



Ostfalia Hochschule für angewandte Wissenschaften
- Hochschule Braunschweig/Wolfenbüttel -

Fakultät für Informatik

Abschlussarbeit

im Studiengang Informatik - Schwerpunkt Computer Engineering

zur Erlangung des akademischen Grades
Bachelor of Science

Thema: Metallentfernung an Dokumenten durch
den Forschungsroboter Baxter

Autor: Timo Schürmann <t.schuermann@ostfalia.de>
MatNr. 70454485

Version vom: 27. Oktober 2020

1. Betreuerin: Prof. Dr.-Ing. Reinhart Gerndt
2. Betreuer: B.Sc. Lars Rüdiger Kelm

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere, dass ich alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet habe, und dass die eingereichte Arbeit weder vollständig noch in wesentlichen Teilen Gegenstand eines anderen Prüfungsverfahrens gewesen ist.

Unterschrift :

Ort, Datum :

Inhaltsverzeichnis

Eidesstattliche Erklärung	I
Abbildungsverzeichnis	III
Tabellenverzeichnis	IV
Listingverzeichnis	V
Glossar	VI
1 Grundlagen	1
1.1 Problembeschreibung	1
1.1.1 Greifkraft	1
1.1.2 Positioniergenauigkeit	1
1.1.3 Statischer Ansatz	2
1.2 Baxter	3
1.3 Robot Operating System (ROS)	7
1.4 Computer Vision	8
2 Positionierung	10
2.1 Architektur	11
2.2 Finden eines systematischen Fehlers	12
2.3 Präzisionserhöhung	23
2.4 Ausgleichen des systematischen Fehlers durch LUT	26
2.4.1 Die Datenstruktur LUT	26
2.4.2 Das LUT Interface	27
2.5 Auswertung	28
3 Heftklammererkennung	32
3.1 Versuchsaufbau	32
3.2 Architektur	34
3.3 Die Klasse <code>cam_class</code>	35
3.3.1 Anzeigen der Kamerabilder	35
3.3.2 Der Aktionspunkt	35
3.3.3 Kameragestützte Steuerung des Roboterarms	38
3.4 Heftklammererkennung	38
3.4.1 Konturenfunktionen von OpenCV	39
3.4.2 Das <code>detector</code> -Interface	43
3.5 Nutzerschnittstelle	45
3.6 Auswertung	50
4 Greifkrafterhöhung	54
5 Fazit und Ausblick	56
Literaturverzeichnis	58
Anhang	61
A - <code>arm_class.py</code>	61

B - cam_class.py	77
C - const_lib.py	81
D - measure_precision_along_axes.py	82
E - measure_step_speed.py	91
F - measure_precision_workspace.py	95
G - lut_interface.py	100
H - test_if_precision_improved.py	110
I - detector.py	114
J - point_on_paper.py	123
K - control_interface.py	126
L - Diagramme der Abschlussmessungen	134

Abbildungsverzeichnis

1	Baxter	3
2	Effektiver Arbeitsbereich und Koordinatenachsen[27]	4
3	Nahaufnahme eines Greifers[24]	5
4	Baxter SDK[26]	6
5	In diesem Kapitel verwendete Komponenten des Pakets <code>baxter_staples</code>	11
6	Messung entlang der Koordinatenachse: Positiv entlang X-Achse	14
7	Messung entlang der Koordinatenachse: Negativ entlang X-Achse	16
8	Messung entlang der Koordinatenachse: Positiv entlang Y-Achse	18
9	Messung entlang der Koordinatenachse: Negativ entlang Y-Achse	20
10	Ergebnisdaten der Präzisionsmessungen	24
11	LUT mit 50 Durchläufen in 6 Stadien	28
12	Erhöhung der Positioniergenauigkeit: Gesamtergebnisse	30
13	Der Stifthalter	32
14	Der Versuchstischs	33
15	Hier verwendete Komponenten des Pakets <code>baxter_staples</code>	34
16	Durch <code>cam_class</code> angezeigte Bilder	35
17	Aktionspunktkoordinaten in Abhängigkeit von der Z-Koordinate des Greifers	37
18	Nahaufnahme einer applizierten Heftklammer	38
19	Überprüfen eines Punktes auf lokales Maximum nach Sobel[16]	40
20	Beispielbild für Grenzfilterung potenzieller Kanten[16]	40
21	Heftklammer vor und nach Anwendung des Canny Algorithmus	41
22	Im Beispielbild gefundene Konturen	42
23	Beispielhafte Vergleichsmasken	43
24	Dokument auf Arbeitsplatte vor und nach <code>detect_paper</code>	45
25	Momentaufnahme aus der Übersichtspose	46
26	Momentaufnahme mit eingezeichneten potenziellen Heftklammern	47
27	Nahaufnahme der potenziellen Heftklammer	48
28	Finden und Markieren der Heftklammer	49
29	Markieren gefundener Heftklammern	50
30	Orientierung der Dokumentenmaske	51
31	Gefundene Heftklammern unterschiedlicher Orientierung und Position .	52
32	Leitz Entklammerer[8]	54
33	Erhöhung der Positioniergenauigkeit: Pose 1	135
34	Erhöhung der Positioniergenauigkeit: Pose 2	136
35	Erhöhung der Positioniergenauigkeit: Pose 3	137
36	Erhöhung der Positioniergenauigkeit: Pose 4	138
37	Erhöhung der Positioniergenauigkeit: Pose 5	139

Tabellenverzeichnis

1	Messung entlang der Koordinatenachse: Positiv entlang X-Achse	14
2	Messung entlang der Koordinatenachse: Negativ entlang X-Achse	16
3	Messung entlang der Koordinatenachse: Positiv entlang Y-Achse	18
4	Messung entlang der Koordinatenachse: Negativ entlang Y-Achse	20
5	Konfigurationen für Präzisionsmessungen	23
6	Ergebnisdaten der Präzisionsmessungen	24
7	Erhöhung der Positioniergenauigkeit: Gesamtergebnisse	29

Listingverzeichnis

1	LUT Datenstruktur	26
2	arm_class.py	61
3	cam_class.py	78
4	const_lib.py	82
5	measure_precision_along_axes.py	83
6	measure_step_speed.py	92
7	measure_precision_workspace.py	96
8	lut_interface.py	101
9	test_if_precision_improved.py	111
10	detector.py	115
11	point_on_paper.py	124
12	control_interface.py	127

Glossar

Aktionspunkt Der Punkt an dem das verwendete Werkzeug voraussichtlich auf den Tisch beziehungsweise das darauf liegende Dokument treffen wird. 35

Bereich von Interesse (ROI) Abkürzung aus dem Englischen: "Region Of Interest". Der Bereich eines Bildes, der von besonderem Interesse ist.[3] 35

Dictionary Datentyp, der aus Schlüssel-Wert-Kombinationen besteht. Jedem Schlüssel wird ein Wert zugeordnet. Dabei müssen Schlüssel von unveränderlichem Datentyp sein, während der Datentyp des Wertes frei wählbar ist.[23] 26

Kollaborativer Roboter (Cobot) Cobots benötigen keine oder nur minimale Schutzeinrichtungen, um im direkten Umfeld von Menschen eingesetzt werden zu können. Durch die integrierte Sensorik erfasst der Roboter Kollisionen und reagiert durch Anhalten oder ein Verringern der Geschwindigkeit. Dieses Verhalten soll Verletzungen am Menschen verhindern.[7] 3

Umsetzungstabelle (LUT) Wertetabelle, die zwei Größen in eine Beziehung bringt, die keiner mathematischen Reihe folgen muss. Dabei wird jeder Eingangsgröße eine individuelle Ausgangsgröße zugeordnet.[2] 22, 26

1 Grundlagen

Im Vorfeld zu dieser Arbeit wurde ein Praxisprojekt absolviert, dessen Aufgabenstellung die Entfernung von Heftklammern an mehrseitigen Dokumenten durch den Forschungsroboter Baxter war. Der Hintergrund für das Praxisprojekt und die vorliegende Arbeit liegt beim niedersächsischen Landesarchiv. Um die dort gelagerten Dokumente vor Korrosionsschäden zu bewahren und eine ordnungsgemäße Digitalisierung zu ermöglichen, müssen aus vielen Dokumenten Heftklammern entfernt werden. Dies ist eine monotone und simple Aufgabe und somit gut automatisierbar.

Neben diversen grundlegenden Versuchen zur mechanischen Entfernung von Heftklammern, wurde hierbei das Interface `arm_class` geschaffen, welches die Steuerung des Roboters in Bezug auf verschiedene Aspekte dieser Aufgabenstellung ermöglicht. Das Interface ist in Anhang A - `arm_class.py` zu finden. Im Zuge dieses Praxisprojekts wurden verschiedene Probleme sichtbar, die in Unterkapitel 1.1 aufgeführt und näher erläutert werden. Die folgende Arbeit beschäftigt sich mit verschiedenen Ansätzen zur Lösung dieser Problemstellungen.

1.1 Problembeschreibung

1.1.1 Greifkraft

Es stellte sich heraus, dass Baxter mit dem verbauten elektrischen Greifer keine ausreichende Greifkraft aufbringen konnte, um mit dem gewählten Werkzeug, einem "Leitz Entklammerer", erfolgreich eine Heftklammer aus einem Dokument zu entfernen oder überhaupt zu verformen. Da es sich bei Baxter um einen Cobot handelt, ist auch die von seinen Greifern maximal ausübbare Kraft limitiert. Diese Limitierung besteht, um bei möglichen Kollisionen keine Mitarbeiter zu verletzen. In den von Rethink Robotics zur Verfügung gestellten Interfaces besteht keine Option zur Umgehung oder Anpassung dieser Limitierung. Mögliche Lösungsansätze für dieses Problem werden in Kapitel 4 aufgeführt.

1.1.2 Positioniergenauigkeit

Mit dem gewählten Werkzeug zur Heftklammerentfernung, einem "Leitz Entklammerer", besteht bei der Positionierung des Werkzeugs an der Heftklammer eine Toleranz von einem Millimeter in Längsrichtung der Klammer. Bei ersten Messungen hat sich jedoch ergeben, dass der Arm im Schnitt um 1,4 Millimeter je Richtung und in Einzelfällen bis zu 4 Millimeter abweicht. Damit überschreitet er die genannte Toleranz von 1 Millimeter, um ein zuverlässiges Entfernen einer Heftklammer zu gewährleisten. Rethink Robotics selber gibt für die Positioniergenauigkeit eine Toleranz von +/- 5 Millimetern an. Diese Toleranz begründet sich auf der Genauigkeit der Gelenke, welche eine maximale Ungenauigkeit von 0,5 Grad aufweisen.[28]

Auf die Ursachen und verfolgten Lösungsansätze für dieses Problem wird in Kapitel 2 eingegangen.

1.1.3 Statischer Ansatz

Im Praxisprojekt wurde ein eher statischer Ansatz verfolgt, der eine gleichbleibende Positionierung und Orientierung der Interaktionsobjekte voraussetzte, da die in den Roboter integrierte Sensorik nicht genutzt wurde. Durch den verfolgten Ansatz konnte eine Grundlage geschaffen und mehrere grundlegende Problemstellungen aufgedeckt werden. Die, durch Restriktionen wie eine gleichbleibende Positionierung und Orientierung der Heftklammer, geschaffene Arbeitsumgebung des Roboters entspricht jedoch keinesfalls realen Gegebenheiten. Es existieren Drucker, die die gefertigten Dokumente auch automatisch heften können. Bei diesen werden die genannten Restriktionen erfüllt. Häufig ist das Heften von Dokumenten jedoch eine händische Tätigkeit, wodurch Positionierung und Orientierung der gesetzten Heftklammern bei jedem Dokument unterschiedlich sind.

Durch die Nutzung der im Roboter integrierten Kameras sollen die genannten Restriktionen aufgehoben werden. Der hierfür verfolgte Ansatz wird in Kapitel 3 erläutert.

1.2 Baxter



Abbildung 1: Der Roboter Baxter [9]

Baxter ist ein Kollaborativer Roboter (Cobot), der von 2012 bis 2018 von Rethink Robotics produziert und vertrieben wurde.[30] Er ist ausgestattet mit integrierten Kameras, Sonar-Sensorik, Drehmomentsensoren in jedem Gelenk und zwei Armen, mit jeweils sieben Freiheitsgraden. Die Arme werden durch Seriell-Elastische Aktoren (SEA) angetrieben.[5] SEAs bestehen aus einem Motor, der, über ein Getriebe die Spannung einer Feder beeinflusst, an deren anderen Ende die Last hängt.[6, S. 399] Durch diesen Aufbau bieten SEAs diverse Vorteile für natürliche, nicht rein maschinelle, Umgebungen. Diese Vorteile liegen, unter anderen, in der Stoßdämpfung, der genauen und stabilen Kraftkontrolle und der geringeren Schäden an seiner Umwelt bei Kollision.[6, S. 405] Die Drehmomentsensoren erfassen die, in den SEAs auftretenden, Momente und nutzen diese Informationen, unter anderem, zur Kollisionserkennung.[24]

Nachdem er zunächst als Roboter für die Produktion vertrieben wurde, war Baxter seit April 2013 auch als Forschungsmodell erhältlich. Während die Fabrikversion eine intuitive Nutzerschnittstelle bietet, die Bewegungen lernt, indem man sie an seinen Armen vormacht, wurde das Forschungsmodell mit einem Software Development Kit (SDK) und einem ROS-Interface ausgestattet.[29, S. 106] Durch das SDK und das Interface ist es Entwicklern möglich, eigene Programme zu schreiben, deren Befehle durch das Interface über ROS (Robot Operating System) an Baxter übermittelt werden, sodass dieser sie ausführt.[26]

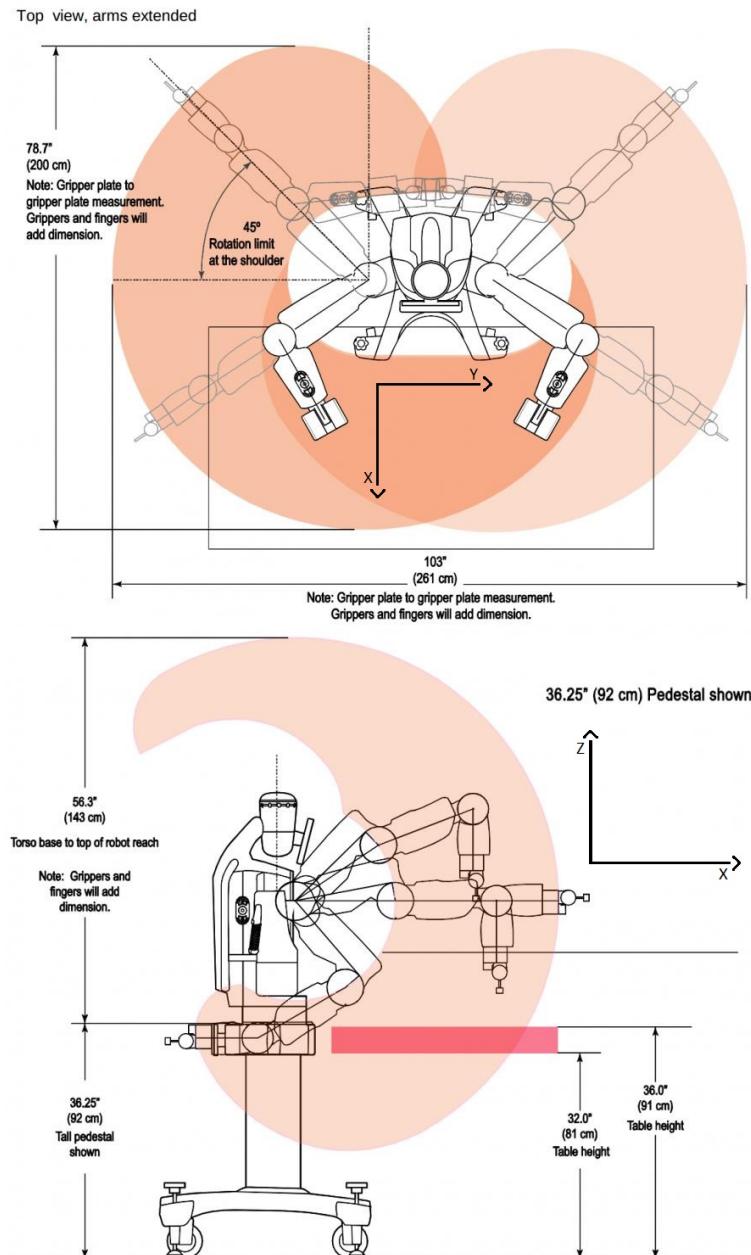


Abbildung 2: Effektiver Arbeitsbereich und Koordinatenachsen[27]

Abbildung 2 zeigt den von Baxters Armen erreichbaren Raum. Zusätzlich wurden Koordinatenachsen in die Bilder eingefügt, auf die im weiteren Verlauf dieser Arbeit mehrfach Bezug genommen wird. Anhand der Farbintensität des oberen Bildes lässt sich erkennen, dass direkt vor Baxter ein Bereich existiert, der von beiden Armen erreicht werden kann. Alle, im Zuge dieser Arbeit durchgeführten, Messungen und Versuche wurden innerhalb dieses Bereichs durchgeführt. Die Arme verlassen den gemeinsamen Arbeitsbereich, wenn sie eine neutrale Pose einnehmen, um Kollisionen untereinander zu verhindern.

Die Roboterarme können mit auswechselbaren Endeffektoren ausgestattet werden, wobei zwischen elektrischen und pneumatischen Greifern gewählt werden kann.[29, S. 106] Für die vorliegende Arbeit wurde ein elektrischer Greifer am linken, und ein pneuma-

tischer Greifer am rechten Arm verwendet.



Abbildung 3: Nahaufnahme eines Greifers[24]

Abbildung 3 zeigt einen elektrischen Greifer in montiertem Zustand. Oberhalb des Greifers sind die, in der Greifergrundplatte verbauten, optischen Sensoren zu sehen. Diese bestehen aus einer Kamera und einem Infrarotsensor. Weiterhin enthält dieses Segment des Arms einen Beschleunigungssensor.[24]

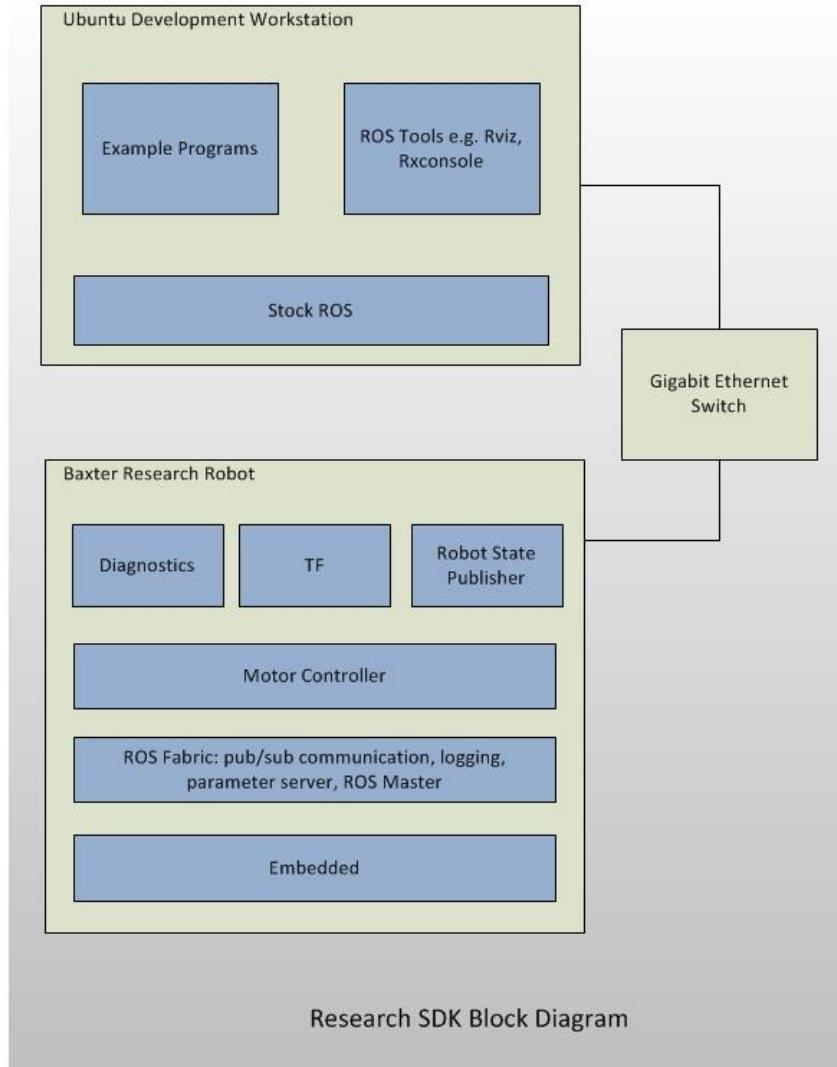


Abbildung 4: Baxter SDK[26]

Abbildung 4 zeigt das, von Rethink Robotics zur Verfügung gestellte, SDK. Auf dem Roboter läuft eine eigene ROS-Instanz. Durch ein Ethernetkabel wird ein lokales Netzwerk zwischen einem Computer und dem Roboter aufgebaut. Jegliche Kommunikation zwischen dem Roboter und dem Computer finden über ROS-Topics statt, wobei diese zum Großteil vom bereitgestellten Interface verwaltet werden. Das Empfangen mancher, vom Roboter gesendeter, Informationen wird jedoch nicht vom Interface übernommen, sodass hierfür eigene Empfänger für die entsprechenden Topics eingerichtet werden müssen. Ein Beispiel hierfür sind die von den Kameras des Roboters aufgenommenen Bilder. Entwickler haben somit insgesamt nur wenige Berührungspunkte mit ROS.

1.3 Robot Operating System (ROS)

Während der Name des Robot Operating System (ROS) vermuten lässt, dass es sich dabei um ein Betriebssystem für Roboter handelt, ist dies nicht der Fall. Tatsächlich ist ROS ein Framework, bestehend aus verschiedenen Tools, Bibliotheken und Konventionen. Durch diese bereitgestellten Komponenten soll die Entwicklung komplexer und robuster Robotersteuerungsalgorithmen vereinfacht werden. Das Ziel hinter ROS liegt darin, eine einheitliche Grundlage zu schaffen, auf der kollaborative Softwareentwicklung, zwischen verschiedenen Institutionen, stattfinden kann.[11]

Die erste Kernversion wurde 2007 von Willow Garage unter der BSD Open-Source Lizenz veröffentlicht. Seitdem wurde ROS dezentral durch verschiedene Institutionen und Einzelpersonen weiterentwickelt. Jede Erweiterung durch ein ROS-Package verbleibt dabei auf dem Server des Entwicklers, sodass er volle Kontrolle darüber behält. Wenn vom Entwickler gewollt, kann ein ROS-Package bei Open Robotics registriert und damit veröffentlicht werden, sodass auch andere Entwickler Zugriff darauf erhalten.[13]

ROS besteht im Kern aus drei Hauptkomponenten.

1. Kommunikationsinfrastruktur

Diese stellt die grundlegenste Komponente dar, da jegliche Interprozesskommunikation hierüber geregelt wird. Die Nachrichtenübermittlung zwischen Prozessen findet auf zwei Arten statt.[12]

Die erste ist das Senden und Empfangen über Publisher und Subscriber. Dabei werden die Nachrichten, vom Publisher, unter einer ROS-Topic veröffentlicht. Ein Subscriber kann diese ROS-Topic abonnieren, um so alle darüber veröffentlichten Informationen zu erhalten. Diese Form der Kommunikation ist anonym und asynchron, da die Publisher die Empfänger nicht kennen und die Empfänger nicht wissen, wer unter dieser Topic veröffentlicht.[12]

Die zweite Art findet über Services statt. Diese folgen dem Prinzip von Anfrage und Antwort. Ein Prozess kann also eine Anfrage veröffentlichen, die an einen anderen Prozess gerichtet ist. Dieser empfängt die Anfrage und übermittelt die gewünschten Informationen auf dem gleichen Weg.[12]

Beide Kommunikationsarten nutzen dabei das gleiche Nachrichtenformat, sodass ein einheitlicher Nachrichtenstandard in ROS verwendet wird.[12]

Eine weitere Kommunikationsmöglichkeit wird durch den Parameterserver dargestellt. Dieser besteht aus einem Dictionary, das von allen Prozessen innerhalb eines Netzwerks genutzt werden kann. Es ist dazu gedacht, Konfigurationsparameter, die für mehrere Prozesse eines Netzwerks relevant sind, netzwerkweit verfügbar zu machen.[12]

2. Roboterbibliotheken und -tools

Neben der Kommunikation stellt ROS eine Reihe an Bibliotheken und Tools

zur Verfügung, die die Arbeit an Robotern vereinfachen sollen. Hierzu gehören Nachrichtenstandards, die quasi universell in der Robotik verwendet werden, wie Posen, Transformationen oder Sensorinformationen. Durch Nutzung dieser Standards können alle ROS Komponenten, Tools und Packages problemlos zusammenwirken.[12]

Weiterhin stellt ROS eine Bibliothek zur Verfügung, die das Speichern der Robotergeometrie vereinfacht. Diese beschreibt, wo welche Hardware montiert ist, und in welcher Ausrichtung.[12]

Um dem Computer den Aufbau des verwendeten Roboters verständlich zu machen, nutzt ROS das URDF (Unified Robot Description Format). Dieses liegt als XML-Dokument vor, in welches physikalische Parameter des Roboters eingetragen werden, wie die Zusammensetzung, Länge und Form von Gliedmaßen.[12]

3. Das grafische Toolset

ROS stellt mit Rviz ein Tool zur Verfügung, über das die Steuerung eines Roboters rein über eine grafische Oberfläche möglich ist, sowie mit Rqt ein Tool, welches die Entwicklung eigener grafischer Oberflächen für Roboter ermöglicht.[12]

1.4 Computer Vision

Computer Vision bezeichnet das Analysieren und Interpretieren von Bildern, um daraus, für die Maschine verständliche, Informationen zu gewinnen.[4] In der vorliegenden Arbeit wird OpenCV-Python verwendet, um Aspekte der Computer Vision umzusetzen. Dabei wird sich auf die Version OpenCV 2 bezogen.

OpenCV-Python ist eine Wrapper-Bibliothek, für die C++ API (Application Programming Interface, dt. Programmierschnittstelle). Der Aufruf einer OpenCV-Funktion unter Python wird also weitergeleitet, sodass im Hintergrund C++-Funktionen die eigentlichen Berechnungen ausführen. So können die kurzen Rechenzeiten von C++ mit der einfachen Handhabung von Python kombiniert werden.[20]

OpenCV wurde unter Intel entwickelt und im Jahr 2000 erstmalig veröffentlicht.[20] Sie steht unter der BSD Open-Source Lizenz und wurde entwickelt, um eine einheitliche Grundlage für Computer Vision Anwendungen zu bieten. Inzwischen wird sie von einer Vielzahl an Firmen, Forschungsgruppen und Regierungen, zu unterschiedlichen Zwecken, eingesetzt. Dabei reichen die Einsatzszenarien vom Zusammenfügen einzelner Bildteile, bis zur Gesichtserkennung in Menschenmengen.[14]

OpenCV folgt einer modularen Struktur, wobei viele verschiedene Module existieren. Einige der, für die Aufgabenstellung interessanten, Hauptmodule werden im Folgenden aufgeführt, wobei in dieser Arbeit ausschließlich Funktionen des Bildverarbeitungs- und Kernmoduls verwendet werden.

- **Core:**

Das Kernmodul definiert grundlegende Datenstrukturen, wie cv:Mat, und stellt Funktionen zur Verfügung, die von allen anderen Modulen benötigt werden.[20]

- **Image Processing:**

Das Bildverarbeitungsmodul enthält eine Vielzahl an Funktionen zur Bildmanipulation, von denen einige im Folgenden aufgezählt werden.[20]

- Bildfilter
- geometrische Transformationen, wie Größenveränderungen oder perspektivische Verschiebungen
- Farbraumkonversionen
- Histogramme
- Kantenerkennung, zum Beispiel nach Canny oder Hough
- Konturerkennung

- **2D Feature Framework:**

In diesem Modul sind Funktionen zum Finden und Vergleichen von Objekt Features enthalten.[20]

- **Object Detection:**

Dieses Modul bietet Funktionen zum Erkennen von Objekten vorgefertigter Klassen. Anhand der Klassen können die Algorithmen verschiedene Objekte erkennen, wie Gesichter, Autos, Tassen und viele weitere. Heftklammern sind hierbei nicht enthalten.[20]

- **Machine Learning:**

Das Modul enthält Funktionen, die verschiedene Ansätze des maschinellen Lernens umsetzen.[20]

Informationen werden in OpenCV in Objekten der Klasse Mat gespeichert, welche als n-dimensionale numerische Arrays vorliegen. Dort können Objekte verschiedener Datentypen, wie Matrizen, Grauwert- oder Farbbilder, Vektorfelder und Histogramme gespeichert werden.[19] Grauwertbilder entsprechen dabei zweidimensionalen und Farbbilder dreidimensionalen Matrizen. Koordinaten im Bild können anhand ihres Reihen- und Spaltenwerts beschrieben werden, wobei der Nullpunkt beider Koordinatenachsen in der oberen linken Ecke eines Bildes liegt.[15]

2 Positionierung

Das folgende Kapitel beschäftigt sich mit der Aufführung und Erläuterung der verfolgten Ansätze und durchgeführten Schritte, die zur Erhöhung der Positioniergenauigkeit des Roboters Baxter führen sollten.

Rethink Robotics stellt zur Kalibrierung des Roboters zwei ausführbare Skripte zur Verfügung. Bei Ausführung dieser werden in verschiedenen Armposen die gemessenen Drehmomente mit gespeicherten Sollwerten verglichen und die Abweichung in die weiteren Bewegungsberechnungen integriert.[25] Diese Kalibrierung wurde in regelmäßigen Abständen durchgeführt, um bei den durchgeführten Messungen aussagekräftige Ergebnisse zu erhalten. Aufgrund der, in Kapitel 1.2 genannten, Greiferwahl des Roboters ist eine erhöhte Positionierungsgenauigkeit vor allem für den linken Arm notwendig. Aus diesem Grund werden die folgenden Messungen und Versuche nur auf diesem durchgeführt. Jegliche entworfene Interfaces und Skripte sind jedoch so entwickelt worden, dass sie mit nur minimalen Änderungen auch für den anderen Arm oder teilweise auch andere Roboter genutzt werden können.

2.1 Architektur

Die, im Zuge dieser Arbeit entwickelte, Software wurde in Form eines ROS-Package zusammengefasst, sodass sie einfach in einen ROS-Workspace integriert werden kann. Das geschaffene Package `baxter_staples` besteht aus den drei Hauptkomponenten `base`, `precision` und `cv_scripts`. Im folgenden Kapitel finden nur die Komponenten `base` und `precision` Anwendung und werden im Folgenden erläutert. Auf die dritte Komponente, sowie die Klasse `cam_class` der `base`-Komponente, wird in Kapitel 3.2 näher eingegangen. `cam_class` wird in diesem Kapitel nicht angewandt, da Baxters Kameras zur Erhöhung der Positioniergenauigkeit nicht benötigt werden.

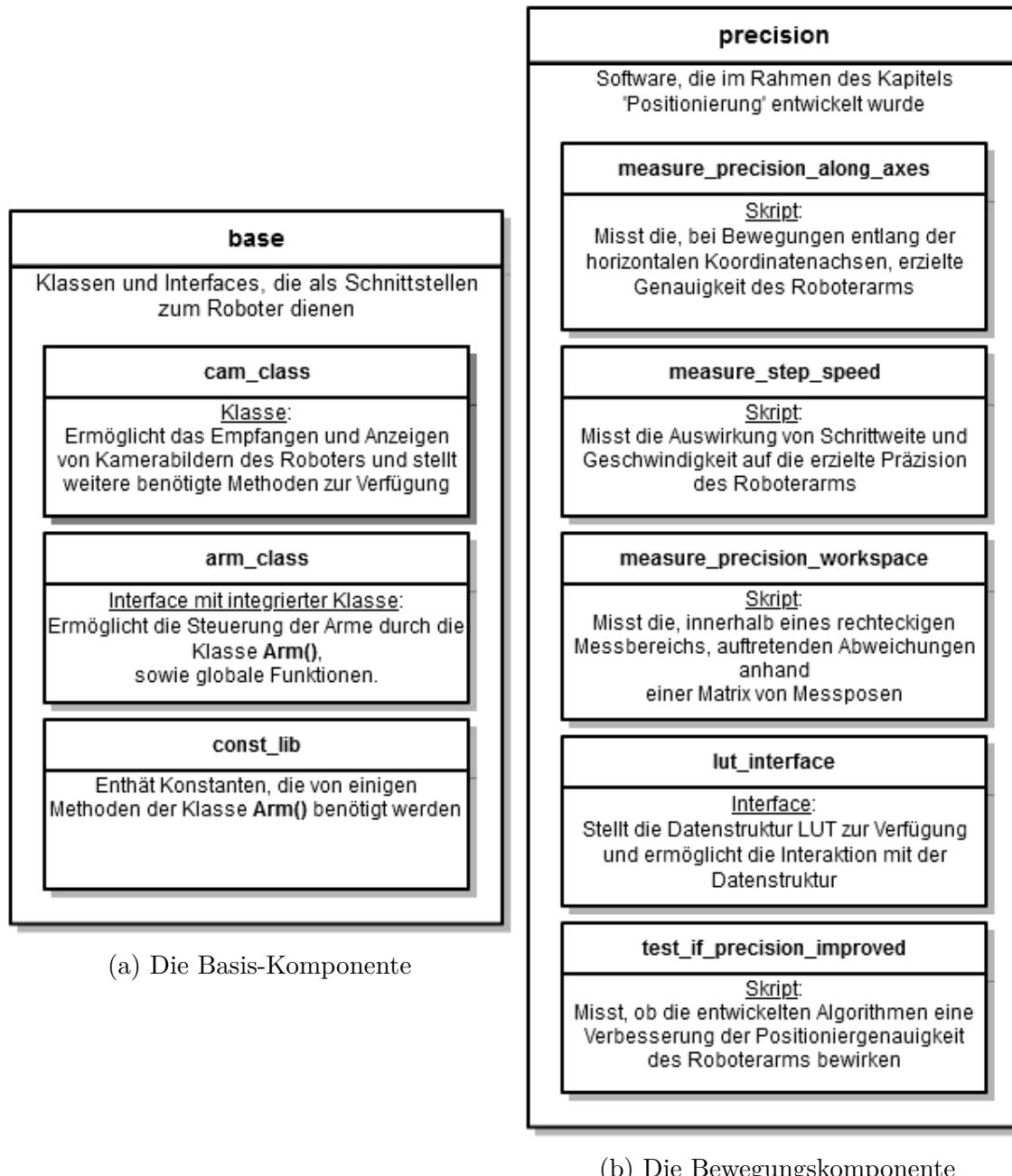


Abbildung 5: In diesem Kapitel verwendete Komponenten des Pakets `baxter_staples`

Abbildung 5a stellt die, in der Komponente `base` enthaltene, Software dar. `arm_class` fungiert als Schnittstelle zwischen dem Package `baxter_staples` und dem Roboter. Hierzu erweitert es das, von Rethink Robotics bereitgestellte, `baxter_interface`, welches diverse Methoden und Konfigurationsmöglichkeiten für jede Hardwarekomponente des Roboters zur Verfügung stellt. Die Kommunikation zwischen dem `baxter_interface` und dem Roboter geschieht über ROS. `arm_class` stellt verschiedene Methoden zur Manipulation von Posen und der Steuerung der Roboterarme zur Verfügung. Auf diese wird in der vorliegenden Arbeit nicht näher eingegangen, da `arm_class` im Zuge einer anderen Prüfungsleistung entwickelt und erklärt wurde. Jegliche, im Zuge dieser Arbeit an `arm_class` vorgenommenen, Änderungen werden in den folgenden Kapiteln kenntlich gemacht und erläutert.

