetzt im Grafikfenster angezeigt. Wie in den vorherigen Fällen kann mittels Entstaubung, Vermeidung von Schatten und Verschiebung des Schachbrettes das Problem in den meisten Fällen behoben werden.

- „**The distance between A and H is out of range. Please try again!**“

Zur korrekten Berechnung der Koordinaten der Felder müssen A und H in der gleichen Zeile gefunden werden. Falls nur jeweils ein A und ein H gefunden werden und die beiden Markierungen auf dem Schachbrett diagonal gegenüberliegen, wird diese Fehlermeldung ausgegeben, hierbei kann das Vorgehen der ersten Fehlermeldung angewandt werden.

- „**The chess starting position could not be detected. Please try again!**“

Diese Fehlermeldung wird ausgegeben, falls die Schachanfangsposition nicht erkannt werden konnte, obwohl exakt 32 Figuren gefunden wurden. Zuerst sollte hierbei überprüft werden, ob die Figuren korrekt aufgestellt worden sind (man beachte die Stellung von König und Dame). Anschließend kann das Vorgehen der zweiten Fehlermeldung angewandt werden.

6.5 Spezialfälle

Dieses Unterkapitel behandelt verschiedene Spezialfälle, die vermieden werden sollten, um einen reibungslosen Ablauf des Schachspiels zu garantieren.

- Zwei oder mehr Figuren stehen zu nah beieinander:

In diesem Fall kann es sein, dass der Roboter nicht in der Lage ist, die Figur zu greifen, da die Greiffinger zu wenig Platz haben. Im schlimmsten Fall kann es zu einer Kollision kommen.

- Eine Figur befindet sich auf der Kante zwischen zwei Feldern:

Hierbei ist es möglich, dass die FEN nicht korrekt berechnet wird.

- Die Kamera wurde minimal bewegt:

Falls die Kamera, aufgrund eines Transports oder dergleichen bewegt wurde, ist es möglich, dass die Figuren nicht mehr gefunden oder die Koordinaten falsch berechnet werden. Durch minimale Bewegungen der Kamera nach unten bzw. oben kann das Problem behoben werden, im schlimmsten Fall müssen die Hyperparameter, die die Entfernung von Kamera und Figur angeben, neu eingestellt und die Transformationsmatrix für die HAK neu berechnet werden.

- Bestimmte Drehungen der Figuren:

Wenn die Figuren in Bezug auf das Schachbrett in einem Winkel von 30° – 60° ausgerichtet sind, ist die Wahrscheinlichkeit deutlich erhöht, einen schwarzen Bauern auf einem schwarzen Feld zwischen verdrehter Figur und Kante des Feldes zu finden. Obwohl das Schachprogramm in der Lage ist, verdrehte Figuren korrekt zu greifen, werden potenzielle Fehler minimiert, wenn die Figuren gerade auf das Brett gestellt werden.

- Drehung des Brettes während des Spiels:

Nach der Initialisierung von HALCON darf das Schachbrett nicht mehr verschoben werden, da die Markierungen aus zeitlichen Gründen nicht für jeden Zug erneut im Bild gesucht werden.

- Aus-Feld zu nah am Brett:

Wie bereits erwähnt, sollte die Ablage, die das Aus-Feld repräsentiert, 1 cm vom Brett entfernt platziert werden. Falls der Rand der Ablage (z. B. ein Karton) eine Höhe von 3 cm überschreitet, eine gewisse Dicke aufweist und sich zu nah am Brett befindet, kann es passieren, dass die Ablage bei der 3D-Erkennung der Figuren gefunden wird. Das hat eine falsche Berechnung der Transformationsmatrizen und den daraus resultierenden Koordinaten zur Folge.

6.6 Berechnung der Transformationsmatrix für die Hand-Augen-Kalibrierung

In diesem Kapitel wird die Berechnung einer neuen Transformationsmatrix für die HAK beschrieben. Hierfür werden das HALCON-Programm *HandEyeCalibration_Halcon* sowie das Sunrise-Projekt benötigt.

Vorbereitung:

- Im Sunrise-Programm wird die Datei *ServerTask.java* im Paket *server* verändert, wofür lediglich die Zeilen 62 und 67 auskommentiert werden müssen, um eine Client-Server-Kommunikation über zwei Geräte aufzubauen zu können.
- Anschließend muss das Projekt mit der Steuerung synchronisiert werden (vgl. Kap. [6.2](#)).
- Die Stereokamera muss den Status „Available“ aufweisen (vgl. Kap. [6.2](#)).
- Eine Figur ($3 \times 3 \times 3$ cm Würfel) wird möglichst mittig an der Spitze des TCP angebracht. Für eine bessere Erkennung der Figur sollte die beklebte Seite nach unten zeigen.
- Der Roboter muss in eine Position gefahren werden, sodass die Figur sichtbar für die Kamera ist.

Programmausführung:

Anfangs muss auf dem SmartPAD das Programm *HandEyeCalibration* ausgewählt und gestartet, dann kann das HALCON-Programm *HandEyeCalibration_Halcon* ausgeführt werden. Nach der Initialisierung stoppt das Programm kurz vor der while-Schleife in Zeile 45. Falls die aktuelle Position des Roboters zufriedenstellend ist, kann das Programm weiter ausgeführt werden. Am Ende jeder Iteration wird der gefundene Würfel in einer 3D-Szene visualisiert, wobei es zwei Möglichkeiten gibt:

- Würfel wurde nicht korrekt gefunden:

In diesem Fall muss der Roboter in eine neue Position gefahren und das Programm an den Anfang der while-Schleife in Zeile 52 gesetzt werden, um die falschen Koordinaten nicht abzuspeichern.

- Würfel wurde korrekt gefunden:

In diesem Fall muss im Grafikfenster auf „Continue“ gedrückt und anschließend das Programm bis zum Ende der while-Schleife ausgeführt werden, um die Koordinaten des Würfels im Roboter- und im Kamerakoordinatensystem zu speichern.

Daraufhin wird das Programm vor der nächsten Iteration gestoppt. Jetzt kann der Roboter in eine neue Position gefahren werden und obiger Prozess wiederholt sich mehrere Male. Die Anzahl der Iterationen kann gegebenenfalls in den Hyperparametern des HALCON-Programms geändert werden.

Nachdem die Koordinaten aller Positionen in den Listen gespeichert wurden, wird die Transformationsmatrix berechnet und in dem Ordner gespeichert, in dem das HALCON-Programm aufgerufen wurde.

Zur Überprüfung der Genauigkeit der Transformationsmatrix wird im letzten Schritt der quadratische Fehler berechnet. Falls der Wert zu hoch ausfällt, sollte das Programm wiederholt werden.

Bevor ein neues Schachspiel gestartet werden kann, muss die Kommentierung der Zeilen 62 und 67 wieder entfernt und das Projekt neu synchronisiert werden.

7 Implementierung

Dieses Kapitel geht ausführlich auf die Implementierung der Programme aus Kap. 5.3 ein und dient als Dokumentation des Quellcodes.

7.1 Client-Server-Architektur

Das Serverprogramm hat eine feste IP-Adresse und läuft als Hintergrundanwendung auf dem Roboter (vgl. Abb. 5.9). Nach dem Starten des Hauptprogramms von Sunrise arbeitet der Server ununterbrochen bis zum Empfangen eines terminalen Strings, welcher das Programm schließt. Der Server wurde von Herrn Jürgen Schwittek entwickelt und besteht aus drei Klassen, die nachfolgend erläutert werden:

- **RobertaServer:**

Die Klasse *RobertaServer* ist in der Lage, Strings zu empfangen und zu versenden. Um das zu bewerkstelligen, wird in der Funktion *start* ein Serversocket geöffnet, außerdem werden Instanzen der Read- und Write-Klassen angelegt (*InputStreamReader*, *BufferedReader*, *PrintWriter*), um Zeichen-Ströme verwalten zu können. Alle anderen Funktionen verwenden die Read- und Write-Klassen, um Nachrichten zu senden, zu empfangen oder den Server zu stoppen.

- **ProgramServerMaster:**

Mithilfe der Klasse *ProgramServerMaster* wird eine Kommunikation mit mehreren Clients ermöglicht, indem zunächst mehrere Instanzen der Klasse *RobertaServer* initialisiert werden. Die Funktion *startServerMaster* legt die Portnummern der einzelnen Instanzen fest und startet die Verbindung über die oben beschriebene Funktion *start*. Die wichtigsten Funktionen der Klasse sind für das Senden und Empfangen der Strings zuständig und werden anschließend beschrieben:

- *receiveString* ist für das Empfangen von Strings sowie deren Verteilung an die einzelnen Server verantwortlich.

- *setSendString* überprüft zunächst, ob der zu versendende String eine Endung der Form „@xxx“ enthält, wobei „xxx“ den Namen eines Clients entspricht. In diesem Fall wird der String ausschließlich an den angegebenen Server gesendet. Ohne diese Endung wird der String an alle Instanzen von *RobertaServer* weitergeleitet.

- **ServerTask:**

Die Klasse *ServerTask* arbeitet mit *RoboticsAPICyclicBackgroundTask*, einer Erweiterung, um bestimmte Aufgaben im Hintergrund des Hauptprogramms auszuführen, die zyklisch gestartet werden sollen. Im Prinzip ist *ServerTask* ein Programm zur Verwendung der *ProgramServerMaster*-Klasse. Es werden zunächst die Portnummern und Namen der einzelnen Server initialisiert sowie der terminale String zur Beendigung der Verbindung definiert. Anschließend befindet sich das Programm in einer while-Schleife, um Strings zu empfangen und an die Server zu verteilen. Die Schleife wird erst nach Erhalt des terminalen Strings verlassen und das Programm beendet.

Jedes der drei Hauptprogramme (Python, HALCON und Sunrise) benötigt eine Client-Klasse, um mithilfe des Servers mit den anderen Clients kommunizieren zu können. Die Initialisierung sowie Anwendung der Klassen in den einzelnen Programmen ist sehr ähnlich und wird in den dafür vorgesehenen Unterkapiteln behandelt.

7.2 Python

Das Python-Programm kann als eine Art Verteilungsprogramm interpretiert werden. Es verarbeitet das gesamte Schachspiel auf einem virtuellen Brett, kommuniziert über die Konsole mit dem Benutzer und sorgt dafür, dass alle in HALCON berechneten Koordinaten zum richtigen Zeitpunkt an das Sunrise-Programm weitergeleitet werden. Um diese Aufgaben umzusetzen, werden in Python drei Klassen inklusive Hauptprogramm verwendet, welche in den folgenden Unterkapiteln beschrieben werden.

7.2.1 ClientClass

Die Klasse *ClientClass* ist für die Kommunikation mit dem Serverprogramm zuständig. Hierfür wird mit dem Modul *socket* gearbeitet. Das Programm beinhaltet verschiedene Funktionen, die anschließend erklärt werden:

- **start:**

Die Funktion *start* öffnet mit der angegebenen IP und Portnummer eine Socket-

Verbindung auf dem Computer. Hierfür wird die interne socket-Funktion *connect* verwendet.

- **setHost:**

Hiermit kann die IP-Adresse eingestellt werden. Der Standardwert ist „127.0.0.1“.

- **setPort:**

Damit kann die Portnummer festgelegt werden. Der Standardwert ist „30003“.

- **setSendString:**

Die Funktion *setSendString* ist für das Senden beliebiger Strings mit der internen socket-Funktion *send* zuständig. Vor dem Senden muss dem String ein Zeilenumbruch hinzugefügt werden, um das Ende des Strings kenntlich zu machen.

- **setReceivedString:**

Diese Funktion ist dafür zuständig, alle empfangenen Strings in der Liste *receivedString* zu speichern. Das Anlegen einer Liste hilft dabei, alle empfangenen Daten sukzessiv bearbeiten zu können.

- **receiveString:**

Mit der Funktion *receiveString* können beliebige Strings empfangen werden. Hierfür wird die Funktion *recv* verwendet, die das Modul socket bereitstellt. Anschließend wird der Zeilenumbruch vom String entfernt und die Funktion *setReceivedString* aufgerufen, um den empfangenen String abzuspeichern.

- **hasReceivedString:**

Diese booleschen Funktion gibt an, ob ein String empfangen wurde. Hierfür wird die obige Funktion *receiveString* aufgerufen. Daraufhin wird abgefragt, ob die Länge des empfangenen Strings in der Liste *receivedString* größer als null ist. Abhängig hiervon wird der Rückgabewert auf True oder False gesetzt.

- **getReceivedString:**

Mithilfe der Funktion *getReceivedString* können die empfangen Strings abgefragt werden, wobei das erste Element aus der Liste *receivedString* als Rückgabewert verwendet wird, anschließend wird der String aus der Liste entfernt.

- **stop:**

Die Funktion *stop* schließt die bestehende Socket-Verbindung mit der internen socket-Funktion *close*.

7.2.2 chessCalculator

Die Klasse *chessCalculator* beinhaltet verschiedene Funktionen zur Berechnung schachspezifischer Fragestellungen, die im Hauptprogramm benötigt und nicht von der Bibliothek chess bereitgestellt werden. Die Hauptaufgabe besteht darin, einen Zug aus zwei aufeinanderfolgenden FEN-Strings zu bestimmen (vgl. Kap. 5.3), wofür eine Funktion namens *move_calculator* zuständig ist, die mehrere Unterfunktionen verwendet, welche zunächst erklärt werden.

- **boardConfig_to_matrix:**

Diese Funktion konvertiert die Figurenstellung einer FEN in einen 8×8 -numpy-Array, der das Schachbrett repräsentiert. In Abb. 7.1 ist ein Beispiel für eine bestimmte Schachposition inklusive der Figurenstellung und dem resultierenden Array gegeben.

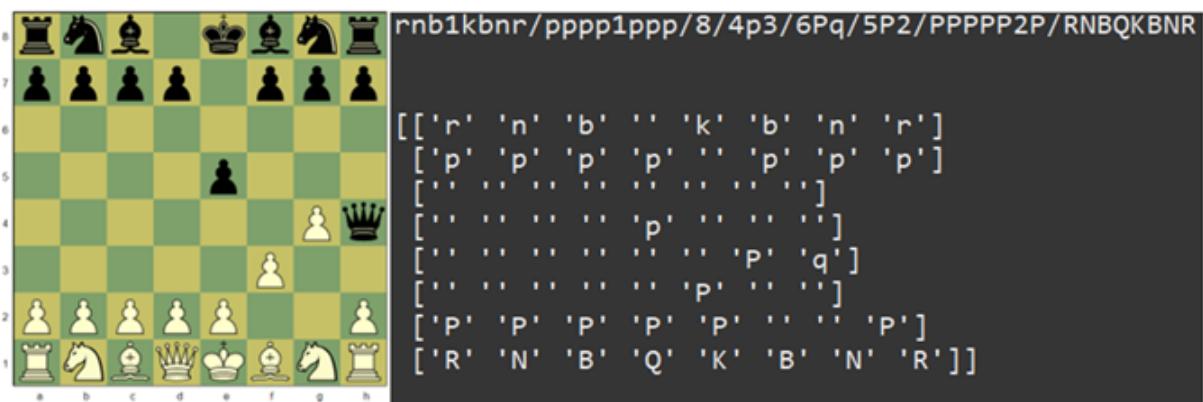


Abbildung 7.1 Beispiel einer Schachposition mit dazugehöriger Figurenstellung und dem mit *boardConfig_to_matrix* resultierenden numpy-Arrays

- **fens_to_difference_list:**

Diese Funktion ist dafür zuständig, aus zwei 8×8 -numpy-Arrays eine Liste mit Unterschieden zwischen den Arrays zu generieren. Die Einträge der Liste beinhalten jeweils ein spezifisches Feld und geben an, welche Figur sich vor und nach einem Zug auf dem Feld befindet, wodurch ein Element der Liste aus einem Tripel der Form [Feld, Figur vor dem Zug, Figur nach dem Zug] besteht. Im Quellcode werden die Arrays zeilenweise verglichen und die Ergebnisse in einer Liste gespeichert. In Abb. 7.2 sind zwei aufeinanderfolgende Schachpositionen abgebildet sowie die dazugehörige Liste der Unterschiede. Diese zeigt, dass auf dem Feld B4 ein schwarzer Läufer durch ein leeres Feld ersetzt wurde und auf dem Feld C3 ein schwarzer Läufer den Platz eines weißen Springer ersinnimmt.

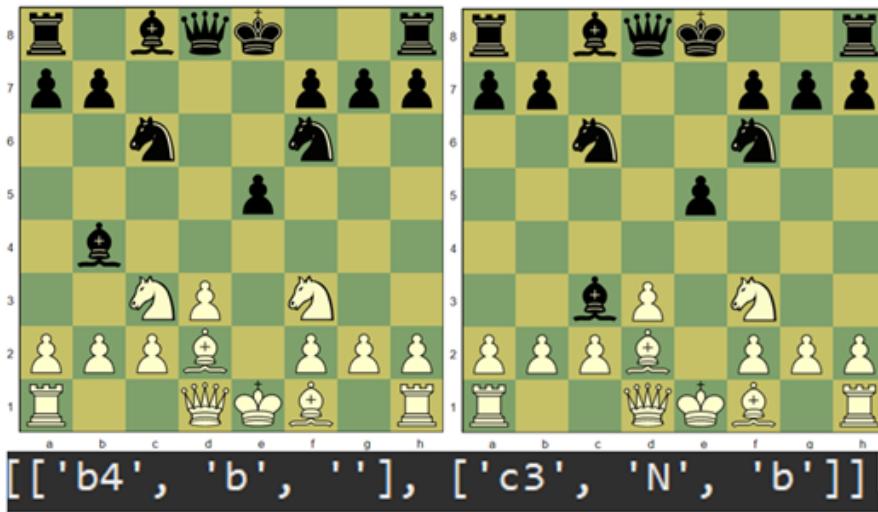


Abbildung 7.2 Zwei aufeinanderfolgende Schachpositionen und die dazugehörige Liste der Unterschiede, die mit der Funktion *fens_to_difference_list*, generiert wurde

- **difference_list_to_move:**

Diese Funktion verwendet die zuvor generierte Liste der Unterschiede und die FEN der ursprünglichen Position, um daraus einen Schachzug zu generieren. Nebenbei wird zusätzlich eine Liste der Bewegungen angelegt, die für den Roboter zur Umsetzung dieses Zuges notwendig ist. Da die Liste für jeden legalen Zug zwei, drei oder vier Elemente enthält, gibt es drei große Fallunterscheidungen. Wenn die Liste zwei Elemente beinhaltet, kann es sich um einen normalen Zug, das Schlagen einer Figur, eine Bauernumwandlung oder das Schlagen einer Figur inklusive Bauernumwandlung handeln. Im Spezialfall En-passant kann die Liste nur drei Elemente und für jede der Rochaden vier Elemente enthalten. Um aus der Liste einen Zug zu generieren, werden die Start- und Ziel-Felder zusammengefügt. Im Beispiel von Abb. 7.2 resultiert mit der Funktion *difference_list_to_move* der Zug B4C3 und die Liste der Bewegungen [‘c3’, ”, ‘b4’, ‘c3’]. Beginnend bei 0 geben die geraden Einträge der Liste die Start-Felder und die ungeraden Einträge die Ziel-Felder an, d. h. der Roboter muss zunächst die Figur auf dem Feld C3 in das Aus-Feld und anschließend die Figur von B4 auf das Feld C3 platzieren. Eine weitere Variable, die die Funktion zurückgibt, gibt mit einem booleschen Wert an, ob der Zug legal sein kann. Hierfür werden aus der ursprünglichen FEN die Informationen bzgl. eines möglichen En-passant-Schlags und den Rochaderechten abgefragt. Falls die Liste der Unterschiede beispielsweise drei Einträge besitzt, sich jedoch in der ursprünglichen FEN an der Stelle eines möglichen En-passant-Schlags ein „-“ befindet, wird die boolesche Variable auf False gesetzt.

- **move_calculator:**

Die Funktion *move_calculator* berechnet einen Zug aus zwei aufeinanderfolgenden Positionen. Hierfür wird die FEN der ursprünglichen Position und die Figurenstellung der neuen Position übergeben. Zunächst werden aus den beiden Figurenstellungen der Positionen mit der Funktion *boardConfig_to_matrix* zwei 8×8 -numpy-Arrays erzeugt. Anschließend wird aus den Arrays und der Funktion *fens_to_difference_list* eine Liste der Unterschiede erstellt. Mit *difference_list_to_move* kann daraufhin aus der Liste und der ursprünglichen FEN der Zug, die Bewegungsliste und eine boolesche Variable generiert werden, die angibt, ob der Zug legal sein kann. Abschließend werden für den Fall eines illegalen Zuges die Bewegungsliste und der Zug geleert.

- **is_move_legal:**

Diese Funktion gibt an, ob ein spezifischer Zug in einer Position legal ist oder nicht. Hierfür wird die ursprüngliche FEN und der Zug als String übergeben. Mithilfe der chess-Bibliothek wird zunächst ein virtuelles Schachbrett mithilfe der FEN erstellt, über die interne chess-Funktion *legal_moves* kann daraufhin eine Liste aller legalen Züge generiert werden. Abschließend überprüft die Funktion, ob sich der angegebene Zug in der Liste aller legalen Züge befindet, speichert das Resultat in einer booleschen Variable und gibt diese zurück.

- **move_to_move_list:**

Die Funktion *move_to_move_list* erstellt aus einer FEN und einem Zug eine Bewegungsliste für den Roboter. Hierfür wird aus der FEN und der chess-Bibliothek ein virtuelles Schachbrett erstellt, auf dem der Zug ausgeführt wird. Aus der resultierenden Position wird die Figurenstellung gespeichert, abschließend kann die Funktion *move_calculator* aus der ursprünglichen FEN und der neuen Figurenstellung die Bewegungsliste generieren.

- **kings_in_boardConfig:**

Um ein Schachspiel gegen den Roboter aufzugeben, müssen beide Könige so gedreht werden, dass sie von der Kamera nicht mehr erkannt werden können. Die Funktion *kings_in_boardConfig* überprüft nun, ob sich in der Figurenstellung der Buchstabe „k“ bzw. „K“ befindet.

7.2.3 PyClient

Die Klasse *PyClient* ist das Hauptprogramm des Python-Codes und für die Kommunikation mit dem Benutzer und die Verteilung der Koordinaten zuständig. Nachfolgend wird die Struktur der Klasse zum besseren Verständnis stichpunktartig dargestellt und anschließend die unterstrichenen Abschnitte im Detail erklärt.

Definitionen und Initialisierungen

Warten auf die Initialisierung von HALCON (vgl. Kap. 7.3.2.1)

Kommunikation mit dem Benutzer (Stärke des Roboters, Aufgabe, Farbe des Spielers)

while(Spiel ist nicht vorbei)

Fallunterscheidung:

1. Benutzer ist am Zug:

 Verarbeite Figurenstellung von HALCON

2. Roboter ist am Zug:

 Berechne Zug mithilfe einer Schachengine bzw. einer Schach-KI

end while

Abschluss (Trennung der Verbindung, Ausgabe des Siegers)

Definitionen und Initialisierungen:

Für das Hauptprogramm werden folgende Objekte angelegt:

- Der Client wird initialisiert, die IP und Portnummer festgelegt und mit der Funktion *start* gestartet.
- Eine Instanz der Klasse *chessCalculator* wird initialisiert.
- Die Schachengine stockfish muss initialisiert werden, indem der Pfad zur Datei angegeben wird.
- Mit der internen chess-Funktion *Board* wird ein virtuelles Schachbrett angelegt, dass dem realen Schachspiel folgt.

Warten auf die Initialisierung von HALCON:

Um auf die Initialisierung von HALCON zu warten, befindet sich das Programm in einer while-Schleife. In jedem Durchlauf wird mit der Funktion *hasReceivedString* abgefragt, ob der Python-Client einen String erhalten hat. Mit der Funktion *getReceivedString* kann der

empfangene String daraufhin gespeichert werden. Falls es sich bei dem String um die Schachanfangsposition inklusive der Rotation des Schachbrettes handelt, wird die while-Schleife verlassen. Für jeden anderen Fall gibt es eine spezifische Fehlermeldung, die in der Konsole ausgegeben wird, um dem Nutzer mitzuteilen, wieso das Schachbrett nicht korrekt erkannt werden konnte. Dadurch können minimale Anpassungen am Brett oder den Figuren vorgenommen werden, um die Position in der nächsten Iteration zu erkennen. In Kap. 6.4 sind alle Fehlermeldungen inklusive einer kurzen Anleitung aufgelistet.

Kommunikation mit dem Benutzer:

In diesem Abschnitt muss der Benutzer über die Konsole drei Variablen angeben, die den Ablauf des Schachspiels beeinflussen. Mit der internen Python-Funktion *input* kann eine Konsoleneingabe erfolgen, anschließend wird überprüft, ob die Eingabe im gewünschten Wertebereich liegt. Abhängig hiervon wird die Eingabe entweder verwendet oder verworfen und nach einem neuen Input gefragt.

- Stärke des Roboters:

Mithilfe der Python-Bibliothek stockfish werden die Züge des Roboters berechnet, wobei über die Funktion *set_elo_rating* die Stärke des Roboters festgelegt werden kann. Daher wird der Benutzer in diesem Abschnitt nach einer Elo-Zahl zwischen 0 und 3000 gefragt.

- Aufgabe:

Daraufhin wird der Nutzer gebeten mit „y“ für ja oder „n“ für nein festzulegen, ob der Roboter die Möglichkeit haben soll, das Spiel aufzugeben. Da stockfish mit der Funktion *get_evaluation* in der Lage ist, die aktuelle Position numerisch zu bewerten, wird er im Falle von „y“ das Spiel aufgeben, wenn der Wert unter bzw. über einen bestimmten Grenzwert fällt.

- Farbe des Spielers:

Abschließend muss der Benutzer die gewünschte Spielfarbe mit „w“ für Weiß oder „b“ für Schwarz festlegen.

Benutzer ist am Zug:

Zu Beginn wird mit der Funktion *setSendString* der String „boardConfig“ an HALCON gesendet, um nach der aktuellen Position in Form einer Figurenstellung der FEN zu fragen. Anschließend befindet sich das Programm in einer while-Schleife, die erst verlassen wird,

wenn entweder ein legaler Zug erkannt wurde oder der Benutzer das Spiel aufgegeben hat. Zunächst wird in der Schleife mit den Funktionen *hasReceivedString* und *getReceivedString* auf die Antwort von HALCON gewartet und diese abgespeichert. Daraufhin wird mit der Funktion *kings_in_boardConfig* aus der Klasse *chessCalculator* überprüft, ob die empfangene Figurenstellung beide Könige enthält. Falls das nicht der Fall ist, hat der Benutzer das Spiel aufgegeben und das Programm endet. Im nächsten Schritt wird mit der Funktion *move_calculator* der Zug zwischen der ursprünglichen FEN und der empfangenen Figurenstellung berechnet. Danach kann mit der Funktion *is_move_legal* festgestellt werden, ob der Zug legal ist, woraufhin es eine Fallunterscheidung gibt:

- Zug ist legal:

Der Zug wird auf dem virtuellen Brett ausgeführt, und es wird überprüft, ob das Spiel vorbei ist. Anschließend wird die while-Schleife verlassen.

- Zug ist illegal:

Python sendet erneut den String „boardConfig“ an HALCON, um auf eine neue Figurenstellung zu warten und durchläuft die obige while-Schleife erneut.

Roboter ist am Zug:

Zunächst wird der Instanz der stockfish-Bibliothek mit der Funktion *set_fen_position* die aktuelle FEN übergeben. Anschließend werden anhand der Aussage des Benutzers zur Aufgabe des Roboters sowie der aktuellen Bewertung der Position festgelegt, ob der Roboter das Spiel aufgeben soll. Hierbei muss auf die Farbe des Nutzers geachtet werden, da sich dadurch das Vorzeichen der Bewertung ändert. Daraufhin wird die Stärke von stockfish mit *set_elo_rating* angepasst und der Zug mit *get_best_move* berechnet. Um aus diesem Zug die Koordinatenliste für den Roboter zu generieren, sind mehrere Schritte notwendig, die nachfolgend stichpunktartig aufgelistet sind.

- Zunächst wird eine Bewegungsliste mit der Funktion *move_to_move_list* aus der Klasse *chessCalculator* erstellt.
- Anschließend wird über alle geraden Einträge in der Bewegungsliste iteriert und diese jeweils in der Variable *field* gespeichert. Der String „get_pose_“ + *field* wird daraufhin an HALCON gesendet, um die Pose der Figur im Weltkoordinatensystem des Roboters zu erhalten. Alle Posen werden in einer Liste gespeichert.
- Analog zum vorherigen Stichpunkt wird über alle ungeraden Einträge in der Bewegungsliste iteriert und in der Variable *field* gespeichert. Daraufhin wird der String

„get_empty_field_pose“ + *field* an HALCON gesendet, um die Translationskoordinaten der Mitte des leeren Feldes im Weltkoordinatensystem des Roboters berechnen zu lassen. Alle empfangenen Nachrichten von HALCON werden in einer Liste gespeichert.

- Um nicht mehrere Strings senden zu müssen, werden alle nötigen Informationen in einem String zusammengefasst und an Sunrise gesendet. Hierfür werden zunächst jeweils alle Einträge aus den obigen Listen mit einem „_“ voneinander getrennt und die resultierenden Strings separiert durch ein \$-Zeichen zusammengefügt. Zusätzlich wird die Rotation des Schachbrettes, die zu Beginn des Hauptprogramms mit der Schachanfangsposition gesendet worden ist, ebenfalls getrennt mit einem \$-Zeichen, zu dem String hinzugefügt. Der resultierende String hat im Falle des Schlagens einer Figur folgende Form (die Details werden in Kap. 7.3.2.2 erklärt, wo die Erstellung des Strings erläutert wird):

```
„TransX/TransY/TransZ/RotX/RotY/RotZ/angle2D/nopromotion_
TransX/TransY/TransZ/RotX/RotY/RotZ/angle2D/nopromotion$
TransX/TransY/TransZ/out_TransX/TransY/TransZ/in$anglechessboard“.
```

Nach dem Senden des Strings wartet Python in einer while-Schleife auf die Antwort von Sunrise, die nach Ausführung des Zuges durch den Roboter gesendet wird. Abschließend wird der Zug auf das virtuelle Schachbrett angewendet und überprüft, ob das Spiel vorbei ist.

Abschluss:

Dieser Abschnitt wird nur ausgeführt, wenn das Spiel vorbei ist. Zunächst wird der terminale String an alle Clients weitergeleitet, um die Verbindung zu trennen. Danach wird mithilfe des virtuellen Brettes der Ausgang des Spiels bestimmt und in der Konsole ausgegeben. Abschließend wird der Client mit der Funktion *stop* geschlossen.

7.3 HALCON

Zur Umsetzung aller nötigen Aufgaben werden in HALCON lediglich zwei Programme benötigt. Zunächst wird in Kap. 7.3.1 auf alle Client-Funktionen eingegangen, die in den beiden Programmen verwendet werden. Daraufhin wird im Hauptprogramm die Implementierung der Unterprogramme aus Kap. 5.3 ausführlich erklärt und dabei beschrieben, wie eine kontinuierliche Kommunikation mit dem Python-Programm während des gesamten Schachspiels

aufrecht erhalten wird. Abschließend wird in Kap. 7.3.3 die HAK durchgeführt, um eine Transformationsmatrix zwischen Kamera- und Roboterkoordinatensystem zu generieren.

7.3.1 Client-Programm

Das Client-Programm für HALCON zur Kommunikation mit anderen Clients, die mit dem Server auf dem Roboter verbunden sind, wie z. B. der Python-Client, arbeitet mit vier elementaren Funktionen, welche nachfolgend beschrieben werden:

Client_Connect:

Client_Connect verwendet die interne HALCON-Funktion *open_socket_connect*, um über die angegebene IP und Portnummer eine Socket-Verbindung zu einem akzeptierenden Socket auf dem Rechner zu öffnen [44], außerdem wird eine Liste angelegt, welche die empfangenen Strings speichert.

Client_Send:

Client_Send ist für das Senden beliebiger Daten an externe Geräte oder Anwendungen über eine generische Socket-Verbindung zuständig, welche mit der internen HALCON-Funktion *send_data* arbeitet [45]. Es ist zu beachten, dass an den zu sendenden String stets ein Zeilenumbruch angehängt wird, sodass HALCON das Ende einer Nachricht erkennt.

Client_Receive:

Client_Receive ist für das Empfangen beliebiger Daten von externen Geräten oder Anwendungen über eine generische Socket-Verbindung zuständig [46]. Hierfür wird zunächst die interne HALCON-Funktion *receive_data* verwendet. Anschließend wird, gegenteilig zur Funktion *Client_Send*, der Zeilenumbruch vom String entfernt. Daraufhin speichert die Funktion den String in die Liste, die in *Client_Connect* angelegt wurde, und verwendet den ersten Eintrag als empfangenen String. Dieses Vorgehen verhindert die Überlagerung mehrerer Zeichenströme und alle Nachrichten können sukzessiv abgearbeitet werden.

Client_Stop:

Client_Stop sorgt mithilfe der internen HALCON-Funktion *close_socket* dafür, dass die Socket-Verbindung geschlossen wird.

7.3.2 Hauptprogramm

Zur besseren Übersichtlichkeit wird das Hauptprogramm zunächst stichpunktartig dargestellt. Es kann grob in die Initialisierung bzw. Vorbereitung und die eigentliche Kommunikation mit Python während des Schachspiels aufgeteilt werden. In den darauf folgenden Unterkapiteln werden die unterstrichenen Abschnitte im Detail erklärt.