Die Datei `const_lib` enthält Posen und andere Konstanten, die zur Ausführung verschiedener, in `arm_class` implementierter, Methoden benötigt werden. Aus diesem Grund wird sie von `arm_class` importiert.

Abbildung 5b zeigt die in der Komponente `precision` enthaltene Software. Abgesehen von `lut_interface` bestehen diese aus ausführbaren Python-Skripten, die jeweils automatisierte Messungen, bezüglich der Positioniergenauigkeit der Roboterarme, ausführen. Das `lut_interface` enthält eine, im Zuge dieser Arbeit entwickelte, Datenstruktur und die dazugehörigen Funktionen. Alle, in Abbildung 5b aufgeführten, Bestandteile der Komponente `precision` nutzen, in `arm_class` implementierte, Funktionen oder erzeugen ein Objekt der darin enthaltenen Arm-Klasse, zur Interaktion mit dem Roboter.

2.2 Finden eines systematischen Fehlers

Um einen geeigneten Ansatz zum Ausgleich der in Unterkapitel 1.1 genannten Abweichungen entwickeln zu können, muss zuerst festgestellt werden, ob es sich hierbei um einen systematischen oder zufälligen Fehler handelt. Ein systematischer Fehler bedeutet in diesem Zusammenhang, dass bei mehrfachem Anfahren eines Punktes, bzw. Verfahren in eine Richtung, eine größtenteils gleichbleibende Abweichung messbar ist. Solch eine erwartbare Abweichung lässt sich durch eine Kalibrierung ausgleichen. Hierbei wird die auftretende Abweichung vor dem eigentlichen Programmdurchlauf gemessen und bei späteren Bewegungen in die Berechnungen mit einbezogen. Ein zufälliger Fehler hingegen würde bei mehreren Messdurchläufen jedes mal eine unterschiedliche Abweichung aufweisen, was sich nicht durch eine Kalibrierung lösen lässt. Stattdessen müsste bei jeder, vom Roboter ausgeführten, Bewegung überprüft werden, ob die gewünschte Pose ausreichend präzise erreicht wurde, um diese gegebenenfalls zu korrigieren. Um herauszufinden, um welche Art von Fehler es sich handelt, wurden entsprechende Messungen mittels des in Anhang D - `measure_precision_along_axes.py` zu sehenden Skripts durchgeführt. Bei Ausführung des Skripts fährt der Roboter nacheinander zehn

Posen an, die in Zweizentimeterabständen entlang einer, der in Abbildung 2 zu sehenden, Koordinatenachsen liegen und misst die auftretende X- und Y-Differenz zu der erwarteten Pose. Diese Bewegungsfolge wird je 20 mal in beide Richtungen entlang der X-Achse und Y-Achse durchgeführt. Die Ergebnisse dieser Messungen sind beispielhaft in den Abbildungen 6 bis 9 zu sehen und werden anhand dieser erläutert.

Die in Abbildung 6 dargestellten Messdaten wurden erfasst, während der Greifer sich in positiver Richtung entlang der X-Achse bewegte, sich demnach in gerader Form und ohne Höhenänderung vom Roboter entfernte. Daher wird die Bewegung entlang der X-Achse im Folgenden, während der Beschreibung und Interpretation von Abbildung 6, als primäre Richtung bezeichnet. Die in der Ebene senkrecht hierzu stehende Y-Achse wird dementsprechend als sekundäre Richtung bezeichnet.

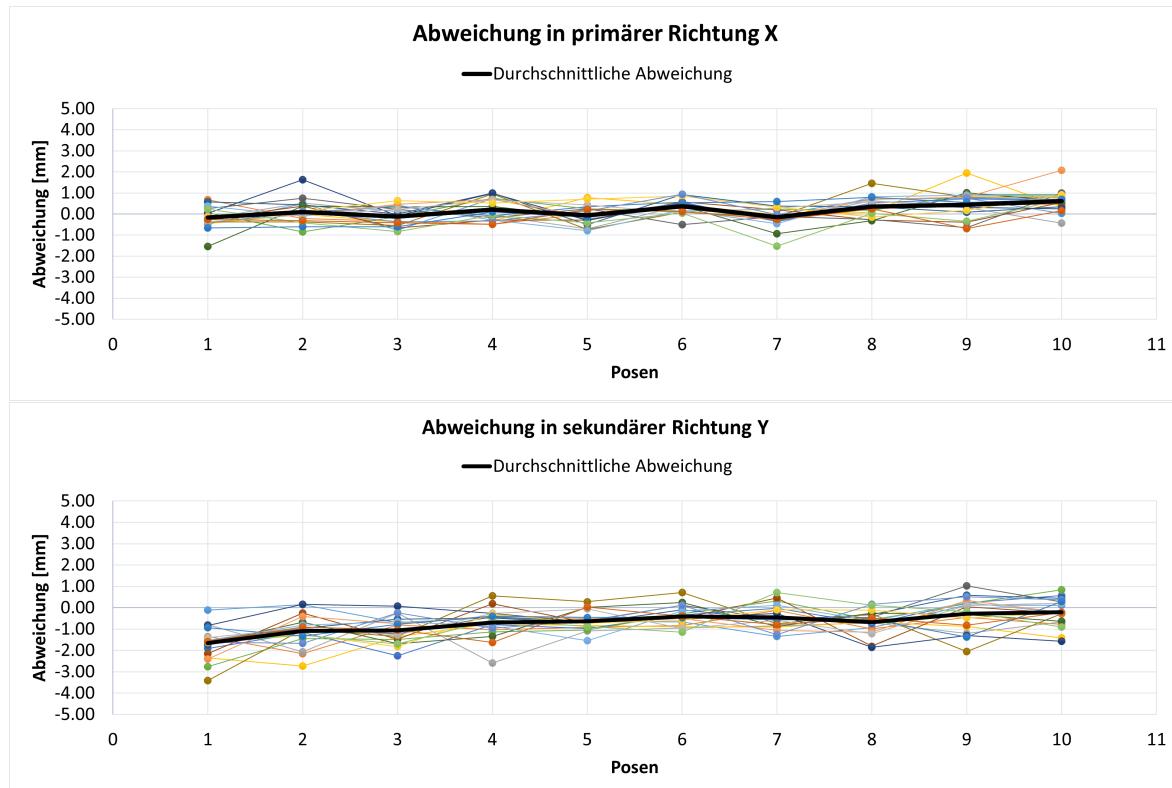


Abbildung 6: Messung entlang der Koordinatenachse: Positiv entlang X-Achse

Tabelle 1: Messung entlang der Koordinatenachse: Positiv entlang X-Achse

Pose	Mittlere Abweichung in primäre Richtung [mm]	Mittlere Abweichung in sekundäre Richtung [mm]
1	-0.17	-1.66
2	0.09	-1.09
3	-0.12	-1.06
4	0.21	-0.70
5	-0.07	-0.63
6	0.37	-0.41
7	-0.15	-0.45
8	0.35	-0.67
9	0.44	-0.28
10	0.62	-0.22

Ein Vergleich der, in Tabelle 1 aufgeführten, mittleren Abweichung in sekundärer Richtung zeigt eine nichtstetige Steigung mit fortlaufender Posenzahl. Dieses Verhalten wird durch die schwarze Linie in Abbildung 6, die die Werte aus der Tabelle repräsentiert, verbildlicht. Die Werte der mittleren Abweichung in primäre Richtung zeigen kein eindeutiges Verhalten, jedoch lassen der Verlauf der Werte, sowie die dazugehörige schwarze Linie eine Steigung vermuten. Dieses Verhalten lässt darauf schließen, dass der Roboterarm mit steigender Distanz zum Robotertorso eher zum Über- als Untersteuern neigt. Die in Abbildung 6 je Pose eingezeichneten Punkte stellen die gemessenen Abweichungen der Messreihen dar. Eine Häufung dieser um die schwarze Linie weist auf einen systematischen Fehler hin. Solche Häufungen sind sowohl in primärer, als auch sekundärer Richtung bei allen Posen zu erkennen.

Die in Abbildung 7 dargestellten Messdaten wurden erfasst, während der Greifer sich in negativer Richtung entlang der X-Achse bewegte, demnach in gerader Form und ohne Höhenänderung dem Roboter näherkommend. Daher wird die Bewegung entlang der X-Achse weiterhin als primäre Richtung bezeichnet. Die in der Ebene senkrecht hierzu stehende Y-Achse wird dementsprechend als sekundäre Richtung bezeichnet.

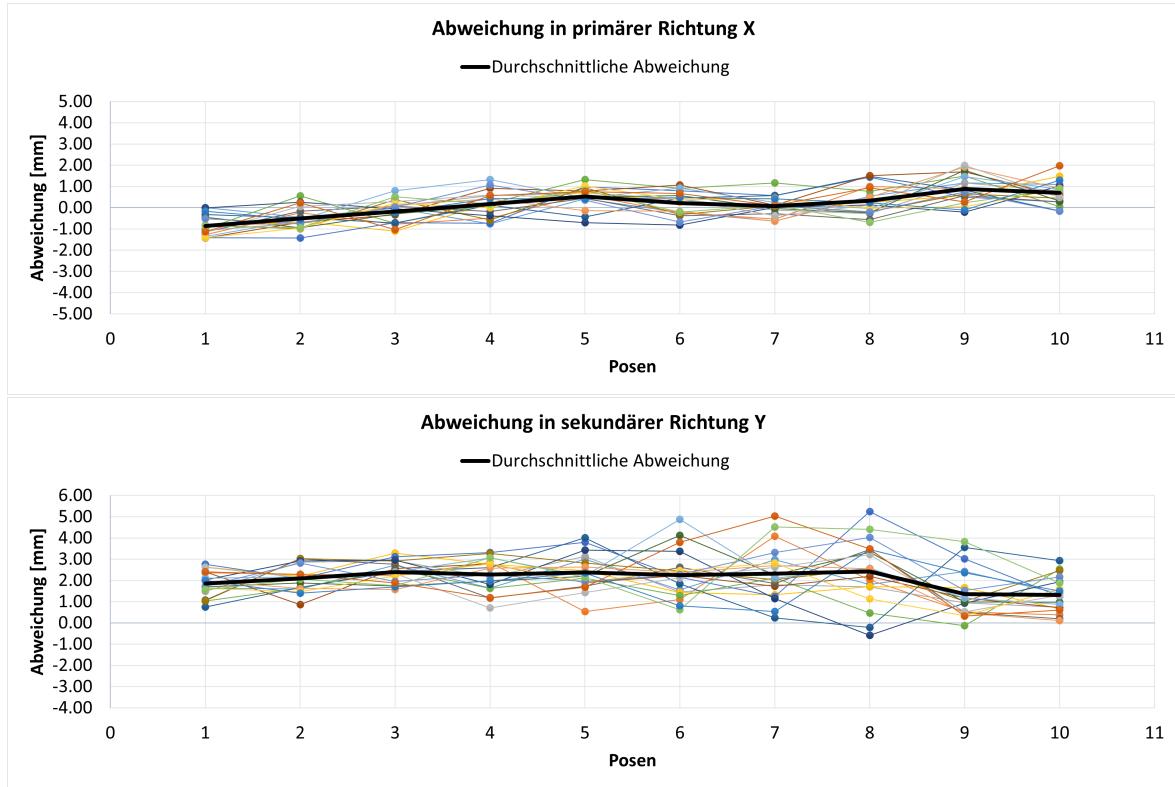


Abbildung 7: Messung entlang der Koordinatenachsen: Negativ entlang X-Achse

Tabelle 2: Messung entlang der Koordinatenachse: Negativ entlang X-Achse

Pose	Mittlere Abweichung in primäre Richtung [mm]	Mittlere Abweichung in sekundäre Richtung [mm]
1	-0.86	1.85
2	-0.51	2.10
3	-0.17	2.40
4	0.18	2.28
5	0.52	2.37
6	0.21	2.25
7	0.07	2.32
8	0.33	2.42
9	0.88	1.35
10	0.70	1.31

Anhand der Werte, der mittleren Abweichung in primärer Richtung, aus Tabelle 2 lässt sich eine stetige Steigung von Pose 1 bis 5 erkennen, sowie von Pose 6 bis 10. Dabei liegen Start- und Endwert der zweiten genannten Steigung über denen der ersten. Diese Beobachtungen lassen darauf schließen, dass die mittlere Abweichung in primärer Richtung insgesamt einer Steigung unterliegt. In sekundärer Richtung lässt sich kein solches Verhalten feststellen, da die Werte der mittleren Abweichung zunächst ansteigen, sich daraufhin jedoch nur gering verändern, bevor sie zu Pose 10 hin wieder absteigen. Es fällt jedoch auf, dass die mittlere Abweichung in sekundärer Richtung bei allen Posen im positiven Wertebereich zu finden sind.

Bei allen Posen der primären, sowie Pose 1 bis 3 der sekundären Richtung lassen sich erneut Häufungen der Punkte, die die gemessene Abweichung je Messreihe und Pose darstellen, um die schwarze Linie erkennen. Von Pose 4 bis 8 in sekundärer Richtung steigt die Distanz der Punkte untereinander jedoch stark an, was die Aussagekraft der mittleren Abweichung für diese Posen beeinträchtigt. Dieses von den anderen Ergebnissen abweichende Streuungsverhalten kann nicht durch äußere Einflüsse begründet werden, da bei der Durchführung der Messungen auf eine gleichbleibende Messumgebung geachtet wurde.

Die in Abbildung 8 dargestellten Messdaten wurden erfasst, während der Greifer sich in positiver Richtung entlang der Y-Achse bewegte, demnach in gerader Form und ohne Höhenänderung quer vor dem Roboter entlangfuhr. Daher wird die Bewegung entlang der Y-Achse im Folgenden während der Beschreibung und Interpretation von Abbildung 8 als primäre Richtung bezeichnet. Die in der Ebene senkrecht hierzu stehende X-Achse wird dementsprechend als sekundäre Richtung bezeichnet.

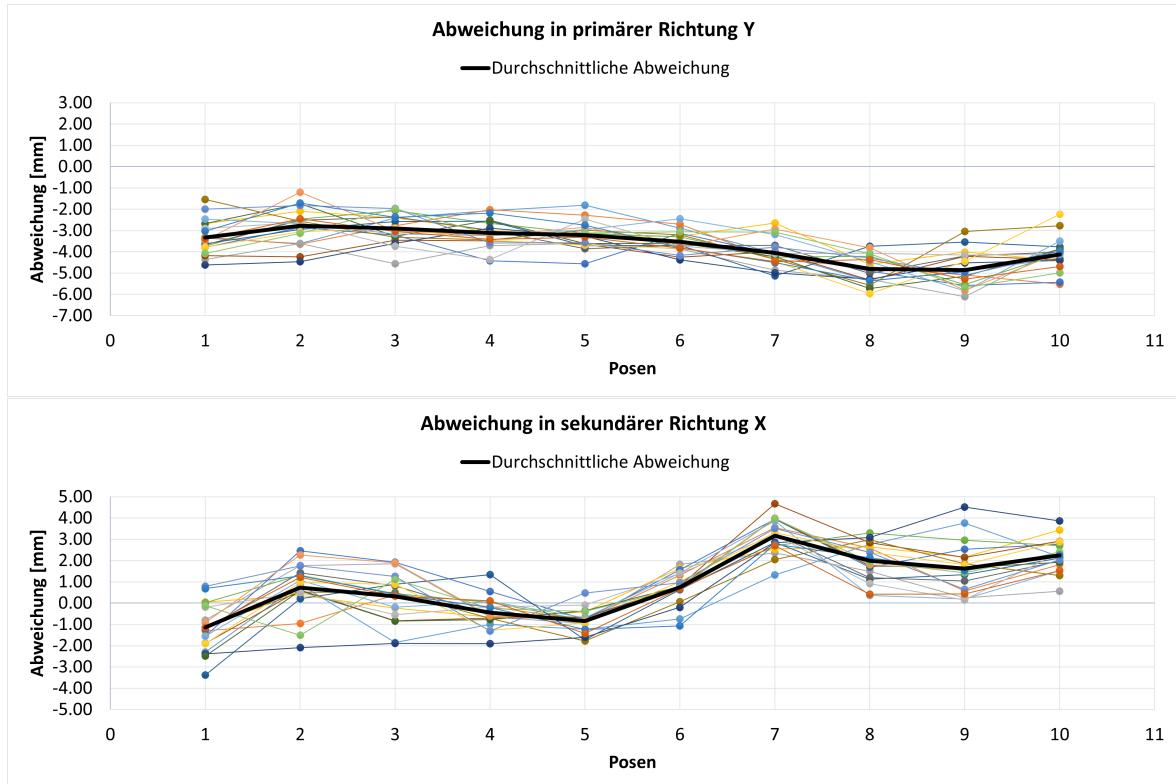


Abbildung 8: Messung entlang der Koordinatenachse: Positiv entlang Y-Achse

Tabelle 3: Messung entlang der Koordinatenachse: Positiv entlang Y-Achse

Pose	Mittlere Abweichung in primäre Richtung [mm]	Mittlere Abweichung in sekundäre Richtung [mm]
1	-3.33	-1.14
2	-2.77	0.73
3	-2.91	0.32
4	-3.12	-0.44
5	-3.22	-0.84
6	-3.53	0.75
7	-4.06	3.17
8	-4.81	1.99
9	-4.86	1.63
10	-4.12	2.24

In primärer Richtung liegt die mittlere Abweichung bei allen Posen um mehrere Millimeter unter null. Weiterhin lassen sich an den gemittelten Werten in Tabelle 8 keine, einem Muster folgenden, Veränderungen der Abweichung feststellen. Die mittlere Abweichung in sekundärer Richtung gleicht ebenfalls keinem eindeutigen Muster. Stattdessen weisen die Werte, im Vergleich zu denen der anderen Bewegungsrichtungen, starke Sprünge auf, wie besonders am Übergang von Pose 6 zu 7 zu sehen ist. Wie bereits in den vorigen Abbildungen sind auch hier in primärer und sekundärer Richtung Häufungen der Punkte entlang der schwarzen Linie zu erkennen, die die Abweichungen je Messreihe und Pose darstellen. Besonders das Auftreten dieser geringen Streuung der Punkte an Pose 7, im Zusammenhang mit dem Sprungverhalten deutet darauf hin, dass die auftretenden Abweichungen als systematische Fehler klassifiziert werden können.

Die in Abbildung 9 dargestellten Messdaten wurden erfasst, während der Greifer sich in negativer Richtung entlang der Y-Achse bewegte, demnach in gerader Form und ohne Höhenänderung quer vor dem Roboter entlangfuhr. Daher wird die Bewegung entlang der Y-Achse im folgenden weiterhin als primäre Richtung bezeichnet. Die in der Ebene senkrecht hierzu stehende X-Achse wird dementsprechend als sekundäre Richtung bezeichnet.

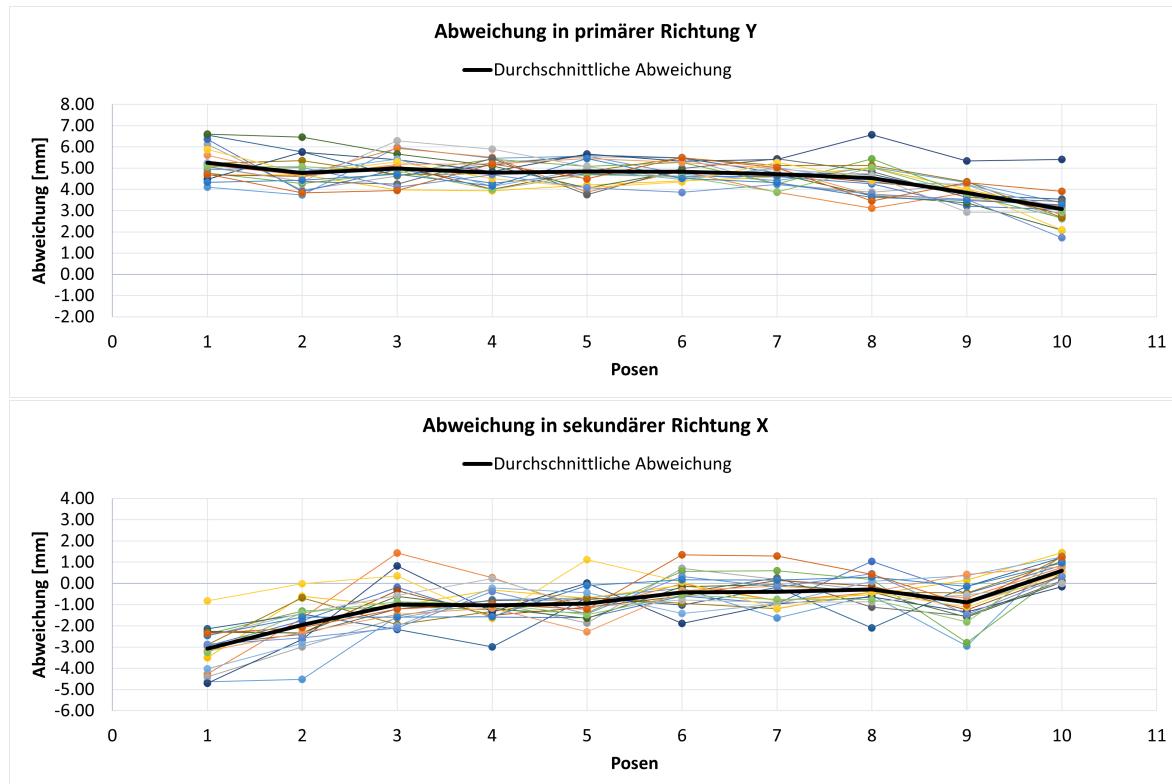


Abbildung 9: Messung entlang der Koordinatenachse: Negativ entlang Y-Achse

Tabelle 4: Messung entlang der Koordinatenachse: Negativ entlang Y-Achse

Pose	Mittlere Abweichung in primäre Richtung [mm]	Mittlere Abweichung in sekundäre Richtung [mm]
1	5.26	-3.07
2	4.77	-1.96
3	4.99	-0.99
4	4.78	-1.03
5	4.84	-0.95
6	4.82	-0.42
7	4.72	-0.40
8	4.53	-0.29
9	3.83	-0.89
10	3.08	0.60

In primärer Richtung liegt die mittlere Abweichung bei allen Messposen um mehrere Millimeter im positiven Wertebereich, wie Tabelle 4 entnommen werden kann. Insgesamt weisen die Werte der mittleren Abweichung hier keine explizit steigende oder sinkende Tendenz auf. Besonders bei Pose 4 bis 6 unterscheiden sich die Werte nur leicht. Ab Pose 7 nähern sie sich Null an. Es lässt sich hier kein eindeutiges Muster erkennen. In sekundärer Richtung ist insgesamt eine weniger starke mittlere Abweichung zu erkennen, die bei Pose 1 ihr Minimum und Pose 10 ihr Maximum aufweist. Mit Ausnahme von Pose 9 und 3 weisen die Werte insgesamt eine Tendenz zur Steigung auf. Wie bereits bei den vorigen Abbildungen und Tabellen neigen die Punkte, die die auftretende Abweichung je Messreihe und Pose darstellen, dazu sich um die schwarze Linie, die die mittlere Abweichung darstellt, zu häufen. Dabei scheinen diese Punkte in sekundärer Richtung eine stärkere Streuung aufzuweisen, was sich besonders an Pose 3 zeigt.

Insgesamt lassen sich aus den Abbildungen 6 bis 9 mehrere wichtige Erkenntnisse erlangen. In ihrer Gesamtheit lässt sich aus den Messdaten auf einen systematischen Fehler schließen. Hierauf weist vor allem die, über mehrere Messreihen hinweg, stets ähnliche Abweichung je Pose. Ein Vergleich der Abbildungen 6 bis 9 zeigt, dass die sekundäre Richtung stets eine insgesamt höhere Streuung je Pose aufweist, als die Primäre. Diese Beobachtung legt nahe, dass, wenn in einer Koordinatenachse eine hohe Präzision erforderlich ist, eine Bewegung entlang dieser Koordinatenachse ausgeführt werden sollte.

Hier, sowie im Folgenden, wird das Wort Präzision explizit verwendet, um eine geringe Streuung der Ergebnisse zu beschreiben, unabhängig von ihrer durchschnittlichen Abweichung von der Zielpose.

Unabhängig davon, entlang welcher Koordinatenachse eine Bewegung ausgeführt wurde, weisen die Ergebnisse stets eine größere mittlere Abweichung in Y-, als in X-Richtung, auf. Das lässt darauf schließen, dass zum Erlangen einer möglichst geringen Abweichung von der Zielpose eine Bewegung entlang der X-Koordinatenachse ausgeführt werden sollte, also auf den Robotertorso zu oder von diesem weg.

Auf Grundlage dieser Beobachtungen wurden für das weitere Vorgehen mehrere Entscheidungen getroffen. Die Erhöhung der Positioniergenauigkeit soll durch zwei kombinierte Ansätze erreicht werden.

1. Verringerung der Streuung

Der erste Ansatz hat die Präzisionserhöhung des Roboterarms zum Ziel. Um dieses Ziel zu erreichen, werden Messungen durchgeführt, die die Auswirkungen von Schrittweite und Verfahrgeschwindigkeit auf die erzielte Präzision erfassen. Die erfolgversprechendste der getesteten Konfigurationen wird anschließend als Methode im, zur Steuerung des Roboters genutzten, Interface `arm_class` implementiert.

2. Verringerung der Abweichung

Die Beobachtungen zeigen, dass für die getesteten Posen systematische Fehler, also reproduzierbare Abweichungen von bis zu mehreren Millimetern, in beide Achsrichtungen vorliegen. Da diese jedoch keinem klaren Muster folgen, sondern von nicht direkt ersichtlichen Faktoren der angesteuerten Pose abhängig sind, kann hier keine normale Kalibrierung angewandt werden. Stattdessen wird ein Ansatz verfolgt, bei dem eine Umsetzungstabelle (LUT) für einen definierten Arbeitsbereich erstellt wird.

Um die Arbeitsmenge und die Menge an Messdaten ökonomisch zu halten, wurde entschieden, sich im weiteren Projektverlauf auf nur eine Bewegungsrichtung zu beschränken. Da bei den zuvor durchgeföhrten Messungen die eindeutigsten Ergebnisse in Bezug auf Streuung und mittlere Abweichung bei der Bewegung in positive Richtung entlang der X-Achse gefunden wurden, fiel die Entscheidung auf Ebendiese. Für eine bessere Verständlichkeit wird sie im Folgenden als primäre Richtung, sowie die horizontal senkrecht dazu liegende positive Richtung entlang der Y-Achse als sekundäre Richtung, bezeichnet. Dementsprechend wird die vertikal vorliegende positive Richtung entlang der Z-Achse im folgenden als tertiäre Richtung bezeichnet.

2.3 Präzisionserhöhung

Wie in Unterkapitel 2.2 erwähnt, dient der erste Ansatz der Erhöhung der Präzision, also der Verringerung der Streuung beim Anfahren eines Punktes. Zur Messung der Auswirkungen von Schrittweite und Geschwindigkeit auf die Präzision, wurde das in Anhang E - `measure_step_speed.py` zu sehende Skript entwickelt. Die enthaltene Funktion lässt den Roboter einen festgelegten Punkt in zwei Schritten anfahren. Dabei sind sowohl Schrittweite, als auch Geschwindigkeit frei wählbar. Die erste Bewegung dient hierbei dazu etwaiges Spiel in den Gelenken zu minimieren, während die zweite Bewegung den Messpunkt ansteuert. An diesem wird die tatsächlich erreichte Pose mit der Erwarteten verglichen und die Differenz zwischengespeichert. Die genannte Bewegungsfolge wird 20 Mal wiederholt.

Tabelle 5: Konfigurationen für Präzisionsmessungen

Name	Geschwindigkeitswert	Schrittweite [m]
<code>slow_short</code>	0.1	0.01
<code>slow_medium</code>	0.1	0.02
<code>slow_far</code>	0.1	0.03
<code>fast_short</code>	0.3	0.01
<code>fast_medium</code>	0.3	0.02
<code>fast_far</code>	0.3	0.03

Bei Ausführung des Skripts wird die zuvor genannte Funktion sechs mal aufgerufen. Die hierbei verwendeten Konfigurationen sind in Tabelle 5 aufgeführt. Die aufgeführten Geschwindigkeitsangaben entsprechen keiner SI-Einheit, sondern werden in diesem Format vom Baxter Interface erwartet und dürfen in einem Bereich von 0.0 bis 1.0 vorliegen. Der Geschwindigkeitswert 0.3 entspricht dabei Baxters Standardgeschwindigkeit. Aufgrund der Annahme, dass der Roboter bei geringeren Geschwindigkeiten eine höhere Präzision erzielt, wurde eine Vergleichsgeschwindigkeit von 0.1 gewählt.

In Abbildung 10 und Tabelle 6 sind die Ergebnisse der durchgeführten Messungen zu sehen. Die erreichte Präzision wird hierbei an der Größe des Interquartilsabstands festgemacht. Somit ergibt sich bei einem geringeren Interquartilsabstand eine höhere Präzision. Entsprechend der in Unterkapitel 2.2 genannten Definition von Präzision, wird die durchschnittliche Abweichung von der Zielpose bei diesem Versuch nicht betrachtet, da sie in diesem Fall für die Präzisionserhöhung nicht von Bedeutung ist.

Tabelle 6: Ergebnisdaten der Präzisionsmessungen

Name	Interquartilsabstand in primäre Richtung X [mm]	Interquartilsabstand in sekundäre Richtung Y [mm]	Summe [mm]
slow_short	0.367	0.800	1.167
slow_medium	0.696	0.789	1.484
slow_far	0.570	0.979	1.549
fast_short	0.480	0.586	1.066
fast_medium	0.606	0.872	1.479
fast_far	0.737	1.382	2.120

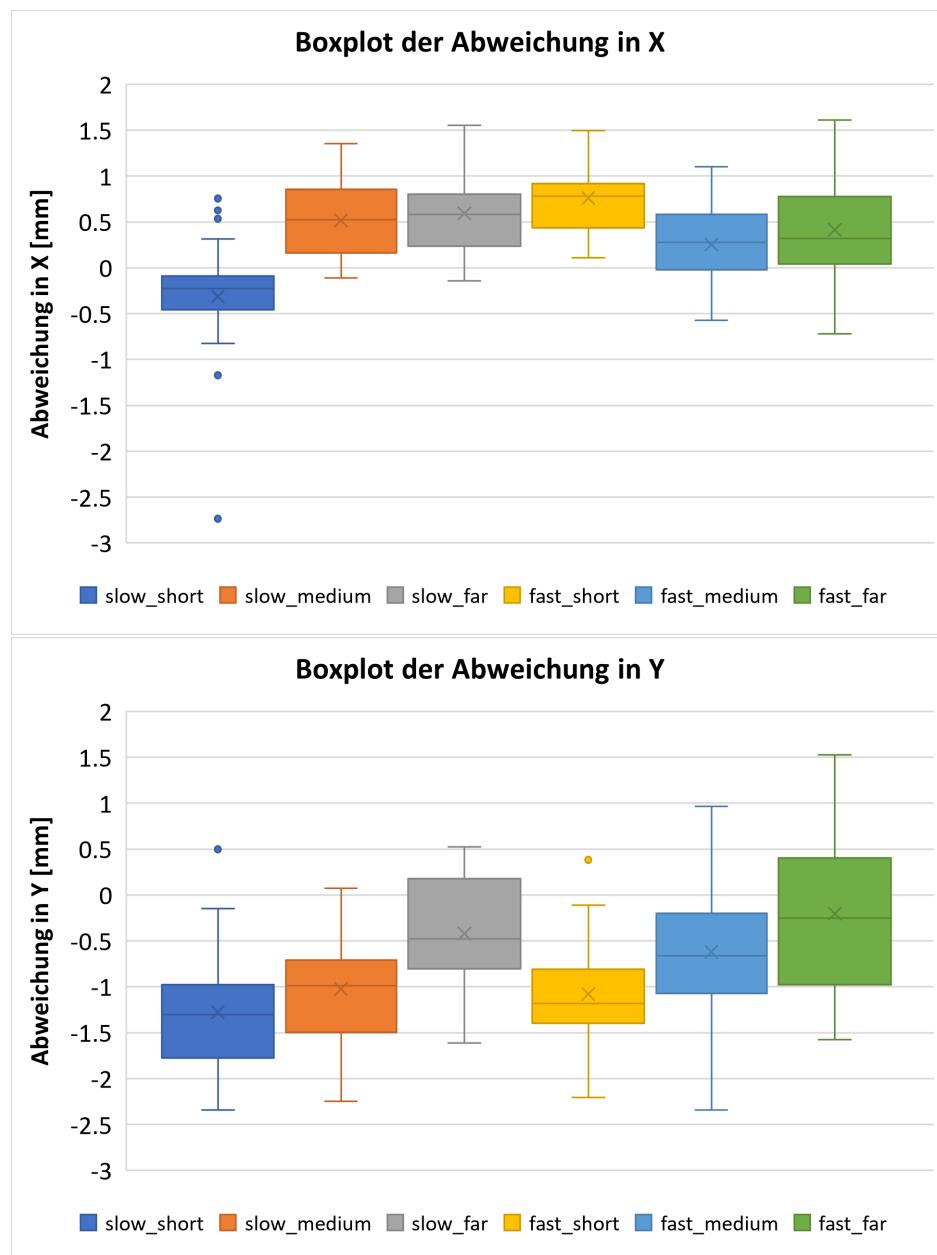


Abbildung 10: Ergebnisdaten der Präzisionsmessungen

Wie in Tabelle 6 zu sehen, ist die höchste Präzision in primäre Richtung mit der Konfiguration 'slow_short' zu erzielen, mit einem Wert von 0.367 Millimetern, gefolgt von 'fast_short' mit einem Wert von 0.480 Millimetern. In sekundärer Richtung hingegen weist 'fast_short' die höchste Präzision auf, mit einem Wert von 0.586 Millimetern. An zweiter und dritter Stelle folgen 'slow_medium', mit 0.789 Millimetern, und 'slow_short', mit einem Wert von 0.800 Millimetern.

Wenn beide Achsen gleichzeitig betrachtet werden, erzielt somit 'fast_short' die höchste Präzision. Zwar liegt die Summe der Interquartilsabstände bei 'slow_short' mit 1.167 Millimetern nur leicht über dem Wert von 1.066 Millimetern, bei 'fast_short', jedoch zeigen die in Abbildung 10 außerhalb der Box liegenden Punkte, dass bei 'slow_short' deutlich mehr und stärkere Ausreißer auftreten, als bei den anderen Konfigurationen. Das Ziel der Präzisionserhöhung ist eine möglichst gleichbleibende, vorherbestimmbare Abweichung von der Zielpose. Ausreißer sind hierbei unerwünscht, da sie eine unerwartete Abweichung von der geplanten Abweichung bedeuten.

Auf Grundlage der genannten Beobachtungen wurde entschieden die Konfiguration 'fast_short' als Methode `move_precise` in dem Interface in Anhang A - `arm_class.py` zu implementieren. Bei Aufruf der Methode wird die übergebene Pose in primäre Richtung in zwei Schritten angefahren. Der erste Schritt soll das Gelenkspiel des Roboterarms minimieren, woraufhin mit dem zweiten Schritt die Zielpose möglichst präzise erreicht werden soll. Jegliche benötigte Bewegung entlang der Y- oder Z-Koordinatenachse werden beim Anfahren der Initialpose des ersten Schrittes ausgeführt.

2.4 Ausgleichen des systematischen Fehlers durch LUT

Wie in Kapitel 2.2 aufgeführt, ist kein Zusammenhang zwischen an benachbarten Zielposen auftretenden Abweichungen erkennbar. Das bedeutet, dass kein allgemeiner Ausgleichswert für einen Roboterarm oder eine Bewegungsrichtung genutzt werden kann, da die auftretende Abweichung von der Zielpose abhängig ist. Um die auftretenden Abweichungen dennoch ausgleichen zu können, wurde das in Anhang G - `lut_interface.py` zu sehende Interface entwickelt. Dieses Interface dient der Erschaffung und Nutzung einer mehrschichtigen Datenstruktur, die die in einem zweidimensionalen Arbeitsbereich auftretenden Abweichungen darstellt.

2.4.1 Die Datenstruktur LUT

Die grundlegende Funktionsweise der Datenstruktur entspricht der einer Umsetzungstabelle (LUT). Das heißt, dass in ihr die für verschiedene Posen zu erwartenden Abweichungen gespeichert werden, um diese im Programmablauf ausgleichen zu können. Ihr Aufbau entspricht dabei einer mehrschichtigen Struktur des Datentyps Dictionary. Dadurch hat jede Ebene eine dynamische Größe. Die beiden untersten Ebenen stellen zusammen einen zweidimensionalen rechteckigen Arbeitsbereich beliebiger Größe als Matrix gleichmäßig verteilter Punkte dar. Die Schlüssel der beiden Ebenen entsprechen der X- und Y-Koordinate der Pose als Integer, während das enthaltene Point-Objekt zur Speicherung der auftretenden Abweichung in drei Dimensionen dient. Die darüberliegenden Ebenen ordnen den dargestellten Arbeitsbereich einer Bewegungsrichtung und diese wiederum einem Arm zu. Durch den dynamischen Aufbau kann die Datenstruktur auf verschiedenen Robotersystemen eingesetzt werden, unabhängig von deren Anzahl an Armen, der genutzten Bewegungsrichtungen oder der Größe des definierten Arbeitsbereiches.

```

1 lut = {
2     'Arm': {
3         'Richtung': {
4             'y [..]': {
5                 'x [..]': Point(
6                     x = 0.000,
7                     y = 0.000,
8                     z = 0.000
9                 )
10            }
11        }
12    }
13}

```

Listing 1: LUT Datenstruktur

2.4.2 Das LUT Interface

In dem in Anhang G - `lut_interface.py` zu sehenden Interface wurde die zuvor erläuterte Datenstruktur, sowie verschiedene Funktionen zur Interaktion mit umgesetzt. Das Interface ermöglicht das Erschaffen eines LUT-Objektes für einen der Norm DIN A4 entsprechenden Arbeitsbereich mit einer Schrittweite von drei Zentimetern. Hierbei nutzt die Funktion die in Kapitel 2.3 genannte Funktion `move_precise`, wodurch die gewählte Schrittweite keine Auswirkung auf die erzielte Präzision sondern nur die Auflösung der Messpunkte im Arbeitsbereich hat. Zusätzlich ist es für diesen Ansatz notwendig, dass bei Erstellung und Anwendung der LUT die gleichen Bewegungen ausgeführt werden, da diese sich auf die auftretenden Abweichungen auswirken.

Mit den genannten Parametern wird somit eine Auflösung von 10x7 Messpunkten erreicht. Bei Wahl einer höheren Auflösung, steigen sowohl der Ressourcenverbrauch, als auch die zur Erschaffung und Anwendung der Struktur benötigte Zeit. Zur Erschaffung eines LUT-Objektes wird der Arbeitsbereich mehrfach entlang der Messpunkte abgefahren. Hierbei wird an jedem Messpunkt die auftretende Abweichung gemessen. Aus allen Durchläufen wird schließlich für jede Koordinate jedes Messpunkts der Mittelwert berechnet. Die erhaltene Struktur wird abschließend auf dem System gespeichert und von der Funktion zurückgegeben, um direkt verwendet werden zu können.

Zum Speichern eines LUT-Objektes wird eine Funktion des Interfaces genutzt. Diese speichert das Objekt als .csv-Datei, sodass es mittels eines Tabellenkalkulationsprogramms ausgewertet werden kann. Zusätzlich kann die Datei durch eine weitere Interface-Funktion ausgelesen und in ein neues LUT-Objekt umgewandelt werden. Dieser Vorgang des Speicherns und Wiederherstellens eines LUT-Objekts ist notwendig, aufgrund des hohen, für die Schaffung eines LUT-Objekts benötigten, Zeitaufwands. Mit den zuvor genannten Parametern beträgt dieser bei zehn Durchläufen etwa 50 Minuten. Angesichts des mit Erhöhung der Auflösung steigenden Zeitaufwands wurden keine Messungen bei höherer Auflösung des Arbeitsbereichs durchgeführt.

Eine weitere Funktion des Interfaces ermöglicht die Beeinflussung einer übergebenen Pose durch ein LUT-Objekt. Diese wendet die entsprechenden Abweichungswerte auf die übergebene Pose an. Wird eine Pose übergeben, welche sich zwischen zwei Messpunkten befindet, werden die Werte der angrenzenden Messpunkte miteinander verrechnet, um einen der Pose entsprechenden Annäherungswert zu schaffen.

2.5 Auswertung

Abbildung 11 zeigt die Daten einer nach den im vorigen Unterkapitel genannten Parametern erstellten LUT in verschiedenen Stadien der Entstehung. Hierzu wurde eine LUT mit 50 Durchläufen erschaffen, wobei alle zehn Durchläufe ein Zwischenstand gespeichert wurde.

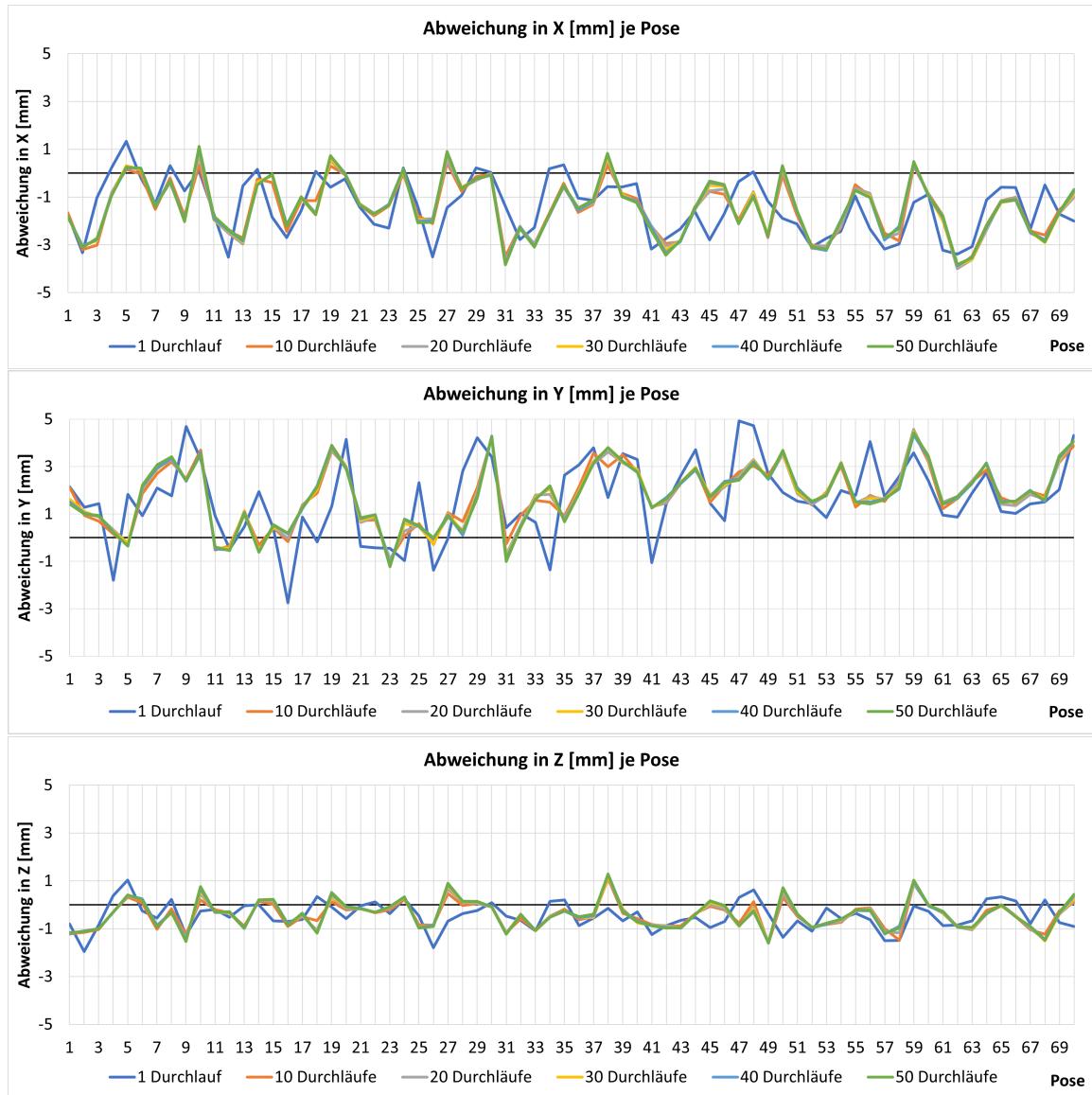


Abbildung 11: LUT mit 50 Durchläufen in 6 Stadien

In Abbildung 11 lässt sich erkennen, dass sich die Kurven der fortlaufenden Stadien der letzten, grünen Kurve, welche 50 Durchläufe repräsentiert, immer weiter angleichen. Es fällt auf, dass bereits die Kurve bei zehn Durchläufen vor allem in X nur geringe Abweichungen zu der mit 50 Durchläufen aufweist. Um einen näherungsweisen Eindruck der durchschnittlichen Abweichung je Pose zu bekommen, ist somit bereits eine Messung mit einer geringen Anzahl an Durchläufen ausreichend. Weiterhin fällt auf, dass die Kurve des einzelnen Durchlaufs an mehreren Stellen deutlich von dem

jeweiligen Durchschnittswert einer Pose abweicht. Vor allem in Y zeigt sich dies mit Abweichungen vom Mittelwert, also der grünen Kurve, von bis zu 2.5 Millimetern (siehe Pose 16). Auch in X und Z sind Abweichungen sichtbar, wobei diese in X maximal 1.5 Millimeter betragen an Pose 26 und in Z an Pose 2 ungefähr 0.7 Millimeter.

Um zu überprüfen, wie sehr sich die umgesetzten Ansätze tatsächlich auf die Positioniergenauigkeit des Roboters auswirken, wurde das in

Anhang H - `test_if_precision_improved.py` zu sehende Skript entwickelt. Bei Aufruf führt dieses 30 Messreihen durch, bei denen entsprechend der vier in Tabelle 7 entnehmbaren Konfigurationen jeweils fünf Messposen abgefahren und die auftretenden Abweichungen gemessen werden. Diese Messposen wurden so gewählt, dass sie sich nahe jeder Ecke und der Mitte des Arbeitsbereichs befinden. Weiterhin wurden Posen gewählt, die sich mit beiden, einer oder keiner Koordinate mit Punkten aus der genutzten LUT decken, sodass hier alle Möglichkeiten durch diesen Versuch abgedeckt werden.

Die Ergebnisse der Abschlussmessungen werden im Folgenden anhand von Boxplots und Tabelle 7 vorgestellt. Abbildung 12 zeigt hierbei die Ergebnisse für alle Messposen zusammengenommen. Weitere Diagramme für die einzelnen Posen befinden sich in Anhang K - Diagramme der Abschlussmessungen.

Tabelle 7: Erhöhung der Positioniergenauigkeit: Gesamtergebnisse

Konfiguration	Mittlere Abweichung in			Absolute
	X [mm]	Y [mm]	Z [mm]	Summe [mm]
1: Normal	-0.659	-0.204	-0.357	1.220
2: Nur <code>move_precise</code>	-1.495	1.051	-0.309	2.855
3: Nur LUT	0.704	-1.739	-0.213	2.656
4: Mit LUT und <code>move_precise</code>	-0.112	-0.412	-0.107	0.631
Interquartilsabstand in				
	X [mm]	Y [mm]	Z [mm]	Summe [mm]
	5.406	2.128	0.802	8.336
1: Normal	1.424	1.028	0.798	3.250
2: Nur <code>move_precise</code>	5.386	4.775	1.581	11.742
3: Nur LUT	2.468	1.619	1.159	5.246

Wie bereits in Unterkapitel 2.3 wird die erreichte Präzision an der Größe des Interquartilsabstands festgemacht. Somit steht ein geringer Interquartilsabstand für eine hohe Präzision und umgekehrt.

Anhand der in Tabelle 7 aufgeführten Interquartilsabstände lässt sich erkennen, dass durch den Einsatz der Methode `move_precise` eine deutliche Erhöhung der Präzision beim Anfahren einer Pose erzielt werden kann, im Vergleich zur durch Konfiguration 1: 'Normal' dargestellten Ausgangssituation. Eine solche Verbesserung zeigt sich bei allen drei Koordinatenachsen, wobei die stärkste in X und die schwächste in Z erzielt wurde. Die in Z erzielte Verbesserung ist hier so gering, dass ihre Aussagekraft fraglich ist.

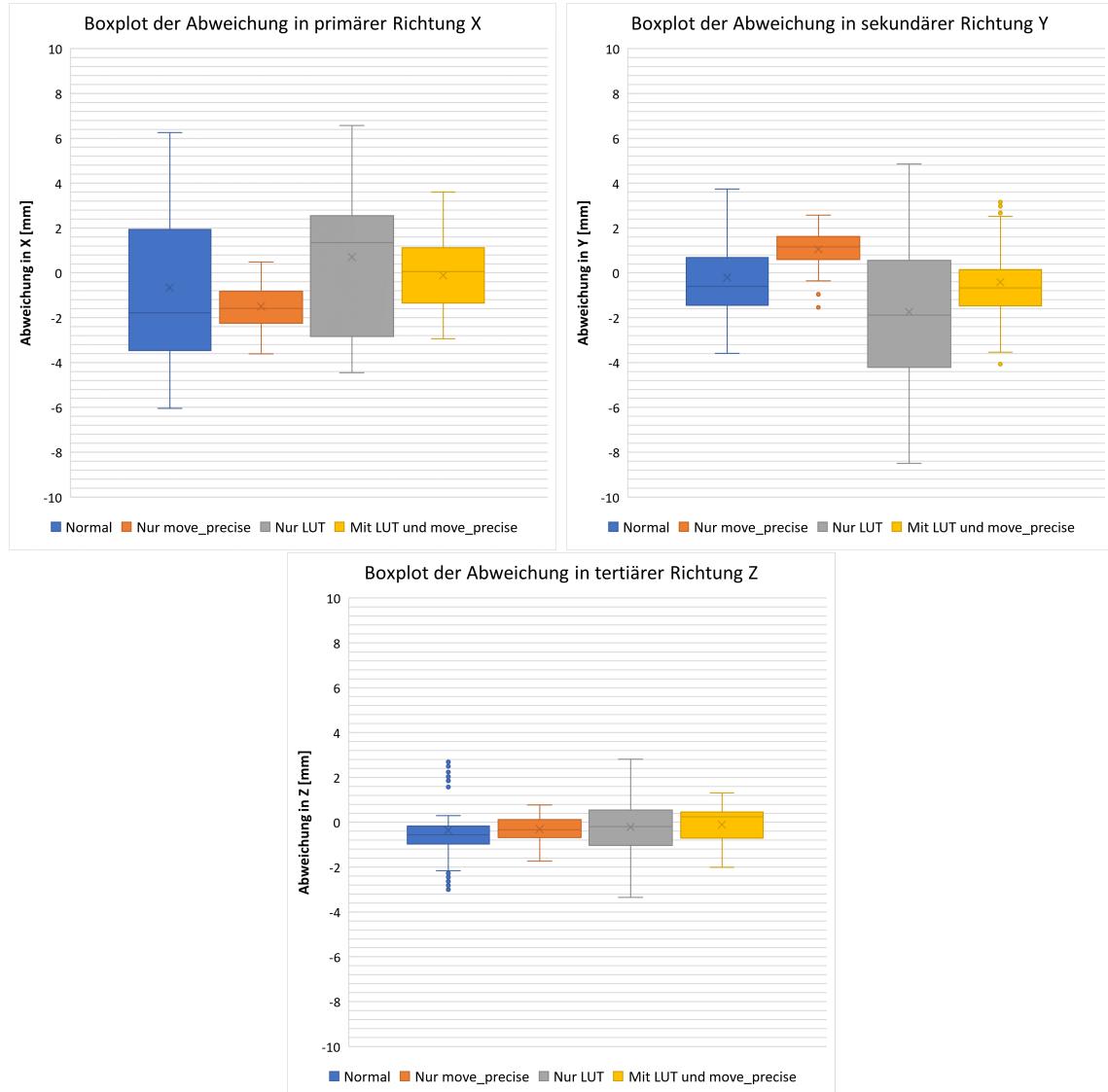


Abbildung 12: Erhöhung der Positioniergenauigkeit: Gesamtergebnisse

Abbildung 12, in tertiärer Richtung, zeigt jedoch, dass bei Konfiguration 1 eine große Zahl an Ausreißern aufgetreten ist, zu sehen an den außerhalb des Kastens liegenden Punkten, während Konfiguration 2: 'Nur move_precise' keine solchen Ausreißer aufweist. Das lässt darauf schließen, dass der Einsatz von move_precise zur Verringerung der auftretenden Ausreißer führt. Wie bereits in Unterkapitel 2.3 erwähnt, ist dies ein positiver Effekt, da die Präzisionserhöhung dazu dient eine möglichst vorhersagbare und damit ausgleichbare Abweichung von der Zielpose zu erreichen.

Der Vergleich von Konfiguration 3: 'Nur LUT' und 4: 'Mit LUT und move_precise' zeigt eine Erhöhung der Präzision durch den Einsatz der Methode move_precise. Auch hier wurde die stärkste Verbesserung in X erzielt und die schwächste in Z, wobei die Verbesserung in Z stärker ausfällt als beim Vergleich von Konfiguration 1 und 2.

Der Einsatz der LUT scheint die Präzision zu verringern. Dies zeigt sich sowohl im Vergleich der Konfigurationen 1 und 3, als auch im Vergleich von 2 und 4. Abgesehen von X im Vergleich zwischen Konfiguration 1 und 3 lässt sich immer eine Vergrößerung

des Interquartilsabstands feststellen.

Der Vergleich von Konfiguration 1 und 2 zeigt in X und Y eine erhöhte mittlere Abweichung von Null bei Einsatz der Methode `move_precise`. Hierbei wird der Mittelwert der Abweichung, statt des bei Boxplots üblicherweise verwendeten Medians, als Kriterium verwendet, weil dieser auch durch auftretende Ausreißer beeinflusst wird. Da Ausreißer nicht nur in den Versuchen, sondern auch dem späteren Einsatz des Roboters vorkommen können, werden sie als nicht vernachlässigbar angesehen und durch Nutzung des Mittelwerts in die Ergebnisse einbezogen.

Der reine Einsatz der LUT ohne Nutzung der Methode `move_precise`, also nach Konfiguration 3, zeigt insgesamt keine nennenswerten Verbesserungen gegenüber Konfiguration 1. Dieses Ergebnis entspricht den Erwartungen, da wie in Unterkapitel 2.4 genannt bei Erstellung und Anwendung der LUT möglichst gleiche Bewegungen ausgeführt werden sollten. Entsprechend gleichartige Bewegungen werden bisher nur durch den Einsatz von `move_precise` sichergestellt. Der Vergleich von Konfiguration 1 und 4 zeigt eine Verringerung der mittleren Abweichung in X und Z, während diese in Y jedoch vergrößert wurde. Der Interquartilsabstand wiederum wurde durch den Einsatz der entwickelten Algorithmen in X und Y verringert, in Z jedoch vergrößert.

Um die Auswirkungen der einzelnen Konfigurationen in ihrer Gesamtheit erfassen zu können, wurden die mittlere Abweichung und die Interquartilsabstände über die Achsen absolut summiert. Anhand der Summen der mittleren Abweichungen lässt sich erkennen, dass der Einsatz eines einzelnen Optimierungsalgorithmus insgesamt keine Verbesserung zum Ausgangszustand darstellt, sondern stattdessen die auftretenden Abweichungen verstärkt. Der kombinierte Einsatz beider Algorithmen zeigt allerdings eine Verbesserung um 48% gegenüber dem Ausgangszustand.

Ein Vergleich der Summen der Interquartilsabstände zeigt ein anderes Ergebnis. Wie bereits zuvor erwähnt wirkt sich der Einsatz der LUT negativ auf die Präzision aus. Dennoch lässt sich durch den kombinierten Einsatz beider Algorithmen eine Verbesserung um 37%, gegenüber dem Ausgangszustand, erzielen. Der reine Einsatz der Methode `move_precise` ermöglicht eine Verbesserung der Präzision um 61%, gegenüber dem Ausgangszustand, bei einer Vergrößerung der mittleren Abweichung um 134%. Auf Grundlage dieser Beobachtungen wird der kombinierte Einsatz aus LUT und `move_precise` empfohlen, um eine verbesserte Positioniergenauigkeit des Roboters zu erzielen.

In Kapitel 2 wurden diverse Messungen zur Genauigkeit von Baxter beim Anfahren verschiedener Posen durchgeführt und ausgewertet. Auf Grundlage dieser Messungen wurden Algorithmen zur Genauigkeitserhöhung entwickelt, die sowohl auf Baxter, als auch anderen Robotersystemen eingesetzt werden können. Schließlich wurde anhand weiterer Messungen festgestellt, dass die entwickelten Algorithmen tatsächlich eine Erhöhung der Positioniergenauigkeit bewirken können.

3 Heftklammererkennung

Wie in Unterkapitel 1.1 erwähnt, ist es notwendig, dass der Roboter Teile seiner Umwelt wahrnimmt, um für die Heftklammerentfernung eingesetzt werden zu können. In Unterkapitel 1.2 wurden die Systeme aufgezählt, die Baxter hierzu befähigen. Unter anderem sind an den Greifergrundplatten Kameras verbaut, über welche Bilder aufgenommen und über ROS an den angeschlossenen Computer übertragen werden können. Durch die Position der Kameras können diese das Umfeld der angeschlossenen Greifer erfassen, was nicht nur eine Überwachung der Tätigkeit der Greifer ermöglicht, sondern auch eine bedingte Steuerung der Arme durch das Kamerabild. Eine Möglichkeit der kameragestützten Steuerung des Roboterarms, in Bezug auf die Aufgabenstellung dieser Arbeit, wird im folgenden Kapitel aufgezeigt und die dafür durchgeführten Schritte erläutert. Bei der Entwicklung, der im weiteren Verlauf vorgestellten Software, wurde darauf geachtet, diese möglichst systemunabhängig zu gestalten, sodass sie auch auf anderen Robotern eingesetzt werden kann. Vor allem bei Funktionen und Methoden, die für die Berechnung von Bewegungen benötigt werden, kann die erfolgreiche Durchführung jedoch nicht für andere Robotersysteme garantiert werden, da Formeln entwickelt und verwendet wurden, die von den Bewegungsmustern des Roboters und der verbauten Hardware abhängig sind.

3.1 Versuchsaufbau

Eines der Ziele der vorliegenden Arbeit liegt darin, wie in Unterkapitel 1.1 genannt, einige der Restriktionen aufzuheben, die im Zuge des vorangegangenen Praxisprojekts aufgestellt wurden. Dennoch wird für die, im Folgenden vorgestellten, Versuche und Lösungen ein spezifizierter Arbeitsbereich benötigt, der gewissen Vorgaben entspricht. Um die, für diese Arbeit essentiellen, Aspekte des Entklammerungsprozesses darstellen zu können, werden Heftklammern mittels eines Stifts markiert. Hierzu wird ein mehrteiliger Adapter für den elektrischen Greifer des Roboters verwendet, der es ihm ermöglicht, Stifte verschiedener Durchmesser zu halten. Dieser Adapter ist in Abbildung 13 zu sehen. Er wurde im Zuge dieser Arbeit entworfen und gefertigt.



Abbildung 13: Der Stifthalter

Der Greiferadapter wurde zur Montage an einem elektrischen Greifer von Rethink Robotics entwickelt. Im folgenden Kapitel wird hierzu der Greifer des linken Arms des Roboters genutzt. Am rechten Arm ist ein pneumatisch betriebener Saugkopf montiert, der für die Bereitstellung von Dokumenten genutzt wird.

Wie in Abbildung 14 zu sehen, wurde der Tisch für die durchgeführten Versuche zentriert vor Baxter aufgestellt. Dabei wurden die Tischbeine direkt hinter den Standfüßen des Roboters platziert. Durch diese Positionierung kann Baxter mit beiden Armen Großteile der Tischfläche erreichen.

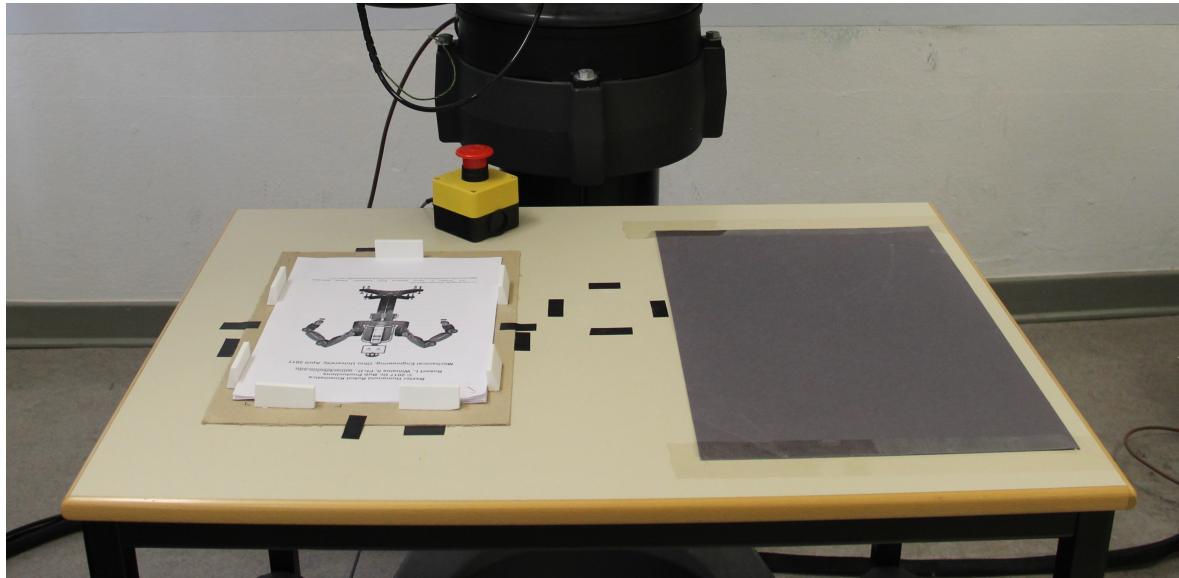
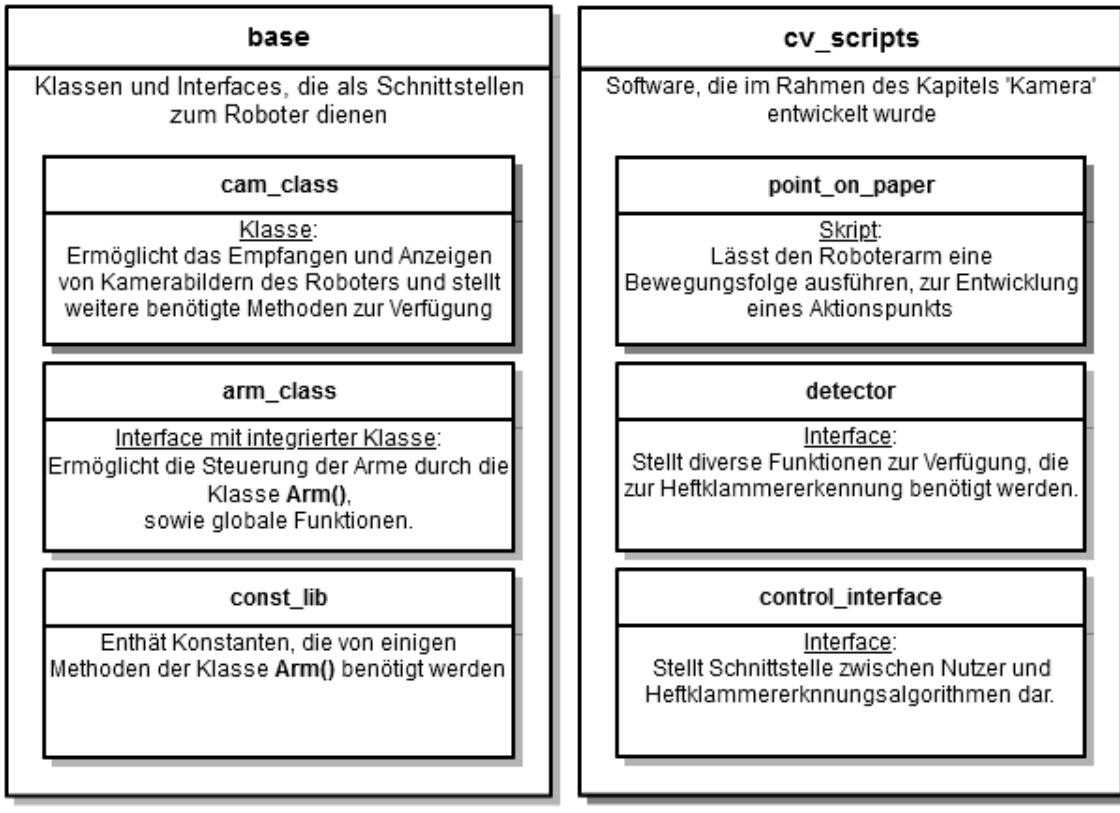


Abbildung 14: Der Versuchstischs

Abbildung 14 zeigt den Versuchsaufbau auf dem Tisch. Die linke Tischhälfte dient der Bereitstellung von Dokumenten, während die rechte Seite für die Heftklammererkennung und -markierung genutzt wird. Wie im Bild zu sehen, wird eine dunkle Unterlage als Ablagefläche für die Dokumente genutzt. Die Maße dieser Ablagefläche entsprechen dabei der Norm DIN A3. Sie dient der Kontrasterhöhung für die angewandten Objekterkennungsalgorithmen. Die Dokumentenbereitstellung, auf der linken Tischhälfte, liegt als Kartonunterlage mit applizierten Zentrierbacken vor und wurde bereits in dem, dieser Arbeit vorangegangenen, Praxisprojekt erarbeitet. Die jeweiligen Unterlagen wurden mit Klebeband auf dem Tisch fixiert, um ihre Position zu sichern.

3.2 Architektur

Wie in Unterkapitel 2.1 genannt, wurde die, im Zuge dieser Arbeit entwickelte, Software in Form eines ROS-Package zusammengefasst. Die dort erläuterte Komponente `precision` findet in diesem Kapitel keine Anwendung.



(a) Die Basis-Komponente

(b) Die Kamerakomponente

Abbildung 15: Hier verwendete Komponenten des Pakets `baxter_staples`

Die, in Abbildung 15a aufgeführte, Komponente `base` wurde teilweise bereits in Unterkapitel 2.1 behandelt. Die in `arm_class` enthaltenen Klasse `Arm`, wurde in Kapitel 2 um die Methode `move_precise` erweitert. Für das folgende Kapitel wurde die Klasse zusätzlich um ein Objekt der Klasse `Cam` aus `cam_class` erweitert, sodass die Nutzung einer Handkamera über den Roboterarm gesteuert werden kann, an dem sie befestigt ist. `cam_class` importiert die Packages `cv2` und `baxter_interface`, sowie `cv_bridge`. `baxter_interface` stellt hierbei einen Controller für die Kameras zur Verfügung, der benötigt wird, um die Kameras zu steuern. `cv_bridge` wird zur Formatumwandlung der, über ROS empfangenen, Bildnachrichten in, von OpenCV verwendbare, Bilddaten genutzt. Das Package `cv2` wird weiterhin von allen Bestandteilen der `cv_scripts`-Komponente genutzt. Es stellt diverse Funktionen zur Bildbearbeitung, sowie verschiedenste Objekterkennungsalgorithmen zur Verfügung. Da `cam_class` über `arm_class` verfügbar ist, importieren `point_on_paper` und `control_interface` nur `arm_class`. Die in `detector` enthaltenen Funktionen stehen in keinem direkten Bezug zum Roboter, weshalb hierfür keine Bestandteile der `base`-Komponente benötigt werden.

3.3 Die Klasse `cam_class`

3.3.1 Anzeigen der Kamerabilder

Die im folgenden Unterkapitel vorgestellte Klasse ist in Anhang 5 zu finden. In Unterkapitel 1.2 wurde bereits erwähnt, dass Baxter, von ihm aufgenommene, Informationen, wie Kamerabilder, über ROS-Topics veröffentlicht. Ein Objekt der Klasse `cam_class` abonniert die, für die Kamera des verwendeten Arms zuständige, Übertragung und kann so die Bilder empfangen. Die empfangenen Bilder werden durch Nutzung der ROS-Bibliothek CvBridge in ein von OpenCV verwendbares Format umgewandelt und angezeigt. Hierbei erschafft die Klasse, auf dem Bildschirm des Computers, drei Fenster zum Anzeigen von Bildern, wie in Abbildung 16 zu sehen.

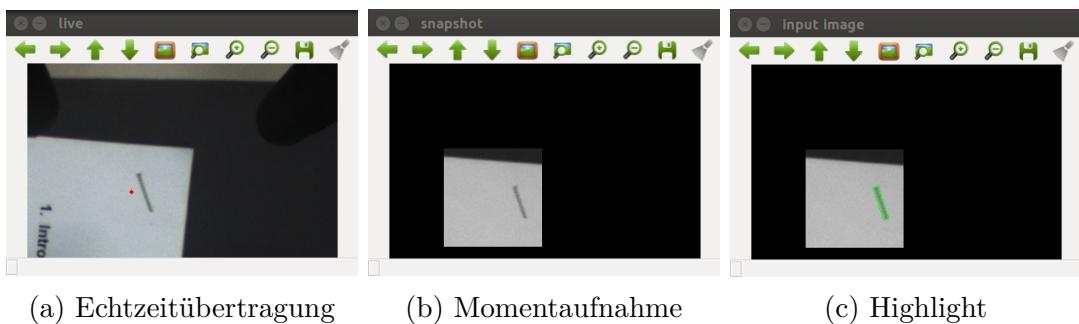


Abbildung 16: Durch `cam_class` angezeigte Bilder

Das erste Fenster, zu sehen in Abbildung 16a, zeigt die Echtzeitübertragung des Kamerabildes. In dessen Frames wird ein roter Punkt eingezeichnet, welcher den Aktionspunkt markiert. Die, zur Berechnung dieses Punkts, verwendeten Formeln wurden, mittels der in Unterkapitel 3.3.2 erläuterten, Software erstellt. Das zweite Fenster dient der Darstellung von Momentaufnahmen der Echtzeitübertragung, zu sehen in Abbildung 16b. Diese Momentaufnahmen werden an verschiedenen Punkten des Programmablaufs festgehalten und für Bildbearbeitung und Objekterkennung genutzt. Das dritte Fenster, zu sehen in Abbildung 16c, wird genutzt um verschiedene Schritte der Bildbearbeitung und Objekterkennung darzustellen, die im Folgenden als Highlights bezeichnet werden. Diese dienen vor allem der Überwachung der Objekterkennungsalgorithmen.