Initialisierung (vgl. Kap. 7.3.2.1):

Definitionen (Anlegen der Hyperparameter)

Verbindungsaufbau (Server, Kamera)

Modellgenerierung (Oberflächenmodell eines Würfels, formbasierte Modelle für
die Figuren sowie die Schachbrettmarkierungen)

while(FEN-String entspricht nicht der Schachanfangsposition)

Bildanalyse (Erstellung eines FEN-Strings der aktuellen Position)

end while

Transformationsmatrix (Generierung aller Transformationsmatrizen)

Kommunikation mit Python (vgl. Kap. 7.3.2.2):

while(empfangener String ist ungleich dem terminalen String)

 Warte auf String von Python → Fallunterscheidung:

 1. „boardConfig“:

 Neues 2D-Bild aufnehmen, FEN berechnen und an Python senden

 2. „get_pose“ + Feld:

 Berechne die Pose der Figur auf dem Feld und sende sie an Python

 3. „get_empty_field_pose“ + Feld:

 Berechne Koordinaten eines leeren Feldes und sende sie an Python

 4. Terminaler String:

 Beendet die while-Schleife und damit das Hauptprogramm

end while

7.3.2.1 Initialisierung

Die Initialisierung behandelt alle nötigen Vorbereitungen, die getroffen werden müssen, um die Aufgaben, die HALCON während eines Schachspiels von Python erhält, abarbeiten zu können. Hierfür wird das Unterkapitel in vier Abschnitte aufgeteilt, welche in der Zusammenfassung zu Beginn des Kapitels unterstrichen sind. Auf den Abschnitt „Definitionen“ wird

nicht gesondert eingegangen, da hier lediglich alle veränderbaren Hyperparameter angelegt werden, die im Quellcode Verwendung finden. Bei einer Änderung bestimmter Bedingungen, wie z. B. der Abstand der Stereokamera zum Schachbrett oder die verwendete IP-Adresse, müssen diese Hyperparameter gegebenenfalls angepasst werden.

Verbindungsauflaufbau:

In diesem Abschnitt wird die Socket-Verbindung zum Server über *Client_Connect* geöffnet (vgl. Kap. 7.3.1). Anschließend öffnet der Operator *open_framegrabber* ein angegebenes Bildaufnahmegerät, in diesem Fall die Stereokamera IDS Ensenso N35. Dadurch wird die Verbindung zur Kamera getestet, die Kamera für andere Prozesse gesperrt und ggf. Speicher für den Puffer reserviert. Die eigentliche Bildaufnahme erfolgt über die Operatoren *grab_image_async* für 2D-Bilder und *grab_data* für 3D-Bilder [47].

Modellgenerierung:

Dieser Abschnitt ist für die Generierung zahlreicher Modelle zuständig, welche nachfolgend beschrieben werden:

- Oberflächenmodell eines Würfels:

Zur Erstellung eines Oberflächenmodells für den Würfel wird zunächst ein CAD-Modell erstellt und in Form einer PLY-Datei über den Operator *read_object_model_3d* eingelesen. Das PLY-Dateiformat ist ein einfaches Format zur Beschreibung eines einzelnen Objekts als polygonales Modell [48]. Es finden sich zahlreiche vorgefertigte PLY-Dateien eines einfachen Würfels im Netz. Dabei muss auf die Größe des Würfels geachtet werden, welche innerhalb der Datei über den Aufenthaltsort der Eckpunkte definiert werden kann. Da hier als Spielfiguren Holzwürfel mit der Kantenlänge 3 cm verwendet werden (vgl. Kap. 5.2), eine PLY-Datei die Maßeinheit mm verwendet und der Schwerpunkt des Würfels im Nullpunkt des Koordinatensystems liegt, wurden die folgenden acht Eckpunkte des Würfels definiert:

$$\begin{aligned}
 x_1 &= -15 & y_1 &= -15 & z_1 &= -15 \\
 x_2 &= 15 & y_2 &= -15 & z_2 &= -15 \\
 x_3 &= 15 & y_3 &= 15 & z_3 &= -15 \\
 x_4 &= -15 & y_4 &= 15 & z_4 &= -15 \\
 x_5 &= -15 & y_5 &= -15 & z_5 &= 15 \\
 x_6 &= 15 & y_6 &= -15 & z_6 &= 15 \\
 x_7 &= 15 & y_7 &= 15 & z_7 &= 15 \\
 x_8 &= -15 & y_8 &= 15 & z_8 &= 15
 \end{aligned}$$

Das eingelesene 3D-Objektmodell wird mit der Funktion `create_surface_model`, welche in Kap. 4.2 erläutert wird, in ein oberflächenbasiertes Modell transformiert. Mithilfe von Tests konnte der beste Kompromiss zwischen Geschwindigkeit und Robustheit des Suchalgorithmus mit einem *RelSamplingDistance*-Wert von 0.15 erreicht werden. Zusätzlich werden die Parameter `train_3d_edge` und `train_view_based` auf True gesetzt, um später eine optimale Suche zu gewährleisten.

- Formbasierte Modelle aller wichtigen Schachbrettmarkierungen:

Für die Erkennung des Schachbrettes sowie die spätere Berechnung der Figurenstellung einer Position werden Markierungen auf dem Brett benötigt. Um das gesamte Schachbrett zu erkennen, werden die Sterne an den Ecken verwendet, welche später ein Viereck bilden, das das Brett einschließt. Zur Berechnung aller Felder werden die Koordinaten eines Referenzfelds sowie ein Vektor, der parallel zum Schachbrett ausgerichtet ist, benötigt. In dieser Arbeit werden das Feld A1 und der Vektor \overrightarrow{AH} verwendet. Insgesamt werden die Markierungen von A, H, 1 und * benötigt, welche in Abb. 7.3 mit einer roten Umrandung dargestellt sind. Das Bild, das verwendet wird, um formbasierte Modelle der Markierungen zu generieren, wird von der Stereokamera IDS Ensenso N35 aufgenommen. Dies hat den Vorteil, dass ein neues Bild, in dem die Markierungen während eines Schachspiels gesucht werden, dasselbe Schachbrett, dieselben Markierungen und denselben Abstand zwischen Brett und Kamera verwenden und somit die Suche vereinfacht wird.

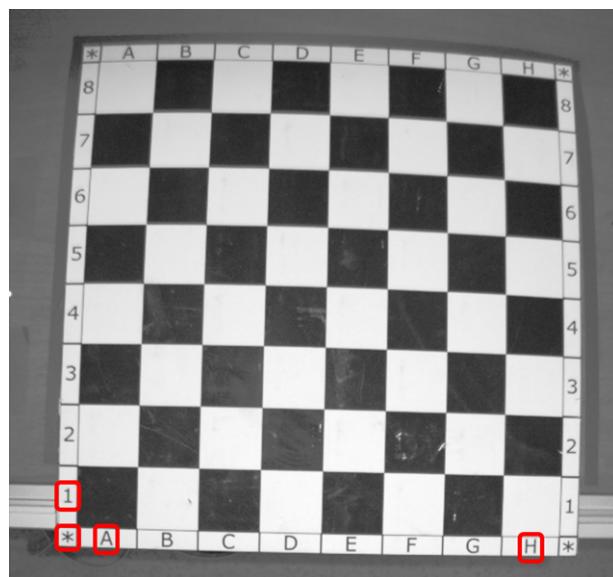


Abbildung 7.3 Musterbild eines leeren Schachbrettes zur Erstellung der formbasierten Modelle aller wichtigen Schachbrettmarkierungen

Im Quellcode wird zunächst das Bild aus Abb. 7.3 (ohne rote Umrandung) eingelesen. Daraufhin werden für die vier Markierungen Region of Interests (ROI) erstellt, welche aus dem Bild ausgeschnitten werden (rote Umrandungen in Abb. 7.3). Mit der Funktion `reduce_domain` werden die Vorlagen zur Erstellung des formbasierten Modells vorbereitet. Jetzt kann `create_shape_model` (vgl. Kap. 4.2) ein Modell für jede Schachbrettmarkierung erstellen. Für die Parameter der Funktion sollte noch darauf geachtet werden, dass `AngleStart = 0` und `AngleExtent = rad(360)` eingestellt sind, um das formbasierte Modell in jeder möglichen Rotation finden zu können.

- Formbasierte Modelle der Figuren:

Das Vorgehen für die formbasierten Modelle der Figuren orientiert sich am Fall der Schachbrettmarkierungen. Es wird zunächst ein Bild eingelesen, eine ROI ausgeschnitten, das Bild reduziert und mit der Funktion `create_shape_model` ein formbasiertes Modell erstellt. Für jede der zwölf verschiedenen Figuren werden mehrere Modelle erstellt, um die Figur besser finden zu können. Wenn sich beispielsweise eine Figur an einer komplett anderen Stelle im Bild befindet, als der Ort, an dem das Modell erstellt wurde, kann es aufgrund von Verzerrungen durch das 2D-Bild passieren, dass die Figur später nicht erkannt wird. Um dabei Abhilfe zu schaffen, wurden für jede Figur 18 Modelle erstellt, die in zwei 3×3 -Gittern über dem Schachbrett angeordnet sind. Die ersten neun Figuren sind richtig herum, wie in Abb. 7.4 am Beispiel eines schwarzen Läufers dargestellt, die restlichen neun Figuren sind an denselben Positionen und um 180° rotiert, um verdrehte Figuren besser erkennen zu können.

Bildanalyse:

Der Abschnitt zur Bildanalyse sowie die nachfolgend beschriebene Generierung der Transformationsmatrizen sind ganz oder teilweise in while-Schleifen verpackt. Das hat den Grund, dass der Abschnitt bei einer falschen Erkennung erneut durchlaufen werden kann, bis das gewünschte Ergebnis erzielt wird. Primär wird dafür gesorgt, dass das Spiel erst starten kann, wenn die Schachanfangsposition erkannt wurde. Es werden aber auch intern weitere Tests durchgeführt, die dafür sorgen, das Programm zu wiederholen. Um dem Nutzer im Ausgabefenster von Python Fehlermeldungen mitteilen zu können, befindet sich das Python-Programm währenddessen ebenfalls in einer while-Schleife, um Nachrichten von HALCON zu empfangen. Python verlässt diese Schleife erst, wenn HALCON die Schachanfangsposition sendet (vgl. Kap. 7.2.3).



Abbildung 7.4 Musterbilder zur Erstellung der ersten neun formbasierten Modelle am Beispiel des schwarzen Läufers

Der Abschnitt beginnt mit der Funktion `get_images_2D_3D`, die vom Autor in HALCON geschrieben wurde. Sie erstellt zunächst ein 2D-Bild mit `grab_image_async` und ein 3D-Bild mit `grab_data`. Daraufhin wird aus dem 3D-Bild ein Objektmodell erstellt und geprüft, ob sich zwischen der Kamera und dem Schachbrett Punkte des Objektmodells befinden. Falls das der Fall ist, werden die Bilder verwendet. Andernfalls werden so lange neue Bilder generiert, bis der Bereich zwischen Kamera und Schachbrett leer ist. Hauptsächlich wird hiermit verhindert, dass während eines Zuges vom Benutzer die Bilder verarbeitet werden, da eine Hand das Brett verdeckt und die Bilder ohnehin unbrauchbar wären.

Anschließend werden mit `find_shape_model` (vgl. 4.2) die im Abschnitt „Modellgenerierung“ erstellten formbasierten Modelle für A, H und 1 auf dem 2D-Bild des Schachbrettes gesucht. Den optimalen *MinScore*-Wert zu wählen, um die Modelle in jedem Bild finden zu können, ohne zusätzliche Instanzen der Modelle zu erhalten, die nicht vorhanden sind, ist sehr schwierig. Durch Testen wurden für A der Wert 0.75 und für H und 1 der Wert 0.9 gewählt, welche gute Ergebnisse erzielen konnten. Daraufhin wird die Anzahl der gefundenen Schachbrettmarkierungen überprüft. Es müssen mindestens eine und maximal zwei Instanzen pro Markierung gefunden werden (vgl. Abb. 5.8). Im Falle einer falschen Erkennung wird mit `Client_Send` eine Nachricht an Python gesendet, welche als Fehlermeldung im Ausgabefenster angezeigt wird.

Um den Winkel des Schachbrettes zu erhalten, welcher später noch gebraucht wird, wird die Rotation für ein gefundenes A verwendet. Danach werden die Koordinaten von A (links unten), H (rechts unten) und 1 (links unten) mit der Funktion `get_coordinates_A_H_1` gespeichert. Die Funktion `get_coordinates_A1` verwendet diese anschließend, um die Koordinaten des Feldes A1 mit einfacher Vektoralgebra zu berechnen. Dazu wird der Schnittpunkt zweier Geraden berechnet. Die erste Gerade hat als Aufpunkt die Koordinaten des Feldes 1 und den Richtungsvektor \overrightarrow{AH} . Die zweite Gerade hat einen Richtungsvektor, der um 90° zu \overrightarrow{AH} rotiert ist und als Aufpunkt die Koordinaten von A.

Daraufhin wird der Abstand zwischen A und H bestimmt, um die Größe eines Feldes zu berechnen, indem der Wert durch sieben geteilt wird. Außerdem wird direkt getestet, ob das korrekte A bzw. H zur Berechnung des Abstandes verwendet worden ist. Falls beispielsweise der Abstand von dem A, welches sich auf dem Brett links unten befindet, und dem H, welches rechts oben zu finden ist, berechnet wird, erfolgt eine Fehlermeldung an Python, da hierdurch ein Feld zu groß angegeben wird.

Im nächsten Schritt wird die ROI des Schachbrettes erstellt. Dafür werden zunächst die vier Sterne an den Ecken des Brettes mit der Funktion `find_shape_model` gesucht ($MinScore = 0.5$, $NumMatches = 4$). Die Funktion `gen_stretched_region_polygon_filled`, die vom Autor in HALCON angelegt wurde, nutzt die HALCON-interne Funktion `gen_region_polygon_filled`, um eine ROI mit den Koordinaten der Sterne als Eckpunkte zu generieren. Diesbezüglich gibt es zwei Probleme. Die ROI ist durch das Verbinden der Sterne nicht groß genug, und ein Teil des Randes wird abgeschnitten. Um das zu verhindern, werden die Koordinaten mit dem sogenannten `stretch_factor` weiter nach außen gestreckt. Das zweite Problem ist die Reihenfolge der Eckpunkte, wenn sie der Funktion übergeben werden. Es kann passieren, dass die Sterne nicht zu einem Quadrat verbunden werden, sondern sich die Form einer Sanduhr ergibt. Deshalb werden die Koordinaten sortiert.

Anschließend wird getestet, ob die korrekten Sterne gefunden worden sind, indem die Innenwinkel der ROI berechnet werden, welche mit einer bestimmten Toleranz einen rechten Winkel ergeben müssen. Sollte das nicht der Fall sein, wird eine Fehlermeldung an Python gesendet.

Im nächsten Schritt wird der Teil der Figurenstellung eines FEN-Strings berechnet, welcher auch als „board configuration“ bezeichnet werden kann. Zunächst werden für jede der zwölf Figuren jeweils 18 verschiedene, formbasierte Modelle im 2D-Bild mit `find_shape_model` gesucht. Die Ergebnisse für `Row`, `Column`, `Angle` und `Score` werden in zweidimensionalen

Vektoren gespeichert, wobei der erste Index die Art der Figur beschreibt und der zweite angibt, welches der 18 Modelle verwendet wurde. Als *MinScore* der einzelnen Figuren werden folgende Werte genutzt, die stetig verbessert werden können, um die Suche fortlaufend zu optimieren: $b = 0.8, B = 0.7, k = 0.8, K = 0.7, n = 0.8, N = 0.7, p = 0.85, P = 0.8, q = 0.7, Q = 0.7, r = 0.9, R = 0.8$ (vgl. Kap. 2.4 für die Bezeichnungen der Figuren)

Um im Anschluss das beste Match (in Bezug auf den höchsten *Score*) der 18 Modelle aller Figuren zu nutzen, wird die Funktion *get_best_scoring_indices* verwendet, die vom Autor erstellt wurde. Die Funktion iteriert zunächst über alle Matches und speichert diejenigen, die sich auf die gleiche Figur beziehen, in einer Liste ab. Innerhalb dieser Listen wird nach dem Match mit dem höchsten *Score* gesucht, dessen Indizes in der sogenannten *winner_list* abgespeichert wird. Eine *winner_list* für die Schachanfangsposition ist in Abb. 7.5 dargestellt. Es sind zwölf Zeilen, die der Art der Figur entsprechen (alphabetisch sortiert, wie im Fall der *MinScore*-Werte (s. o.)). Eine Zeile enthält so viele Einträge wie gefundene Figuren, z. B. zwei schwarze Läufer in der ersten Zeile. Die Einträge bestehen aus jeweils zwei Zahlen, die benötigt werden, um die richtigen Koordinaten in den zweidimensionalen Vektoren der *Row*- und *Column*-Werte auswählen zu können. Der erste Eintrag gibt an, welches der 18 Matches verwendet wird. Da es für jeden Match gegebenenfalls mehrere gefundene Instanzen gibt, gibt der zweite Eintrag an, welche dieser Instanzen den größten *Score* aufweist.

Variableninspektion: winner_list	
	winner_list
0	▶ {[0, 0],[5, 0]}
1	▶ {[6, 0],[6, 1]}
2	▶ {[1, 0]}
3	▶ {[6, 0]}
4	▶ {[1, 0],[2, 0]}
5	▶ {[6, 1],[6, 0]}
6	▶ {[0, 0],[2, 0],[2, 1],[2, 2],[2, 3],...}
7	▶ {[6, 2],[4, 0],[5, 0],[6, 0],[4, 2],...}
8	▶ {[1, 0]}
9	▶ {[7, 0]}
10	▶ {[2, 1],[2, 0]}
11	▶ {[6, 0],[6, 1]}
Typen	
Dimension	

Abbildung 7.5 Beispiel einer *winner_list* der Schachanfangsposition

Anschließend wird die board configuration mit der Funktion `gen_board_config` berechnet. Hierfür wird über jedes Element der Liste `winner_list` iteriert und die Figur auf dem zugehörigen Feld in einem Vektor gespeichert. In Abb. 7.6 ist ein Beispiel zur Berechnung des Feldes einer Figur visualisiert. Es wird zunächst die Differenz zwischen den *Row*-Werten der betrachteten Figur und A1 bestimmt und durch die Größe eines Feldes geteilt. Nach Rundung dieses Wertes auf eine ganze Zahl, erhält man die Anzahl der Felder zwischen der Figur und A1 in vertikaler Richtung. Gleiches Verfahren wird anschließend mit den *Column*-Werten in horizontaler Richtung durchgeführt. Hierbei werden die Abstände in Abhängigkeit der Rotation des Schachbrettes berechnet, um alle möglichen Fälle abzudecken. Das Ergebnis, also der ganzzahlige Abstand der betrachteten Figur zum Feld A1, wird daraufhin in einen 64-dimensionalen Vektor namens `boardConfig` gespeichert. Der Vektor enthält alle Felder in folgender Reihenfolge: A8 – H8, A7 – H7, ..., A2 – H2, A1 – H1. Daraufhin wird der Vektor `boardConfig` in die Figurenstellung eines FEN-Strings umgewandelt und als `FEN_part1` bezeichnet.

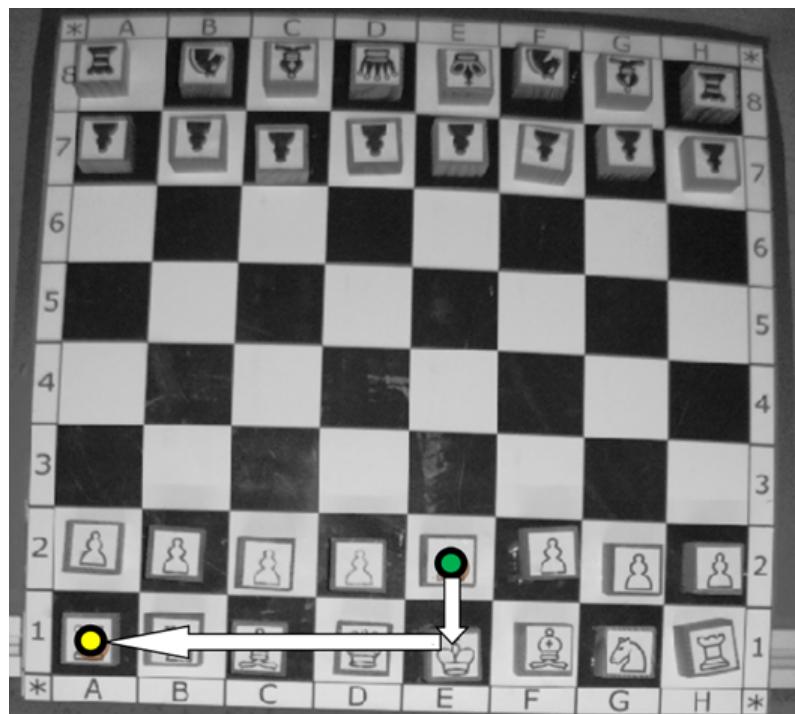


Abbildung 7.6 Visualisierung der Berechnung des Feldes einer Figur am Beispiel des Bauern auf E2

Falls es sich bei `FEN_part1` nicht um die Figurenstellung der Schachanfangsposition handelt, wird eine Fehlermeldung an Python gesendet. Andernfalls startet daraufhin die Berechnung der Transformationsmatrizen.

Transformationsmatrix:

Dieser Abschnitt beschreibt die Berechnung bzw. Beschaffung aller notwendigen Transformationsmatrizen. Vom 2D-Bild des Schachbrettes bis hin zu den Translationskoordinaten der einzelnen Figuren im Roboterkoordinatensystem sind drei Transformationen nötig, welche nachfolgend erläutert werden. Im Anschluss daran wird auf die Berechnung bzw. Beschaffung der Matrizen im Quellcode eingegangen:

- TransformationsMatrix_2D_3D:

Die Aufnahme eines 2D-Bildes mit der Funktion `grab_image_async` sorgt dafür, dass die Stereokamera ein Bild mit einer der beiden Kameras macht (vgl. Abb. 7.7). Im resultierenden Bild sind die Aufkleber auf den Schachfiguren gut erkennbar. Es weist jedoch leichte Verzerrungen an den Rändern auf, weswegen im Abschnitt „Modellgenerierung“ mehrere Modelle für alle Figuren erstellt wurden.



Abbildung 7.7 2D-Bild einer Szene

Die Generierung eines 3D-Bildes über die Funktion `grab_data` liefert, aufgrund der Mustermaske (vgl. Kap. 2.2), für jede der zwei Kameras ein verrausches Bild (vgl. Abb. 7.8). Da mithilfe der projizierten Textur die schwache Oberflächenstruktur ergänzt wird, können die Tiefeninformationen der Szene besser verarbeitet werden. Die Stereokamera nutzt diese Informationen, um den Verzerrungen am Rand des Bildes entgegenzuwirken und das Schachbrett quadratisch darzustellen. Der Nachteil ist, dass die Aufkleber auf den Figuren und weitere Markierungen zu verrauscht sind, um erkannt werden zu können.

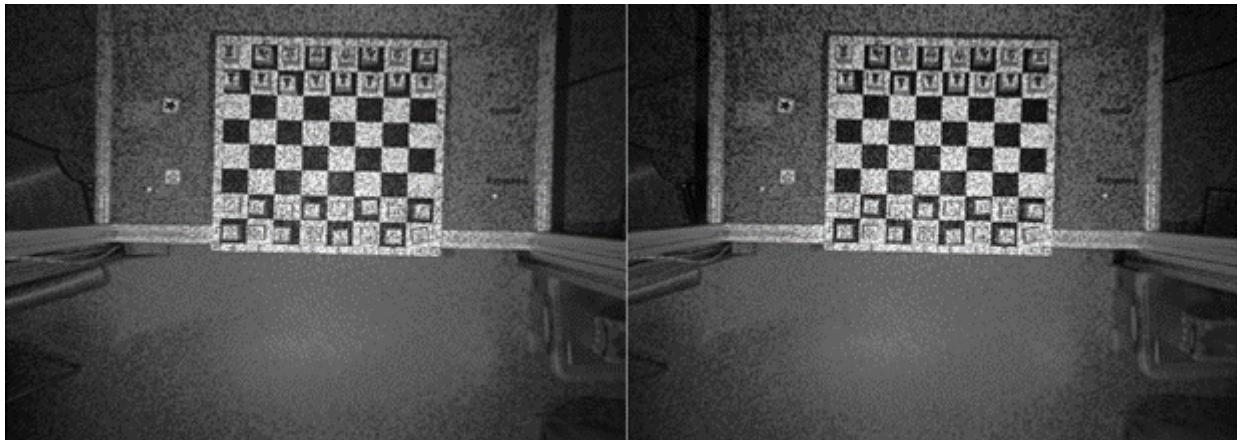


Abbildung 7.8 Bilder einer Szene mit Mustermaske mit der linken bzw. rechten Kamera der Stereokamera

Anschließend wird aus den beiden verrauschten Bildern der Stereokamera mit dem internen Algorithmus ein Überlagerungsbild, bestehend aus drei Einzelbildern, generiert, welches in Abb. 7.9 dargestellt ist.



Abbildung 7.9 Überlagerungsbild bestehend aus x-, y- und z-Koordinaten einer Szene

Die drei Einzelbilder, aus dem sich das Überlagerungsbild zusammensetzt, beinhalten jeweils die x-, y- bzw. z-Koordinaten der aufgenommenen Szene, welche in Abb. 7.10 veranschaulicht werden. Das linke Bild entspricht den x-Koordinaten, da die Helligkeit von links nach rechts zunimmt. Das Bild in der Mitte entspricht den y-Koordinaten,

da es von oben nach unten immer heller wird. Und das dritte Bild repräsentiert die z-Koordinaten, in denen weiter entfernte Objekte heller erscheinen.

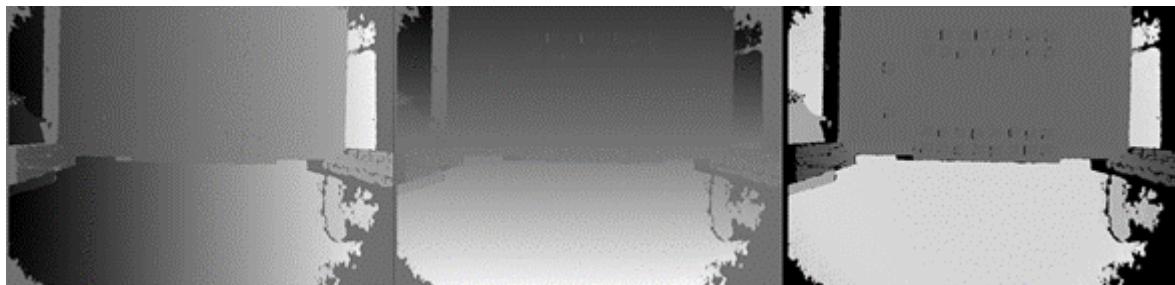


Abbildung 7.10 Einzelbilder der x-, y- und z-Koordinaten einer 3D-Szene

Das Einzelbild der z-Koordinate ist für diese Anwendung am aufschlussreichsten, da die Würfel bzw. Figuren eindeutig vom glatten Untergrund unterschieden werden können. Leider kann aus dem 3D-Bild nicht die Information gezogen werden, um welche Figur es sich genau handelt, da die Bilder der Figuren, aufgrund der Mustermaske, nicht erkennbar sind. Um die beiden Informationen aus 2D- und 3D-Bild aufeinander abzustimmen, wird die sogenannte *TransformationsMatrix_2D_3D* erstellt. Diese Matrix ist in der Lage, die *Row-* und *Column-Koordinaten* einer Figur im 2D-Bild in die *Row-* und *Column-Koordinaten* der zugehörigen Figur im Einzelbild der z-Koordinaten zu transformieren. Dieser Übergang wird in Abb. 7.11 visuell veranschaulicht. Die Matrix ist vom Typ 3×3 , da es sich um eine homogene Matrix handelt, die zweidimensionale Koordinaten wieder in zweidimensionale Koordinaten übersetzt.

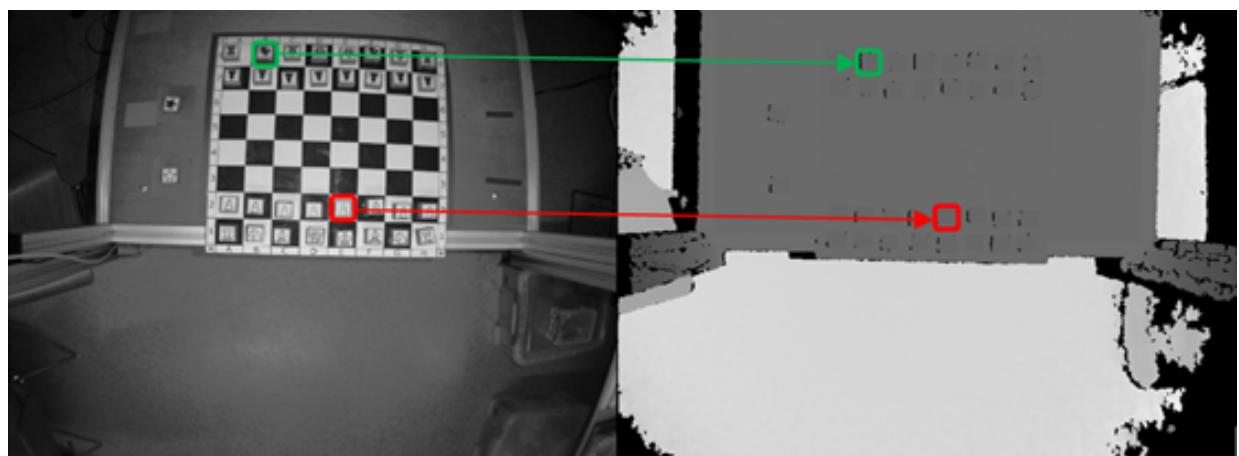


Abbildung 7.11 Veranschaulichung des Koordinatenübergangs mithilfe der *TransformationsMatrix_2D_3D* zwischen 2D-Bild und z-Komponenten des 3D-Bildes

- TransformationsMatrix_coordinate_pose:

Im nächsten Schritt müssen die Koordinaten der z-Komponenten des 3D-Bildes in die zugehörigen Translationskoordinaten der Posen der einzelnen Würfel übersetzt werden. Hierfür ist die homogene 3×3 -Matrix *TransformationsMatrix_coordinate_pose* zuständig. Um daraufhin den dreidimensionalen Aufenthaltsort des Würfels im Kamerakoordinatensystem zu erhalten, muss an die beiden Translationskoordinaten noch die z-Koordinate der Würfel angefügt werden, die sich aus der Entfernung zwischen Kamera und Würfel ergibt.

- TransformationsMatrix_camera_robot:

Im letzten Schritt werden die zuvor generierten Koordinaten im Kamerakoordinatensystem in drei Translationskoordinaten des Weltkoordinatensystems des Roboters transformiert. Diese Koordinaten können vom Roboter angefahren werden, um die Figur zu greifen. Dafür wird eine homogene 4×4 -Matrix verwendet, die im Quellcode als *TransformationsMatrix_camera_robot* bezeichnet wird.

Die Berechnungen der Transformationsmatrizen werden mithilfe der internen HALCON-Funktionen *vector_to_hom_mat2d* im zweidimensionalen Fall und *vector_to_hom_mat3d* im dreidimensionalen Fall durchgeführt. Dabei wird das Optimierungsverfahren, welches in Kap. 2.5 beschrieben wird, angewendet. Aus diesem Grund müssen den Funktionen mehrere Objekte mit Koordinaten bzgl. beider Koordinatensysteme übergeben werden. Es werden zunächst die einzelnen Translationskoordinaten in Vektoren gespeichert und anschließend in den Funktionen verwendet. Wichtig ist, darauf zu achten, dass die Reihenfolge der Koordinaten in den Vektoren korrekt ist. Nachfolgend wird nun der eigentliche Quellcode beschrieben.

Zur Berechnung der Matrix *TransformationsMatrix_2D_3D* werden die *Row-* und *Column-*Koordinaten der Figuren im 2D-Bild verwendet, welche im vorherigen Abschnitt „Bildanalyse“ generiert wurden. Diese müssen lediglich mithilfe der *winner_list* aus den zweidimensionalen *Row-* und *Column-*Vektoren in zwei eindimensionale Vektoren abgespeichert werden. Des Weiteren werden die *Row-* und *Column-*Koordinaten der Würfel in den z-Komponenten des 3D-Bildes benötigt, wofür zunächst eine ROI mit *gen_stretched_region_polygon_filled* erstellt wird. Der Unterschied zur ROI aus dem Abschnitt „Bildanalyse“ ist der größere *stretch_factor*, der dafür sorgt, dass sich trotz der Verschiebung zwischen 2D- und 3D-Bild das vollständige Schachbrett innerhalb der ROI befindet. Mithilfe der Funktion *threshold* können daraufhin alle Würfel mit einem bestimmten Grauwert innerhalb der ROI ausgewählt werden, wie in Abb. 7.12 dargestellt.