3.3.2 Der Aktionspunkt

Der Aktionspunkt erfüllt mehrere Funktionen. Die Offenkundigste besteht darin, dem Nutzer zu verdeutlichen, an welcher Stelle der Greifer auf das Papier treffen wird, wenn dieser eine gerade Bewegung auf den Tisch zu ausführt. Eine weitere Funktion ist das Begrenzen des Kamerabildes, auf einen Bereich von Interesse (ROI), um den Aktionspunkt herum, was durch Aufruf einer Funktion des `detector`-Interface möglich

ist. Weiterhin wird der Aktionspunkt genutzt, um den zurückzulegenden Weg zu einem Punkt im Bild zu berechnen.

Die Reihen- und Spalten-Koordinate des Aktionspunkts im Kamerabild werden mittels polynomischer Formeln dritten Grades in Abhängigkeit der aktuellen Z-Koordinate der Greifergrundplatte berechnet. Um die Korrektheit des Aktionspunkts sicherzustellen, ist es notwendig, dass der Greifer absolut vertikal ausgerichtet ist. Zur Entwicklung der benötigten Formel, wurde das Skript aus Anhang J - `point_on_paper.py` und ein Greiferadapter genutzt, welcher dem Roboter das Halten eines Filzstifts ermöglicht und in Unterkapitel 3.1 zu sehen ist.

Bei Ausführung des Skripts, senkt sich der Roboterarm auf den Tisch herab und markiert einen Punkt auf einem bereitliegenden Dokument. Im Anschluss entfernt sich der Greifer schrittweise in einer geraden senkrechten Bewegung vom Dokument. Jeder Schritt entspricht dabei einer Länge von zwei Zentimetern. Mit jedem Schritt wird die Position der Markierung im übertragenen Kamerabild ausgelesen. Die gesammelten Positionsdaten wurden anschließend in Diagrammen zusammengefasst und Trendlinien darübergelegt, wie in Abbildung 17 zu sehen. Hierbei wurden polynomische Trendlinien dritten Grades gewählt, da sie mit einem Bestimmtheitsmaß von

$$R^2 = 0.949$$

für die Spaltenkoordinaten bzw.

$$R^2 = 0.995$$

für die Reihenkoordinaten das nichtlineare Verhalten der zu sehenden Kurven ausreichend genau darstellen sollten. Die, den Abbildungen entnehmbaren, beschreibenden Formeln dieser Trendlinien werden zur Berechnung des Aktionspunkts verwendet.

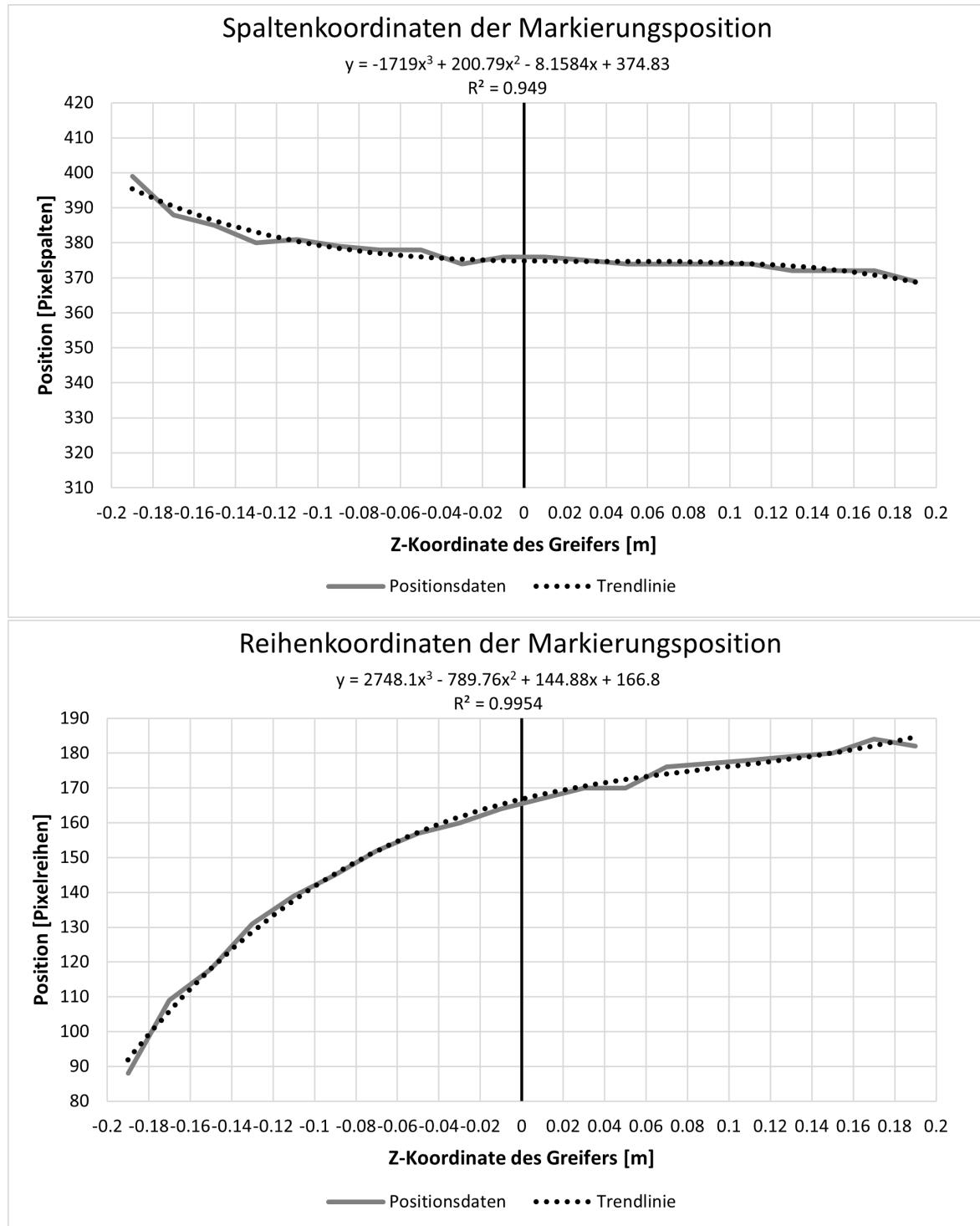


Abbildung 17: Aktionspunktkoordinaten in Abhängigkeit von der Z-Koordinate des Greifers

3.3.3 Kameragestützte Steuerung des Roboterarms

Wie zu Beginn des vorigen Unterkapitels 3.3.2 erwähnt, wird der Aktionspunkt für die kameragestützte Steuerung des Roboterarms genutzt. Hierbei dient er als Anhaltspunkt für die Berechnung der Distanz zwischen dem Greifer und einem Punkt im Bild. Die Distanz wird als X- und Y-Komponente berechnet. Diese Komponenten werden mittels eines, vom Z-Koordinatenwert des Greifers abhängigen, Umrechnungsfaktor von Pixel in Meter übertragen. Die aktuelle Pose des Greifers kann mit den so erhaltenen Distanzen modifiziert werden, um den gewünschten Punkt zu erreichen.

Um die Formel zur Berechnung des Umrechnungsfaktors, in Abhängigkeit des Z-Koordinatenwerts des Greifers, zu ermitteln, wurden, mit der Handkamera von Baxter, Bilder eines DIN A4 Dokuments aus verschiedenen Höhen gemacht. Anschließend wurden die Kantenlängen des Dokuments in den Aufnahmen ausgelesen und mit den, der Norm DIN A4 entnehmbaren, realen Kantenlängen jeweils zu einem Faktor verrechnet. Anhand dieser Faktoren wurde die im Folgenden aufgeführte Formel entwickelt.

$$f = -9530.9 \frac{m}{Pixel} * Z + 1949.7 \frac{m}{Pixel}$$

3.4 Heftklammererkennung

Um Heftklammern aus einem Dokument entfernen zu können, muss der Roboter diese zunächst als solche erkennen. Wie in Unterkapitel 1.4 aufgezählt, stellt OpenCV viele Möglichkeiten der Bildbearbeitung und Objekterkennung zur Verfügung. Einige Objekterkennungsalgorithmen nutzen maschinelles Lernen zur Objekterkennung. Ein Beispiel hierfür ist die Haar-Feature Kaskadenklassifizierung.[1] Wie der Name bereits vermuten lässt, lernt hierbei eine Maschine, anhand von positiven und negativen Vergleichsbildern, ein Objekt, unter Einfluss verschiedener Faktoren, auf Bildern zu erkennen. Sowohl das Generieren von Vergleichsbildern, als auch der Lernprozess, den eine Maschine durchlaufen muss, bedeuten einen zu großen Arbeits- und Zeitaufwand für den Rahmen dieser Arbeit. Aus diesem Grund wurde ein anderer Ansatz verfolgt, bei dem jeweils nur ein Vergleichsobjekt, je Situation und zu erkennendem Objekt, genutzt wird.



Abbildung 18: Nahaufnahme einer applizierten Heftklammer

Abbildung 18 zeigt den Ausschnitt einer, von einer Handkamera Baxters, erzeugten Aufnahme. Sie zeigt eine, in einem gebleichten Dokument applizierte, Heftklammer aus Stahl, des Typs 3 nach DIN 7405. An dem Bild lässt sich erkennen, dass eine applizierte Heftklammer nur wenige Features aufweist, die sie als einzigartig in einem Bild ausweisen können. Sie besitzt weder eine besondere Farbe, die sie explizit vom Hintergrund abhebt, noch eine einzigartige Form. Somit lässt sich nur ein dunkleres längliches Rechteck auf hellerem Grund erkennen. Aufgrund dieser eingeschränkten Anzahl an Features wurde entschieden, die Kontur der Klammer als Grundlage für die folgende Objekterkennung zu nutzen.

3.4.1 Konturenfunktionen von OpenCV

Zum Finden von Konturen in einem Bild, stellt OpenCV die Funktion `findContours` zur Verfügung. Dabei kann eine Kontur als Kurve beschrieben werden, die alle zusammenhängenden Punkte, mit gleicher Farbe oder Farbintensität, entlang einer Kante verbindet.[17]

Um die Genauigkeit von `findContours` zu erhöhen, sollte ein binäres Bild übergeben werden.[17] Um das erhaltene Kamerabild in ein binäres Bild umzuwandeln, wird die, durch OpenCV zur Verfügung gestellte, Funktion `Canny` genutzt. Diese wendet den, nach seinem Entwickler John F. Canny benannten, Algorithmus der Canny Kanten detektion an.[16] Dieser Algorithmus besteht aus vier nacheinander ausgeführten Schritten.

1. Filtern des Bildes

Um das Bildrauschen zu verringern, wird ein Gauß-Filter der Größe 5x5 Pixel angewandt, der Elemente mit hoher Frequenz, also Rauschen und Kanten, aus dem Bild entfernt.[16] Hierzu fährt der Filter über das Bild und verändert den Wert des zentralen Pixels der 5x5-Matrix nach dem Prinzip der Faltung.[21]

2. Intensitätsgradienten des Bildes finden

Auf das geglättete Bild werden Sobel-Filter in vertikaler und horizontaler Richtung angewandt, um, durch Helligkeitsübergänge, Kanten in beide Richtungen zu finden. Die Sobel-Filter fahren reihen- bzw. spaltenweise mit einer 3x3-Matrix das Bild ab.[16] Jeder Filter bildet Ableitungen ersten Grades des Bildes in X- und Y-Richtung, anhand derer Helligkeitsübergänge erkannt werden. Außerdem werden deren Stärke und Ausrichtung erkannt.[22] Das Bild liegt am Ende als Beschreibung der Helligkeitsübergänge zwischen den Pixeln vor.

3. Finden lokaler Maxima

Um aus den Helligkeitsübergängen Kanten zu erkennen, wird jeder Pixel des Bildes darauf überprüft, ob er ein lokales Maximum entlang der Gradientenausrichtung darstellt. Der Vorgang wird anhand von Abbildung 19 erläutert.

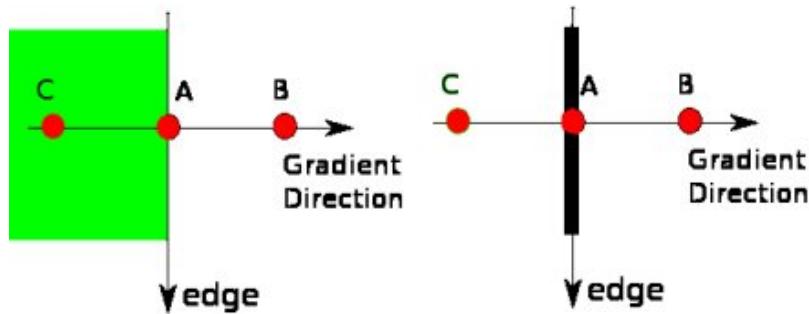


Abbildung 19: Überprüfen eines Punktes auf lokales Maximum nach Sobel[16]

Punkt A liegt auf einer Kante, während die Gradientenausrichtung senkrecht zu dieser vorliegt. Da B und C die nächsten Punkte zu A entlang der Gradientenrichtung darstellen, werden die Werte der drei Punkte verglichen. Wenn A ein lokales Maximum darstellt, wird der Punkt als potenzielle Kante beibehalten. Ansonsten wird sein Wert auf Null gesetzt.[16] Durch diesen Schritt wird das Bild auf potenzielle schmale Kanten reduziert.

4. Grenzfilterung der potenziellen Kanten

Der letzte Schritt dient der Feststellung, ob bisher als potenziell eingestufte Kanten eine sichere Kante darstellen. Hierzu werden diese mit einer Unter- und Obergrenze verglichen. Punkte unterhalb der Untergrenze werden als 'Nicht-Kante' eingestuft, während Punkte oberhalb der Obergrenze als 'sichere Kante' eingestuft werden. Liegt der Wert eines Punktes zwischen den Grenzen, muss er mit einem Punkt verbunden sein, der als 'sichere Kante' eingestuft wurde, da er sonst als 'Nicht-Kante' auf Null gesetzt wird.[16] Anhand von Abbildung 20 wird dazu ein Beispiel aufgeführt.

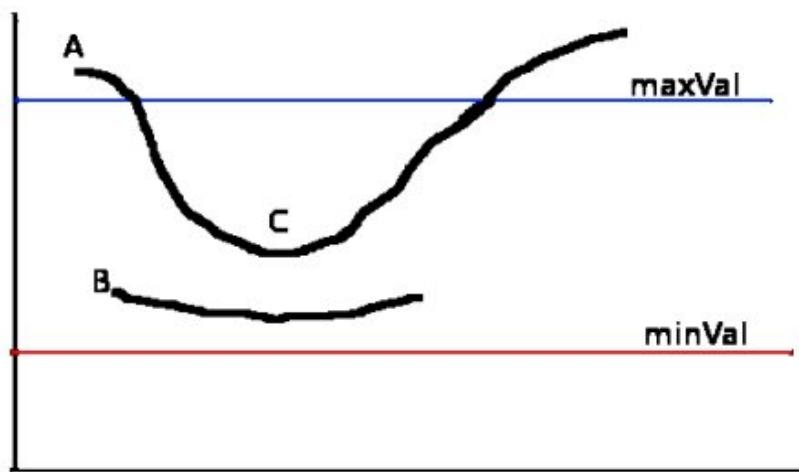


Abbildung 20: Beispielbild für Grenzfilterung potenzieller Kanten[16]

Punkt A liegt oberhalb der Obergrenze 'maxVal' und wird somit als 'sichere Kante' eingestuft, bzw. auf 1 gesetzt. Punkt C liegt zwar zwischen den Grenzen, ist jedoch Teil der gleichen Kurve wie A und wird somit als sicher eingestuft. Punkt B liegt zwischen den Grenzen und da kein Punkt seiner Kurve über der Obergrenze liegt, wird dieser als 'Nicht-Kante' auf Null gesetzt.[16]

Insgesamt wandelt die Funktion **Canny** also ein normales in ein binäres Bild um, welches nur die starken Kanten des Ausgangsbilds enthält. Das Ergebnis dieser Umwandlung ist in Abbildung 21 zu sehen.

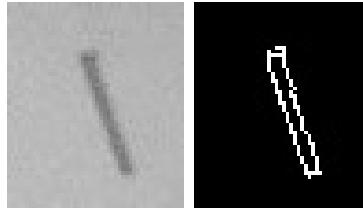


Abbildung 21: Heftklammer vor und nach Anwendung des Canny Algorithmus

Wie zu Beginn erwähnt, wird die von OpenCV bereitgestellte Funktion **findContours** zur Heftklammererkennung genutzt. Hierfür bekommt sie ein, durch **Canny** vorbereitetes, binäres Bild, welches auf die darin vorkommenden Kanten reduziert wurde. Sie gibt alle gefundenen Konturen, sowie deren Hierarchie zurück.[17] Die Hierarchie der gefundenen Konturen wird im gewählten Lösungsansatz nicht verwendet und deshalb hier nicht näher erläutert.

Der Funktion **findContours** liegt ein Algorithmus zur Kantenverfolgung zu Grunde, dessen Ablauf im folgenden Absatz aufgeführt und erläutert wird.

Bei der Kantenverfolgung wird grundlegend ein Rasterscan des Bildes durchgeführt. Dieser durchsucht das Bild nach einem Pixel, welches sich als Kantenverfolgungsstartpunkt eignet. Das bedeutet, dass das Bild reihenweise von oben nach unten und innerhalb der Reihen von links nach rechts pixelweise überprüft wird. Ein als Startpunkt geeignetes Pixel ist gefunden, wenn das aktuelle Pixel einen Wert von 1 oder höher besitzt und a) das linke Pixel einen Wert von 0 hat, oder b) das rechte Pixel einen Wert von 0 hat. Liegt Fall a) vor, wurde eine Außenkante gefunden. Fall b) markiert eine Innenkante, also gewissermaßen die Kante zu einem Loch im Objekt. Wenn eine Kante nur einen Pixel breit ist, können beide Fälle zutreffen. Das Pixel wird dann nach Fall a) behandelt und die gefundene Kante als Außenkante markiert. Sobald ein geeignetes Pixel gefunden wurde, beginnt der eigentliche Kantenverfolgungsalgorithmus. Die acht anliegenden Pixel werden im Uhrzeigersinn nach einem 'Nicht-Null-Pixel' durchsucht, im folgenden NNP1 genannt. Bei einer Außenkante startet diese Suche am linken Pixel, des aktuell Betrachteten. Bei einer Innenkante startet die Suche am Pixel rechts des aktuell Betrachteten. Wird kein Pixel, mit einem Wert ungleich null gefunden, wird der

Kantenverfolgungsalgorithmus beendet und der Rasterscan fortgeführt. Ansonsten werden die acht anliegenden Pixel gegen den Uhrzeigersinn, beginnend bei NNP1, nach einem weiteren 'Nicht-Null-Pixel', im Folgenden NNP2 genannt, durchsucht. NNP1 und NNP2 werden als Teil der Kante markiert. Entspricht NNP1 dem Startpunkt der Kante, wird die Kantenverfolgung beendet und der Rasterscan fortgeführt. Ansonsten werden die genannten Schritte der Kantenverfolgung mit NNP1 als zentralem Pixel wiederholt.[31, S. 42f]

In Abbildung 22 sind die gefundenen Konturen rot eingezeichnet, sowohl im Ausgangs- als auch im, vom Canny-Algorithmus umgewandelten, Canny-Bild. 22a zeigt Ausgangs- und Canny-Bild, 22b die Außenkontur und 22c die gefundene Innenkontur. Anhand der Canny-Bilder in 22b und 22c lässt sich erkennen, dass einige Pixel Teil beider Konturen sind, während andere Pixel jeweils nur Teil einer Kontur sind. Das zeigt, dass die Breite, der durch die Funktion **Canny** erzeugten Kanten, stellenweise mehr als einen Pixel betragen kann. Weiterhin lässt sich erkennen, dass keine der gefundenen Konturen ein einfaches längliches Rechteck darstellen, wie erwartet. Besonders die Außenkontur weicht hiervon, durch eine hakenförmige Erweiterung am oberen Ende, ab. Die Innenkontur weist abgerundete Enden auf.

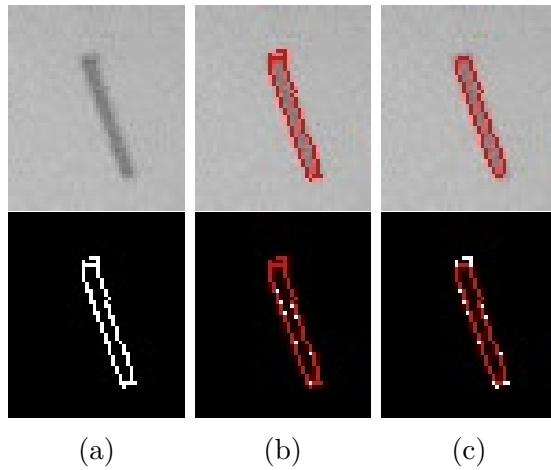


Abbildung 22: Im Beispielbild gefundene Konturen

Wie in Unterkapitel 3.4 genannt, nutzt der verfolgte Ansatz die Kontur der Heftklammer als Erkennungsmerkmal. Das Erkennen eines Objektes, auf Grundlage der Kontur, geschieht durch das Vergleichen zweier Konturen, oder Bilder, die diese Konturen darstellen, im folgenden Masken genannt. Um zwei Konturen, oder Masken, zu vergleichen, stellt OpenCV die Funktion **matchShapes** zur Verfügung. Diese berechnet die Abweichung zweier übergebener Objekte voneinander, basierend auf Hu-Momenten.[18] Hu-Momente sind eine Menge aus sieben Werten, die, anhand zentraler Momente eines Bildes, berechnet werden. Alle sieben Momente sind invariant gegenüber Translation, Skalierung und Rotation. Die ersten sechs zentralen Momente sind zusätzlich invariant

gegenüber Spiegelung, während das Vorzeichen des siebten Moments bei auftretender Spiegelung negativ ist. Ein Bildmoment entspricht hierbei einem gewichteten Mittelwert aller Helligkeitswerte eines Bildes.[10]

`matchShapes` nutzt zur Berechnung der Abweichung die folgende Formel.[10]

$$D(A, B) = \sum_{i=0}^6 |H_i^B - H_i^A|$$

Der von `matchShapes` zurückgegebene Wert ist somit die Summe der absoluten Differenzen, der sieben Hu-Momente, beider übergebener Objekte. Dementsprechend bedeutet ein geringerer Rückgabewert eine höhere Übereinstimmung der Objekte.

3.4.2 Das detector-Interface

Um das Erkennen von Heftklammern zu ermöglichen, wurde das Interface in Anhang I - `detector.py` entwickelt, welches die in Unterkapitel 3.4.1 aufgeführten Funktionen nutzt.

Wie bereits im vorigen Unterkapitel erwähnt, 'erkennt' ein Computer ein Objekt nicht einfach, sondern vergleicht dieses mit einem Vergleichswert oder -objekt. In dieser Arbeit werden als Vergleichsobjekte die Konturen von Heftklammern genutzt, in Abhängigkeit der Pose des Greifers. Diese Vergleichsobjekte müssen auf dem Computer vorliegen, um bei Programmdurchlauf genutzt werden zu können. Die Funktion `matchShapes` erwartet Konturen, oder bildliche Darstellungen dieser. Um Konturen, die in OpenCV als Vektoren von Punkten vorliegen, effektiv speichern zu können, wurden aus diesen Vergleichsmasken erschaffen. Abbildung 23 zeigt zwei beispielhafte Vergleichsmasken, die aus den zuvor in Abbildung 22 gezeigten Konturen erschaffen wurden. Das Erschaffen der Vergleichsmasken wird durch die Interface-Funktion `create_box_mask` ermöglicht.

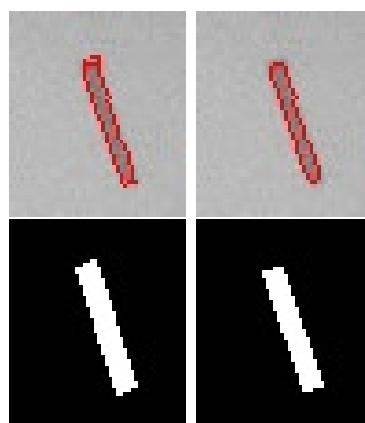


Abbildung 23: Beispielhafte Vergleichsmasken

Im Vergleich lässt sich erkennen, dass die auf den Masken abgebildeten Formen längliche Rechtecke darstellen und nur von den Maßen her den ursprünglichen Konturen

entsprechen. Diese Form wurde gewählt, da die durch `findContours` gefundenen Konturen einer Heftklammer, je nach Distanz zwischen Kamera und Dokument, stark vom Umgebungslicht, und den dadurch entstehenden Schatten, beeinflusst werden können. Durch die Umwandlung in ein, der Kontur möglichst entsprechendes, Rechteck wird der Einfluss durch unregelmäßig auftretende Schatten verringert.

Zum Finden von Heftklammern in einem Bild bietet das `detector`-Interface die Funktion `detect_staple`. Der Ablauf der Funktion ist abhängig von den Maßen des übergebenen Bildes. Wie später in Unterkapitel 3.5 näher erläutert wird, besteht der Prozess der Heftklammererkennung aus zwei Schritten. Beim ersten Schritt wird das gesamte Dokument in einem Bild erfasst und nach potenziellen Heftklammern gescannt. Die dabei genutzte Momentaufnahme der Kameraübertragung liegt mit einer Auflösung von 640x400 Pixeln vor. Der zweite Schritt dient der genauen Überprüfung der potenziellen Heftklammern und liefert ein Bild der Auflösung 320x200 Pixel. Die Funktion `detect_staple` erkennt automatisch die Auflösung des übergebenen Bildes. Entsprechend dieser setzt sie verschiedene Vergleichsparameter, sowie die Vergleichsmaske, und ruft die dem Fall entsprechende Vergleichsfunktionen auf.

Liegt ein Bild der Auflösung 640x400 Pixel vor, wird die Funktion `find_matches` genutzt. Diese vergleicht alle im Bild gefundenen Konturen mit der übergebenen Vergleichsmaske, sowie Grenzwerten für die Größe der gefundenen Kontur. Alle Konturen, deren Abweichung von der Maske maximal 0.15 beträgt, werden in einer Liste gespeichert. Diese Liste wird aufsteigend nach Abweichungswert sortiert und zurückgegeben. Der Wert 0.15 entspricht hierbei keiner SI-Einheit und wurde empirisch als geeigneter Grenzwert ermittelt.

Liegt ein Bild der Auflösung 320x200 Pixel vor, wird die Funktion `find_match` genutzt. Diese vergleicht alle im Bild gefundenen Konturen mit der übergebenen Vergleichsmaske und den Grenzwerten für die Maximalgröße der Kontur. Hier wird jedoch nur die Kontur mit der geringsten Abweichung von der Vergleichsmaske zurückgegeben.

Die genannten Vergleichsfunktionen wurden nach einem möglichst dynamischen Konzept entwickelt, um auch für andere Objekte, als Heftklammern, eingesetzt werden zu können. `detect_staple` gibt schließlich die, von der Vergleichsfunktion erhaltenen, Daten, zusammen mit einer Erfolgsmeldung, zurück. Diese Erfolgsmeldung sagt aus, ob im Bild, den Parametern entsprechende, Konturen gefunden wurden, oder nicht.

Neben dem Finden von potenziellen Heftklammerkonturen in Bildern, bietet das Interface die Möglichkeit ein Dokument auf einem übergebenen Bild zu finden. Um den korrekten Ablauf, der hierfür genutzten Funktion `detect_paper`, zu gewährleisten, muss der darauf abgebildete Arbeitsbereich dem, in Unterkapitel 3.1 aufgeführten, Versuchsaufbau entsprechen. Die Funktion erwartet ein Bild der Auflösung 640x400 und blendet auf diesem zunächst den Hintergrund aus. Anschließend wird die Funktion

`find_match` genutzt, um das Dokument, anhand einer geeigneten Vergleichsmaske, zu finden. Wenn ein Dokument gefunden wurde, wird dessen Mitte maskiert, sodass nur noch der Randbereich erhalten bleibt. Die Maskierung geschieht in Abhängigkeit der Orientierung der gefundenen Dokumentenkontur. Abbildung 24a zeigt ein für die Nutzung von `detect_paper` geeignetes Bild. Abbildung 24b zeigt das durch `detect_paper` bearbeitete Bild. Durch die Maskierung der Dokumentenmitte wird das Bild des Dokuments auf den Bereich beschränkt, in dem Heftklammern, in der Regel, zu finden sind.

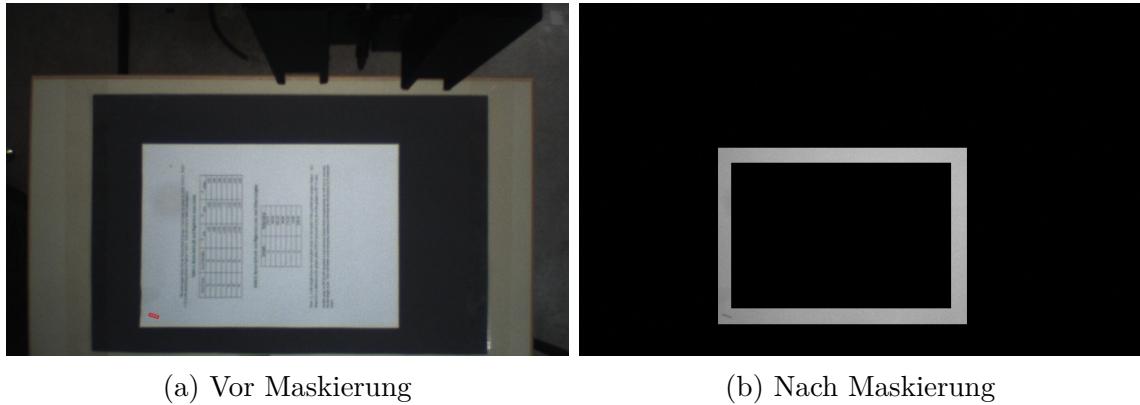


Abbildung 24: Dokument auf Arbeitsplatte vor und nach `detect_paper`

Wie zuvor erwähnt, besteht der Heftklammererkennungsprozess aus zwei Schritten, wobei der zweite Schritt der genaueren Überprüfung gefundener potenzieller Heftklammern dient. Wie bei der Maskierung der Dokumentenschritte im vorigen Absatz, sollte auch hier das Bild auf seine ROI beschränkt werden. Die Funktion `mask_window` maskiert das übergebene Bild, sodass nur ein Feld von 120x70 Pixeln um den Aktionspunkt herum sichtbar bleibt.

3.5 Nutzerschnittstelle

Das im Folgenden vorgestellte Interface stellt gewissermaßen das Endprodukt dieser Arbeit dar, da die Ergebnisse der bisherigen Kapitel darin enthalten sind und angewandt werden. Es muss jedoch darauf hingewiesen werden, dass die in Kapitel 2 entworfene LUT hier keine Anwendung findet. Dieser Umstand begründet sich darauf, dass die Funktion zur Anwendung der LUT ursprünglich einen Fehler enthielt. Durch diesen Fehler führten die Ergebnisse der Abschlussmessungen in Kapitel 2 zu dem Schluss, die LUT im weiteren Projektverlauf nicht einzusetzen. Die Existenz des Fehlers wurde erst im weit fortgeschrittenen Projektverlauf bemerkt, sodass die Abschlussmessungen wiederholt und neu ausgewertet werden konnten, jedoch die Zeit fehlte, um anschließend Änderungen an der, in diesem Unterkapitel vorgestellten, Nutzerschnittstelle vorzunehmen und abschließend zu verifizieren.

Das Interface kann in Anhang K - `control_interface.py` eingesehen werden.

Das Kontroll-Interface ermöglicht dem Nutzer die Interaktion mit Baxter über die Systemkonsole des verbundenen Computers. Hierbei stehen dem Nutzer verschiedene Interaktionsmöglichkeiten zur Verfügung. Die Befehle `cam` und `arm` öffnen Untermenüs, über die Kameraparameter verändert, oder der linke Arm des Roboters gesteuert werden können. Weiterhin stehen die Befehle `find`, `run` und `quit` zur Verfügung. `quit` setzt alle Kameraparameter zurück, verfährt den Arm in eine neutrale Pose und schaltet ihn ab. `find` startet den Heftklammersuchalgorithmus, durch Aufruf der Funktion `find_staple`, während `run` die Funktion `full_run` aufruft. Diese aktiviert den rechten Arm, lässt ihn ein Dokument von der Dokumentenbereitstellung auf den Arbeitsbereich der Heftklammererkennung transportieren und ruft anschließend die Funktion `find_staple` auf.

Die Funktion `find_staple` bewegt den linken Arm in eine Pose, von der aus der gesamte, in Unterkapitel 3.1 beschriebene, Arbeitsbereich der Heftklammererkennung von der Handkamera eingefangen werden kann. Sobald der Greifer diese Übersichtspose eingenommen hat, wird eine Momentaufnahme der Echtzeitübertragung gespeichert. Eine beispielhafte Momentaufnahme aus der Übersichtspose ist in Abbildung 25 zu sehen.

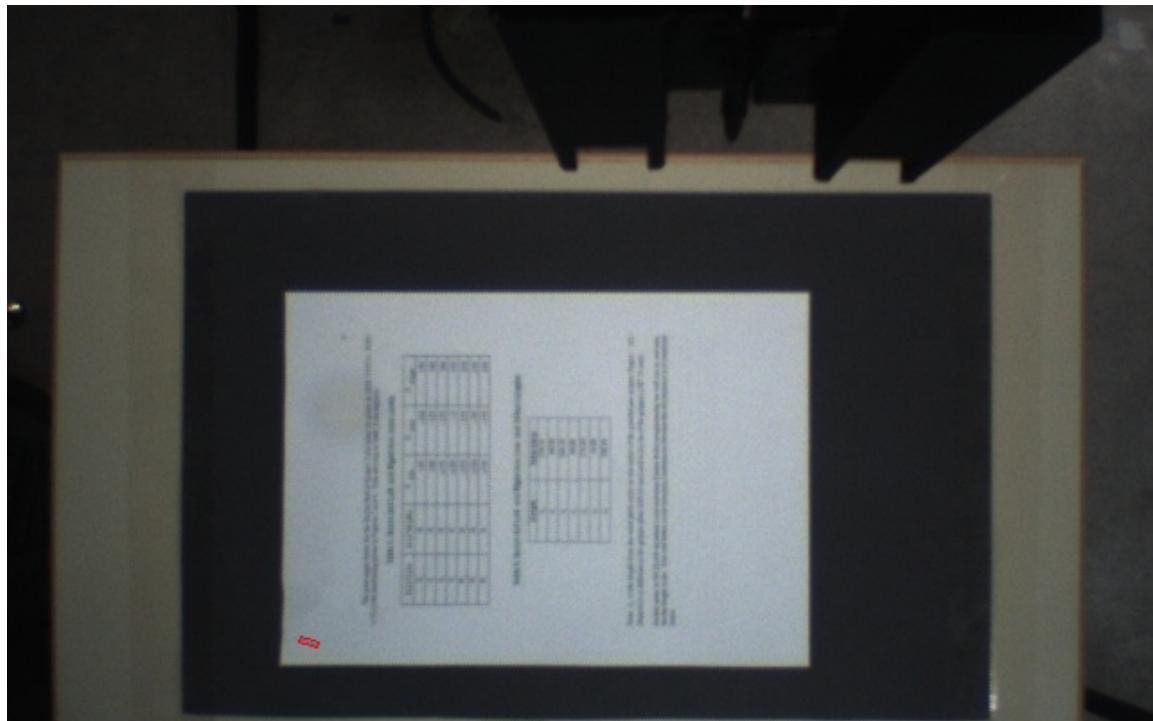


Abbildung 25: Momentaufnahme aus der Übersichtspose

Anhand dieser Momentaufnahme wird der erste Schritt der Heftklammererkennung durchgeführt. Hierbei wird die, in Unterkapitel 3.4.2 erklärte, Funktion `detect_paper` des Interface `detector` genutzt, um ein Dokument in der Momentaufnahme zu finden und auf die entsprechende ROI zu beschränken. Das maskierte Bild wird, wie in Unterkapitel 3.3 erläutert, als Highlight auf dem Bildschirm angezeigt und an die Funktion `detect_staple` des Interface `detector` übergeben. Das übergebene Bild hat eine Auf-

lösung von 640x400, weshalb, wie in Unterkapitel 3.4.2 beschrieben, eine geordnete Liste aller potenziellen Heftklammern zurückgegeben wird. Abbildung 26 zeigt die im Ausgangsbild eingezeichneten potenziellen Heftklammern.