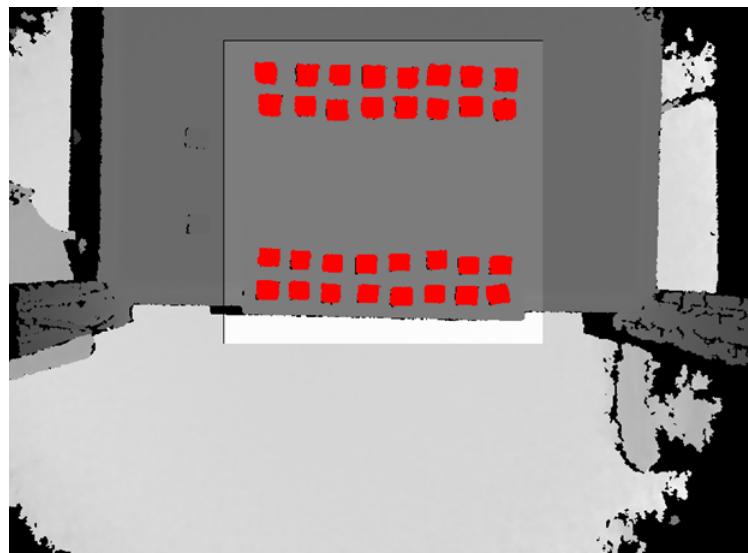


Abbildung 7.12 Bild der z-Komponenten des 3D-Bildes innerhalb einer ROI mit rot markierten Figuren, dessen z-Koordinaten in einem bestimmten Intervall liegen

Anschließend wird die Region der Figuren (in Abb. 7.12 rot dargestellt) mithilfe der Funktion `erosion_rectangle1` an den Rändern verkleinert. Dadurch kann vermieden werden, dass Figuren, die zu nah beieinander liegen, als einzelne große Figur erkannt werden. Zudem wird verhindert, eine Ablage, die das Aus-Feld darstellt, als Figur zu klassifizieren (vgl. Kap. 6.5). Daraufhin kann die gesamte Region in einzelne Teilregionen aufgeteilt werden, welche jeweils eine Figur repräsentieren. Mit der Funktion `area_center` können dann die *Row*- und *Column*-Koordinaten der Mittelpunkte aller Regionen in zwei Vektoren gespeichert werden. Vor dem nächsten Schritt werden die Anzahl der Elemente in den Vektoren miteinander verglichen. Falls sich die Anzahl unterscheidet, also mehr bzw. weniger Figuren im 2D-Bild gefunden werden als im 3D-Bild, wird eine Fehlermeldung an Python gesendet und die Bildanalyse wiederholt. Danach müssen die Vektoren sortiert werden, sodass jeweils der *i*-te Eintrag der Koordinaten im 2D-Bild und der *i*-te Eintrag der Koordinaten im 3D-Bild die gleiche Figur beschreiben. Hierfür wird die mit HALCON realisierte Sortierfunktion `gen_indices_sequence_to_sort_matches` verwendet, die eine Indexfolge für die Vektoren erstellt, mit der sie sortiert werden können. Da die Matrizen stets für den Fall der Schachanfangsposition berechnet werden, sind vier Reihen auf dem Schachbrett mit je acht Figuren gegeben. Der Sortieralgorithmus teilt die 32 Figuren zunächst in vier Reihen auf, indem eine zufällige Figur herausgenommen wird. Diese zufällige Figur bildet jeweils mit einer anderen Figur eine Gerade. Diese Gerade und die x-Achse schließen einen Winkel ein, welcher für alle Figuren in einer Liste abgespeichert wird (vgl. Abb. 7.13).

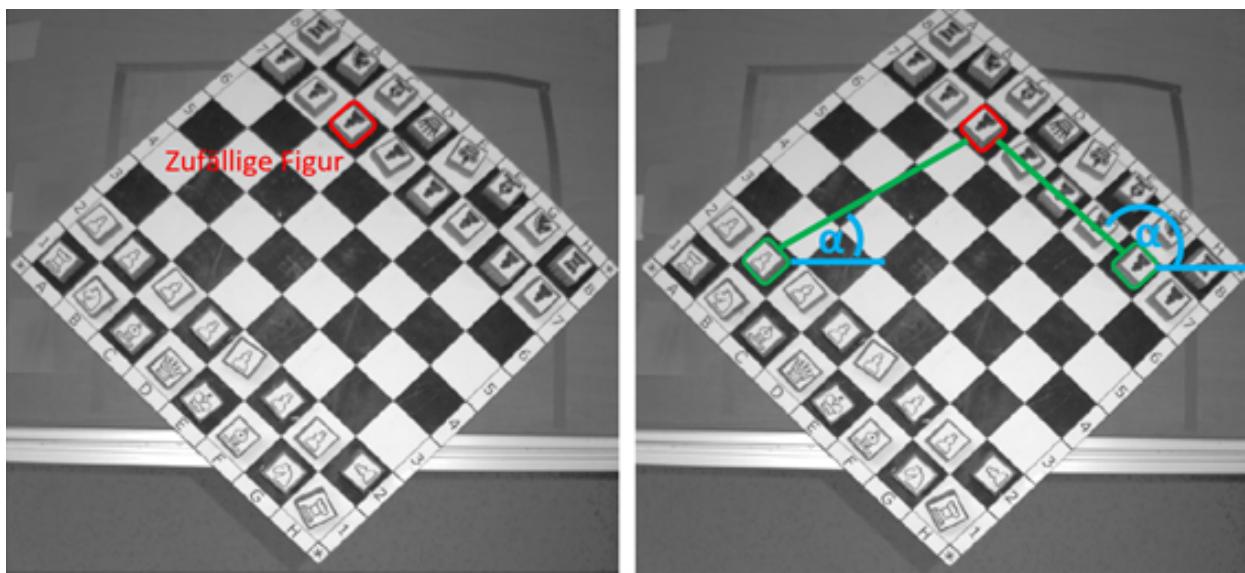


Abbildung 7.13 Visualisierung des ersten Schrittes des Sortieralgorithmus

Um derselben Reihe anzugehören, muss sich der berechnete Winkel um ein Vielfaches von 180° vom Winkel des Schachbretts unterscheiden, der im Abschnitt „Bildanalyse“ abgespeichert wurde. Es werden nun, zusätzlich zur zufällig ausgewählten Figur, die sieben Figuren, deren Winkel diese Voraussetzung erfüllen, in einer Liste gespeichert. Mit den restlichen Figuren wird dieses Verfahren noch dreimal wiederholt, um insgesamt vier Gruppen für die erste, zweite, siebte und achte Reihe zu erhalten.

Anschließend werden die Elemente innerhalb der Gruppen sortiert. In den meisten Fällen werden die *Column*-Koordinaten der Figuren absteigend sortiert. Falls das Brett jedoch um ungefähr 90° oder 270° gedreht ist, werden die Elemente in den Gruppen mithilfe der *Row*-Koordinate aufsteigend sortiert. Die dadurch generierte Indexfolge kann nun die Koordinaten der ursprünglichen Vektoren in folgende Reihenfolge bringen: A8–H8, A7–H7, A2–H2, A1–H1. Falls das Brett zwischen 90° und 270° gedreht wird, ergibt sich die Reihenfolge: H1–A1, H2–A2, H7–A7, H8–A8.

Abschließend wird mithilfe der Funktion `vector_to_hom_mat2d` und den sortierten Vektoren der Koordinaten die Matrix *Transformationsmatrix_2D_3D* generiert.

Zum Verständnis der Berechnung von *Transformationsmatrix_coordinates_pose* muss im Folgenden zunächst die Funktion `get_piece_pose` beschrieben werden, die vom Autor erstellt wurde und für das Verfahren nötig ist.

Mithilfe der Funktion `get_piece_pose` und der Übergabe eines bestimmten Schachfeldes im Stringformat kann die Pose der Figur, die sich auf dem Feld befindet, zurückgegeben werden.

Hierfür wird eine weitere vom Autor erstellte Funktion namens `get_field_coordinates_2D` aufgerufen, die mithilfe der Koordinaten von A1, welche im Abschnitt „Bildanalyse“ gespeichert wurden, die Koordinaten der Mitte des Feldes berechnet. Diese Koordinaten werden anschließend mit der `Transformationsmatrix_2D_3D` transformiert. Daraufhin wird eine quadratische ROI erstellt, die als Mittelpunkt die transformierten Koordinaten verwendet. Die Größe des Quadrats wurde so gewählt, dass es in der Lage ist, einen kompletten Würfel einzuschließen. Schließlich wird mit den durch die ROI reduzierten z-Komponenten des 3D-Bildes zusammen mit den x- und y-Komponenten des 3D-Bildes ein 3D-Objektmodell durch die Funktion `xyz_to_object_model_3d` erstellt. Um alle Punkte des Objektmodells auszuwählen, die sich innerhalb des Arbeitsbereiches befinden, werden mithilfe der Funktion `select_points_object_model_3d` die Punkte abhängig von der z-Koordinate aussortiert. Das entstandene Modell ist in Abb. 7.14 visualisiert.

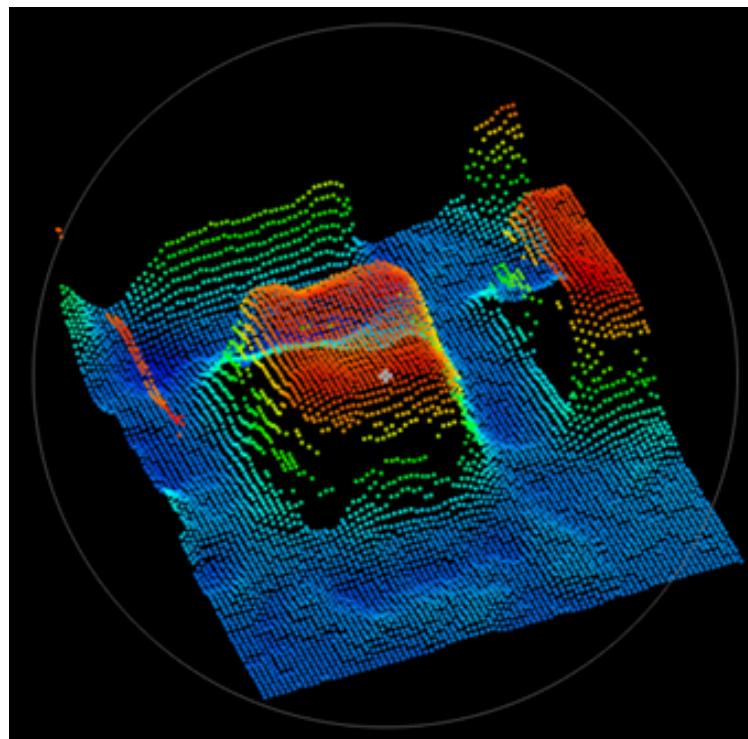


Abbildung 7.14 Durch eine quadratische ROI reduziertes 3D-Objektmodell eines Bereiches auf dem Schachbrett

Um den Würfel im 3D-Objektmodell optimal zu finden wird im nächsten Schritt der Hintergrund mit `threshold` entfernt. Diese Vorgehensweise liefert deutlich bessere Ergebnisse, da die Würfel sonst des Öfteren nicht parallel zum Schachbrett gefunden werden. Das resultierende Modell ist in Abb. 7.15 dargestellt.

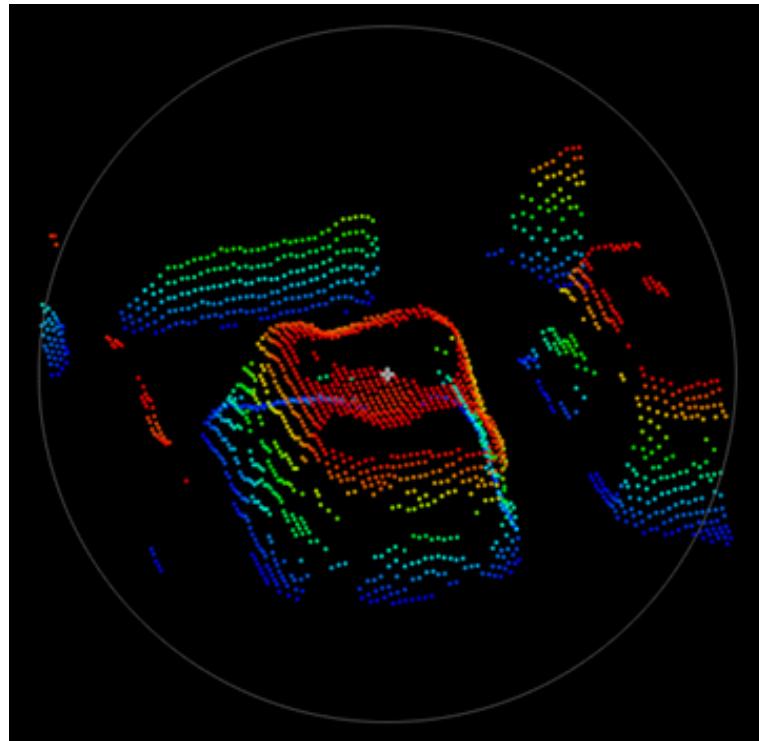


Abbildung 7.15 Durch eine quadratische ROI reduziertes 3D-Objektmodell eines Bereiches auf dem Schachbrett ohne Hintergrund

Im nächsten Schritt kann das Oberflächenmodell des Würfels im 3D-Objektmodell ohne Hintergrund gesucht werden. Hierfür wird die Funktion *find_surface_model* (s. Kap. 4.2) verwendet. Die Parameter der Funktion wurden für die Suche folgendermaßen eingestellt:

- *RelSamplingDistance* = 0.05,
- *KeyPointFraction* = 0.2,
- *num_matches* = 1,
- *use_3d_edges* = *true*,
- *use_view_based* = *true*.

Für den finalen Quellcode ist eine Visualisierung nicht notwendig. Zur Veranschaulichung wird in Abb. 7.16 dennoch gezeigt, wie das Oberflächenmodell des Würfels im 3D-Objektmodell mit Hintergrund dargestellt ist. Es ist darauf zu achten, dass die Suche des Würfels mit *find_surface_model* im 3D-Objektmodell ohne Hintergrund durchgeführt wurde. Das Modell mit Hintergrund wird lediglich für die Visualisierung verwendet.

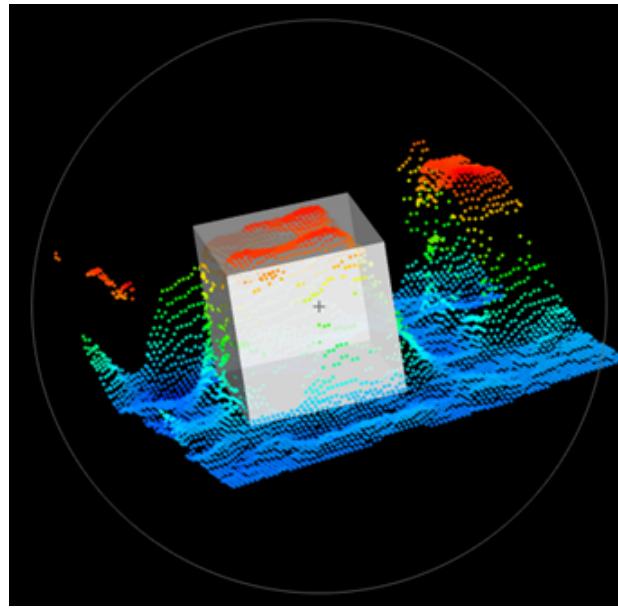


Abbildung 7.16 Gefundene Instanz des Oberflächenmodells eines Würfels visualisiert in dem 3D-Objektmodell einer Szene

Die Funktion `get_piece_pose` liefert zusätzlich zur Pose des gefundenen Oberflächenmodells, aufgrund der Anwendung der `TransformationsMatrix_2D_3D` zu Beginn der Funktion, die *Row*- und *Column*-Koordinaten in den z-Komponenten des 3D-Bildes. Somit werden alle Daten, die zur Berechnung der `TransformationsMatrix_coordinate_pose` nötig sind, von der Funktion bereitgestellt.

Im Quellcode beginnt die Berechnung mit einer doppelten while-Schleife zur Bereitstellung aller wichtigen Schachfelder im Stringformat, wozu die Felder A1–H1, A2–H2, A7–H7 und A8–H8 zählen. Anschließend wird die Funktion `get_piece_pose` auf diese Felder angewendet, womit die Posen der Figuren sowie die dazugehörigen Koordinaten in den z-Komponenten des 3D-Bildes zurückgegeben werden. Die Daten werden daraufhin in Vektoren gespeichert, die zur Berechnung der Transformationsmatrix benötigt werden. Nachdem über alle 32 Felder iteriert wurde, wird die Funktion `vector_to_hom_mat2d` aufgerufen und die Matrix `TransformationsMatrix_coordinate_pose` zurückgegeben.

Die Berechnung der `TransformationsMatrix_camera_robot` wird in einem separaten HALCON-Programm durchgeführt (s. Kap. 7.3.3) und wurde auf dem Computer abgespeichert. Im Quellcode des Hauptprogramms wird über den Operator `read_tuple` die Matrix eingelesen.

7.3.2.2 Kommunikation mit Python

Dieser Abschnitt des Hauptprogramms beginnt mit dem Senden des Strings *FEN_part1* und des Schachbrettwinkels, welcher später im Sunrise-Programm benötigt wird. Anschließend befindet sich das Programm in einer while-Schleife, um verschiedene Anfragen von Python zu verarbeiten. Hierfür wird in jeder Iteration die Funktion *Client_Receive* aufgerufen (vgl. 7.3.1). Abhängig vom empfangenen String wird eines der folgenden vier Programme ausgeführt (vgl. den Anfang des Kapitels 7.3.2):

- „boardConfig“:

In diesem Fall benötigt Python die aktuelle Figurenstellung der FEN. Diesbezüglich wird zunächst mit der Funktion *get_images_2D_3D* ein momentanes Bild des Schachbrettes aufgenommen. Die darauf folgende Vorgehensweise ist exakt die gleiche wie im Fall der Bildanalyse aus Kap. 7.3.2.1, weswegen das Programm nur knapp beschrieben wird. Es werden mithilfe der vier Sterne an den Ecken des Schachbrettes und der Funktion *gen_stretched_region_polygon_filled* eine ROI erstellt, in dem die Figuren im nächsten Schritt mit *find_shape_model* gesucht werden. Mit den Funktionen *get_best_scoring_indices* und *gen_boardConfig* wird daraufhin die Figurenstellung der FEN bzw. die board configuration erstellt. Anschließend wird die neu berechnete board configuration mit der vorherigen verglichen, um zu testen, ob sich etwas verändert hat. Falls keine Veränderung zwischen den Figurenstellungen besteht, wird obige Prozedur, beginnend mit der Aufnahme eines neuen Bildes, wiederholt. Falls sich die Stellungen unterscheiden, wird die neue board configuration an Python gesendet.

- „get_pose“ + Feld:

Für den Fall eines solchen Strings benötigt Python die Pose der Figur auf dem angegebenen Feld, um sie später mit dem Roboter anfahren zu können. Der Ausdruck „Feld“ repräsentiert ein Schachfeld im Stringformat (Bsp.: „get_pose_e2“) oder eine Promotionsfigur (Bsp.: „get_pose_q“). Aus diesem Grund gibt es eine Fallunterscheidung für ein normales Feld und eine Promotionsfigur.

- Normales Feld:

Für ein normales Feld wird die Funktion *get_piece_pose*, die bereits im Abschnitt „Transformationsmatrix“ aus Kap. 7.3.2.1 beschrieben wurde, verwendet, um die Pose der Figur auf dem angegebenen Feld zu erhalten. Danach werden die Translationskoordinaten der Pose mit *TransformationsMatrix_camera_robot* in das Weltkoordinatensystem des Roboters transformiert. Um später den Würfel

optimal greifen zu können, werden nicht nur die Rotationskoordinaten der Pose des Würfels benötigt, sondern ebenfalls der Winkel des gefundenen 2D-Matches der Schachfigur. Die genaue Verwendung der Daten zur Berechnung der Rotation des Greifers wird in Kap. 7.4 erläutert. Zur Bestimmung des 2D-Winkels einer Figur wird die Funktion *get_2D_match_angle_on_field*, die vom Autor erstellt wurde, verwendet. Hiermit wird aus dem aktuellen Bild des Schachbrettes, der Figurenstellung der FEN und dem angegebenen Feld die entsprechende Figur mit *find_shape_model* gesucht. Für die Suche werden alle 18 verschiedenen Matchbilder der zu bestimmenden Figur benötigt und der Winkel zurückgegeben, der für den zugehörigen Match den höchsten *Score* liefert. Der Winkel wird mit *angle_piece* bezeichnet. Abschließend werden die transformierten Translationskoordinaten, die Rotationskoordinaten und der Winkel *angle_piece* in Strings konvertiert, um sie an Python senden zu können. Um nicht mehrere Nachrichten an Python zu senden, werden alle Daten, getrennt mit einem Schrägstrich, zu einem String zusammengefügt, um später von Python zerlegt zu werden. Zusätzlich wird am Ende des Strings noch der Ausdruck „/nopromotion“ angehängt, um zu kennzeichnen, dass es sich nicht um eine Bauernumwandlung handelt. Der vollständige String sieht folgendermaßen aus:

„TransX/TransY/TransZ/RotX/RotY/RotZ/*angle_piece/nopromotion*“.

- Promotionsfigur:

Da die Promotionsfigur des Roboters stets auf einer 3 cm hohen Ablage links vom Schachbrett platziert ist, wird zunächst eine ROI über den gesamten linken Bereich gelegt und mit der Funktion *threshold* die Oberfläche der Figur gefunden. Über den Mittelpunkt der Figur, der mit dem Operator *area_center* gespeichert werden kann, wird anschließend ein Quadrat definiert, dessen Kantenlänge so gewählt wird, dass die Figur vollständig im Inneren des Quadrates liegt. Mithilfe von *reduce_domain* wird das Einzelbild der z-Koordinate auf diesen Bereich reduziert und zusammen mit den Bildern der x- und y-Koordinaten ein 3D-Objektmodell über *xyz_to_object_model_3d* erstellt. Mit der Funktion *find_shape_model* wird in dieser 3D-Szene anschließend das oberflächenbasierte Modell des Würfels gesucht. Hierfür werden die gleichen Parameter, wie in der Funktion *get_piece_pose*, die im Abschnitt „Transformationsmatrix“ aus Kap. 7.3.2.1 beschrieben wird, verwendet. Die zurückgegebene Pose des Würfels muss im nächsten Schritt mit der *TransformationMatrix_camera_robot* in das Weltkoordinatensystem des Roboters transformiert werden. Wie im Fall eines normalen Zuges werden nun alle

Daten in Strings konvertiert und getrennt mit einen Schrägstrich an Python gesendet. Der vollständige String sieht folgendermaßen aus:

„TransX/TransY/TransZ/RotX/RotY/RotZ/*angle_piece*/promotion_piece“.

Der Parameter *angle_piece* wird mit 0 vorbelegt, da die Promotionsfigur parallel zum Tisch ausgerichtet ist. Der Parameter *promotion_piece* gibt an, welche Promotionsfigur verwendet wird (*q*, *r*, *n*, *b*).

- „get_empty_field_pose“ + Feld:

Eine nichttriviale Fragestellung für dieses Projekt ist die Bestimmung der Koordinaten von leeren Feldern, indem ausschließlich die Stereokamera verwendet wird. Im vorherigen Projekt ChessRoberta (vgl. Kap. 3.1) konnte dieses Problem leicht gelöst werden, dadurch dass die Koordinaten der Felder A1 und H8 zu Beginn einer Partie mit dem Roboter eingelesen werden. Zur Steigerung der Benutzerfreundlichkeit wird diese Methode hier vermieden, obwohl sie eine einfache Lösung des Problems darstellt. Stattdessen werden die bereits berechneten Transformationsmatrizen verwendet, um aus einem 2D-Bild die Koordinaten leerer Felder im Weltkoordinatensystem des Roboters zu berechnen.

Der Ausdruck „Feld“ kann in diesem Fall alle 64 Felder eines Schachbrettes sowie das in Kap. 5.2 beschriebene Aus-Feld repräsentieren. Für das Aus-Feld muss zudem unterschieden werden, welche Farbe der Spieler gewählt hat. Das hat den Grund, dass es ebenfalls ein Schachfeld beschreibt, welches sich außerhalb des Schachbrettes befindet. Beispielsweise entspricht das Feld E2 numerisch dem Tupel (4, 1), da es vier Felder in horizontaler und ein Feld in vertikaler Richtung von A1 entfernt ist. Für den Fall, dass der Spieler Weiß wählt, erhält das Aus-Feld das Tupel (10, 4), und falls der Spieler Schwarz wählt das Tupel (-3, 3). In beiden Fällen ist das Feld an der gleichen Stelle im Raum. Im Quellcode wird der String, um das Aus-Feld zu beschreiben, mit „out_w“, wenn der Spieler Weiß wählt, bzw. „out_b“, wenn der Spieler Schwarz wählt, bezeichnet.

Mit der Unterfunktion *get_field_coordinates_2D*, die bereits im Abschnitt „Transformationsmatrix“ in Kap. 7.3.2.1 beschrieben wurde, sowie dem angegebenen Feld im Stringformat werden zunächst die *Row*- und *Column*-Koordinaten des Feldes im 2D-Bild bestimmt. Daraufhin werden die Koordinaten mit *TransformationMatrix_2D_3D* und anschließend mit *TransformationMatrix_coordinate_pose* transformiert, um die Translationskoordinaten der Pose in x- und y-Richtung zu erhalten. Die z-Koordinate ist der Abstand zwischen Kamera und leerem Feld, welcher zu Beginn des Quellco-

des definiert wird, und muss nachträglich zu den Koordinaten hinzugefügt werden. Dann kann der dreidimensionale Vektor mit *TransformationMatrix_camera_robot* in das Weltkoordinatensystem des Roboters transformiert werden. Die resultierenden Koordinaten werden in Strings konvertiert und, getrennt durch einen Schrägstrich, zusammengefügt. Zusätzlich wird an den String für den Fall des Aus-Feldes „/out“ und für die übrigen Fälle „/in“ angehängt, um dem Roboter mitzuteilen, welcher Bewegungsablauf bei den gegebenen Koordinaten verwendet werden soll. Der zusammengesetzte String wird abschließend an Python gesendet und kann folgendermaßen aussehen:

- „TransX/TransY/TransZ/out“,
 - „TransX/TransY/TransZ/in“.
- Terminaler String:
Für den Fall, dass der HALCON-Client den terminalen String empfängt, wird die while-Schleife verlassen.

Nach Beendigung der while-Schleife werden die laufenden Verbindungen getrennt, mit der Funktion *Client_Stop* wird die Socket-Verbindung und mit *close_framegrabber* die Stereokamera geschlossen. Im letzten Schritt werden noch alle Modelle, die im Abschnitt „Modellgenerierung“ in Kap. 7.3.2.1 erstellt wurden, mit der Funktion *clear_shape_model* geschlossen, um den Speicher freizugeben.

7.3.3 Hand-Augen-Kalibrierung (HAK) in HALCON

Für dieses Projekt ist es nötig, die Kamera in ein Robotersystem zu integrieren, um Objekte im Kamerasytem mit dem Roboter greifen zu können, wofür die Kamera zu einem Roboter-Koordinatensystem kalibriert werden muss. Die sogenannte HAK ist eine separate Software, welche einmalig und unabhängig vom Hauptprogramm eine Transformationsmatrix erstellt. Es gibt verschiedene Möglichkeiten, mithilfe von HALCON eine HAK durchzuführen. Für dieses Programm hält der Roboter eine Schachfigur bzw. einen $3 \times 3 \times 3$ cm Würfel mit seinem Greifer, während die Stereokamera Bilddaten aufnimmt. Mit den entsprechenden Posen in mehreren Positionen kann die Transformationsmatrix vom Kamerakoordinatensystem in das Weltkoordinatensystem des Roboters berechnet werden. Um in HALCON auf die Koordinaten des TCP vom Roboter zugreifen zu können, wird die Client-Server-Verbindung verwendet, die in Kap. 7.1 erklärt wird. Somit besteht die HAK aus einem HALCON-Programm und einem Sunrise-Programm, welches in Kap. 7.4.3 beschrieben wird.

Zu Beginn wird wie im Abschnitt „Verbindungsauftbau“ aus Kap. 7.3.2.1 über die Funktion *Client_Connect* eine Verbindung zu Sunrise aufgebaut und die Stereokamera mit dem Operator *open_framegrabber* geöffnet. Daraufhin wird das Oberflächenmodell des Würfels über die PLY-Datei eingelesen (vgl. Abschnitt „Modellgenerierung“ aus Kap. 7.3.2.1).

Anschließend wird mit einer while-Schleife über die Anzahl der verschiedenen Positionen iteriert. Zunächst sendet HALCON mit *Client_Send* den String „requestPose“ an Sunrise, um die Koordinaten des TCP in der aktuellen Position anzufragen. Danach werden mit der Funktion *Client_Receive* die Koordinaten in der Form „x/y/z“ empfangen und in separaten Variablen abgespeichert. Für die Generierung der Pose des Würfels im Kamerakoordinatensystem wird ein 3D-Bild mit *grab_image* eingelesen, welches mithilfe des Operators *decompose3* in die drei 2D-Bilder der x-, y- und z-Koordinaten zerlegt wird. Schließlich wird mit *threshold* und *reduce_domain* das Einzelbild der z-Koordinate reduziert, um den Arbeitsbereich, in dem sich der Würfel aufhalten kann, zu definieren. Im nächsten Schritt wird mit der Funktion *xyz_to_object_model_3d* das 3D-Objektmodell aus den drei Einzelbildern erstellt. In dieser Szene wird das oberflächenbasierte Modell des Würfels mit dem Operator *find_surface_model* gesucht und die Pose gespeichert. Für die Suche wurden die Parameter der Funktion folgendermaßen gewählt, da durch sie gute Ergebnisse erzielt wurden:

- *RelSamplingDistance* = 0.05,
- *KeyPointFraction* = 0.2,
- *num_matches* = 1,
- *use_3d_edges* = true.

Die gefundene Instanz kann in der 3D-Szene mit der Funktion *visualize_object_model_3d* dargestellt werden, um sicherzugehen, dass der Würfel an der korrekten Stelle gefunden wurde. Dafür ist eine euklidische Transformation des Würfels nötig, die mit dem Operator *rigid_trans_object_model_3d* durchgeführt wird. Falls der Würfel an einer falschen Stelle visualisiert wird, muss vor der Ausführung des kommenden Schrittes das Programm manuell an den Anfang der while-Schleife gesetzt und der Roboter in eine anderen Position gefahren werden. Danach kann die Iteration wiederholt werden. Falls der Würfel korrekt gefunden wurde, werden im letzten Schritt der while-Schleife jeweils die x-, y- und z-Koordinaten des Würfels im Kamera- und Roboterkoordinatensystem in sechs verschiedenen Listen gespeichert.

Nach Beendigung der Schleife sendet HALCON mit *Client_Send* den String „end“ an Sunrise, um das Programm zu beenden. Die Transformationsmatrix wird daraufhin mit der

Funktion `vector_to_hom_mat3d` berechnet und mit `write_tuple` in den aktuellen Ordner, in dem die HAK ausgeführt wird, abgespeichert.

Abschließend kann der quadratische Fehler der Transformation berechnet werden, um das Programm gegebenenfalls zu wiederholen. Hierfür wird für alle Positionen die quadratische Differenz der echten und der transformierten Kamerakoordinaten aufsummiert und dessen Durchschnitt berechnet.

7.4 Sunrise

Das Sunrise-Programm ist für die Steuerung des Roboters zuständig, dabei werden die empfangenen Nachrichten in einen Bewegungsablauf für den Roboter übersetzt. Für einen Schachzug gibt es jeweils Paare aus Start- und Ziel-Feldern oder auch Von- und Zu-Felder genannt, die sukzessiv abgearbeitet werden. Die Koordinaten der leeren Felder entsprechen jeweils der Mitte des Feldes in einer Höhe von etwa 1 cm über dem Schachbrett. Der exakte Bewegungsablauf soll stets folgende Form haben, wobei sich der Roboter zwischen den Positionen größtenteils linear bewegt:

1. Startposition mit offenen Greiffingern,
2. 10 cm über das Von-Feld,
3. Von-Feld,
4. Schließen der Greiffinger,
5. 10 cm über das Von-Feld,
6. 10 cm über das Zu-Feld,
7. Zu-Feld (falls es sich nicht um das Aus-Feld handelt),
8. Öffnen der Greiffinger,
9. 10 cm über das Zu-Feld (falls es sich nicht um das Aus-Feld handelt),
10. Ggf. Wiederholung von Schritt 2–9,
11. Startposition.

Für diesen Bewegungsablauf muss gewährleistet sein, dass ein Punkt im Raum, der nicht zwischen Kamera und Schachbrett liegt, im Roboter als Startposition eingelesen wird. Zudem ist der Bewegungsablauf während einer Bauernumwandlung komplizierter und wird im Abschnitt „Bewegungsablauf des Roboters“ in Kap. 7.4.2 beschrieben. Um ein Programm in Sunrise zu implementieren, welches diese Voraussetzungen erfüllt, ist ein Client-Programm zur Kommunikation mit anderen Clients und ein Hauptprogramm zur Ausführung der Bewegungen nötig, welche in den folgenden Unterkapiteln erläutert werden.

7.4.1 Client

Der Sunrise-Client funktioniert analog zur *ClientClass* in Python und verwendet lediglich eine andere Programmiersprache. Im Folgenden werden die Funktionen der Klasse nur knapp erläutert und für eine detailliertere Beschreibung auf das Kap. 7.2.1 verwiesen.