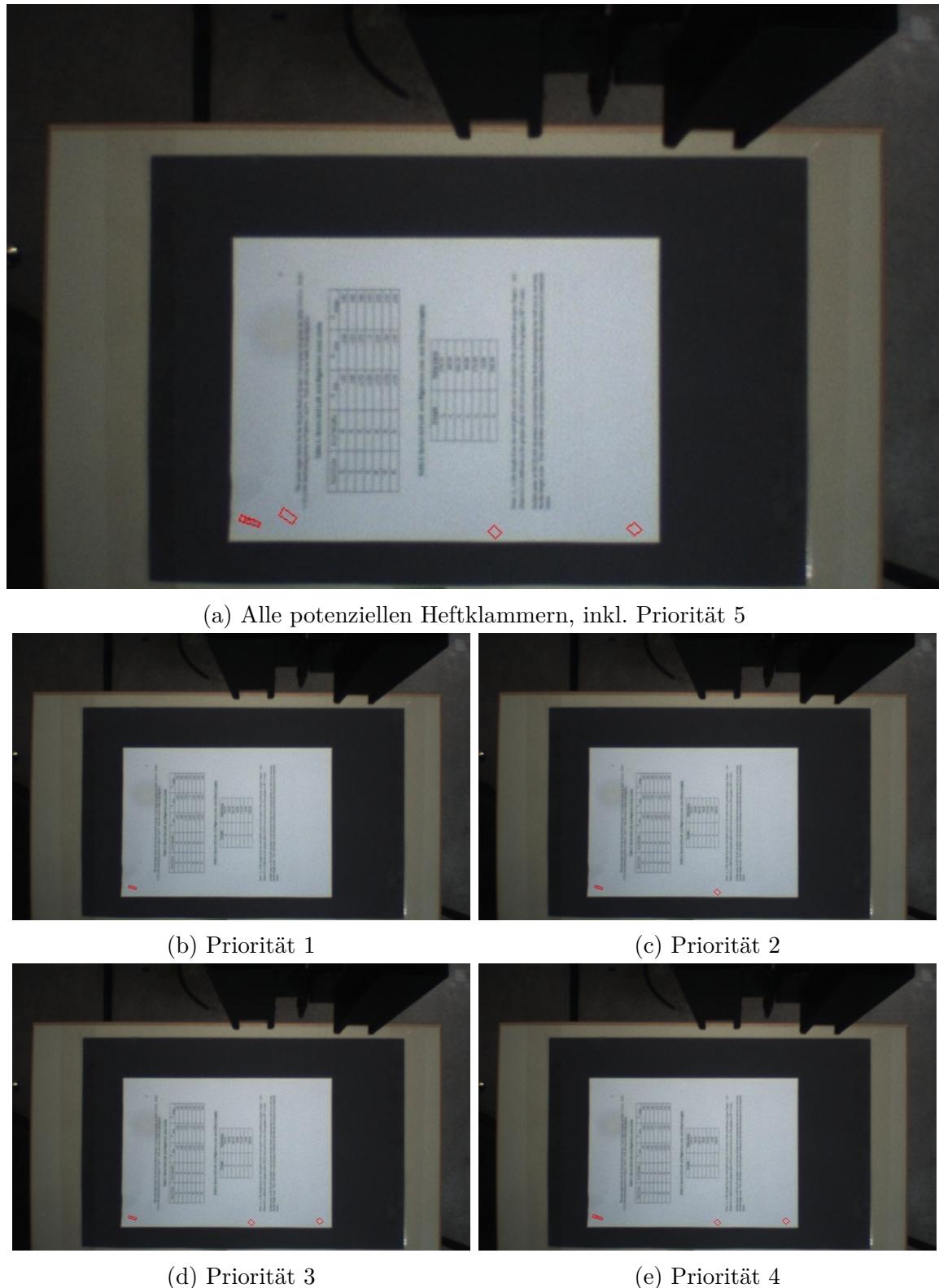
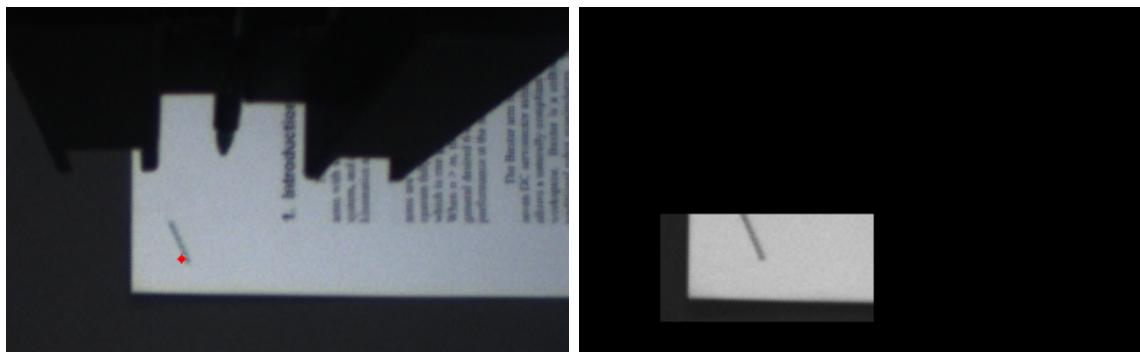


Abbildung 26: Momentaufnahme mit eingezeichneten potenziellen Heftklammern

Die in den Bildunterschriften der Abbildung genannten Prioritäten zeigen die Reihenfolge an, in der die potenziellen Heftklammern vorliegen. Hierbei steht ein niedriger Prioritätswert, entsprechend der Erläuterung in Unterkapitel 3.4.2, für eine geringere Abweichung, der gefundenen Kontur, von der Vergleichsmaske.

Die Liste der potenziellen Heftklammern wird an die, im Kontroll-Interface enthaltene, Funktion `approve_matches` übergeben, welche die Überprüfung der potenziellen Heftklammern, und damit den zweiten Schritt der Heftklammererkennung, durchführt. Die Notwendigkeit des zweiten Schrittes zeigt sich an der Anzahl der potenziellen Heftklammern. Zwar wurde in diesem Beispiel die tatsächliche Heftklammer mit der geringsten Abweichung eingestuft, jedoch ist dies nicht immer der Fall.

Um dem Nutzer einen Überblick über die vorläufigen Ergebnisse zu ermöglichen, werden die gefundenen Konturen, ihrer Reihenfolge in der Liste nach, nacheinander in das Ausgangsbild eingezeichnet und dieses, als Highlight, auf dem Bildschirm angezeigt. Mittels der Funktion `distance_to_point` werden die realen Positionen der potenziellen Heftklammern ermittelt und anschließend nacheinander entsprechend ihrer Priorität angefahren. Dabei werden die Distanz zwischen Greifer und Dokument, sowie die Auflösung des Kamerabilds verringert. Die Verringerung der Distanz bewirkt, dass die Kontur der Heftklammer genauer erkannt werden kann, was zu zuverlässigeren Ergebnissen der Heftklammererkennung führt. Die Verringerung der Auflösung dient der Effizienzsteigerung, da alle im Folgenden eingesetzten Algorithmen hierdurch auf kleinere Pixelmatrizen angewandt werden. Weiterhin wird der Ablauf der Funktion `detect_staple` anhand der Bildauflösung beeinflusst, wie in Unterkapitel 3.4.2 aufgeführt. Vor Aufruf der Funktion `detect_staple`, wird die Momentaufnahme der Echtzeitübertragung auf die ROI begrenzt, mittels der in Unterkapitel 3.4.2 beschriebenen Funktion `mask_window`.



(a) Normale Nahaufnahme

(b) Maskierte Nahaufnahme

Abbildung 27: Nahaufnahme der potenziellen Heftklammer

Abbildung 27a zeigt, dass im normalen Kamerabild Wörter, sowie die Konturen des Greiferaufapters, zu sehen sind. Da sich diese Bestandteile des Bildes vom hellen Hintergrund des Dokuments abheben, würden sie als Konturen gefunden werden und somit

in den Vergleichsalgorithmus mit einbezogen. Das erhöht die benötigte Rechenzeit und erschafft die Möglichkeit, andere Bildelemente als Heftklammer zu erkennen.

Abbildung 27b zeigt, dass die störenden Bildelemente durch die Maskierung ausgebendet werden und nur der Dokumentenrand, sowie die Heftklammer innerhalb der ROI enthalten sind. Hierbei fällt auf, dass die Heftklammer, aufgrund ihrer vertikalen Orientierung, den Rand der ROI berührt. Abbildung 28a zeigt, dass die Heftklammer dennoch als solche verifiziert werden konnte, was durch die, in Rot eingezeichnete, Kontur verdeutlicht wird.

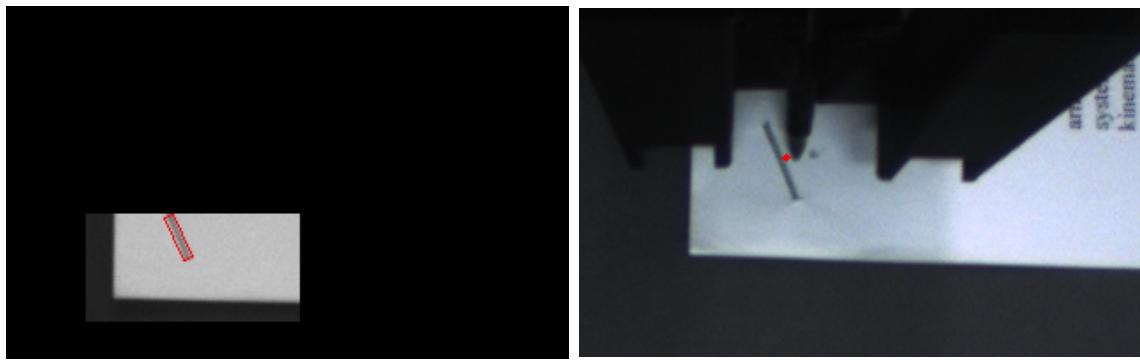


Abbildung 28: Finden und Markieren der Heftklammer

Abbildung 28b zeigt einen Ausschnitt der Echtzeitübertragung, während Baxter die gefundene Heftklammer mit einem Stift markiert. Um einem zukünftigen Entfernen der Heftklammern aus einem Dokument möglichst nah zu kommen, wurde sich, im Rahmen dieser Arbeit, dazu entschieden Baxter die Klammer mit einem Stift markieren zu lassen. Um eine gefundene Klammer zu markieren, ruft die aktuell durchlaufene Funktion `approve_matches` die Funktion `mark_staple` auf. Diese übergibt das, in Abbildung 28a zu sehende, Bild als Highlight an das übergeordnete `cam`-Objekt, wodurch dem Nutzer das erfolgreiche Verifizieren der potenziellen Heftklammer mitgeteilt wird. Anschließend berechnet sie die Distanz zwischen Aktionspunkt und Heftklammer und fährt diese an. Dabei werden zuerst die Bewegungen entlang der X- und Y-Koordinatenachse ausgeführt, bevor der Greifer sich auf die Heftklammer herabsenkt, um sie punktuell zu markieren. Sobald die Heftklammer markiert wurde, verfährt der Greifer vertikal, um den Abstand zum Dokument zu erhöhen. Das Erreichen der erhöhten Pose beendet den Heftklammererkennungsprozess. Wenn dieser durch den Aufruf der Funktion `full_run` gestartet wurde, verfährt der linke Arm zusätzlich in eine neutrale Pose.

3.6 Auswertung

Abbildung 29 zeigt drei beispielhafte Ergebnisse, bei denen erfolgreich Heftklammern auf Dokumenten erkannt wurden und der Roboter versucht hat diese zu markieren. Die Abbildungen 29a und 29b zeigen hierbei zwei nacheinander ausgeführte Erkennungsprozesse am gleichen Dokument.

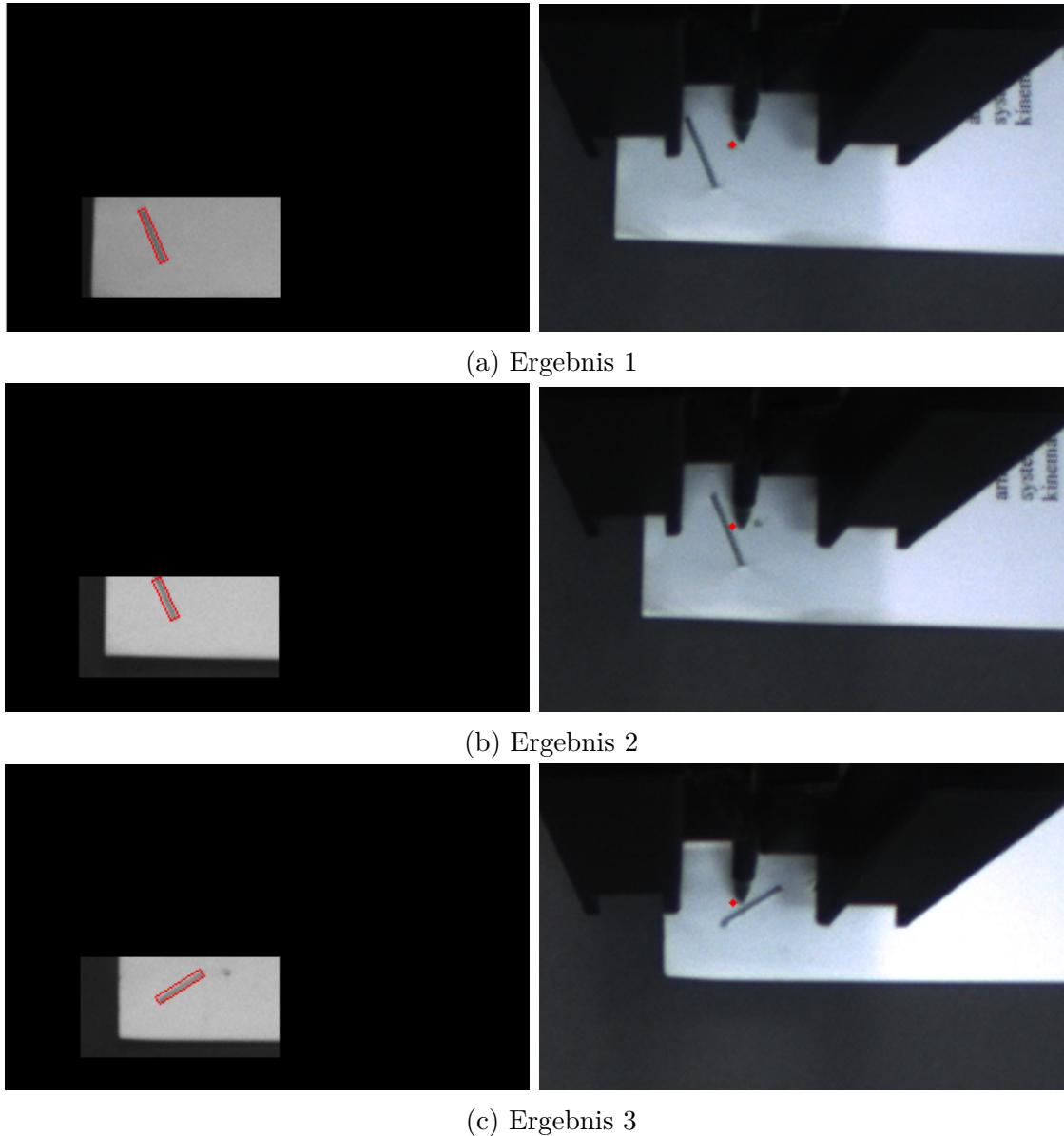


Abbildung 29: Markieren gefundener Heftklammern

Alle drei Abbildungen zeigen eine leichte Abweichung der Stiftspitze und des Aktionspunkts von der Heftklammer. Diese ist in 29a besonders stark ausgeprägt, während die anderen Abbildungen nur eine minimale Abweichung aufweisen. Ein möglicher Grund für das Auftreten der Abweichungen ist, dass die zuletzt ausgeführte Bewegung nicht durch die Funktion `move_precise` ausgeführt wird, sondern eine Bewegung entlang der Z-Koordinatenachse darstellt. Der Einsatz von `move_precise` ist beim Markieren der Klammer nicht möglich, da die hierbei ausgeführte Bewegungsfolge zuerst die vertikale

Distanz zwischen Stift und Dokument minimieren würde, bevor sie zwei horizontale Bewegungen ausführt. Stattdessen wurde als Markierung das Verursachen eines Punkts gewählt, ohne Ausführung horizontaler Bewegungen. Da das Dokument nur lose auf dem Tisch aufliegt und in keiner Form befestigt ist, würde eine horizontale Bewegung dieses auf der Unterlage verschieben und hierbei voraussichtlich das Dokument bemaßen.

Es wurde eine Vielzahl an Versuchsdurchläufen durchgeführt, bei denen Parameter, wie die Position und Orientierung des Dokuments und der darauf befestigten Heftklammer variiert wurden. Hierbei hat sich gezeigt, dass die Positionierung des Dokuments sich nicht entscheidend auf die Heftklammererkennung auswirkt, solange es sich komplett auf der Unterlage befindet. Die Orientierung des Dokuments hingegen hat einen Einfluss auf die Erfolgschance der Heftklammererkennung, da die Maskierung der Dokumentenmitte bei steigender Auslenkung schlechtere Ergebnisse liefert. Anhand Abbildung 30 lässt sich erkennen, dass die Orientierung der mittleren Maskierung von der des Dokuments abweicht. Ein Grund für dieses Verhalten konnte nicht gefunden werden und auch die Verwendung anderer Algorithmen, zur Berechnung der Dokumentenmaske, lieferten ähnliche Ergebnisse. Bei einer zu starken Auslenkung kann es somit vorkommen, dass die Dokumentenmaske eine applizierte Heftklammer verdeckt, wodurch eine Erkennung dieser nicht möglich ist.

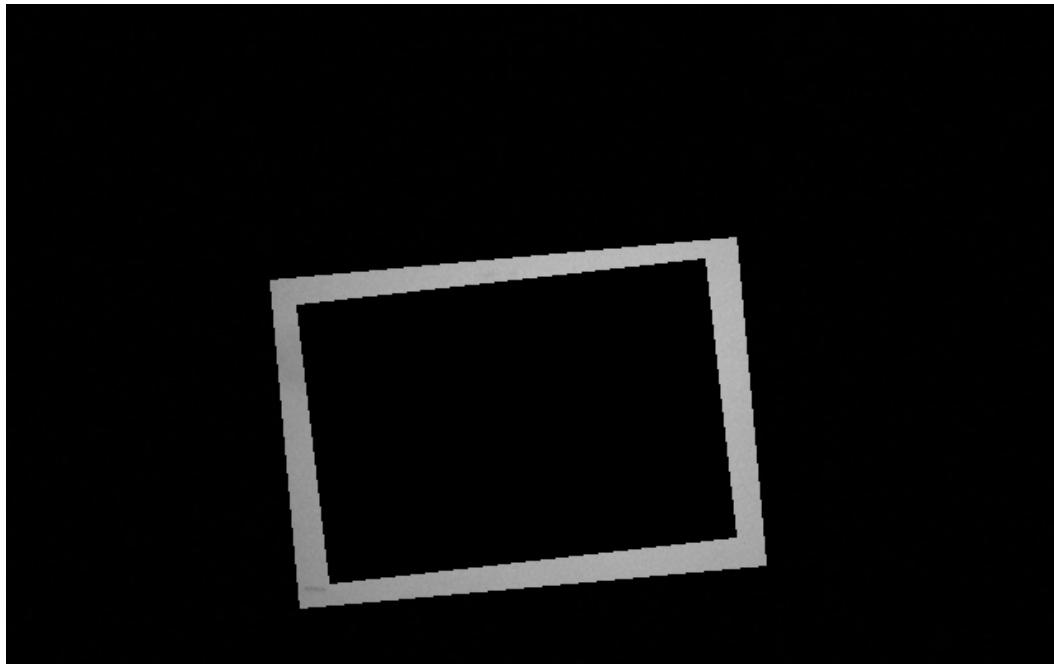


Abbildung 30: Orientierung der Dokumentenmaske

Die beispielhaften Ergebnisse in Abbildung 31 zeigen, dass Heftklammern unabhängig ihrer Orientierung und Position erkannt wurden.

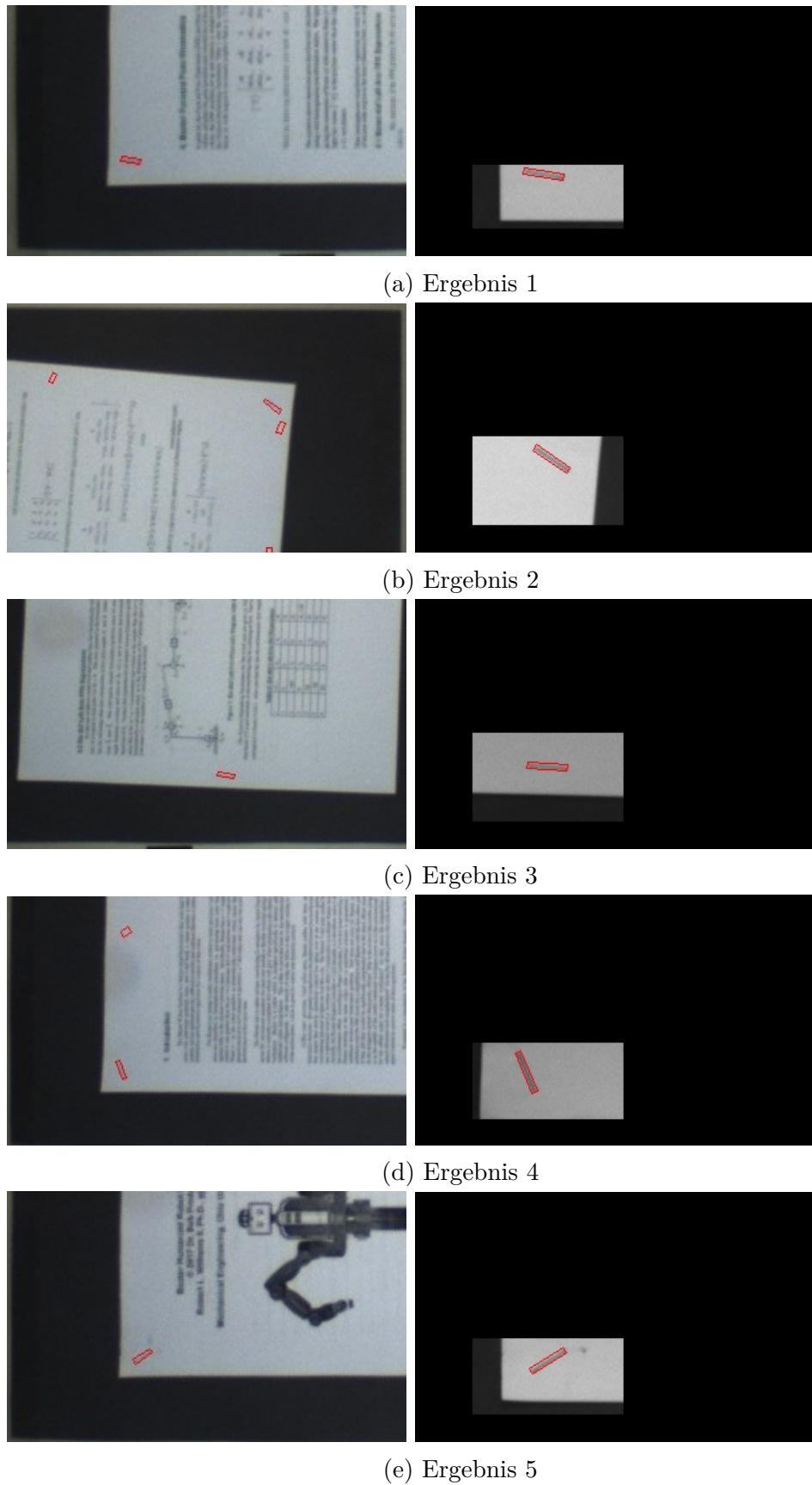


Abbildung 31: Gefundene Heftklammern unterschiedlicher Orientierung und Position

Im Zuge der Versuchsreihen hat sich gezeigt, dass die Position und Orientierung der Heftklammern nur geringe bis keine Auswirkung auf die Heftklammererkennung haben. Weiterhin hat sich herausgestellt, dass die Beleuchtung der Versuchsumgebung einen starken Einfluss auf die Heftklammererkennung hat. Aufgrund der Gegebenheiten des, für die Versuche genutzten, Raums, ließ sich die Versuchsumgebung nicht komplett abdunkeln. Es hat sich ergeben, dass zu bestimmten Uhrzeiten, also bei einem gewissen Lichteinfallswinkel durch die Fenster des Versuchsraums, besonders ungeeignete Lichtumgebungen geschaffen wurden. Diese konnten, weder durch automatische, noch manuelle Konfiguration der Kamera ausgeglichen werden. Diese Lichtumgebungen erzeugten, dass vermehrt nicht-existente Konturen erkannt wurden, während der Kontrast zwischen Heftklammer und Dokument so stark herabgesetzt wurde, dass weder mit dem menschlichen Auge, noch durch die Algorithmen eine Heftklammer im Kamerabild erkannt werden konnte. Der Erkennungsalgorithmus profitiert somit von einer gleichmäßigen und gleichbleibenden Beleuchtung des Arbeitsbereichs.

In Kapitel 3 wurde ein Ansatz zur Erkennung von Heftklammern, sowie die zugrundeliegenden Algorithmen, vorgestellt und erläutert. Der aufgeführte Ansatz wurde in mehreren Interfaces umgesetzt, die eine mehrschrittige Erkennung von, in Dokumenten applizierten, Heftklammern ermöglichen. Zusätzlich ermöglichen die geschaffenen Interfaces die kameragestützte Steuerung des linken Roboterarms, wodurch dieser gefundene Heftklammern mit einem Stift markieren kann.

4 Greifkrafterhöhung

Wie in Kapitel 1.1 erwähnt, kann der elektrische Greifer keine ausreichende Kraft aufbringen, um mit dem verwendeten Werkzeug eine Heftklammer aus einem Dokument zu entfernen. Das dabei genutzte Werkzeug ist ein 'Leitz Entklammerer', entsprechend Abbildung 32.



Abbildung 32: Leitz Entklammerer[8]

Das Werkzeug wurde ausgewählt, da es bei Versuchen, im Zuge des Praxisprojekts, gute Entklammerungsergebnisse erbracht hat und ohne Nutzung eines Greiferadapters vom Roboter gehalten werden kann. Im Zuge des Praxisprojekts hat sich ergeben, dass die, durch den elektrischen Greifer, ausübbare Kraft nicht ausreicht, um eine Heftklammer aus Stahl, nach Typ 3 der Norm DIN 7405, zu verformen. Dabei wurde das Werkzeug vom Greifer an den designierten Griffflächen ergriffen, da diese Position die meiste Kontrolle über das Werkzeug bietet.

Die, zum vollständigen Entfernen einer Heftklammer, benötigte Kraft, bei Nutzung des genannten Werkzeugs, wurde, aufgrund des zeitlichen Rahmens der Arbeit, nicht ermittelt. Im Folgenden werden mögliche Ansätze genannt, sowie einige bei der Umsetzung zu beachtende Restriktionen. Aufgrund des begrenzten zeitlichen Rahmens, der vorliegenden Arbeit, konnten zu der, in diesem Kapitel behandelten, Problemstellung keine Versuche und Lösungen umgesetzt werden.

Ein einfacher mechanischer Lösungsansatz besteht darin, einen Greiferadapter nach dem Prinzip einer handelsüblichen Schere oder Zange zu nutzen. Dieser kann mit einer großen Hebellänge entworfen werden, sodass die an der Klammer wirkende Kraft, nach dem Hebelgesetz von Archimedes, zunimmt, je weiter die Hebellänge erhöht wird.

Der genannte Ansatz könnte jedoch voraussichtlich nicht in Kombination mit der in

Kapitel 3 vorgestellten Software verwendet werden, da diese einen möglichst gerin- gen Abstand zwischen Kamera und Dokument benötigt, um zuverlässige Ergebnisse zu erzielen. Der Abstand zwischen Greiferspitze und Dokument beträgt, bei Nutzung der aktuellen Konfiguration, ungefähr zehn Zentimeter. Gleichzeitig darf der zu entwickelnde Adapter nicht die Handkamera des Roboters verdecken, wodurch die Maße der umzusetzenden Lösung in zwei Richtungen begrenzt werden.

Ein weiterer mechanischer Lösungsansatz entspricht dem Prinzip sogenannter Ratschenscheren. Dabei handelt es sich um handelsübliche Gartenscheren. Durch die namensgebende Ratschenmechanik besteht das Schließen der Schere aus mehreren Schritten, bei denen jeweils stärkere Hebelwirkungen erzielt werden. Durch den, im Vergleich zum zuvor genannten Ansatz, geringen Platzbedarf dieser Mechanik, könnte dieser Ansatz, trotz der genannten Restriktionen, umsetzbar sein.

Neben rein mechanischen Ansätzen, ermöglicht Baxter die Nutzung von Pneumatik, wie der Saugkopfgreifer von Rehtink Robotics zeigt. Dafür wird ein externer Kompressor benötigt.

Weitere Möglichkeiten stehen zur Verfügung, sobald sich nicht auf die Nutzung der originalen Greifer von Rethink Robotics beschränkt, sondern stattdessen ein neuer Greifer entwickelt wird. Wie in Unterkapitel 1.2 genannt, können die Greifer des Roboters ausgewechselt werden. Die Funktionen der `gripper`-Komponente, des `baxter_interface`, können auch zur Steuerung nicht-offizieller Greifer verwendet werden. Entsprechende Abfragen innerhalb der Funktionen überprüfen jeweils nach Aufruf, um welche Art von Greifer es sich handelt und führen, der erkannten Greiferart entsprechend, unterschiedliche Befehle aus. Die dabei möglichen Greiferarten lauten 'electric', 'suction' und 'custom'.

5 Fazit und Ausblick

In Kapitel 1.1 wurden mehrere Problemstellungen aufgeführt, für die, in den darauf folgenden Kapiteln, Lösungsansätze, sowie deren Umsetzung, in den Kapiteln 2 und 3, vorgestellt wurden.

Die Problemstellung, der unzureichenden Positioniergenauigkeit des linken Roboterarms, konnte, durch verschiedene Messungen, in zwei zusammenwirkende Teilprobleme aufgeteilt werden. Das erste Teilproblem besteht in der erzielten Präzision des Roboterarms, die von der Bewegungsrichtung des Roboterarms, sowie der Bewegungsweite abhängig ist. Es hat sich gezeigt, dass durch Einsatz einer spezifischen, mehrschrittigen Verfahrweise die erzielte Präzision erhöht werden kann, indem Bewegungsweite, -richtung und -geschwindigkeit auf empirisch ermittelte Werte festgelegt werden. Die entwickelte Verfahrweise konnte als Methode implementiert werden und wurde im weiteren Verlauf der Arbeit mehrfach eingesetzt. Das zweite Teilproblem liegt in einer, systematisch auftretenden, Abweichung von der Zielpose. Diese kann bis zu mehreren Millimeter betragen und tritt nach keinen erkennbaren Mustern, in Abhängigkeit von der angesteuerten Pose, auf. Zur Lösung dieses Teilproblems wurde sich am Prinzip der Umsetzungstabelle orientiert und eine Datenstruktur entwickelt, durch deren Einsatz die auftretenden systematischen Abweichungen innerhalb eines definierten Bereichs verringert werden konnten, solange sie in Kombination mit der, für Teilproblem 1 entwickelten, Methode angewandt wird. Der kombinierte Einsatz beider Lösungsansätze führt zu einer erhöhten Präzision und der Verringerung der systematischen Abweichung und somit zu einer Verbesserung der Positioniergenauigkeit in allen betrachteten Aspekten.

Eine weitere, in Kapitel 1.1 vorgestellte, Problemstellung besteht in den Restriktionen, die dem Ausgangszustand des Projekts auferlegt wurden. Diese erzielten, dass der Versuchsaufbau dem Bewegungsablauf des Roboters angepasst werden musste. Durch den Einsatz der im Roboter verbauten Handkameras konnte erreicht werden, dass der Bewegungsablauf des Roboters sich, zu einem gewissen Grad, dem Versuchsaufbau anpasst. Hierzu werden Bildbearbeitungs- und Objekterkennungsalgorithmen auf die, von der Handkamera empfangenen, Bilder angewandt. Einige der, vom Roboterarm eingenummernen, Posen werden dabei, anhand der im Bild erkannten Konturen, berechnet. Durch den verfolgten Ansatz wurde erreicht, dass Heftklammern auf Dokumenten erkannt und markiert werden konnten, unabhängig von Position und Orientierung der Dokumente und Heftklammern.

In Kapitel 4 wurden mögliche Ansätze zur Überwindung der Begrenzung der Greifkraft aufgeführt. Aufgrund des begrenzten Zeitrahmens und der fachlichen Ausrichtung dieser Arbeit, konnten in diesem Themenfeld jedoch keine Versuche und Lösungen umgesetzt werden. Auch die anderen Themenfelder dieser Arbeit bieten viele Möglichkeiten zur weiteren Forschung. Besonders die Objekterkennung ist ein sehr vielseitiges The-

ma, das eine Vielzahl unterschiedlicher Ansätze und Algorithmen bietet. Während in dieser Arbeit, aufgrund des zeitlichen Rahmens, ein vergleichsweise einfacher Ansatz verfolgt wurde, könnte der Einsatz komplexerer Ansätze, wie das maschinelle Lernen, zu vielversprechenden robusten Lösungen führen. Neben der Heftklammer existieren weitere metallische Bindungsarten, die an Dokumenten angewandt werden. Ein Beispiel hierfür ist die Büroklammer. Die in dieser Arbeit entwickelte Software könnte, im Zuge weiterer Projekte, auf andere Bindungsarten erweitert werden.

Insgesamt stellt diese Arbeit somit nur einen Einstieg in das Thema der automatisierten Metallentfernung an Dokumenten dar.