- Die Funktion *start* öffnet eine Socket-Verbindung.
- Mit den Funktionen *setHostname* und *setPort* können die IP-Adresse und die Portnummer angepasst werden.
- Für das Senden von Nachrichten ist die Funktion *sendString* zuständig.
- Die Funktion *hasReceivedString* öffnet die Unterfunktion *receiveString*, um festzustellen, ob ein String empfangen wurde.
- Mit der Funktion *getReceivedString* kann daraufhin auf den empfangenen String zugegriffen werden.
- Die Funktion *stop* schließt die bestehende Socket-Verbindung.

7.4.2 ChessProgram

Die Klasse *ChessProgram* ist das Hauptprogramm des Sunrise-Codes und für die Ausführung aller Bewegungen des Roboters zuständig. Nachfolgend wird die Struktur der Klasse zur besseren Übersichtlichkeit stichpunktartig dargestellt. Anschließend werden die unterstrichenen Abschnitte im Detail erklärt.

Injektionen

Anlegen einer Funktion: `testTerminate`

Anlegen einer Funktion: `getRotation`

Hauptfunktion/Run-Funktion:

Initialisierungen und Definitionen

while(Spiel ist nicht vorbei)

 Warte auf String von Python → Fallunterscheidung:

 1. Terminaler String:

Beendigung des Programms

 2. Koordinaten:

Bewegungsablauf des Roboters

 end while

Injektionen:

Um auf bestimmte Funktionen der Sunrise-Bibliotheken zuzugreifen, die in Kap. 4.4 aufgelistet werden, müssen drei der Module mit dem Operator `@Inject` initialisiert werden. Dies gilt für die Bibliotheken mit den Endungen „LBR“, „iApplicationData“ und „ZimmerR840“.

testTerminate:

Die Funktion `testTerminate` gibt eine boolesche Variable zurück, die angibt, ob das Programm sofort beendet werden soll.

getRotation:

Die Funktion `getRotation` ist dafür zuständig, aus den gegebenen Rotationsdaten die Drehung des Greifers zu berechnen, um die Schachfigur optimal greifen zu können. Die Funktion benötigt die drei Rotationskoordinaten der zu greifenden Figur sowie den 2D-Winkel der Figur namens `angle_piece`, der in Kap. 7.3.2.2 erläutert wird. Die optimale Lösung für den Drehwinkel unter der Voraussetzung, dass der Würfel perfekt gefunden wurde, ist die Rotationskoordinate der gegebenen drei zu verwenden, die die Drehung in der Schachbrett-Ebene beschreibt. Hierbei gibt es zwei Probleme:

- Es ist nicht eindeutig, welche der drei Rotationskoordinaten die Drehung in der Schachbrett-Ebene beschreibt.
- Der Würfel kann an der richtigen Stelle mit einer falschen Verdrehung gefunden werden.

Der folgende Algorithmus ermittelt die korrekte Rotationskoordinate und vergleicht sie mit dem 2D-Winkel der Figur, welcher als *angle2D* bezeichnet wird. Wenn sich die Werte um über 10° unterscheiden, wird *angle2D* für das Greifen des Roboters verwendet, obwohl dieser immer eine Ungenauigkeit aufweist. Wenn die Differenz der Werte kleiner als 10° ist, wird die korrekte Rotationskoordinate genutzt. Dieser Kompromiss sorgt dafür, dass in den meisten Fällen die Rotation des Greifers nahezu perfekt berechnet wird. In den wenigen Ausnahmefällen, in denen der Würfel verdreht gefunden wurde, weicht die Rotation nur leicht vom Realwert ab, was in der Praxis kaum bemerkbar ist.

Unter der Voraussetzung, dass der Würfel perfekt gefunden wird, müssen zwei der Rotationskoordinaten ungefähr einem Vielfachen von 90° entsprechen. Das hat den Grund, dass die Seiten des Würfels, welcher mit einer CAD-Datei in HALCON eingelesen wird, parallel zu den xy-, xz- bzw. yz-Ebenen definiert wurde (vgl. Abschnitt „Modellgenerierung“ in Kap. 7.3.2.1). Daraus folgt, dass es nur eine Drehung in der Schachbrett-Ebene geben kann, da der Würfel auf dem Tisch liegt. Im ersten Schritt des Algorithmus wird also für jede Rotationskoordinate die kleinste Differenz zu einem Vielfachen von 90° ermittelt. Anschließend wird die Koordinate, die den größten Abstand hat, weiterverwendet und mit *rot* bezeichnet. Da es nicht möglich ist zu bestimmen, in welcher Richtung das Koordinatensystem innerhalb des Würfels ausgerichtet ist, ist die Drehrichtung der ermittelten Rotationskoordinate nicht eindeutig. Aus diesem Grund wird die Koordinate und die Inverse der Koordinate, also *rot_inverted* = 360 – *rot*, abgespeichert. Im nächsten Schritt werden die beiden Variablen aufgrund der Würfelsymmetrie um ein Vielfaches von 90° reduziert:

$$\begin{aligned} \text{rot_modulo} &= \text{rot \% } 90 \text{ und} \\ \text{rot_inverted_modulo} &= \text{rot_inverted \% } 90. \end{aligned}$$

Um alle Variablen in der gleichen Einheit zu verwenden, werden sie in das Bogenmaß transformiert. Für den Definitionsbereich des in HALCON generierte 2D-Winkel *angle2D* gilt $D = [-0.05, 2\pi - 0.05]$, weswegen er in den Definitionsbereich $D = [0, 2\pi]$ übersetzt wird. Da der Greifer den Würfel aus zwei verschiedenen Richtungen greifen kann, die um 90° verdreht zueinander sind, wird zusätzlich eine Variable namens *direction* definiert, die mit einem Wert von 0 oder 1 angibt, ob der Greifer um zusätzliche 90° rotiert wird oder nicht. Das Ziel ist es, dass die Greifinger die Figuren stets von links-rechts in Bezug auf das Schachbrett greifen. Hierfür wird eine Formel mit folgenden Voraussetzungen benötigt:

$$\begin{aligned} \text{direction} &= 1 \text{ für } \text{angle2D} \in [0, \pi/2] \cap [\pi, 3\pi/2[\text{ und} \\ \text{direction} &= 0 \text{ für } \text{angle2D} \in [\pi/2, \pi] \cap [3\pi/2, 2\pi[. \end{aligned}$$

Anhand dieser Voraussetzungen wird folgende Formel konstruiert:

$$direction = (\lfloor \frac{angle2D}{(\pi/2)} \rfloor + 1) \% 2.$$

Danach muss *angle2D* analog zu den Rotationskoordinaten mittels

$$angle2D_modulo = angle2D \% (\pi/2)$$

um ein Vielfaches von $\pi/2$ reduziert werden. Im nächsten Schritt wird von den beiden Rotationskoordinaten diejenige verwendet, deren Differenz zu *angle2D_modulo* geringer ist und mit *rot_real* bezeichnet. Daraufhin wird überprüft, ob sich *rot_real* und *angle2D_modulo* um weniger als 10° unterscheiden. In diesem Fall wird *rot_real* beibehalten, andernfalls wird *rot_real* = *angle2D_modulo* gesetzt.

Abschließend gilt für den vollständigen Drehwinkel *rot_complete* des Greifers, den die Funktion *getRotation* zurückgibt:

$$rot_complete = rot_real + direction \cdot (\pi/2).$$

Hauptfunktion/Run-Funktion:

Die Hauptfunktion, auch Run-Funktion genannt, ist das Kernstück des Sunrise-Programms und wird bei der Ausführung des Projektes auf der Steuerung des Roboters abgearbeitet. Zunächst werden einige wichtige Initialisierungen und Definitionen angelegt:

- Der Client wird initialisiert und die IP und Portnummer festgelegt.
- Der Client wird mit der Funktion *start* gestartet.
- Die Geschwindigkeit des Roboters muss definiert werden, sie sollte aus sicherheitstechnischen Gründen nicht zu hoch eingestellt werden.
- Der Abstand der Greiffinger beim Öffnen des Greifers muss definiert werden.
- Mithilfe der Bibliothek iApplicationData können die Koordinaten der Startposition eingelesen und im Programm verwendet werden.
- Der Greifer wird initialisiert und geöffnet.

Anschließend wird die neue Startposition berechnet, welche die Translationskoordinaten der alten Startposition verwendet. Die Rotationskoordinaten sollen so eingestellt werden, dass der Greifer exakt nach unten und parallel zum Tisch ausgerichtet ist:

- AlphaRad = -0.38

Dieser Wert ergibt sich, da der TCP und der Medien-Flansch um ein Bogenmaß von 0.38 verdreht zueinander eingebaut sind. Wenn der Medien-Flansch neu mit dem Roboterarm verbunden wird, muss der Wert angepasst werden.

- BetaRad = 0 und GammaRad = $-\pi$

Mit dieser Einstellung zeigt der Greifer des Roboters exakt nach unten.

Jetzt wird eine PTP-Bewegung zu der neuen Startposition durchgeführt. Danach startet die while-Schleife, die mit den Funktionen *hasReceivedString* und *getReceivedString* in jeder Iteration auf einen String vom Python-Programm wartet. Abhängig vom empfangenen String wird einer der folgenden Programmteile ausgeführt.

Beendigung des Programms:

Falls der empfangene String gleich dem terminalen String ist, wird das Programm sofort beendet, indem die Funktion *testTerminate* aufgerufen wird. In diesem Fall wird auch die Server-Client-Verbindung mit der Funktion *stop* geschlossen.

Bewegungsablauf des Roboters:

Falls der empfangene String nicht dem terminalen String entspricht, wird ein Bewegungsablauf vom Roboter verlangt. Zur Erinnerung hat der empfangene String, im Fall des Schlagens einer Figur, folgende Form:

```
„TransX/TransY/TransZ/RotX/RotY/RotZ/angle2D/nopromotion_
TransX/TransY/TransZ/RotX/RotY/RotZ/angle2D/nopromotion$
TransX/TransY/TransZ/out_TransX/TransY/TransZ/in$anglechessboard“.
```

Im ersten Schritt werden alle wichtigen Informationen in separate Variablen gespeichert. Außerdem muss der Definitionsbereich vom Winkel des Schachbrettes namens *Angle_chessboard* verschoben werden, da eine Drehung um 2π in den meisten Situationen mechanisch nicht möglich ist. Um den Bereich $[0, 2\pi]$ in $[-\pi, \pi]$ zu konvertieren, gilt für den Winkel:

$$Angle_chessboard = Angle_chessboard - 2\pi \text{ für } Angle_chessboard \in]\pi, 2\pi]$$

Anschließend wird mit einer for-Schleife über die Anzahl der Bewegungen iteriert (normaler Zug $\hat{=}$ eine Bewegung, Schlagen einer Figur $\hat{=}$ zwei Bewegungen, Rochade $\hat{=}$ zwei Bewegungen, usw.). In jeder Iteration gibt es ein Von-Feld, bestehend aus Translations- und Rotationskoordinaten, und ein Zu-Feld, welches nur Translationskoordinaten beinhaltet. Mit

den Rotationskoordinaten und dem 2D-Winkel der Figur *angle2D* kann nun der Drehwinkel des Greifers mithilfe der Funktion *getRotation* berechnet werden. Im empfangenen String ist mit „/in“ und „/out“ ebenfalls gekennzeichnet, ob die Figur in das Aus-Feld gesetzt werden soll. Das hat einen Einfluss auf den Bewegungsablauf, da der Roboter die Figur aus einer Höhe von 10 cm in das Aus-Feld fallen lässt (vgl. Anfang des Kapitels 7.4). Die boolesche Variable *is_out* speichert die Information und limitiert die Bewegung, wenn eine Figur in das Aus-Feld gesetzt werden soll.

Für das Greifen der Figur auf dem Von-Feld gibt es zwei verschiedene Bewegungsabläufe. Es muss zwischen normalen Zügen und Bauernumwandlungen unterschieden werden. Alle nachfolgenden Bewegungsabläufe verwenden die LINREL-Bewegung, für die die Differenz zwischen der aktuellen Position und der Zielposition berechnet wird.

- Normaler Zug:

In diesem Fall wird die Position 10 cm über dem Von-Feld angefahren. Die x-, y- und z-Koordinaten berechnen sich über die Differenz der Translationskoordinaten des Von-Feldes und den Translationskoordinaten der Startposition. Die AlphaRad-Koordinate entspricht dem zuvor berechneten Drehwinkel des Greifers namens *rot_complete*.

- Bauernumwandlung:

Im Falle einer Bauernumwandlung wird der Würfel, abhängig von der Promotionsfigur, gedreht, bis die korrekte Figur oben liegt. In Abb. 7.17 ist das Würfelnetz der Promotionsfigur abgebildet.

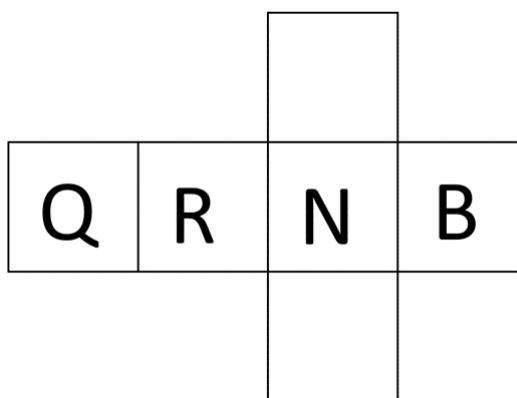


Abbildung 7.17 Würfelnetz der Promotionsfigur

Da die Promotionsfigur mit der Dame nach oben ausgerichtet ist, ergibt sich hieraus folgende Anzahl an Drehungen, die notwendig sind. (Man beachte, dass die Figur um 180° rotiert ist, um aus Sicht des Roboters richtig herum zu liegen):

- Dame $\hat{=}$ 0 Drehungen
- Läufer $\hat{=}$ 1 Drehung nach links
- Springer $\hat{=}$ 2 Drehungen nach links
- Turm $\hat{=}$ 1 Drehung nach rechts

Zunächst fährt der Roboter 10 cm über die Promotionsfigur und rotiert den Greifer um 90° . Danach wird der Greifer so gedreht, dass er, abhängig von der Promotionsfigur, an der linken bzw. rechten oberen Ecke des Würfels ansetzt ($\text{GammaRad} = 45^\circ$) und diesen um 90° in die entgegengesetzte Richtung rotiert. Dazu wird der Würfel 2 cm nach oben angehoben. Diese Prozedur wird im Falle des Springers erneut wiederholt. Daraufhin fährt der Greifer 10 cm über das Von-Feld, sodass der Greifer parallel zum Tisch nach unten ausgerichtet ist und die gewünschte Promotionsfigur oben aufliegt.

In beiden Fällen befindet sich der Greifer mit der korrekten Rotation über dem Von-Feld. Anschließend fährt der Roboter 10 cm nach unten und greift die Figur mit *gripAsync*. Jetzt muss eine Sekunde gewartet werden, um den Roboter die Möglichkeit zu geben, die Figur vollständig zu greifen. Andernfalls schließen sich die Greiffinger während der Roboter bereits in die nächste Position fährt. Nachdem der Würfel im nächsten Schritt 10 cm nach oben bewegt wurde, wird er, parallel zum Schachbrett, 10 cm über das Von-Feld gefahren. An dieser Stelle wird die Figur im Falle des Aus-Feldes abgeworfen und in allen anderen Fällen zunächst 10 cm nach unten gefahren, die Greiffinger geöffnet und wieder 10 cm nach oben gefahren.

Nachdem alle Paare aus Von- und Zu-Feldern abgearbeitet wurden, fährt der Roboter mit einer PTP-Bewegung zurück zur Startposition, damit die Kamera das Schachbrett für den nächsten Zug vollständig erkennen kann. Abschließend sendet Sunrise mit *sendString* den String „done“ an Python, um zu signalisieren, dass der Bewegungsablauf beendet wurde.

7.4.3 Hand-Augen-Kalibrierung (HAK) in Sunrise

Dieses Kapitel beschreibt das HAK-Programm von Sunrise, das für die Berechnung der Transformationsmatrix im HAK-Programm von HALCON benötigt wird (s. Kap. 7.3.3). Zunächst wird eine Instanz der Klasse *Client*, welche in Kap. 7.4.1 beschrieben wird, erstellt und die Verbindung mit HALCON über die Funktion *start* aufgebaut.

Die Kommunikation erfolgt über eine while-Schleife, die mithilfe der beiden Funktionen *hasReceivedString* und *getReceivedString* auf einen String von HALCON wartet. Falls Sunrise

„requestPose“ empfängt, werden die aktuellen x-, y- und z-Koordinaten des TCP mit der Funktion *getCurrentCartesianPosition* gespeichert. Hier muss darauf geachtet werden, den TCP als Frame zu übergeben. Die Koordinaten werden daraufhin als String in der Form „x/y/z“ mit *sendString* an HALCON gesendet. Falls der String „end“ empfangen wird, wird die while-Schleife geschlossen und die Verbindung mit der Funktion *stop* getrennt.