Literaturverzeichnis

- [1] Ana Huamán. *OpenCV: Cascade Classifier*. Hrsg. von OpenCV. 2020. URL: https://docs.opencv.org/master/db/d28/tutorial_cascade_classifier.html (besucht am 22. 10. 2020).
- [2] BET-Fachwörterbuch, Hrsg. *Lookup Table - Definition im Fachwörterbuch / Lexikon - BET - Michael Mücher - Seminare und Workshops, Fernsehen und Video*. 2020. URL: <https://www.bet.de/lexikon/lookuptable/> (besucht am 14. 10. 2020).
- [3] DATACOM Buchverlag GmbH, Hrsg. *ROI (region of interest) :: ITWissen.info*. 22.10.2015. URL: <https://www.itwissen.info/ROI-region-of-interest.html> (besucht am 25. 10. 2020).
- [4] Fraunhofer IGD, Hrsg. *Computer Vision (CV) / Fraunhofer IGD*. Fraunhofer IGD. o.J. URL: <https://www.igd.fraunhofer.de/kompetenzen/forschungslinien/computer-vision-cv> (besucht am 27. 10. 2020).
- [5] Erico Guizzo. *Rethink Robotics Opens Up Baxter Robot For Researchers - IEEE Spectrum. Researchers now have access to an open source SDK to "hack" Baxter*. IEEE. 2013. URL: <https://spectrum.ieee.org/automaton/robotics/humanoids/rethink-robotics-baxter-research-robot> (besucht am 01. 08. 2020).
- [6] IEEE, Hrsg. *IROS '95 IEEE/RSJ International conference on Intelligent Robots and Systems. Human robot interaction and cooperative robots : proceedings : August 5-9, 1995, Pittsburgh, Pennsylvania*. eng. Unter Mitarb. von Gill A. Pratt und Matthew M. Williamson. Bd. 3. 3 Bde. Computer Science and Scientific Computing. Los Alamitos, Brussels und Tokyo: IEEE, 1995. 399–406. ISBN: 0818671084. URL: <http://ieeexplore.ieee.org/servlet/opac?punumber=3952>.
- [7] KUKA. *Mensch-Roboter-Kollaboration in der Produktion - KUKA AG*. de-DE. KUKA. 4.08.2020. URL: <https://www.kuka.com/de-de/future-production/mensch-roboter-kollaboration> (besucht am 04. 08. 2020).
- [8] Leitz, Hrsg. *Leitz Entklammerer - Heftklammern und Zubehör / LEITZ*. Leitz Acco Brands GmbH & Co KG. 2020. URL: https://www.leitz.com/de-de/products/leitz-entklammerer_55900085/ (besucht am 26. 10. 2020).
- [9] Kate Malczewski. *Children train robots in school to prepare for the future workplace - Factor*. 2014. URL: <https://www.factor-tech.com/future-cities/6277-children-train-robots-in-school-to-prepare-for-the-future-workplace/> (besucht am 01. 08. 2020).
- [10] Satya Mallick. “Shape Matching using Hu Moments (C++/Python)”. In: *Learn OpenCV* (11. Dez. 2018). URL: <https://www.learnopencv.com/shape-matching-using-hu-moments-c-python/> (besucht am 24. 10. 2020).
- [11] Open Robotics, Hrsg. *ROS.org / About ROS*. Creative Commons Attribution 3.0. 1.08.2020. URL: <https://www.ros.org/about-ros/> (besucht am 01. 08. 2020).
- [12] Open Robotics, Hrsg. *ROS.org / Core Components*. Open Robotics. o.J. URL: <https://www.ros.org/core-components/> (besucht am 27. 10. 2020).
- [13] Open Robotics, Hrsg. *ROS.org / History*. Open Robotics. o.J. URL: <https://www.ros.org/history/> (besucht am 27. 10. 2020).

- [14] OpenCV, Hrsg. *OpenCV - About*. 2020. URL: <https://opencv.org/about/> (besucht am 27.10.2020).
- [15] OpenCV, Hrsg. *OpenCV: Basic Operations on Images*. 2020. URL: https://docs.opencv.org/4.5.0/d3/df2/tutorial_py_basic_ops.html (besucht am 27.10.2020).
- [16] OpenCV, Hrsg. *OpenCV: Canny Edge Detection*. 2020. URL: https://docs.opencv.org/master/da/d22/tutorial_py_canny.html (besucht am 22.10.2020).
- [17] OpenCV, Hrsg. *OpenCV: Contours : Getting Started*. 2020. URL: https://docs.opencv.org/master/d4/d73/tutorial_py_contours_begin.html (besucht am 22.10.2020).
- [18] OpenCV, Hrsg. *OpenCV: Contours : More Functions*. 2020. URL: https://docs.opencv.org/master/d5/d45/tutorial_py_contours_more_functions.html (besucht am 24.10.2020).
- [19] OpenCV, Hrsg. *OpenCV: cv::Mat Class Reference*. 2020. URL: https://docs.opencv.org/4.5.0/d3/d63/classcv_1_1Mat.html#details (besucht am 27.10.2020).
- [20] OpenCV, Hrsg. *OpenCV: Introduction*. 2020. URL: <https://docs.opencv.org/4.5.0/d1/dfb/intro.html> (besucht am 27.10.2020).
- [21] OpenCV, Hrsg. *OpenCV: Smoothing Images*. 2020. URL: https://docs.opencv.org/master/d4/d13/tutorial_py_filtering.html (besucht am 22.10.2020).
- [22] OpenCV, Hrsg. *OpenCV: Sobel Derivatives*. 2020. URL: https://docs.opencv.org/3.4/d2/d2c/tutorial_sobel_derivatives.html (besucht am 22.10.2020).
- [23] Python Software Foundation, Hrsg. *5. Data Structures — Python 3.9.0 documentation*. Python Software Foundation. 16.10.2020. URL: <https://docs.python.org/3/tutorial/datastructures.html#dictionaries> (besucht am 16.10.2020).
- [24] Rethink Robotics, Hrsg. *Arms - sdk-wiki*. Rethink Robotics. 8.10.2014. URL: <https://sdk.rethinkrobotics.com/wiki/Arms> (besucht am 27.10.2020).
- [25] Rethink Robotics. *Arm Calibration - sdk-wiki*. Hrsg. von Rethink Robotics. Rethink Robotics. 2014. URL: https://sdk.rethinkrobotics.com/wiki/Arm_Calibration (besucht am 13.10.2020).
- [26] Rethink Robotics. *Baxter Research Robot Software Developers Kit (SDK) - sdk-wiki*. Hrsg. von Rethink Robotics. Rethink Robotics. 2014. URL: [https://sdk.rethinkrobotics.com/wiki/Baxter_Research_Robot_Software_Developers_Kit_\(SDK\)](https://sdk.rethinkrobotics.com/wiki/Baxter_Research_Robot_Software_Developers_Kit_(SDK)) (besucht am 13.10.2020).
- [27] Rethink Robotics. *Workspace Guidelines - sdk-wiki*. Hrsg. von Rethink Robotics. Rethink Robotics. 2015. URL: https://sdk.rethinkrobotics.com/wiki/Workspace_Guidelines (besucht am 13.10.2020).
- [28] Rethink Robotics. *Hardware Specifications - sdk-wiki*. Hrsg. von Rethink Robotics. Rethink Robotics. 2016. URL: https://sdk.rethinkrobotics.com/wiki/Hardware_Specifications#Peak_Torque (besucht am 13.10.2020).

- [29] S. Cremer, L. Mastromoro und D. O. Popa. "On the performance of the Baxter research robot". In: *2016 IEEE International Symposium on Assembly and Manufacturing (ISAM)*. 2016 IEEE International Symposium on Assembly and Manufacturing (ISAM). Hrsg. von IEEE. 2016, S. 106–111. DOI: 10.1109/ISAM.2016.7750722. URL: <https://ieeexplore.ieee.org/document/7750722> (besucht am 01.08.2020).
- [30] Matt Simon. "A Long Goodbye to Baxter, a Gentle Giant Among Robots". In: *WIRED* (10. Aug. 2018). URL: <https://www.wired.com/story/a-long-goodbye-to-baxter-a-gentle-giant-among-robots/> (besucht am 01.08.2020).
- [31] Satoshi Suzuki und KeiichiA be. "Topological structural analysis of digitized binary images by border following". In: *Computer Vision, Graphics, and Image Processing* 30.1 (1985). PII: 0734189X85900167, S. 32–46. ISSN: 0734-189X. DOI: 10.1016/0734-189X(85)90016-7. URL: <http://www.sciencedirect.com/science/article/pii/0734189X85900167>.

Anhang

A - arm_class.py

```
1 #!/usr/bin/env python
2 -*- coding:utf-8 -*-
3
4 import struct
5 import sys
6 import time
7
8 import const_lib
9 import cam_class
10
11 import rospy
12 import baxter_interface
13
14 from copy import deepcopy
15
16 from std_msgs.msg import Header
17
18 from geometry_msgs.msg import (
19     PoseStamped,
20     Pose,
21     Point,
22     Quaternion
23 )
24
25 from baxter_core_msgs.srv import (
26     SolvePositionIK,
27     SolvePositionIKRequest
28 )
29
30 def alter_pose_abs(pose, verbose=False, posx = None, posy = None, posz
= None, orx = None, ory = None, orz = None, orw = None):
31     """
32         Replaces the given poses parameters with given parameter
33         modifiers.
34
35             Parameter:
36                 pose:          The pose to be modified
37                 verbose:       True -> print verbose information; False
38                 -> dont print verbose information; Default: False
39                 posx:          The value for the positions x-value to be
40                 replaced; Default: None
41                 posy:          The value for the positions y-value to be
42                 replaced; Default: None
```

```
39         posz:           The value for the positions z-value to be
40             replaced; Default: None
41
42         orx:           The value for the orientations x-value to
43             be replaced; Default: None
44
45         ory:           The value for the orientations y-value to
46             be replaced; Default: None
47
48         orz:           The value for the orientations z-value to
49             be replaced; Default: None
50
51         orw:           The value for the orientations w-value to
52             be replaced; Default: None
53
54     Return:
55
56         workpose:   The modified pose
57
58     """
59
60     if verbose:
61         print(" --- func: alter_pose_abs ---")
62         print("posx={} posy={} posz={} orx={} ory={} orz={} orw={}")
63     .format(posx, posy, posz, orx, ory, orz, orw)
64
65     if posx is None: posx = pose.position.x
66     if posy is None: posy = pose.position.y
67     if posz is None: posz = pose.position.z
68     if orx is None: orx = pose.orientation.x
69     if ory is None: ory = pose.orientation.y
70     if orz is None: orz = pose.orientation.z
71     if orw is None: orw = pose.orientation.w
72
73     if verbose:
74         print(" --- given pose: ---")
75         print(pose)
76
77     workpose = deepcopy(pose)
78     workpose.position.x = posx
79     workpose.position.y = posy
80     workpose.position.z = posz
81     workpose.orientation.x = orx
82     workpose.orientation.y = ory
83     workpose.orientation.z = orz
84     workpose.orientation.w = orw
85
86     if verbose:
87         print(" --- new pose: ---")
88         print(workpose)
89         print("-----")
90
91     return workpose
92
93
94 def alter_pose_inc(pose, verbose=False, posx = 0.0, posy = 0.0, posz =
95 0.0, orx = 0.0, ory = 0.0, orz = 0.0, orw = 0.0):
96 """
97
98     Adds the given parameter modifiers to the given poses parameters.
99
100
101     Parameter:
102         pose:       The pose to be modified
```

```

    verbose:      True -> print verbose information; False ->
    dont print verbose information; Default: False

    posx:          The value for the positions x-value to be
    modified; Default: None

    posy:          The value for the positions y-value to be
    modified; Default: None

    posz:          The value for the positions z-value to be
    modified; Default: None

    orx:           The value for the orientations x-value to be
    modified; Default: None

    ory:           The value for the orientations y-value to be
    modified; Default: None

    orz:           The value for the orientations z-value to be
    modified; Default: None

    orw:           The value for the orientations w-value to be
    modified; Default: None

    Return:
        workpose:   The modified pose
    """
if verbose:
    print("--- func: alter_pose_inc ---")
    print("posx={} posy={} posz={} orx={} ory={} orz={} orw={}").format(posx, posy, posz, orx, ory, orz, orw)
    print("--- given pose: ---")
    print(pose)
workpose = deepcopy(pose)
workpose.position.x += posx
workpose.position.y += posy
workpose.position.z += posz
workpose.orientation.x += orx
workpose.orientation.y += ory
workpose.orientation.z += orz
workpose.orientation.w += orw
if verbose:
    print("--- new pose: ---")
    print(workpose)
    print("-----")
return workpose

def convert_to_pose(pose_dict):
    """
    Converts the given dictionary to a pose object.

    Parameters:
        pose_dict: Dictionary containing a pose
    Return:
        pose:       The given dictionary as pose object
    """

```

```
119     pose = Pose()
120     pose.position.x = pose_dict['position'].x
121     pose.position.y = pose_dict['position'].y
122     pose.position.z = pose_dict['position'].z
123     pose.orientation.x = pose_dict['orientation'].x
124     pose.orientation.y = pose_dict['orientation'].y
125     pose.orientation.z = pose_dict['orientation'].z
126     pose.orientation.w = pose_dict['orientation'].w
127
128     return pose
129
130
131 def failsafe(left, right, tool=False):
132     """
133     Moves both arms to neutral positions and disables them.
134
135     Parameters:
136         left:    Left arm to be set neutral
137         right:   Right arm to be set neutral
138         tool:    Flag to determine if the tool shall be stored
139         before setting the left arm neutral; Default: False
140
141     print("Exit routine started\nStoring Tool (if attached) and moving
142         arms to neutral poses")
143     right.set_neutral()
144     if tool:
145         left.store_tool()
146     else:
147         left.set_neutral()
148     left._rs.disable()
149
150
151 class Arm(object):
152     """
153     Class to manage and control one arm of Baxter
154
155     def __init__(self, limb, verbose=False, cam_wanted=False):
156         """
157         Constructor for the Arm class.
158
159         Parameters:
160             limb:        The limb of Baxter to be managed; Options:
161             'left', 'right'
162                 verbose:    True -> print verbose information; False
163                 -> dont print verbose information; Default: False
164                 cam_wanted: Flag to determine if a cam object is
165                 created
166
167                 self._limb_name = limb
168                 self._verbose = verbose
```

```
162     self._limb = baxter_interface.limb.Limb(limb)
163     self._gripper = baxter_interface.Gripper(limb)
164     self._gripper._type = self._gripper.type()
165     self._ns = "ExternalTools/" + limb + "/PositionKinematicsNode/"
166     IKService"
167
168     self._iksvc = rospy.ServiceProxy(self._ns, SolvePositionIK)
169     self._ikreq = SolvePositionIKRequest()
170     self._hdr = Header(stamp=rospy.Time.now(), frame_id='base')
171     self._current_pose = convert_to_pose(self._limb.endpoint_pose
172     ())
173
174     if self._gripper._type is 'suction':
175         self._gripper.set_vacuum_threshold(1.0)
176     if self._gripper._type is 'electric':
177         self._gripper.set_moving_force(100.0)
178     self._limb.set_joint_position_speed(0.3)
179     self.cam_wanted = cam_wanted
180     if cam_wanted:
181         self.cam = cam_class.Cam(limb, self._verbose)
182         self.cam.update_z(self._current_pose.position.z)
183
184     print("Getting robot state...")
185     self._rs = baxter_interface.RobotEnable(baxter_interface.
186     CHECK_VERSION)
187     self._init_state = self._rs.state().enabled
188     if not self._init_state:
189         print("Enabling robot...")
190         self._rs.enable()
191     else:
192         print("Robot already enabled...")
193     if self._gripper._type is 'electric' and not self._gripper.
194     calibrated():
195         print("Calibrating electric gripper...")
196         self._gripper.calibrate()
197
198     def get_solution(self, pose):
199         """
200             Warning: Deprecated. Please use move_to_pose() instead of
201             get_solution() and move_to_solution(). These functions are only
202             kept for compatibility.
203
204             Calculates a joint solution for the given pose if reachable.
205
206             Parameters:
207                 pose:           The pose to be reached by the endpoint
208                 effector.
209
210             Return:
211                 False ->    No valid joint solution could be found for
212                 the given pose
```

```
202             True ->      Valid joint solution found for the given
203             pose
204             """
205
206             if self._verbose:
207                 print("--- func: get_solution ---")
208                 print("--- pose: ---")
209                 print(pose)
210                 print("-----")
211
212             del self._ikreq.pose_stamp[:]
213             self._ikreq.pose_stamp.append(PoseStamped(header=self._hdr,
214             pose=pose))
215
216             try:
217                 rospy.wait_for_service(self._ns, 5.0)
218                 resp = self._iksrv(self._ikreq)
219
220             except (rospy.ServiceException, rospy.ROSEException) as e:
221                 rospy.logerr("Service call failed: %s" %(e,))
222                 return False
223
224
225             #Check if result is valid, and type of seed ultimately used to
226             #get solution
227
228             #convert rospy's string representation of uint8[]'s to int's
229             resp_seeds = struct.unpack('<%dB' % len(resp.result_type),
230             resp.result_type)
231
232             if (resp_seeds[0] != resp.RESULT_INVALID):
233                 seed_str = {
234                     self._ikreq.SEED_USER: 'User Provided Seed',
235                     self._ikreq.SEED_CURRENT: 'Current Joint Angles',
236                     self._ikreq.SEED_NS_MAP : 'Nullspace Setpoints',
237                 }.get(resp_seeds[0], 'None')
238
239                 if self._verbose:
240                     print("SUCCESS - Valid Joint Solution Found from Seed
241 Type: %s" % (seed_str,))
242
243                 #Format solution into Limb API-compatible dictionary
244                 self._ik_solution = dict(zip(resp.joints[0].name, resp.
245             joints[0].position))
246
247                 if self._verbose:
248                     print("\nIK Solution:\n", self._ik_solution)
249                     print("-----")
250                     print("Response Message:\n", resp )
251
252             return True
253
254         else:
255             print ("INVALID POSE - No Valid Joint Solution Found.")
256
257         return False
258
259
260     def move_to_solution(self):
261         """
262
263             Warning: Deprecated. Please use move_to_pose() instead of
264             get_solution() and move_to_solution(). These functions are only
265             kept for compatibility with older versions of the API.
```

```
kept for compatibility.

243
244     Executes the movement to the previously calculated pose.
245     Please run get_solution() beforehand for this to function properly
246
247     """
248
249     if self._verbose:
250         print("moving %s arm..." %self._limb_name)
251         self._limb.move_to_joint_positions(self._ik_solution)
252         self._current_pose = convert_to_pose(self._limb.endpoint_pose
253         ())
254         time.sleep(1)    #to make sure he really finished his movement
255         and reached the pose
256
257     def move_to_pose(self, pose):
258         """
259         Calculates and executes a joint solution for the given pose.
260
261         Parameters:
262             pose:           The pose to be reached by the endpoint
263             effector.
264
265         Return:
266             False ->      No valid joint solution could be found for
267             the given pose
268             True ->       The arm reached the given pose
269
270         """
271
272         if self._verbose:
273             print("--- func: get_solution ---")
274             print("--- pose: ---")
275             print(pose)
276             print("-----")
277
278         del self._ikreq.pose_stamp[:]
279         self._ikreq.pose_stamp.append(PoseStamped(header=self._hdr,
280             pose=pose))
281
282         try:
283             rospy.wait_for_service(self._ns, 5.0)
284             resp = self._iksvc(self._ikreq)
285         except (rospy.ServiceException, rospy.ROSEException) as e:
286             rospy.logerr("Service call failed: %s" %(e,))
287             return False
288
289
290         #Check if result is valid, and type of seed ultimately used to
291         #get solution
292
293         #convert rospy's string representation of uint8[]'s to int's
294         resp_seeds = struct.unpack('<%dB' % len(resp.result_type),
295             resp.result_type)
296
297         if (resp_seeds[0] != resp.RESULT_INVALID):
298             seed_str = {
```

```
281         self._ikreq.SEED_USER: 'User Provided Seed',
282         self._ikreq.SEED_CURRENT: 'Current Joint Angles',
283         self._ikreq.SEED_NS_MAP : 'Nullspace Setpoints',
284     }.get(resp_seeds[0], 'None')
285     if self._verbose:
286         print("SUCCESS - Valid Joint Solution Found from Seed
287 Type: %s" % (seed_str,))
288         #Format solution into Limb API-compatible dictionary
289         self._ik_solution = dict(zip(resp.joints[0].name, resp.
290 joints[0].position))
291         if self._verbose:
292             print("\nIK Solution:\n", self._ik_solution)
293             print("-----")
294             print("Response Message:\n", resp )
295             print("moving %s arm..." %self._limb_name)
296             #execute movement
297             self._limb.move_to_joint_positions(self._ik_solution)
298             time.sleep(1)    #to make sure he really finished his
299             movement and reached the pose
300             self._current_pose = convert_to_pose(self._limb.
301 endpoint_pose())
302             if self.cam_wanted:
303                 self.cam.update_z(self._current_pose.position.z)
304             return True
305         else:
306             print ("INVALID POSE - No Valid Joint Solution Found.")
307             print("Given Pose:\n{}".format(pose))
308             return False
309
310     def move_precise(self, pose):
311         """
312             Calculates and executes a joint solution for the given pose in
313             a more precise way.
314
315             Approaches the given pose in three steps to provide a more
316             precise positioning.
317             The first step is offset by 2cm negative in x after which the
318             arm approaches the given pose with one intermediate step.
319             For more thorough information on this method please see "
320             Metallentfernung an Dokumenten durch den Forschungsroboter Baxter"
321             by "Timo Sch rmann"
322
323             Parameters:
324                 pose:          The pose to be reached by the endpoint
325                 effector.
326
327             Return:
328                 False ->      No valid joint solution could be found for
329                 the given pose
```

```
318             True ->      The arm reached the given pose
319
320         """
321
322         if self._verbose:
323             print("moving {}_arm more precise...").format(self.
324                 _limb_name)
325             if not self.move_to_pose(alter_pose_inc(deepcopy(pose), self.
326                 _verbose, posx=-0.02)):
327                 return False
328             if self.cam_wanted:
329                 self.cam.update_z(self._current_pose.position.z)
330             if not self.move_to_pose(alter_pose_inc(deepcopy(pose), self.
331                 _verbose, posx=-0.01)):
332                 return False
333             if not self.move_to_pose(pose):
334                 return False
335             return True
336
337     def move_direct(self, pose, precise=False):
338         """
339             Calculates and executes a joint solution for the given pose
340             alongside the axes.
341
342             Moves alongside the axes in multiple steps. The movement is
343             executed in following sequence: x-axis, y-axis, z-axis
344             Use this function if the workspace is obstructed by possible
345             obstacles and you need a more linear movement.
346
347             Parameters:
348                 pose:           The pose to be reached by the endpoint
349                 effector.
350
351                 precise:        Flag to determine if the given pose shall be
352                     approached in a more precise manner (for more information see
353                     move_precise()); Default: False
354
355             Return:
356
357                 False ->      No valid joint solution could be found for
358                 the given pose
359                 True ->       The arm reached the given pose
360
361         """
362
363         if self._verbose:
364             print("moving {}_arm on kind of linear way").format(self.
365                 _limb_name))
366             big_move = True
367             step_width = 0.03
368             while big_move:
369                 x_diff = pose.position.x - self._current_pose.position.x
370                 y_diff = pose.position.y - self._current_pose.position.y
371                 z_diff = pose.position.z - self._current_pose.position.z
372                 if self._verbose:
```

```
354         print("way left:\nx: {}\\ny: {}\\nz: {}".format(x_diff,
355             y_diff, z_diff))
356         big_move = False
357         x_step = 0.0
358         y_step = 0.0
359         z_step = 0.0
360         if x_diff > step_width:
361             x_diff -= step_width
362             x_step = step_width
363             big_move = True
364         elif x_diff < -step_width:
365             x_diff += step_width
366             x_step = -step_width
367             big_move = True
368         elif y_diff > step_width:
369             y_diff -= step_width
370             y_step = step_width
371             big_move = True
372         elif y_diff < -step_width:
373             y_diff += step_width
374             y_step = -step_width
375             big_move = True
376         elif z_diff > step_width:
377             z_diff -= step_width
378             z_step = step_width
379             big_move = True
380         elif z_diff < -step_width:
381             z_diff += step_width
382             z_step = -step_width
383             big_move = True
384         elif precise:
385             self.move_precise(pose)
386             return True
387         else:
388             self.move_to_pose(pose)
389             return True
390         self.move_to_pose(alter_pose_inc(self._current_pose,
391             verbose=self._verbose, posx=x_step, posy=y_step, posz=z_step))
392         if self.cam_wanted:
393             self.cam.update_z(self._current_pose.position.z)
394         return False
395
396     def set_neutral(self, open_gripper=True):
397         """
398             Moves the arm to a neutral pose.
399
400             Parameters:
```

```
400         open_gripper: Flag to determine if the gripper shall be
401         opened after reaching the neutral pose
402
403         """
404
405         self._limb.move_to_neutral()
406         if open_gripper:
407             self._gripper.open()
408             self._current_pose = convert_to_pose(self._limb.endpoint_pose
409             ())
410
411
412     def pick(self, pick_pose, remove_staple=False, remove_clip=False,
413             opening=0.0, hover_distance=0.1):
414
415         """
416
417         Performs a classic "pick" movement.
418
419
420         Approaches the given pose from a heightened pose, closes the
421         gripper and retreats to a heightened pose.
422
423         With the optional flag parameters slightly different movements
424         can be performed to remove staples or clippers from documents.
425
426
427         Parameters:
428
429             pick_pose:           Pose at which the gripper shall be closed
430             to perform the picking
431
432             remove_staple:      Flag to determine if a special movement
433             shall be executed for removing staples from documents; Default:
434             False
435
436             remove_clip:        Flag to determine if a special movement
437             shall be executed for removing clips from documents; Default:
438             False
439
440             opening:            Percentage of opening for the gripper.
441             0.0 -> closed, 100.0 -> open; Default: 0.0
442
443             hover_distance:    Distance between the heightened pose and
444             the pick_pose; Default: 0.1
445
446             Return:
447
448                 False ->      No valid joint solution could be found for
449                 the given pose
450
451                 True ->       The arm reached the given pose
452
453         """
454
455         if self._verbose:
456             print("--- func: pick ---")
457             print("--- given pose: ---")
458             print(pick_pose)
459
460             hover_pose = alter_pose_inc(deepcopy(pick_pose), verbose=self.
461             _verbose, posz = hover_distance)
462
463             #hover point
464             if self.get_solution(hover_pose):
465
466                 if self._verbose:
467
468                     print("moving to hover pose...")
469
470                 self.move_to_solution()
```



```
474         if self._verbose:
475             print("----> suction: {}".format(self._gripper.
476                 vacuum_sensor()))
477             if not self._gripper.vacuum_sensor() > 8.0:
478                 print("Gripping with {}_arm failed\nSuction:
479                     {}".format(self._limb_name, self._gripper.vacuum_sensor()))
480             return False
481             self._limb.set_joint_position_speed(0.3)
482             return True
483         else:
484             return False
485
486     def place(self, place_pose, opening=100.0, hover_distance=0.1):
487         """
488             Performs a classic "place" movement.
489
490             Approaches the given pose from a heightened pose, opens the
491             gripper and retreats to a heightened pose.
492
493             Parameters:
494                 place_pose:          Pose at which the gripper shall be opened
495                 to perform the placing
496                 opening:            Percentage of opening for the gripper.
497                 0.0 -> closed, 100.0 -> open; Default: 100.0
498                 hover_distance:    Distance between the heightened pose and
499                 the pick_pose; Default: 0.1
500
501             Return:
502                 False ->      No valid joint solution could be found for
503                 the given pose
504                 True ->       The arm reached the given pose
505
506         if self._verbose:
507             print("--- func: place ---")
508             print("--- given pose: ---")
509             print(place_pose)
510             safe_pose = alter_pose_inc(deepcopy(place_pose), verbose=self.
511             _verbose, posz = hover_distance)
512             #hover point
513             if self.get_solution(safe_pose):
514                 if self._verbose:
515                     print("moving to hover pose...")
516                     self.move_to_solution()
517             else:
518                 return False
519             #place point
520             if self.get_solution(place_pose):
521                 if self._verbose:
522                     print("approaching target...")
```

```
514         self._limb.set_joint_position_speed(0.15)
515         self.move_to_solution()
516         time.sleep(1)
517         print("-----> place_pose: {}".format(self._limb.
518         endpoint_pose()))
518         self._gripper.command_position(position=opening)
519     else:
520         return False
521
522     #retreat
523     if self.get_solution(safe_pose):
524         if self._verbose:
525             print("retreating to safe position...")
526         self.move_to_solution()
527         self._limb.set_joint_position_speed(0.3)
528         return True
529     else:
530         return False
531
532     def place_paper(self):
533         """
534             Performs a special set of movements to place a multipaged
535             document in a repeatable manner.
536
537             For this method to function please provide a proper workspace
538             as described in "Metallentfernung an Dokumenten durch den
539             Forschungsroboter Baxter" by "Timo Sch rmann"
540             as all executed movements are hardcoded.
541
542             Return:
543                 False ->      No valid joint solution could be found for
544                 the given pose
545                 True ->      The arm reached the given pose
546
547             if self._verbose:
548                 print("--- func: place_paper ---")
549                 place_paper_pose = convert_to_pose(const_lib.place_paper_pose)
550                 safe_pose = alter_pose_inc(deepcopy(place_paper_pose), verbose
551                 =self._verbose, posz = const_lib.hover_distance['paper'])
552                 drag_paper_pose = alter_pose_inc(deepcopy(place_paper_pose),
553                 verbose=self._verbose, posx=0.30, posz=const_lib.hover_distance['
554                 paper'], orx=90.0)
555
556                 #initial hover point
557                 if self.get_solution(drag_paper_pose):
558                     if self._verbose:
559                         print("moving to hover pose...")
560                     self.move_to_solution()
561                 else:
```

```

554         return False
555
556     #dragging start point
557     alter_pose_abs(drag_paper_pose, verbose=self._verbose, posz=
const_lib.table_height[self._gripper._type] + 0.015)
558     alter_pose_inc(drag_paper_pose, verbose=self._verbose, posy
=0.015, posx=-0.04)
559     if self.get_solution(drag_paper_pose):
560         if self._verbose:
561             print("lowering paper...")
562         self.move_to_solution()
563     else:
564         return False
565
566     #dragging end point
567     alter_pose_inc(place_paper_pose, verbose=self._verbose, posx
=0.03, posy=0.01)
568     if self.get_solution(place_paper_pose):
569         if self._verbose:
570             print("dragging paper...")
571         self.move_to_solution()
572     else:
573         return False
574
575     #place point
576     alter_pose_inc(place_paper_pose, verbose=self._verbose, posx
=-0.03, posy=-0.01)
577     if self.get_solution(place_paper_pose):
578         if self._verbose:
579             print("approaching target...")
580         self._limb.set_joint_position_speed(0.15)
581         self.move_to_solution()
582         print("-----> place_paper_pose: {}".format(self.
583 _limb.endpoint_pose()))
584         #open gripper
585         self._gripper.open()
586     else:
587         return False
588
589     #retreat
590     if self.get_solution(safe_pose):
591         if self._verbose:
592             print("retreating to safe position...")
593         self.move_to_solution()
594         self._limb.set_joint_position_speed(0.3)
595         return True
596     else:
597         return False
598
599
600     def take_tool(self):
601         """
602
603         Picks up the tool from a hardcoded position.

```

```
597             Return:  
598                 False ->      No valid joint solution could be found  
599                 for the given pose  
600                     True ->      The arm reached the given pose  
601                     """  
602                     if not self._gripper._type is 'electric':  
603                         print("The tool can only be used with an electric gripper\\nThe currently used gripper of {}_arm is {}".format(self.  
604                         _limb_name, self._gripper._type))  
605                         return False  
606                     if not self.pick(convert_to_pose(const_lib.tool_pose), opening  
607 =const_lib.gripper_opening, hover_distance=const_lib.  
608 hover_distance['tool']):  
609                         print("Couldn't take tool")  
610                         return False  
611                     self.set_neutral(open_gripper=False)  
612                     return True  
613  
614             def store_tool(self):  
615                 """  
616                 Places the tool at a hardcoded position.  
617  
618                 Return:  
619                 False ->      No valid joint solution could be found  
620                 for the given pose  
621                     True ->      The arm reached the given pose  
622                     """  
623                     self._gripper.command_position(const_lib.gripper_opening)  
624                     tool_pose = convert_to_pose(const_lib.tool_pose)  
625                     alter_pose_inc(tool_pose, verbose=self._verbose, posx=-0.002,  
626 posy=-0.002, posz=0.03)  
627                     if not self.place(tool_pose, hover_distance=const_lib.  
628 hover_distance['tool']):  
629                         print("Couldn't store tool")  
630                         return False  
631                     self.set_neutral()  
632                     return True  
633  
634             def simplefailsafe(self, open_gripper=True):  
635                 """  
636                 Moves the arm to a neutral pose and disables the robot  
637                 afterwards.  
638  
639                 For setting both arms neutral please use the global function  
640                 failsafe().  
641  
642                 Parameters:
```

```
635         open_gripper: Flag to determine if the gripper shall be
636         opened after reaching a neutral pose
637         """
638         print("Exit routine started \nShutting down arm in neutral
639         pose")
640         self.set_neutral(open_gripper)
641         self._rs.disable()
```

Listing 2: arm_class.py

B - cam_class.py

```
1 #!/usr/bin/env python
2 #-*- coding:utf-8 -*-
3 import argparse
4 import sys
5 import time
6 import rospy
7 import numpy as np
8 import cv2 as cv
9 from cv_bridge import CvBridge, CvBridgeError
10 from sensor_msgs.msg import Image
11 import baxter_interface
12 from copy import deepcopy
13
14
15 class Cam(object):
16     """
17         Class to control one hand camera of Baxter
18     """
19     def __init__(self, limb, verbose, arm_z=5.0):
20         """
21             Constructor for the Cam class.
22
23             Parameters:
24                 limb:          The limb of Baxter whichs camera shall be
25                         managed; Options: 'left', 'right'
26                 verbose:       True -> print verbose information; False
27                         -> dont print verbose information; Default: False
28                 arm_z:         z value of the cameras arm in meter (
29                         needed for action point prediction); Default: 5.0
30
31             """
32             sub_cam = "/cameras/{}/hand_camera/image".format(limb)
33             self._limb = limb
34             self.controller = baxter_interface.CameraController("{}"
35                     "_hand_camera".format(limb))
36             self.controller.resolution = (640, 400)
37             self.bridge = CvBridge()
38             self.update_snapshot = True
39             self._verbose = verbose
40             self._init = True
41             self.sub = rospy.Subscriber(sub_cam, Image, self.show_callback
42         )
43             self.arm_z = arm_z
44             self.windowed = False
45             self._img = None
46             self.ix = 0
```

```
42
43     def show_callback(self, msg):
44         """
45             Callback method for the image stream of the hand camera.
46
47             Displays the live image stream of the cam, the current
48             snapshot and the current highlight.
49
50             Parameters:
51                 msg:      The image message to be displayed
52             """
53
54         try:
55             img = self.bridge.imgmsg_to_cv2(msg)
56             if self.update_snapshot:
57                 self._snapshot = deepcopy(img)
58                 self.update_snapshot = False
59                 self._update_snapshot_window = False
60             if self.arm_z < 5.0:
61                 action_point = self.get_action_point()
62                 cv.circle(img, action_point, 2, (0,0,255), -1)
63             cv.imshow("Live", img)
64             cv.imshow("Snapshot", self._snapshot)
65             if self._init:
66                 cv.setMouseCallback("Snapshot", self.onMouse)
67                 self._img = deepcopy(img)
68                 self._init = False
69             cv.imshow("Highlight", self._img)
70             cv.waitKey(1)
71         except CvBridgeError as e:
72             print("Bridge-Error: {}".format(e))
73
74     def onMouse(self, event, x, y, flags, param):
75         """
76             Eventhandler for clicking on the 'Snapshot' image.
77
78             Prints the pixel coordinates of the point on which the mouse
79             clicked on the 'Snapshot' display.
80
81             Parameters:
82                 event:    The triggering event type
83                 x:        The x coordinate at which the event got
84             triggered
85                 y:        The y coordinate at which the event got
86             triggered
87                 flags:   Flags that come with the event
88                 param:  Optional parameters that belong to the
89             triggering event
90             """
91
```

```
85     if event == cv.EVENT_LBUTTONDOWN:
86         print("{} , {}" .format(x,y))
87
88     def get_action_point(self):
89         """
90             Calculates the action point depending on the current z value
91             of the arm.
92
93             For further information on the used calculations please see "
94             Metallentfernung an Dokumenten durch den Forschungsroboter Baxter"
95             by "Timo Sch rmann".
96             """
97
98             display_y = 2748.1*np.float_power(self.arm_z, 3)-789.76*np.
99             square(self.arm_z)+144.88*self.arm_z+166.8 #third order
100            polynomial trend line
101
102            display_x = -1719*np.float_power(self.arm_z, 3)+200.79*np.
103            square(self.arm_z)-8.1584*self.arm_z+374.83 #third order
104            polynomial trend line
105
106            if self.windowed:
107                display_x -= 280
108
109            return (int(display_x),int(display_y))
110
111
112    def distance_to_point(self, point):
113        """
114            Calculates the distance between the action point and a given point
115            in meter and split into x and y.
116
117            For more information on the used formula please see "
118            Metallentfernung an Dokumenten durch den Forschungsroboter Baxter"
119            by "Timo Sch rmann".
120
121            Parameters:
122                img:                                Image to be masked
123                gripper_action_point:    Action point of the used end
124                effector
125                arm_z:                                Current z value of the used end
126                effector
127
128            Return:
129                distance:                            Calculated distance in format: (x,
130                y)
131        """
132
133        factor = -9530.9 * self.arm_z + 1949.7
134        gripper_action_point = self.get_action_point()
135        distance_x = (point[0] - gripper_action_point[0]) / factor
136        distance_y = (-(point[1] - gripper_action_point[1]) / factor) -
137                      0.004
138
139        return (distance_x, distance_y)
140
```

```
119     def update_z(self, z):
120         """
121             Saves the given z value for the calculation of the action
122             point.
123
124             Parameters:
125                 z: Current z value of the gripper
126
127             self.arm_z = z
128
129     def set_highlight(self, img):
130         """
131             Saves the given image as the new highlight to be displayed.
132
133             Parameters:
134                 img: Image to be displayed as highlight
135
136             self._img = img
137
138     def write_img(self, img):
139         """
140             Saves the given image to a folder.
141             Each method call will increase the index to be used in the
142             naming of the file
143
144             Parameters:
145                 img: Image to be saved as file
146
147             cv.imwrite("/home/user/schuermann_BA/ros_ws/src/baxter_staples
148             /cv_test_images/bilder/08/pic_{}.jpg".format(self.ix), img)
149             self.ix+=1
```

Listing 3: cam_class.py

C - const_lib.py

```
1 #!/usr/bin/env python
2
3 from geometry_msgs.msg import (
4     PoseStamped,
5     Pose,
6     Point,
7     Quaternion
8 )
9
10 table_height = {'suction':-0.175, 'electric':-0.197}      #length from
11      baseframe of baxter to table. It differs depending on the gripper
12      type
13
14 gripper_base_offset = 1.0                                     #attachement
15      point for gripper is at 1 meter height when posx is 0
16
17 hover_distance = {'paper':0.38, 'tool':0.1}                  #distance
18      between gripper/tool and table
19
20 gripper_opening = 68.0                                       #opening
21      percentage for gripper to gently grab the tool
22
23
24 #Poses
25 tool_pose = {'position': Point(x=0.507, y=0.003, z=-0.194),
26             'orientation': Quaternion(x=0.000, y=0.999, z=0.000, w
27 =0.000)}
28
29 staple_pose = {'position': Point(x=0.405, y=0.108, z=-0.204),
30                 'orientation': Quaternion(x=0.000, y=0.999, z=0.000, w
31 =0.000)}
32
33 pick_paper_pose = {'position': Point(x=0.660, y=-0.150, z=-0.173),
34                     'orientation': Quaternion(x=0.000, y=0.999, z
35 =0.000, w=0.000)}
36 place_paper_pose = {'position': Point(x=0.420, y=0.137, z=-0.18),
37                     'orientation': Quaternion(x=0.999, y=0.012, z
38 =0.000, w=0.000)}
```

Listing 4: const_lib.py

D - measure_precision_along_axes.py

```
1 #!/usr/bin/env python
2 #-*- coding:utf-8 -*-
3
4 """
5 Script to measure the accuracy of a robots limb for the different
6     possible directions.
7 """
8
9
10 import argparse
11 import struct
12 import sys
13 import time
14
15 import rospy
16
17 sys.path.append("/home/user/schuermann_BA/ros_ws/src/baxter_staples/
18     scripts/base_classes")
19
20 import arm_class
21
22 from copy import deepcopy
23
24 from geometry_msgs.msg import (
25     PoseStamped,
26     Pose,
27     Point,
28     Quaternion
29 )
30
31
32 middle_pose = {
33     'left': PoseStamped(
34         pose=Pose(
35             position=Point(
36                 x=0.660,
37                 y=0.300,
38                 z=0.000,
39             ),
40             orientation=Quaternion(
41                 x=-0.000,
42                 y=0.999,
43                 z=0.000,
44                 w=0.000,
45             ),
46         ),
47     ),
48     'right': PoseStamped(
```

```
45         pose=Pose(
46             position=Point(
47                 x=0.660,
48                 y=-0.300,
49                 z=0.000,
50             ),
51             orientation=Quaternion(
52                 x=0.000,
53                 y=0.999,
54                 z=0.000,
55                 w=0.000,
56             ),
57         ),
58     )
59 }
60
61
62 def positive_x(arm, rounds):
63 """
64     Executes ten movements with a positive increment of 2cm along the
65     x axis and measures the difference between expected pose and
66     reached pose.
67
68     This motion gets repeated for a number of given rounds.
69
70     Parameter:
71         arm:          The robots limb to be measured on
72         rounds:       Number of rounds to be executed
73
74     Return:
75         True/False: False if any movement is not executable, else
76         True
77 """
78     print("--- {} arm: positive x".format(arm._limb_name))
79     raw_input("Press Enter to start...")
80     for round_counter in range(rounds):
81         #Minimize joint play
82         if arm.get_solution(arm_class.alter_pose_inc(deepcopy(
83             middle_pose[arm._limb_name].pose), arm._verbose, posx=-0.12)):
84             arm.move_to_solution()
85         else:
86             arm.simple_failsafe()
87             return False
88
89     #Initial Pose
90     initial_pose = deepcopy(middle_pose[arm._limb_name].pose)
91     if arm.get_solution(arm_class.alter_pose_inc(initial_pose,
92         verbose=arm._verbose, posx=-0.1)):
93         arm.move_to_solution()
94     else:
95         arm.simple_failsafe()
```

```
88         return False
89     print("---->Reached: initial pose\nStarting Measurements...")
90     #Readings: ['posenumber', 'setpoint', 'actual', 'difference in
91     #x', 'difference in y']
92     setpoint = list()
93     actual = list()
94     x_diff = list()
95     y_diff = list()
96     #Measuring
97     for x in range(1, 11):
98         next_pose = arm_class.alter_pose_inc(arm_class.
99         alter_pose_inc(deepcopy(initial_pose), arm._verbose, posx=x*0.02))
100        if arm.get_solution(next_pose):
101            arm.move_to_solution()
102        else:
103            arm.simplefailsafe()
104            return False
105        print("Reached Pose {}".format((int)(x)))
106        setpoint.append(next_pose.position.x)
107        actual.append(arm._current_pose.position.x)
108        x_diff.append(arm._current_pose.position.x - next_pose.
109        position.x)
110        y_diff.append(arm._current_pose.position.y - next_pose.
111        position.y)
112        print("Finished: {} arm: positive x").format(arm._limb_name)
113        print("Data following: round {}\n").format(round_counter)
114        for step_counter in range(10):
115            print("{} ,{} ,{} ,{} ,{}").format(step_counter+1, setpoint[
116            step_counter], actual[step_counter], x_diff[step_counter], y_diff[
117            step_counter])
118            print("")
119        return True
120
121    def negative_x(arm, rounds):
122        """
123            Executes ten movements with a negative increment of 2cm along the
124            x axis and measures the difference between expected pose and
125            reached pose.
126            This motion gets repeated for a number of given rounds.
127
128            Parameter:
129                arm:          The robots limb to be measured on
130                rounds:       Number of rounds to be executed
131            Return:
132                True/False: False if any movement is not executable, else
133                True
134                """
135        print("---- {} arm: negative x").format(arm._limb_name)
```

```
127     raw_input("Press Enter to start...")  
128     for round_counter in range(rounds):  
129         #Minimize joint play  
130         if arm.get_solution(arm_class.alter_pose_inc(deepcopy(  
131             middle_pose[arm._limb_name].pose), arm._verbose, posx=0.12)):  
132             arm.move_to_solution()  
133         else:  
134             arm.simplefailsafe()  
135             return False  
136         #Initial Pose  
137         initial_pose = deepcopy(middle_pose[arm._limb_name].pose)  
138         if arm.get_solution(arm_class.alter_pose_inc(initial_pose,  
139             verbose=arm._verbose, posx=0.1)):  
140             arm.move_to_solution()  
141         else:  
142             arm.simplefailsafe()  
143             return False  
144         print("--->Reached: initial pose\nStarting Measurements...")  
145         #Readings: ['posenumber', 'setpoint', 'actual', 'difference in  
146         x', 'difference in y']  
147         setpoint = list()  
148         actual = list()  
149         x_diff = list()  
150         y_diff = list()  
151         #Measuring  
152         for x in range(1, 11):  
153             next_pose = arm_class.alter_pose_inc(arm_class.  
154                 alter_pose_inc(deepcopy(initial_pose), arm._verbose, posx=-(x  
155                 *0.02)))  
156             if arm.get_solution(next_pose):  
157                 arm.move_to_solution()  
158             else:  
159                 arm.simplefailsafe()  
160                 return False  
161             print("Reached Pose {}".format((int)(x)))  
162             setpoint.append(next_pose.position.x)  
163             actual.append(arm._current_pose.position.x)  
164             x_diff.append(arm._current_pose.position.x - next_pose.  
165             position.x)  
166             y_diff.append(arm._current_pose.position.y - next_pose.  
position.y)  
167             print("Finished: {} arm: negative x").format(arm._limb_name)  
168             print("Data following: round {}\n").format(round_counter)  
169             for step_counter in range(10):  
170                 print("{} ,{} ,{} ,{} ,{}").format(step_counter+1, setpoint[  
171                     step_counter], actual[step_counter], x_diff[step_counter], y_diff[  
172                     step_counter])  
173                 print("")
```

```
166     return True
167
168 def positive_y(arm, rounds):
169     """
170         Executes ten movements with a positive increment of 2cm along the
171         y axis and measures the difference between expected pose and
172         reached pose.
173         This motion gets repeated for a number of given rounds.
174
175         Parameter:
176             arm:          The robots limb to be measured on
177             rounds:       Number of rounds to be executed
178
179         Return:
180             True/False: False if any movement is not executable, else
181             True
182
183         """
184
185     print(" --- {} arm: positive y".format(arm._limb_name))
186     raw_input("Press Enter to start...")
187     for round_counter in range(rounds):
188         #Minimize joint play
189         if arm.get_solution(arm_class.alter_pose_inc(deepcopy(
190             middle_pose[arm._limb_name].pose), arm._verbose, posy=-0.12)): # approach starting point with minimal joint play
191             arm.move_to_solution()
192         else:
193             arm.simplefailsafe()
194             return False
195
196         #Initial Pose
197         initial_pose = deepcopy(middle_pose[arm._limb_name].pose)
198         if arm.get_solution(arm_class.alter_pose_inc(initial_pose,
199             verbose=arm._verbose, posy=-0.1)):
200             arm.move_to_solution()
201         else:
202             arm.simplefailsafe()
203             return False
204
205         print(" ---> Reached: initial pose\nStarting Measurements...")
206         #Readings: ['posenumber', 'setpoint', 'actual', 'difference in
207         y', 'difference in x']
208         setpoint = list()
209         actual = list()
210         x_diff = list()
211         y_diff = list()
212
213         #Measuring
214         for x in range(1, 11):
215             next_pose = arm_class.alter_pose_inc(arm_class.
216             alter_pose_inc(deepcopy(initial_pose), arm._verbose, posy=x*0.02))
217             if arm.get_solution(next_pose):
218                 arm.move_to_solution()
```

```
206         else:
207             arm.simplefailsafe()
208             return False
209         print("Reached Pose {}".format((int)(x)))
210         setpoint.append(next_pose.position.y)
211         actual.append(arm._current_pose.position.y)
212         x_diff.append(arm._current_pose.position.x - next_pose.
position.x)
213         y_diff.append(arm._current_pose.position.y - next_pose.
position.y)
214         print("Finished: {} arm: positive y".format(arm._limb_name))
215         print("Data following: round {}\n".format(round_counter))
216         for step_counter in range(10):
217             print("{} ,{} ,{} ,{} ,{} ".format(step_counter+1, setpoint[
step_counter], actual[step_counter], y_diff[step_counter], x_diff[
step_counter]))
218             print("")
219         return True
220
221 def negative_y(arm, rounds):
222     """
223         Executes ten movements with a negative increment of 2cm along the
y axis and measures the difference between expected pose and
reached pose.
224         This motion gets repeated for a number of given rounds.
225
226         Parameter:
227             arm:          The robots limb to be measured on
228             rounds:       Number of rounds to be executed
229         Return:
230             True/False: False if any movement is not executable, else
True
231     """
232     print("--- {} arm: negative y".format(arm._limb_name))
233     raw_input("Press Enter to start...")
234     for round_counter in range(rounds):
235         #Minimize joint play
236         if arm.get_solution(arm_class.alter_pose_inc(deepcopy(
middle_pose[arm._limb_name].pose), arm._verbose, posy=0.12)): #
approach starting point with minimal joint play
237             arm.move_to_solution()
238         else:
239             arm.simplefailsafe()
240             return False
241         #Initial Pose
242         initial_pose = deepcopy(middle_pose[arm._limb_name].pose)
243         if arm.get_solution(arm_class.alter_pose_inc(initial_pose,
verbose=arm._verbose, posy=0.1)):
```

```
244         arm.move_to_solution()
245     else:
246         arm.simplefailsafe()
247     return False
248     print("---->Reached: initial pose\nStarting Measurements...")
249     #Readings: ['posenumber', 'setpoint', 'actual', 'difference in
y', 'difference in x']
250     setpoint = list()
251     actual = list()
252     x_diff = list()
253     y_diff = list()
254     #Measuring
255     for x in range(1, 11):
256         next_pose = arm_class.alter_pose_inc(arm_class.
alter_pose_inc(deepcopy(initial_pose), arm._verbose, posy=-(x
*0.02)))
257         if arm.get_solution(next_pose):
258             arm.move_to_solution()
259         else:
260             arm.simplefailsafe()
261         return False
262         print("Reached Pose {}".format((int) (x)))
263         setpoint.append(next_pose.position.y)
264         actual.append(arm._current_pose.position.y)
265         x_diff.append(arm._current_pose.position.x - next_pose.
position.x)
266         y_diff.append(arm._current_pose.position.y - next_pose.
position.y)
267         print("Finished: {} arm: negative y").format(arm._limb_name)
268         print("Data following: round {}\n").format(round_counter)
269         for step_counter in range(10):
270             print("{} ,{} ,{} ,{} ,{}").format(step_counter+1, setpoint[
step_counter], actual[step_counter], y_diff[step_counter], x_diff[
step_counter])
271             print("")
272     return True
273
274
275 def main():
276     try:
277         # Argument Parsing
278         arg_fmt = argparse.RawDescriptionHelpFormatter
279         parser = argparse.ArgumentParser(formatter_class=arg_fmt,
description=main.__doc__)
280
281         parser.add_argument(
282             '-v', '--verbose',
283             action='store_const',
```

```
284         const=True ,
285         default=False ,
286         help="displays debug information (default = False)"
287     )
288     parser.add_argument(
289         '-l', '--limb',
290         choices=['left', 'right'],
291         required=True,
292         #default='right',
293         help='the limb to run the measurements on'
294     )
295     args = parser.parse_args(rospy.myargv()[1:])
296
297     #Init
298     rospy.init_node("measure_precision", anonymous = True)
299     time.sleep(0.5)
300     print("--- Ctrl-D stops the program ---")
301     print("Init started...")
302     arm = arm_class.Arm(args.limb, args.verbose)
303     print("Init finished...")
304
305     #print(arm_class.convert_to_pose(arm._limb.endpoint_pose()))
306
307     #Move to neutral Pose
308     arm.set_neutral()
309
310     #Measurements
311     print("Starting measurements...")
312     rounds = 10
313     if not positive_x(arm, rounds):
314         return False
315     if not negative_x(arm, rounds):
316         return False
317     if not positive_y(arm, rounds):
318         return False
319     if not negative_y(arm, rounds):
320         return False
321
322     print("\nMeasurements finished...\nExiting program...")
323     arm.simplefailsafe()
324
325     except rospy.ROSInterruptException as e:
326         return e
327     except KeyboardInterrupt as e:
328         return e
329
330 if __name__ == '__main__':
331     main()
```

Listing 5: `measure_precision_along_axes.py`

E - measure_step_speed.py

```
1 #!/usr/bin/env python
2 #-*- coding:utf-8 -*-
3
4 """
5 A simple script to measure the accuracy of the robots arm movements
6     depending on different increments and speed configurations.
7
8 The chosen arm approaches a hardcoded pose multiple times with
9     different configurations and prints the outcome in a neat way to
10    import it in a spreadsheet program as Microsoft Excel to analyze.
11 """
12
13
14
15 import argparse
16 import struct
17 import sys
18 import time
19
20 import rospy
21
22 sys.path.append("/home/user/schuermann_BA/ros_ws/src/baxter_staples/
23                 scripts/base_classes")
24 import arm_class
25
26
27 from copy import deepcopy
28
29
30 start_pose = {
31         'left': PoseStamped(
32             pose=Pose(
33                 position=Point(
34                     x=0.660,
35                     y=0.300,
36                     z=0.000,
37                 ),
38                 orientation=Quaternion(
39                     x=-0.000,
40                     y=0.999,
41                     z=0.000,
42                     w=0.000,
```

```
43         ),
44         ),
45     ),
46     'right': PoseStamped(
47         pose=Pose(
48             position=Point(
49                 x=0.660,
50                 y=-0.300,
51                 z=0.000,
52             ),
53             orientation=Quaternion(
54                 x=0.000,
55                 y=0.999,
56                 z=0.000,
57                 w=0.000,
58             ),
59         ),
60     )
61 }
62
63
64 def measure_positive_x(arm, step_width = 0.01, speed = 0.3):
65 """
66     Approaches one point 20 times and measures the difference between
67     given pose and reached pose.
68
69     The approaches contain two movements. The first is to minimize
70     joint play, whereas the second is to reach the measurement pose.
71
72     Parameter:
73         arm:          The robots limb to perform the motion
74         step_width:  Width of the steps to be made in meter;
75     Default: 0.01
76         speed:        Speed at which the arm shall move; Default:
77         0.3; Range: 0.0-1.0
78
79     Return:
80         diff_dict:  Dictionary containing the measured data.
81     Structure: diff_dict['x' : list(), 'y' : list()]
82 """
83
84     print("--- {} arm: positive x".format(arm._limb_name))
85     arm._limb.set_joint_position_speed(speed)
86     diff_dict = dict()
87     diff_dict['x'] = list()
88     diff_dict['y'] = list()
89     for round_counter in range(20):
90         #Minimize joint play
91         if not arm.move_to_pose(arm_class.alter_pose_inc(start_pose[
92             arm._limb_name].pose, arm._verbose, posx=(step_width * -2))):
```

```
85         arm.simplefailsafe()
86     return False
87 #Initial pose
88     initial_pose = deepcopy(start_pose[arm._limb_name].pose)
89     if not arm.move_to_pose(arm_class.alter_pose_inc(initial_pose,
90     verbose=arm._verbose, posx=-(step_width))):
91         arm.simplefailsafe()
92     return False
93     print("---->Reached: initial pose\nStarting Measurements...")
94 #Measuring
95     next_pose = arm_class.alter_pose_inc(arm_class.alter_pose_inc(
96     initial_pose, arm._verbose, posx=step_width))
97     if not arm.move_to_pose(next_pose):
98         arm.simplefailsafe()
99     return False
100    diff_dict['x'].append(arm._current_pose.position.x - next_pose
101    .position.x)
102    diff_dict['y'].append(arm._current_pose.position.y - next_pose
103    .position.y)
104    return diff_dict
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127 def main():
    try:
        # Argument Parsing
        arg_fmt = argparse.RawDescriptionHelpFormatter
        parser = argparse.ArgumentParser(formatter_class=arg_fmt,
description=main.__doc__)
        parser.add_argument(
            '-v', '--verbose',
            action='store_const',
            const=True,
            default=False,
            help="displays debug information (default = False)"
        )
        parser.add_argument(
            '-l', '--limb',
            choices=['left', 'right'],
            required=True,
            help='the limb to run the measurements on'
        )
        args = parser.parse_args(rospy.myargv()[1:])
        #Init
        rospy.init_node("measure_precision", anonymous = True)
        time.sleep(0.5)
        print("Init started...")
        arm = arm_class.Arm(args.limb, args.verbose)
```

```
128     print("Init finished...")  
129  
130     #Move to neutral Pose  
131     arm.set_neutral()  
132  
133     #Measurements  
134     print("Starting measurements...")  
135     slow_short = measure_positive_x(arm, step_width=0.01, speed  
136     =0.1)  
137     slow_medium = measure_positive_x(arm, step_width=0.02, speed  
138     =0.1)  
139     slow_far = measure_positive_x(arm, step_width=0.03, speed=0.1)  
140     fast_short = measure_positive_x(arm, step_width=0.01, speed  
141     =0.3)  
142     fast_medium = measure_positive_x(arm, step_width=0.02, speed  
143     =0.3)  
144     fast_far = measure_positive_x(arm, step_width=0.03, speed=0.3)  
145  
146     #Print data  
147     print("slow_short,slow_medium,slow_far,fast_short,fast_medium,  
148     fast_far")  
149     print("x")  
150     for k in range(len(slow_short['x'])):  
151         print("{} ,{} ,{} ,{} ,{} ,{} ".format(slow_short['x'][k],  
152             slow_medium['x'][k], slow_far['x'][k], fast_short['x'][k],  
153             fast_medium['x'][k], fast_far['x'][k]))  
154     print("y")  
155     for k in range(len(slow_short['y'])):  
156         print("{} ,{} ,{} ,{} ,{} ,{} ".format(slow_short['y'][k],  
157             slow_medium['y'][k], slow_far['y'][k], fast_short['y'][k],  
158             fast_medium['y'][k], fast_far['y'][k]))  
159  
160     print("\nMeasurements finished...\nExiting program...")  
161     arm.simple_failsafe()  
162  
163  
164     except rospy.ROSInterruptException as e:  
165         return e  
166     except KeyboardInterrupt as e:  
167         return e  
168  
169 if __name__ == '__main__':  
170     main()
```

Listing 6: measure_step_speed.py

F - measure_precision_workspace.py

```
1 #!/usr/bin/env python
2 #-*- coding:utf-8 -*-
3
4 """
5 Script to measure the robots accuracy in a specified workspace with
6 rectangular shape.
7 The default size of this workspace equals that of a DIN A4 document.
8 For this the robot approaches a number of points arranged in a matrix
9     and prints the measured data in a neat way to be importet into a
10    spreadsheet program.
11 """
12
13
14
15 import argparse
16 import struct
17 import sys
18 import time
19
20
21 import rospy
22 from numpy import arange
23
24 sys.path.append("/home/user/schuermann_BA/ros_ws/src/baxter_staples/
25                 scripts/base_classes")
26 import arm_class
27
28
29
30 start_pose = {
31     'left': PoseStamped(
32         pose=Pose(
33             position=Point(
34                 x=0.380,
35                 y=0.100,
36                 z=-0.150,
37             ),
38             orientation=Quaternion(
39                 x=-0.000,
40                 y=0.999,
41                 z=0.000,
```

```
43             w=0.000,
44         ),
45     ),
46 ),
47 'right': PoseStamped(
48     pose=Pose(
49         position=Point(
50             x=0.660,
51             y=-0.300,
52             z=0.000,
53         ),
54         orientation=Quaternion(
55             x=0.000,
56             y=0.999,
57             z=0.000,
58             w=0.000,
59         ),
60     ),
61 )
62 }
63
64 def positive_x(arm, step_width, workspace_x=0.297, workspace_y=0.210):
65 """
66     Measure the accuracy of the given robots limb in a workspace of
67     given size.
68     The number of measurement points depends on the relationship
69     between workspace size and step width.
70
71     Parameter:
72         arm:           The robots limb to be measured on
73         step_width:    Distance between the measurement points
74         workspace_x:  Size of the workspace along the x axis
75         workspace_y:  Size of the workspace along the y axis
76     Return:
77         out_dict:      Dictionary that contains the measured data
78 ; Structure: outdict['x{}y{}'] = Point
79 """
80     print("---- {} arm: positive x").format(arm._limb_name)
81     #raw_input("Press Enter to start...")
82     out_dict = dict()
83     if step_width < 0.02:
84         #Initial Pose
85         if arm.get_solution(arm_class.alter_pose_inc(deepcopy(
86             start_pose[arm._limb_name].pose), arm._verbose, posx=-(step_width
87             *2))): #approach starting point with minimal joint play
88             arm.move_to_solution()
89         else:
90             arm.simplefailsafe()
```

```
86         return False
87     initial_pose = deepcopy(start_pose[arm._limb_name].pose)
88     if arm.get_solution(arm_class.alter_pose_inc(initial_pose,
89         verbose=arm._verbose, posx=-(step_width))):
90         arm.move_to_solution()
91     else:
92         arm.simplefailsafe()
93     return False
94 print("---->Reached: initial pose\nStarting Measurements...")
95 #Measuring
96 next_pose = deepcopy(initial_pose)
97 for y_step in arange(initial_pose.position.y, initial_pose.
98     position.y+workspace_y, step_width):
99     next_pose = arm_class.alter_pose_abs(next_pose, verbose=
100         arm._verbose, posy=y_step)
101     for x_step in arange(initial_pose.position.x, initial_pose.
102         position.x+workspace_x, step_width):
103         next_pose = arm_class.alter_pose_abs(next_pose,
104             verbose=arm._verbose, posx=x_step)
105         if arm.get_solution(next_pose):
106             arm.move_to_solution()
107         else:
108             arm.simplefailsafe()
109         return False
110     diff = Point(
111         x = arm._current_pose.position.x - next_pose.
112         position.x,
113         y = arm._current_pose.position.y - next_pose.
114         position.y,
115         z = arm._current_pose.position.z - next_pose.
116         position.z
117     )
118     out_dict['x{}y{}'.format(((int)(next_pose.position.x
119         *100)), ((int)(next_pose.position.y*100)))] = diff
120 else:
121     next_pose = deepcopy(start_pose[arm._limb_name].pose)
122     for y_step in arange(start_pose[arm._limb_name].pose.position.
123         y, start_pose[arm._limb_name].pose.position.y+workspace_y,
124         step_width):
125         next_pose = arm_class.alter_pose_abs(next_pose, verbose=
126             arm._verbose, posy=y_step)
127         for x_step in arange(start_pose[arm._limb_name].pose.
128             position.x, start_pose[arm._limb_name].pose.position.x+workspace_x
129             , step_width):
130             next_pose = arm_class.alter_pose_abs(next_pose,
131                 verbose=arm._verbose, posx=x_step)
132             if not arm.move_precise(next_pose):
133                 arm.simplefailsafe()
```

```
119         return False
120
121         diff = Point(
122             x = arm._current_pose.position.x - next_pose.
123             position.x,
124             y = arm._current_pose.position.y - next_pose.
125             position.y,
126             z = arm._current_pose.position.z - next_pose.
127             position.z
128         )
129         out_dict['y{}x{}'.format(((int)(next_pose.position.y
130 *100)), ((int)(next_pose.position.x*100)))] = diff
131         print("row {} finished".format(y_step))
132
133     #Print data
134     print("Pose,x_diff[mm],y_diff[mm],z_diff[mm],,")
135     for k in out_dict.keys():
136         print("{}{},{}{},{}{},,".format(k, out_dict[k].x*1000, out_dict[
137             k].y*1000, out_dict[k].z*1000))
138
139     return out_dict
140
141
142 def main():
143     try:
144         # Argument Parsing
145         arg_fmt = argparse.RawDescriptionHelpFormatter
146         parser = argparse.ArgumentParser(formatter_class=arg_fmt,
147                                         description=main.__doc__)
148
149         parser.add_argument(
150             '-v', '--verbose',
151             action='store_const',
152             const=True,
153             default=False,
154             help="displays debug information (default = False)"
155         )
156         parser.add_argument(
157             '-l', '--limb',
158             choices=['left', 'right'],
159             required=True,
160             #default='right',
161             help='the limb to run the measurements on'
162         )
163         args = parser.parse_args(rospy.myargv()[1:])
164
165         #Init
166         rospy.init_node("measure_precision", anonymous = True)
167         time.sleep(0.5)
168         print(" --- Ctrl-D stops the program ---")
169         print("Init started...")
170         arm = arm_class.Arm(args.limb, args.verbose)
```

```
161     lut = dict()
162     lut[args.limb] = dict()
163     print("Init finished...")
164
165     #Move to neutral Pose
166     arm.set_neutral()
167
168     #Measurements
169     print("Starting measurements...")
170     for k in range(10):
171         print("-----> Runde: {}".format(k))
172         lut[args.limb]['x_pos'] = positive_x(arm, step_width=0.03,
173         workspace_x=0.297, workspace_y=0.210)
174         if lut[args.limb]['x_pos'] is False:
175             return False
176
177         print("\nMeasurements finished...\nExiting program...")
178         arm.simplefailsafe()
179
180         #print lut
181
182     except rospy.ROSInterruptException as e:
183         return e
184     except KeyboardInterrupt as e:
185         return e
186
187 if __name__ == '__main__':
188     main()
```