8 Fazit

8.1 Zusammenfassung

Das Ziel der vorliegenden Arbeit ist die Realisierung eines interaktiven Schachroboters mithilfe einer Stereokamera und eines MRK-fähigen Leichtbauroboters. Für diesen Zweck wurde mit einem Client-Server-Modell eine Netzwerkarchitektur aufgebaut, um es den verschiedenen Hardware- und Softwarekomponenten während eines Schachspiels zu ermöglichen, kontinuierlich miteinander zu kommunizieren.

Für die Erkennung der Figuren wurde zunächst eine 3D-Punktwolke der gegebenen Szene mit der 3D-fähigen Kamera extrahiert. Anschließend konnten das Objektmodell der Figuren mit einem speziellen Suchalgorithmus der Software HALCON in der Szene dargestellt und die Koordinaten zurückgegeben werden. Durch die Reduzierung der Punktwolke in der betrachteten Szene haben sich die Ergebnisse des Algorithmus deutlich verbessert. Aus diesem Grund wurden die Aufkleber auf den Figuren zusätzlich in einem 2D-Bild gesucht, um den Bereich einer konkreten Figur im 3D-Bild ausschneiden zu können.

Um im nächsten Schritt in der Lage zu sein, die Figur mit dem Roboter greifen zu können, wurde dessen Pose mit verschiedenen Koordinatentransformationen in das Weltkoordinatensystem des Roboters übersetzt. Hierbei konnte festgestellt werden, dass die Vorgehensweise zur Berechnung der Transformationsmatrizen in der Praxis sehr genaue Ergebnisse liefert. Das wird deutlich, wenn Koordinaten leerer Felder nach dreimaliger Transformation von 2D-Koordinaten, die anhand der Schachbrettmarkierungen berechnet wurden, millimetergenau auf dem realen Brett angefahren werden können.

Mit dieser Vorgehensweise konnte dem Leser eine Möglichkeit präsentiert werden, wie ein sehender und interaktiver Schachroboter realisiert werden kann. Für weitere Fälle von speziellen Pick-and-Place-Anwendungen können die Methoden der vorliegenden Arbeit genutzt und erweitert werden, insbesondere wenn die Form von 3D-Objekten sowie zugehörige 2D-Markierungen auf den spezifischen Objekten gegeben sind.

8.2 Ausblick

Das Projekt liefert zahlreiche Möglichkeiten für Verbesserungen und Erweiterungen, die in diesem Kapitel diskutiert werden.

Zunächst ist es möglich, die Schach-KI ChessAI (vgl. Kap. 3.2), die vor Beginn des Projektes realisiert wurde, in den Quellcode zu integrieren, damit der Benutzer zwischen zwei verschiedenen Spielstilen wählen kann. Diese muss jedoch ausgiebig auf einer soliden Hardware trainiert werden, um einen fähigen Schachspieler zu simulieren.

In der aktuellen Version des Projektes kann eine Partie lediglich in der Schachanfangsposition gestartet werden. Falls das Spiel im schlimmsten Fall abgebrochen werden muss (vgl. Kap. 6.4 und 6.5), ist es daher sinnvoll eine Option in das Projekt zu integrieren, eine Partie aus einer beliebigen Position spielen zu können, um dem Spieler die Möglichkeit zu geben, jedes Spiel zu beenden.

Des Weiteren ist eine Funktion sinnvoll, die die Verdrehung des Brettes während des Spiels ermöglicht. In diesem Fall muss ein Spiel nicht neu gestartet werden, falls das Schachbrett unbeabsichtigt bewegt wird. Allerdings müsste der Algorithmus für jeden Zug alle Markierungen suchen und die Koordinaten des Referenzfelds berechnen, wodurch die Wartezeit während des Roboterzugs erhöht wird.

Eine weitere mögliche Fehlerquelle ist, dass der Roboter die Figuren greift, ohne die Umgebung zu betrachten. Hierbei kann es zu einer Kollision kommen, falls mehrere Figuren zu eng beieinander stehen. Um das zu vermeiden, ist es möglich die umliegenden Felder in die Bewegung des Roboters mit einzubeziehen. Beispielsweise können die Greiffinger um 90° rotiert werden, wenn auf der linken bzw. rechten Seite der Figur nicht ausreichend Platz vorhanden ist. Falls der Roboter den Würfel weder in links-recht-Richtung, noch um 90° verdreht zur links-rechts-Richtung greifen kann, können Alternativstrategien entwickelt werden, um jede Eventualität abzudecken.

Um die Benutzerfreundlichkeit des Projektes weiter zu erhöhen, ist es außerdem möglich, das Aus-Feld zu ersetzen und den Roboter die Schachfiguren nach jeder Partie aufbauen zu lassen. Hierfür kann der Roboter die Würfel blockweise neben dem Brett stapeln, um die Schachanfangsposition nach einer abgeschlossenen Partie wieder aufstellen zu können.

Ein weiterer Nachteil des bestehenden Projektes ist das aufwendige Starten eines Spiels. Ein automatisierter Ablauf der einzelnen Schritte aus Kap. 6.2 und 6.3 würde hier Abhilfe verschaffen und die Bedienung vereinfachen.

Abschließend ist eine Erweiterung der beklebten Holzwürfel auf traditionelle Schachfiguren eine mögliche Verbesserung. In diesem Fall muss jedoch der Großteil des Projektes ange-

passt werden, da die bestehende Architektur auf diesen Komponenten aufgebaut ist. Wie in Kap. 5.2 erwähnt, muss zunächst für jede Figur ein 3D-Objektmodell erstellt werden. Es ist sinnvoll, die Figuren mithilfe der Modelle in einem 3D-Drucker anzufertigen, um die Kompatibilität zwischen Modell und Figur zu erhöhen und damit den Suchalgorithmus zu verbessern. Da für das Projekt lediglich eine Stereokamera über dem Schachbrett verwendet wird, ist eine optimale Erkennung eventuell nicht gewährleistet. Für die Beantwortung dieser Frage ist eine detaillierte Untersuchung erforderlich, und gegebenenfalls muss eine weitere Kamera installiert werden.

Des Weiteren kann die Erkennung der Markierungen auf alle Schachbretter erweitert werden. Hierfür könnte das Speichern der formbasierten Modelle durch eine allgemeine Schrifterkennung ersetzt werden, die die Zahlen und Buchstaben in den vorgesehenen Bereichen sucht. Eine weitere Möglichkeit ist das Verwenden von 3D-Markierungen oder QR-Codes, die beispielsweise an den Ecken eines neuen Schachbrettes befestigt und im Algorithmus gesucht werden.

Literaturverzeichnis

- [1] EUROBOTS: *KUKA LBR IIWA.* <https://www.eurobots.net/gebrauchter-roboter-lbr-iiwa-de.html>. – Aufgerufen am 17.02.2022
- [2] KUKA: *System Software, KUKA Sunrise.OS 1.16, KUKA Sunrise.Workbench 1.16, Bedien- und Programmieranleitung für Systemintegratoren.* – Aufgerufen am 22.02.2022
- [3] IDS: *ENSENSO N35.* <https://en.ids-imaging.com/ensenso-n35.html>. – Aufgerufen am 22.02.2022
- [4] IDS: *ENSENSO 3D OPERATION: HOW DOES PROJECTED TEXTURE STEREO VISION WORK?* <https://en.ids-imaging.com/ensenso-3d-camera-operating.html#ensoflexview>. – Aufgerufen am 22.02.2022
- [5] PRODUCER, Zeta: *Client.* <https://blog.zeta-producer.com/client/>. – Aufgerufen am 28.02.2022
- [6] MVTEC: *2D Transformations.* https://www.mvtec.com/doc/halcon/1805/en/toc_transformations_2dtransformations.html. – Aufgerufen am 04.04.2022
- [7] GMBH, Zimmer: *MONTAGE UND BETRIEBSANLEITUNG.* 2019
- [8] LANDEFELD: *DHAS-GF-60-U-BU (3998967) Adaptiv-Greiffinger.* https://www.landefeld.de/artikel/de/dhas-gf-60-u-bu-3998967-adaptiv-greiffinger/OT-FEST0070172?param_1=%5b%7b%22typ%22:%22fremd%22,%22uebereinstimmung%22:%22unbekannt%22,%22artnr%22:%223998967%22%7d. – Aufgerufen am 17.03.2022
- [9] KUKA: *LBR iiwa.* <https://www.kuka.com/de-de/produkte-leistungen/robotersysteme/industrieroboter/lbr-iiwa>. – Aufgerufen am 17.02.2022
- [10] KUKA: *Mensch-Roboter-Kollaboration.* <https://www.kuka.com/de-de/future-production/mensch-roboter-kollaboration>. – Aufgerufen am 17.02.2022

- [11] KUKA: *KUKA Sunrise.OS*. <https://www.kuka.com/de-de/produkte-leistungen/robotersysteme/software/systemsoftware/sunriseos>. – Aufgerufen am 17.02.2022
- [12] KUKA: *Medien-Flansch - Für Produktfamilie LBR iiwa / Montage- und Betriebsanleitung*. <https://docplayer.org/31077478-Medien-flansch-robot-option-fuer-produktfamilie-lbr-iiwa-montage-und-betriebsanleitung.html>. – Aufgerufen am 17.02.2022
- [13] LUBER, Stefan; Andreas D.: *Was ist das Client-Server-Modell?* <https://www.ip-insider.de/was-ist-das-client-server-modell-a-940627/>. – Aufgerufen am 28.02.2022
- [14] FONIAL: *Client-Server-Modell*. <https://www.fonial.de/wissen/begriff/client-server-modell/>. – Aufgerufen am 28.02.2022
- [15] LEHRERFORTBILDUNG, Baden-Württemberg: *Client-Server-Prinzip*. https://lehrerfortbildung-bw.de/u_matnatech/informatik/gym/bp2016/fb1/3_rechner_netze/1_hintergrund/3_kommunikation/3_client/. – Aufgerufen am 28.02.2022
- [16] MITCHELL, Cory: *IP Address*. <https://www.investopedia.com/terms/i/ip-address.asp>. – Aufgerufen am 28.03.2022
- [17] CLOUD, V2: *Computer Ports Definition*. <https://v2cloud.com/glossary/computer-ports-definition>. – Aufgerufen am 14.03.2022
- [18] CLOUDFLARE: *Was ist ein Computer-Port? / Ports in Netzwerken*. <https://www.cloudflare.com/de-de/learning/network-layer/what-is-a-computer-port/>. – Aufgerufen am 14.03.2022
- [19] ORACLE: *What is a Socket?* <https://docs.oracle.com/javase/tutorial/networking/sockets/definition.html>. – Aufgerufen am 14.03.2022
- [20] CHESSPROGRAMMING.ORG: *Forsyth-Edwards Notation*. https://www.chessprogramming.org/Forsyth-Edwards_Notation. – Aufgerufen am 10.02.2022
- [21] GLOBKE, Wolfgang: *Koordinaten, Transformationen und Roboter*. <https://www.mathkit.edu/iag2/~globke/media/koordinaten.pdf>. – Aufgerufen am 25.02.2022
- [22] VOSEN, Stefan A.: *3D-Transformations*. <http://mapref.org/3D-Transformations.html#Zweig85>. – Aufgerufen am 25.02.2022

- [23] NAUMBURGER, Volkmar: *Homogene Koordinaten*. <https://www.matheretter.de/wiki/homogene-koordinaten>. – Aufgerufen am 25.02.2022
- [24] DANKERT/DANKERT: *Transformationen, homogene Koordinaten*. <http://www.tm-mathe.de/Themen/html/canvastransformationen.html>. – Aufgerufen am 25.02.2022
- [25] MVTEC: *vector_to_hom_mat3d (Operator)*. https://www.mvtec.com/doc/halcon/13/en/vector_to_hom_mat3d.html. – Aufgerufen am 25.02.2022
- [26] SCHISSL, Tobias: *Eclipse IDE*. <https://mission-mobile.de/knowhow/eclipse-ide/>. – Aufgerufen am 17.02.2022
- [27] MARKETPLACE, Eclipse: *PyDev - Python IDE for Eclipse*. <https://marketplace.eclipse.org/content/pydev-python-ide-eclipse>. – Aufgerufen am 17.02.2022
- [28] CHESS python: *python-chess: a chess library for Python*. <https://python-chess.readthedocs.io/en/latest/>. – Aufgerufen am 18.02.2022
- [29] ZHELYABUZHISKY, Ilya: *Stockfish*. <https://pypi.org/project/stockfish/>. – Aufgerufen am 18.02.2022
- [30] W3SCHOOLS: *NumPy Introduction*. [https://www.w3schools.com/python\(numpy_intro.asp](https://www.w3schools.com/python(numpy_intro.asp)). – Aufgerufen am 04.04.2022
- [31] PYTHONTIC: *Socket Module In Python*. <https://pythontic.com/modules/socket/introduction>. – Aufgerufen am 04.04.2022
- [32] MVTEC: *HALCON – THE POWER OF MACHINE VISION*. <https://www.mvtec.com/de/produkte/halcon>. – Aufgerufen am 03.03.2022
- [33] IDS: *HALCON - MEHR ALS SOFTWARE*. <https://de.ids-imaging.com/halcon.html>. – Aufgerufen am 03.03.2022
- [34] SOURCE, The I.: *HALCON - Umfassende Software für Bildverarbeitungstools mit einem weltweit gängigem IDE*. https://www.theimaginingsource.de/produkte/halcon-merlic/halcon/?gclid=Cj0KCQiA64GRBhCZARIIsAHOLriJ0g0cyCFI3l_4CHbvSqgnDY9gCzSJUrpl6dC749690qJLs9T0YqNIaAhHyEALw_wcB. – Aufgerufen am 03.03.2022

- [35] MVTEC: *HALCON - Solution Guide III-C - 3D Vision*. <http://download.mvtec.com/halcon-12.0-solution-guide-iii-c-3d-vision.pdf>. – Aufgerufen am 03.03.2022
- [36] MVTEC: *create_shape_model (Operator)*. https://www.mvtec.com/doc/halcon/12/en/create_shape_model.html. – Aufgerufen am 22.03.2022
- [37] MVTEC: *find_shape_model (Operator)*. https://www.mvtec.com/doc/halcon/12/en/find_shape_model.html. – Aufgerufen am 22.03.2022
- [38] MVTEC: *create_surface_model (Operator)*. https://www.mvtec.com/doc/halcon/12/en/create_surface_model.html. – Aufgerufen am 22.03.2022
- [39] MVTEC: *find_surface_model (Operator)*. https://www.mvtec.com/doc/halcon/12/en/find_surface_model.html. – Aufgerufen am 23.03.2022
- [40] MVTEC: *xyz_to_object_model_3d (Operator)*. https://www.mvtec.com/doc/halcon/13/en/xyz_to_object_model_3d.html. – Aufgerufen am 23.03.2022
- [41] MVTEC: *IMAGE ACQUISITION INTERFACE FOR ENSENSO 3D SENSORS*. <https://www.mvtec.com/de/produkte/schnittstellen/dokumentation/view/53-3d-camera-11-mvtedoc-ensenso-nxlib>. – Aufgerufen am 03.03.2022
- [42] IDS: *ENSENSO SDK -ONE SOFTWARE FOR ALL ENSENSO 3D CAMERAS*. <https://en.ids-imaging.com/ensenso-software-development-kit.html>. – Aufgerufen am 03.03.2022
- [43] KUKA: *KUKA Sunrise.OS 1.7 KUKA / Sunrise.Workbench 1.7 - Bedien- und Programmieranleitung*. <https://docplayer.org/10777331-Kuka-sunrise-workbench-1-7.html>. – Aufgerufen am 18.02.2022
- [44] MVTEC: *open_socket_connect (Operator)*. https://www.mvtec.com/doc/halcon/13/en/open_socket_connect.html. – Aufgerufen am 22.03.2022
- [45] MVTEC: *send_data (Operator)*. https://www.mvtec.com/doc/halcon/13/en/send_data.html. – Aufgerufen am 22.03.2022
- [46] MVTEC: *receive_data (Operator)*. https://www.mvtec.com/doc/halcon/13/en/receive_data.html. – Aufgerufen am 22.03.2022
- [47] MVTEC: *open_framegrabber (Operator)*. https://www.mvtec.com/doc/halcon/11/en/open_framegrabber.html. – Aufgerufen am 22.03.2022

- [48] TURK, Greg: *PLY Files - an ASCII Polygon Format.* <https://people.math.sc.edu/Burkardt/data/ply/ply.html>. – Aufgerufen am 22.03.2022

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Abschlussarbeit selbstständig und nur unter Verwendung der von mir angegebenen Quellen und Hilfsmittel verfasst zu haben. Sowohl inhaltlich als auch wörtlich entnommene Inhalte wurden als solche kenntlich gemacht. Die Arbeit hat in dieser oder vergleichbarer Form noch keinem anderen Prüfungsgremium vorgelegen.

Datum: _____ Unterschrift: _____