Listing 7: measure_precision_workspace.py

G - lut_interface.py

```
1 #!/usr/bin/env python
2 #-*- coding:utf-8 -*-
3
4 """
5 Interface for the handling of the LookUpTable-Datatype (LUT).
6
7 This interface is capable of following actions:
8 - Creating a LUT from scratch
9 - Creating a LUT from file
10 - Printing a LUT
11 - Writing a LUT as a .csv-file
12 - Apply LUT to a given pose
13
14 A LUT is a multilayered dictionary object representing a matrix of
15     measure points across a rectangular workspace.
16 Each value of the matrix holds the average deviation for this pose of
17     one of the robots limbs.
18 For further information please see "Metallentfernung an Dokumenten
19     durch den Forschungsroboter Baxter" by "Timo Schermann".
20 """
21
22
23
24 import argparse
25 import struct
26 import sys
27 import time
28
29
30 import rospy
31 from numpy import arange
32
33 sys.path.append("/home/user/schuermann_BA/ros_ws/src/baxter_staples/
34     scripts/base_classes")
35 import arm_class
36
37 from copy import deepcopy
38
39
40 from geometry_msgs.msg import (
41     PoseStamped,
42     Pose,
43     Point,
44     Quaternion
45 )
46
47
48 start_pose = {
49     'left': PoseStamped(
50         pose=Pose(
```

```
43         position=Point(
44             x=0.380,
45             y=0.100,
46             z=-0.150,
47         ),
48         orientation=Quaternion(
49             x=-0.000,
50             y=0.999,
51             z=0.000,
52             w=0.000,
53         ),
54     ),
55 ),
56 'right': PoseStamped(
57     pose=Pose(
58         position=Point(
59             x=0.660,
60             y=-0.300,
61             z=0.000,
62         ),
63         orientation=Quaternion(
64             x=0.000,
65             y=0.999,
66             z=0.000,
67             w=0.000,
68         ),
69     ),
70 )
71 }
72
73
74 def create_lut(arm, filename, number_of_rounds=10, verbose=False):
75 """
76     Creates a LUT for a specific workspace shaped according to DIN A4.
77
78     This function moves the arm along a matrix of points across the
79     workspace and measures the deviation.
80     The measured deviation will be averaged for each point.
81     Please consider, that 10 rounds will need around 50 minutes with a
82     step_width of 0.03.
83
84     Parameter:
85         arm:                      Robots limb to create a LUT for
86         filename:                  Absolute path to where the created LUT
87         shall be saved
88         number_of_rounds:        Number of rounds to go through for the
89         creation
```

```
86         verbose:           True -> Every 10 rounds an
intermediate LUT will be saved to the filelocation
87     Return:
88         lut_data:           Created LUT-Object; False if creation
failed
89     """
90     lut_data = dict()
91     lut_data[arm._limb_name] = dict()
92     lut_data[arm._limb_name]['x_pos'] = dict()
93     print("Starting measurements...")
94     for r in range(number_of_rounds):
95         lut_data[arm._limb_name]['x_pos'] = measure_positive_x(arm
=arm, data_dict=lut_data[arm._limb_name]['x_pos'], step_width
=0.03, workspace_x=0.297, workspace_y=0.210)
96         if lut_data[arm._limb_name]['x_pos'] is False:
97             return False
98         print("-----> Round: {}".format(r+1))
99         if verbose and (r == 0 or (r+1)%10 == 0) and r <
number_of_rounds-1:
100             temp_lut = deepcopy(lut_data)
101             print("averaging newest measurements...")
102             for y in temp_lut[arm._limb_name]['x_pos'].keys():
103                 for x in temp_lut[arm._limb_name]['x_pos'][y].keys():
104                     temp_lut[arm._limb_name]['x_pos'][y][x].x /= (
r+1)
105                     temp_lut[arm._limb_name]['x_pos'][y][x].y /= (
r+1)
106                     temp_lut[arm._limb_name]['x_pos'][y][x].z /= (
r+1)
107                     time.sleep(0.01) #without this the
calculations for z might generate strange values
108                     write_lut_as_csv(lut=temp_lut, number_of_rounds=r+1,
filename=filename+"_{:}_steps".format(r+1))
109                     print("averaging measurements...")
110                     for y in lut_data[arm._limb_name]['x_pos'].keys():
111                         for x in lut_data[arm._limb_name]['x_pos'][y].keys():
112                             lut_data[arm._limb_name]['x_pos'][y][x].x /= (r+1)
113                             lut_data[arm._limb_name]['x_pos'][y][x].y /= (r+1)
114                             lut_data[arm._limb_name]['x_pos'][y][x].z /= (r+1)
115                             time.sleep(0.01) #without this the calculations for z
might generate strange values
116                     write_lut_as_csv(lut=lut_data, number_of_rounds=number_of_rounds,
filename=filename+"_final")
117                     return lut_data
118
119 #/home/user/schuermann_BA/ros_ws/src/baxter_staples/precision/
my_first_lut/measurements/calibrated/lut
```

```
120
121 def measure_positive_x(arm, data_dict=dict(), step_width=0.03,
122     workspace_x=0.297, workspace_y=0.210):
123     """
124         Measure the accuracy of the given robots limb in a workspace of
125         given size.
126         The number of measurement points depends on the relationship
127         between workspace size and step width.
128         The measured data will be added to that of the given dictionary.
129
130         Parameter:
131             arm:                 The robots limb to run the measurements on
132             data_dict:           The dictionary to be modified; Default:
133                 New dictionary will be created
134             step_width:          Distance between the measure points;
135             Minimum: 0.02; Default: 0.03
136             workspace_x:        Size of the workspace along the x axis
137             workspace_y:        Size of the workspace along the y axis
138         Return:
139             data_dict:           Modified dictionary of measured data;
140             False if any movement failed
141
142         """
143
144     print("--- {} arm: positive x".format(arm._limb_name))
145     diff = Point()
146     if step_width < 0.02:
147         print("running this script with a step_width < 0.02 will need
148             a lot of time and is therefore not supported in this version...")
149         return False
150     next_pose = deepcopy(start_pose[arm._limb_name].pose)
151     for y_step in arange(start_pose[arm._limb_name].pose.position.y,
152         start_pose[arm._limb_name].pose.position.y+workspace_y, step_width):
153
154         next_pose = arm_class.alter_pose_abs(next_pose, verbose=arm._verbose,
155             posy=y_step)
156         y_name = 'y{}'.format(int(y_step*100))
157         if not y_name in data_dict.keys():
158             data_dict[y_name] = dict()
159         for x_step in arange(start_pose[arm._limb_name].pose.position.x,
160             start_pose[arm._limb_name].pose.position.x+workspace_x,
161             step_width):
162
163             x_name = 'x{}'.format(int(x_step*100))
164             next_pose = arm_class.alter_pose_abs(next_pose, verbose=arm._verbose,
165                 posx=x_step)
166             if not arm.move_precise(next_pose):
167                 arm.simplefailsafe()
168                 return False
169             time.sleep(0.1)
```

```
154         diff.x = arm._current_pose.position.x - next_pose.position
155         .x
156         diff.y = arm._current_pose.position.y - next_pose.position
157         .y
158         diff.z = arm._current_pose.position.z - next_pose.position
159         .z
160         if not x_name in data_dict[y_name].keys():
161             data_dict[y_name][x_name] = deepcopy(diff)
162         else:
163             data_dict[y_name][x_name].x += diff.x
164             data_dict[y_name][x_name].y += diff.y
165             data_dict[y_name][x_name].z += diff.z
166             print("row finished: {} / {}".format(y_step, start_pose[arm.
167             _limb_name].pose.position.y+workspace_y))
168
169     return data_dict
170
171
172 def print_lut_as_csv(lut, number_of_rounds):
173     """
174     Prints the given LUT and number of rounds in the same format as it
175     would look in a file.
176
177     Parameter:
178         lut:                      LUT-Object to be printed
179         number_of_rounds:        Number of rounds used for the creation
180         of this LUT
181     """
182     print("\n{}".format(number_of_rounds))
183     for arm in lut.keys():
184         print(arm)
185         for way in lut[arm].keys():
186             print(way)
187             for y in lut[arm][way].keys():
188                 for x in lut[arm][way][y].keys():
189                     print("{} , {} , {} , {} , {} , {} ".format(y, x, lut[arm][way][
190                         y][x].x, lut[arm][way][y][x].y, lut[arm][way][y][x].z))
191
192 def write_lut_as_csv(lut, number_of_rounds, filename):
193     """
194     Writes the given LUT to a .csv-file.
195     This file can be imported to a spreadsheet program or read back in
196     with 'restore_lut_from_file()' .
197
198     Parameter:
199         lut:                      The LUT-Object to be saved
200         number_of_rounds:        Number of rounds used for the creation
201         of this LUT
202         filename:                Absolute path to where the given LUT
203         shall be saved
```

```
192 """
193     writefile = open(filename, 'w')
194     writefile.write("{}\n".format(number_of_rounds))
195     for arm in lut.keys():
196         writefile.write("{}\n".format(arm))
197         for way in lut[arm].keys():
198             writefile.write("{}\n".format(way))
199             for y in lut[arm][way].keys():
200                 for x in lut[arm][way][y].keys():
201                     writefile.writelines("{} ,{} ,{} ,{} ,{}\n".format(y,
202                                         x, lut[arm][way][y][x].x, lut[arm][way][y][x].y, lut[arm][way][y][x].z))
203     writefile.close()
204     print("data written to file")
205
206 def restore_lut_from_file(filename):
207     """
208     Creates a LUT-Object from a suitable file.
209
210     Parameter:
211         filename:           Absolute path to the file to be read
212     Return:
213         restored_data:   Created LUT-Object
214     """
215
216     readfile = open(filename, 'r')
217     lut = dict()
218     lut_string = readfile.readlines()
219     arm = lut_string[1].strip()
220     way = lut_string[2].strip()
221     lut[arm] = dict()
222     lut[arm][way] = dict()
223     for p in range(3, len(lut_string)):
224         string_list = lut_string[p].split(',')
225         if not string_list[0] in lut[arm][way].keys():
226             lut[arm][way][string_list[0]] = dict()
227             lut[arm][way][string_list[0]][string_list[1]] = Point(
228                 x = float(string_list[2]),
229                 y = float(string_list[3]),
230                 z = float(string_list[4]))
231     restored_data = dict()
232     restored_data['number_of_rounds'] = lut_string[0]
233     restored_data['lut'] = lut
234     return restored_data
235
236 def improve_pose(pose, lut, limb_name = 'left'):
237     """
```

```
237     Applies the data of the given LUT-Object to the given pose to
238     improve its accuracy.
239
240     Parameter:
241         pose:          Pose to be improved
242         lut:           LUT to improve pose with
243         limb_name:    Name of the robots limb on which the LUT got
244         created
245
246     Return:
247         pose:          Improved pose
248     """
249
250     #Create possibilities
251     y_name = "y{}".format(int(pose.position.y*100))
252     y_name_plus = "y{}".format(int(pose.position.y*100)+1)
253     y_name_minus = "y{}".format(int(pose.position.y*100)-1)
254     x_name = "x{}".format(int(pose.position.x*100))
255     x_name_plus = "x{}".format(int(pose.position.x*100)+1)
256     x_name_minus = "x{}".format(int(pose.position.x*100)-1)
257
258     #poses y resembles point in LUT
259     if y_name in lut[limb_name]['x_pos'].keys():
260
261         #poses x resembles point in LUT
262         if x_name in lut[limb_name]['x_pos'][y_name].keys():
263             x_diff = lut[limb_name]['x_pos'][y_name][x_name].x
264             y_diff = lut[limb_name]['x_pos'][y_name][x_name].y
265             z_diff = lut[limb_name]['x_pos'][y_name][x_name].z
266
267         #poses x slightly smaller than point in LUT
268         elif x_name_plus in lut[limb_name]['x_pos'][y_name].keys():
269             x_name_minus_2 = "x{}".format(int(pose.position.x*100)-2)
270             x_diff = ((lut[limb_name]['x_pos'][y_name][x_name_plus].x
271 *2)+lut[limb_name]['x_pos'][y_name][x_name_minus_2].x)/3
272             y_diff = ((lut[limb_name]['x_pos'][y_name][x_name_plus].y
273 *2)+lut[limb_name]['x_pos'][y_name][x_name_minus_2].y)/3
274             z_diff = ((lut[limb_name]['x_pos'][y_name][x_name_plus].z
275 *2)+lut[limb_name]['x_pos'][y_name][x_name_minus_2].z)/3
276
277         #poses x slightly bigger than point in LUT
278         elif x_name_minus in lut[limb_name]['x_pos'][y_name].keys():
279             x_name_plus_2 = "x{}".format(int(pose.position.x*100)+2)
280             x_diff = ((lut[limb_name]['x_pos'][y_name][x_name_minus].x
281 *2)+lut[limb_name]['x_pos'][y_name][x_name_plus_2].x)/3
282             y_diff = ((lut[limb_name]['x_pos'][y_name][x_name_minus].y
283 *2)+lut[limb_name]['x_pos'][y_name][x_name_plus_2].y)/3
284             z_diff = ((lut[limb_name]['x_pos'][y_name][x_name_minus].z
285 *2)+lut[limb_name]['x_pos'][y_name][x_name_plus_2].z)/3
286
287         else:
288             print("improve_pose: given pose not in improvable
289 workspace")
290
291     return pose
292
293     #poses y slightly smaller than point in LUT
```

```
276     elif y_name_plus in lut[limb_name]['x_pos'].keys():
277         #poses x resembles point in LUT
278         if x_name in lut[limb_name]['x_pos'][y_name_plus].keys():
279             x_diff = lut[limb_name]['x_pos'][y_name_plus][x_name].x
280             y_diff = lut[limb_name]['x_pos'][y_name_plus][x_name].y
281             z_diff = lut[limb_name]['x_pos'][y_name_plus][x_name].z
282             #poses x slightly smaller than point in LUT
283             elif x_name_plus in lut[limb_name]['x_pos'][y_name_plus].keys():
284                 x_name_minus_2 = "x{}".format(int(pose.position.x*100)-2)
285                 x_diff = ((lut[limb_name]['x_pos'][y_name_plus][
286                     x_name_plus].x*2)+lut[limb_name]['x_pos'][y_name_plus][
287                     x_name_minus_2].x)/3
288                 y_diff = ((lut[limb_name]['x_pos'][y_name_plus][
289                     x_name_plus].y*2)+lut[limb_name]['x_pos'][y_name_plus][
290                     x_name_minus_2].y)/3
291                 z_diff = ((lut[limb_name]['x_pos'][y_name_plus][
292                     x_name_plus].z*2)+lut[limb_name]['x_pos'][y_name_plus][
293                     x_name_minus_2].z)/3
294                 #poses x slightly bigger than point in LUT
295                 elif x_name_minus in lut[limb_name]['x_pos'][y_name_plus].keys():
296                     x_name_plus_2 = "x{}".format(int(pose.position.x*100)+2)
297                     x_diff = ((lut[limb_name]['x_pos'][y_name_plus][
298                         x_name_minus].x*2)+lut[limb_name]['x_pos'][y_name_plus][
299                         x_name_plus_2].x)/3
300                     y_diff = ((lut[limb_name]['x_pos'][y_name_plus][
301                         x_name_minus].y*2)+lut[limb_name]['x_pos'][y_name_plus][
302                         x_name_plus_2].y)/3
303                     z_diff = ((lut[limb_name]['x_pos'][y_name_plus][
304                         x_name_minus].z*2)+lut[limb_name]['x_pos'][y_name_plus][
305                         x_name_plus_2].z)/3
306                     else:
307                         print("improve_pose: given pose not in improvable
308 workspace")
309                         return pose
#Given poses y is slightly bigger than point in the LUT
310             elif y_name_minus in lut[limb_name]['x_pos'].keys():
311                 #poses x resembles point in LUT
312                 if x_name in lut[limb_name]['x_pos'][y_name_minus].keys():
313                     x_diff = lut[limb_name]['x_pos'][y_name_minus][x_name].x
314                     y_diff = lut[limb_name]['x_pos'][y_name_minus][x_name].y
315                     z_diff = lut[limb_name]['x_pos'][y_name_minus][x_name].z
316                     #poses x slightly smaller than point in LUT
317                     elif x_name_plus in lut[limb_name]['x_pos'][y_name_minus].keys():
318                         x_name_minus_2 = "x{}".format(int(pose.position.x*100)-2)
```

```
307         x_diff = ((lut[limb_name]['x_pos'][y_name_minus][
308             x_name_plus].x*2)+lut[limb_name]['x_pos'][y_name_minus][
309                 x_name_minus_2].x)/3
310         y_diff = ((lut[limb_name]['x_pos'][y_name_minus][
311             x_name_plus].y*2)+lut[limb_name]['x_pos'][y_name_minus][
312                 x_name_minus_2].y)/3
313         z_diff = ((lut[limb_name]['x_pos'][y_name_minus][
314             x_name_plus].z*2)+lut[limb_name]['x_pos'][y_name_minus][
315                 x_name_minus_2].z)/3
316         #poses x slightly bigger than point in LUT
317         elif x_name_minus in lut[limb_name]['x_pos'][y_name_minus].keys():
318             x_name_plus_2 = "x{}".format(int(pose.position.x*100)+2)
319             x_diff = ((lut[limb_name]['x_pos'][y_name_minus][
320                 x_name_minus].x*2)+lut[limb_name]['x_pos'][y_name_minus][
321                     x_name_plus_2].x)/3
322             y_diff = ((lut[limb_name]['x_pos'][y_name_minus][
323                 x_name_minus].y*2)+lut[limb_name]['x_pos'][y_name_minus][
324                     x_name_plus_2].y)/3
325             z_diff = ((lut[limb_name]['x_pos'][y_name_minus][
326                 x_name_minus].z*2)+lut[limb_name]['x_pos'][y_name_minus][
327                     x_name_plus_2].z)/3
328         else:
329             print("improve_pose: given pose not in improvable
330 workspace")
331             return pose
332         else:
333             print("improve_pose: given pose not in improvable workspace")
334             return
335         return arm_class.alter_pose_inc(pose, posx=-x_diff, posy=-y_diff,
336                                         posz=-z_diff)
337
338 def main():
339     try:
340         # Argument Parsing
341         arg_fmt = argparse.RawDescriptionHelpFormatter
342         parser = argparse.ArgumentParser(formatter_class=arg_fmt,
343                                         description=main.__doc__)
344
345         parser.add_argument(
346             '-v', '--verbose',
347             action='store_const',
348             const=True,
349             default=False,
350             help="displays debug information (default = False)"
351         )
352         parser.add_argument(
353             '-l', '--limb',
```

```
339         choices=[ 'left' , 'right' ] ,
340         #required=True ,
341         default='left' ,
342         help='the limb to run the measurements on'
343     )
344     args = parser.parse_args(rospy.myargv()[1:])
345
346     #Init
347     rospy.init_node("lut_interface", anonymous = True)
348     time.sleep(0.5)
349     print(" --- Ctrl-D stops the program ---")
350     print("Init started...")
351     arm = arm_class.Arm(args.limb, args.verbose)
352     print("Init finished...")
353
354     #Move to neutral Pose
355     arm.set_neutral()
356
357     create_lut(arm=arm, filename="/home/user/schuermann_BA/ros_ws/
src/baxter_staples/precision/my_first_lut/lut.csv",
number_of_rounds=50)
358
359     print("\nMeasurements finished...\nExiting program...")
360     arm.simplefailsafe()
361
362 except rospy.ROSInterruptException as e:
363     return e
364 except KeyboardInterrupt as e:
365     return e
366
367 if __name__ == '__main__':
368     main()
```

Listing 8: lut_interface.py

H - test_if_precision_improved.py

```
1 #!/usr/bin/env python
2 #-*- coding:utf-8 -*-
3
4 import argparse
5 import struct
6 import sys
7 import time
8
9 import rospy
10 from numpy import arange
11
12 sys.path.append("/home/user/schuermann_BA/ros_ws/src/baxter_staples/
13     scripts/base_classes")
14 import arm_class
15 import lut_interface
16
17 from copy import deepcopy
18
19 from geometry_msgs.msg import (
20     PoseStamped,
21     Pose,
22     Point,
23     Quaternion
24 )
25
26 start_pose = {
27     'left': PoseStamped(
28         pose=Pose(
29             position=Point(
30                 x=0.380,
31                 y=0.100,
32                 z=-0.150,
33             ),
34             orientation=Quaternion(
35                 x=-0.000,
36                 y=0.999,
37                 z=0.000,
38                 w=0.000,
39             ),
40         ),
41     'right': PoseStamped(
42         pose=Pose(
43             position=Point(
44                 x=0.660,
45                 y=-0.300,
```

```
46             z=0.000,
47         ),
48         orientation=Quaternion(
49             x=0.000,
50             y=0.999,
51             z=0.000,
52             w=0.000,
53         ),
54     ),
55 )
56 }
57
58 def save_data(arm, pose, filename):
59 """
60     Adds the distance between the given pose and the current pose of
61     the given arm to the given file.
62
63     Parameter:
64         arm:           Robots limb to get current pose from
65         pose:          Pose to compare to current pose
66         filename:      Absolute path to the file to be modified
67 """
68
69     writefile = open(filename, 'a')
70     x_diff = (arm._current_pose.position.x - pose.position.x)*1000
71     y_diff = (arm._current_pose.position.y - pose.position.y)*1000
72     z_diff = (arm._current_pose.position.z - pose.position.z)*1000
73     writefile.write("{}\n".format(x_diff, y_diff, z_diff))
74     writefile.close()
75     print("saved data to {}".format(filename))
76
77 def main():
78     try:
79         # Argument Parsing
80         arg_fmt = argparse.RawDescriptionHelpFormatter
81         parser = argparse.ArgumentParser(formatter_class=arg_fmt,
82                                         description=main.__doc__)
83
84         parser.add_argument(
85             '-v', '--verbose',
86             action='store_const',
87             const=True,
88             default=False,
89             help="displays debug information (default = False)"
90         )
91         """ parser.add_argument(
92             '-l', '--limb',
93             choices=['left', 'right'],
94             required=True,
```

```
92         #default='left',
93         help='the limb to run the measurements on'
94     ) """
95
96     args = parser.parse_args(rospy.myargv()[1:])
97
98     #Init
99     rospy.init_node("measure_precision", anonymous = True)
100    time.sleep(0.5)
101    print(" --- Ctrl-D stops the program ---")
102    print("Init started...")
103    arm = arm_class.Arm('left', verbose=args.verbose)
104    lut = lut_interface.restore_lut_from_file("/home/user/
schuermann_BA/ros_ws/src/baxter_staples/precision/lut.csv")['lut']
105    number_of_rounds = 10
106    filelocation = "/home/user/schuermann_BA/ros_ws/src/
baxter_staples/precision/"
107    test_poses = [
108        arm_class.alter_pose_inc(deepcopy(start_pose['left'].pose),
109        verbose=args.verbose, posx=0.06, posy=0.06),
110        arm_class.alter_pose_inc(deepcopy(start_pose['left'].pose),
111        verbose=args.verbose, posx=0.06, posy=0.15),
112        arm_class.alter_pose_inc(deepcopy(start_pose['left'].pose),
113        verbose=args.verbose, posx=0.14, posy=0.11),
114        arm_class.alter_pose_inc(deepcopy(start_pose['left'].pose),
115        verbose=args.verbose, posx=0.23, posy=0.06),
116        arm_class.alter_pose_inc(deepcopy(start_pose['left'].pose),
117        verbose=args.verbose, posx=0.23, posy=0.15)
118    ]
119    print("Init finished...")
120
121    #Measurements
122    for n in range(number_of_rounds):
123        print(n)
124        for p in range(len(test_poses)):
125            #not improved measurements
126            if arm.get_solution(test_poses[p]):
127                arm.move_to_solution()
128                time.sleep(0.05)
129                save_data(arm, test_poses[p], filelocation+
not_improved_{}.csv".format(p+1))
130                time.sleep(0.05)
131            else:
132                arm.simplefailsafe()
133                sys.exit()
134            for p in range(len(test_poses)):
135                #improved with move_precise
136                if arm.move_precise(test_poses[p]):
137                    time.sleep(0.05)
```

```
132         save_data(arm, test_poses[p], filelocation+"  
133             move_precise_{}.csv".format(p+1))  
134             time.sleep(0.05)  
135         else:  
136             arm.simplefailsafe()  
137             sys.exit()  
138         for p in range(len(test_poses)):  
139             #improved with lut  
140             if arm.get_solution(lut_interface.improve_pose(  
141                 deepcopy(test_poses[p]), lut=lut, limb_name=arm._limb_name)):  
142                 arm.move_to_solution()  
143                 time.sleep(0.05)  
144                 save_data(arm, test_poses[p], filelocation+"  
145                     with_lut_{}.csv".format(p+1))  
146                     time.sleep(0.05)  
147                 else:  
148                     arm.simplefailsafe()  
149                     sys.exit()  
150                 for p in range(len(test_poses)):  
151                     #improved with lut an move_precise  
152                     if arm.move_precise(lut_interface.improve_pose(  
153                         deepcopy(test_poses[p]), lut=lut, limb_name=arm._limb_name)):  
154                         time.sleep(0.05)  
155                         save_data(arm, test_poses[p], filelocation+"all_  
156                             {}.csv".format(p+1))  
157                         time.sleep(0.05)  
158                     else:  
159                         arm.simplefailsafe()  
160                         sys.exit()  
161  
162  
163  
164 if __name__ == "__main__":  
165     main()
```

Listing 9: test_if_precision_improved.py

I - detector.py

```
1 #!/usr/bin/env python
2 #-*- coding:utf-8 -*-
3 import cv2 as cv
4 import sys
5 import numpy as np
6 from copy import deepcopy
7
8 def create_rim(contour, mask, rim_width, verbose=False):
9     """
10         Modifies the given mask by a smaller version of the given contour
11         to create a white rim zone in accordance to the given width.
12
13         Parameters:
14             contour:      Contour of the document on which the rim shall
15             be created
16             mask:        Mask of the document on which the rim shall be
17             created
18             rim_width:   Width of the rim to be created
19             verbose:     Flag to print messages for debugging
20
21         Return:
22             mask:        Modified version of the given mask; None if
23             rim is not creatable
24
25         """
26
27     rect = cv.minAreaRect(contour)
28     box = cv.boxPoints(rect)
29     box = np.int0(box)
30     main_length, sec_length, angle, mainline_endpoint = get_box_info(
31     box)
32
33     if main_length == 0:
34         return None
35
36     if verbose:
37         print("main_length: {} sec_length: {} angle: {}".format(
38             main_length, sec_length, angle))
39
40     rim_corner_length = np.sqrt(np.square(main_length*(rim_width
41 [1]/297.0)) + np.square(sec_length*(rim_width[0]/210.0)))
42
43     rim_box = deepcopy(box)
44
45     if mainline_endpoint == 3 :
46
47         angle += (np.pi/4)
48
49         cols = (rim_corner_length*np.sin(angle))
50         rows = (rim_corner_length*np.cos(angle))
51
52         rim_box[0][0] = box[0][0] + cols #bottom left col
53         rim_box[0][1] = box[0][1] - rows #bottom left row
54         rim_box[1][0] = box[1][0] + cols #top left col
55         rim_box[1][1] = box[1][1] + rows #top left colrow
56         rim_box[2][0] = box[2][0] - cols #top right col
57         rim_box[2][1] = box[2][1] + rows #top right row
```

```
40         rim_box[3][0] = box[3][0] - cols #bottom right col
41         rim_box[3][1] = box[3][1] - rows #bottom right row
42     else:
43         angle -= (np.pi/4)
44         cols = (rim_corner_length*np.cos(angle))
45         rows = (rim_corner_length*np.sin(angle))
46         rim_box[0][0] = box[0][0] - cols #bottom right col
47         rim_box[0][1] = box[0][1] - rows #bottom right row
48         rim_box[1][0] = box[1][0] + cols #bottom left col
49         rim_box[1][1] = box[1][1] - rows #bottom left row
50         rim_box[2][0] = box[2][0] + cols #top left col
51         rim_box[2][1] = box[2][1] + rows #top left row
52         rim_box[3][0] = box[3][0] - cols #top right col
53         rim_box[3][1] = box[3][1] + rows #top right row
54     cv.drawContours(mask, [rim_box], 0, 0, -1)
55     return mask
56
57 def create_box_mask(contour, image_shape):
58     """
59     Creates a mask with a box contour of the given contour.
60
61     Parameters:
62         contour:          Contour as base for the mask
63         image_shape:      Shape of the box to be created
64     Return:
65         mask:            Created mask of box contour
66     """
67     mask = np.ones(image_shape, np.uint8)
68     box = create_box_contour(contour)
69     cv.drawContours(mask, [box], 0, 255, -1)
70     return mask
71
72 def create_box_contour(contour):
73     """
74     Creates a box contour on basis of the given contour.
75
76     Parameters:
77         contour:          Basis for the box contour to be created
78     Return:
79         box:              Box contour
80     """
81     rect = cv.minAreaRect(contour)
82     box = cv.boxPoints(rect)
83     box = np.int0(box)
84     return box
85
86 def get_box_info(box_cnt):
87     """
```

```
88     Returns a number of informations of the given box contour.  
89  
90     The mainline of the box contour refers to the longer one of the  
edges starting from the first point of box_cnt.  
91     The first point of a box contour is always the one with the  
highest row value, therefore the one closest to the bottom of the  
screen if displayed.  
92     The following points of a box contour are then clockwise arranged.  
93     So the mainline goes either from 0 to 1 or 0 to 3.  
94     For more information on this function please see "Metallentfernung  
an Dokumenten durch den Forschungsroboter Baxter" by "Timo  
Sch rmann"  
95  
96     Parameters:  
97         box_cnt:           Box contour  
98     Return:  
99         main_length:       Length of the given box / Lenght of  
the mainline  
100        secondary_length: Width of the given box  
101        angle:           Angle of the mainline of the box  
102        mainline_endpoint: Endpoint of the mainline of the box.  
Either 1 or 3  
103  
104     103 = np.sqrt(np.square(box_cnt[3][0] - box_cnt[0][0]) + np.square  
(box_cnt[3][1] - box_cnt[0][1]))  
105     101 = np.sqrt(np.square(box_cnt[1][0] - box_cnt[0][0]) + np.square  
(box_cnt[1][1] - box_cnt[0][1]))  
106     ankathete = float(box_cnt[3][1] - box_cnt[0][1])  
107     if ankathete != 0.0:  
108         gkdak = float(box_cnt[3][0] - box_cnt[0][0]) / ankathete  
109         angle03 = np.arctan(gkdak)+(np.pi/2)  
110     else:  
111         angle03 = 0.0  
112     if 103 > 101 :  
113         main_length = 103  
114         secondary_length = 101  
115         angle = angle03  
116         mainline_endpoint = 3  
117     else:  
118         main_length = 101  
119         secondary_length = 103  
120         angle = angle03  
121         mainline_endpoint = 1  
122     return main_length, secondary_length, angle, mainline_endpoint  
123  
124 def find_match(img, comparator, maxL=1280.0, minL=0.0, verbose=False):  
125     """
```

```
126     Extracts all contours from the given image, finds the best
127     matching one to the comparator and returns it in form of different
128     datatypes.
129
130     Parameters:
131         img:           Image on which the designated contour
132         shall be found
133         comparator:    Mask which resembles the box contour to be
134         found
135         maxL:          Maximal mainlength of the box contour to
136         be found
137         minL:          Minimal mainlength of the box contour to
138         be found
139         verbose:        Flag to print messages for debugging
140
141     Return:
142         True/False:     Whether a match was found or not
143         best_contour:   Box contour of the best match; None if no
144         match was found
145         best_mask:      Mask of the box contour of the best match;
146         None if no match was found
147         show_img:       Grayscale input image with found best
148         matching box contour drawn on it in green; None if no match was
149         found
150
151     """
152
153     if len(img.shape) > 2:
154         img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
155         edges = cv.Canny(img, 10, 100)
156         contours = cv.findContours(edges.copy(), cv.RETR_TREE, cv.
157         CHAIN_APPROX_SIMPLE)[1]
158         best_match = 1.0
159         best_contour = None
160         i = 0
161
162         for c in contours:
163             work_mask = create_box_mask(c, img.shape)
164             match = cv.matchShapes(comparator, work_mask, 2, 0.0)
165             arcL = cv.arcLength(create_box_contour(c), True)
166             if arcL < maxL and arcL > minL and match < best_match:
167                 i+=1
168                 if verbose:
169                     print(arcL)
170                 best_match = match
171                 best_mask = deepcopy(work_mask)
172                 best_contour = create_box_contour(c)
173
174             if verbose:
175                 print("deviation for best match at: {}%\nnumber of best
176 matches: {}".format(float(best_match)*100, i))
```

```
162     if not best_contour is None:
163         show_img = cv.cvtColor(deepcopy(img), cv.COLOR_GRAY2BGR)
164         cv.drawContours(show_img, [best_contour], 0, (0,255,0), 1)
165         return True, best_contour, best_mask, show_img
166     else:
167         return False, None, None, None
168
169 def sortkey_first(x):
170     """
171     Sortkey for find_matches. Sorts the list in accordance to the
172     first element.
173     """
174     return x[0]
175
176 def find_matches(img, comparator, maxL=1280.0, minL=0.0,
177                 deviation_thresh=0.15, verbose=False):
178     """
179     Extracts all contours from the given image and returns all found
180     matches with a deviation lower than 0.15 as a sorted list.
181
182     img and comparator must have the same resolution.
183
184     Parameters:
185         img:                         Image on which the designated contour
186         shall be found
187         comparator:                  Mask which resembles the box contour
188         to be found
189         maxL:                        Maximal mainlength of the box contour
190         to be found; Default: 1280.0
191         minL:                        Minimal mainlength of the box contour
192         to be found; Default: 0.0
193         deviation_thresh:           Maximal deviation of found contour to
194         comparator; Default: 0.15
195         verbose:                     Flag to print messages for debugging
196
197     Return:
198         True/False:                  Whether one or more matches were found
199         or not
200         best_matches:                List of all found matches, sorted by
201         deviation in ascending order; Empty if no matches were found
202     """
203
204     if len(img.shape) > 2:
205         img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
206     edges = cv.Canny(img, 10, 100)
207     contours = cv.findContours(edges.copy(), cv.RETR_TREE, cv.
208                               CHAIN_APPROX_SIMPLE)[1]
209     best_matches = list()
210     for c in contours:
211         work_mask = create_box_mask(c, img.shape)
```

```
199     match = cv.matchShapes(comparator, work_mask, 2, 0.0)
200     arcL = cv.arcLength(create_box_contour(c), True)
201     if arcL < maxL and arcL > minL and match < deviation_thresh:
202         if verbose:
203             print(arcL)
204         best_matches.append([match, create_box_contour(c), deepcopy
205 (work_mask)])
206         best_matches.sort(key=sortkey_first)
207     if len(best_matches) > 0:
208         print("max deviation for matches at: {}%\n\n".format(
209             deviation_thresh*100))
210         print("number of matches: {}".format(len(best_matches)))
211         print("deviation for best match at {}%".format(best_matches
212 [0][0]))
213     return True, best_matches
214 else:
215     return False, best_matches
216
217
218 def detect_staple(img):
219     """
220     Function to detect staples in a given image.
221
222     The given image must have a resolution of 640x400 or 320x200.
223     This functions uses different approaches and return parameters for
224     the different image resolutions.
225     So please choose the right resolution for the effect you want to
226     accomplish.
227
228     640x400 -> The Method searches for all contours on the image that
229     resemble that of a staple from afar. It returns a sorted list of
230     all found contours with a deviation of maximum 15%
231     320x200 -> The Method searches for the contour that resembles that
232     of a staple from near with the least deviation.
233
234     Parameters:
235         img:                      Image on which the designated contour
236         shall be found
237
238         Return:
239             640x400:
240                 success:      True if at least one match was found,
241                 else False
242
243                 found_matches: List of all found matches, sorted by
244                 deviation in ascending order; None if no matches were found
245
246                 None:          Only for compatibility with the return
247                 parameters for 320x200 images
248
249             320x200:
250                 success:      True if a match was found, else False
251                 found_contour: Box contour of the found match
```

```
235             cnt_img:           Given image with the found contour
236             drawn on it
237             """
238
239             if len(img.shape) > 2:
240                 img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
241
242                 if img.shape[0] == 400:
243                     cmp_mask = cv.imread("/home/user/schuermann_BA/ros_ws/src/
244 baxter_staples/cv_images/masks/small_staple.jpg", 0)
245                     maxL = 50.0
246                     minL = 20.0
247
248                     success, found_matches = find_matches(img, cmp_mask, maxL=maxL,
249 , minL=minL)
250
251                     if not success:
252                         print("cannot find any staple")
253
254                     return success, None, None
255
256                     return success, found_matches, None
257
258                 elif img.shape[0] == 200:
259                     cmp_mask = cv.imread("/home/user/schuermann_BA/ros_ws/src/
260 baxter_staples/cv_images/masks/staple1.jpg", 0)
261                     maxL = 120.0
262                     minL = 60.0
263
264                     img = cv.GaussianBlur(img, (5,5), 0)
265
266                     success, found_contour, best_mask, cnt_img = find_match(img,
267 cmp_mask, maxL=maxL, minL=minL)
268
269                     if not success:
270                         print("cannot find any staple")
271
272                     return success, img, None
273
274                     return success, found_contour, cnt_img
275
276             else:
277
278                 print("given image has wrong resolution. Please provide a
279 picture with following format: 640x400")
280
281             return success, img, None
282
283
284
285     def detect_paper(img):
286         """
287
288             Function to detect a document on an appropriate workplate and
289             apply a mask on it, so that only the clear rim of the document
290             remains.
291
292
293             For this to function a specially arranged workspace is needed.
294             Please provide such as described in "Metallentfernung an
295             Dokumenten durch den Forschungsroboter Baxter" by "Timo Sch rmann
296             ".
297
298             Parameters:
299
300                 img:           Image on which the designated contour shall be
301 found. Must be of resolution 640x400
```

```
271     Return:
272         success:    True if a document was found, else False
273         only_rim:   Given image masked, so that only the clear rim
274         remains; img if no document was found
275         paper_cnt:  Box contour of the found document; None if no
276         document was found
277         """
278
279     if len(img.shape) > 2:
280         img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
281     if img.shape[0] != 400:
282         print("given image has wrong resolution. Please provide a
283             picture with following format: 640x400")
284     return False, img, None
285 #get background mask
286 workplate_mask = cv.imread("/home/user/schuermann_BA/ros_ws/src/
287 baxter_staples/cv_images/masks/background.jpg", 0)
288 workplate_mask = cv.resize(workplate_mask, img.shape[1::-1])
289 #get workplate mask
290 paper_mask = cv.imread("/home/user/schuermann_BA/ros_ws/src/
291 baxter_staples/cv_images/masks/document.jpg", 0)
292 paper_mask = cv.resize(paper_mask, img.shape[1::-1])
293 #subtract background
294 erode_kernel = np.ones((11,13), np.uint8)
295 workplate_mask = cv.erode(workplate_mask, erode_kernel, iterations
296 =1)
297 minus_background = cv.bitwise_and(img, workplate_mask)
298 #find document and subtract workplate
299 success, paper_cnt, paper_mask, cnt_img = find_match(
300 minus_background, paper_mask)
301 if not success:
302     print("cannot find document... please rearrange on workplate")
303     return False, img, None
304 paper_mask = cv.erode(paper_mask, erode_kernel, iterations=1)
305 minus_workplate = cv.bitwise_and(img, paper_mask)
306 #create ROI as mask and apply to image
307 rim_mask = create_rim(paper_cnt, paper_mask, (20.0, 25.0))[1]
308 only_rim = cv.bitwise_and(img, rim_mask)
309 return True, only_rim, paper_cnt
310
311
312 def mask_window(img, gripper_action_point):
313     """
314     Masks the given image, so that only a field around the given
315     action point remains visible.
316
317     Parameters:
318         img:                      Image to be masked
319         gripper_action_point:    Action point of the used end
320         effector
```

```
310     Return:  
311         masked_img:           Given image with applied mask  
312     """  
313     img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)  
314     mask = np.zeros(img.shape, np.uint8)  
315     field = 60  
316     cv.rectangle(mask, (gripper_action_point[0]-field,  
317                   gripper_action_point[1]+field), (gripper_action_point[0]+field,  
318                   gripper_action_point[1]-10), 255, -1)  
319     masked_img = cv.bitwise_and(img, mask)  
320     return masked_img  
321  
322 def draw_cnt_on_img(cnt, img):  
323     """  
324     Draws a given contour on a given image.  
325  
326     Parameters:  
327         cnt:      Contour to be drawn  
328         img:      Image to be drawn on  
329     Return:  
330         Image:   Given image with contour drawn on it  
331     """  
332     if len(img.shape) == 2:  
333         img = cv.cvtColor(img, cv.COLOR_GRAY2BGR)  
334     return cv.drawContours(img, [cnt], 0, (0,0,255), 1)
```

Listing 10: detector.py

J - point_on_paper.py

```
1 #!/usr/bin/env python
2 #-*- coding:utf-8 -*-
3
4 """
5 A simple script to measure the relationship between the z axis value
6   of the arm and the action point of it.
7
8 The arm executes a gradual movement approaching a paper with attached
9   pen to draw a point on it.
10 Afterwards the arm retreats gradually. At each step the user can click
11   on the painted point on the "Snapshot" display to get the
12   associated coordinates.
13
14 In combination with the pose, the coordinates can be used to create a
15   descriptive formula for the relationship between pixel coordinate
16   and z axis value.
17
18 For further information please see "Metallentfernung an Dokumenten
19   durch den Forschungsroboter Baxter" by "Timo Schuermann".
20 """
21
22
23
24
25
26
27 sys.path.append("/home/user/schuermann_BA/ros_ws/src/baxter_staples/
28   scripts/base_classes")
29 import arm_class
30
31
32
33 pose = Pose(
34   position=Point(
35     x=0.500,
36     y=0.200,
37     z=-0.190,
38   ),
```

```
39     orientation=Quaternion(
40         x=-0.700,
41         y=0.700,
42         z=0.000,
43         w=0.000
44     ),
45 )
46
47 def main():
48     # Argument Parsing
49     arg_fmt = argparse.RawDescriptionHelpFormatter
50     parser = argparse.ArgumentParser(formatter_class=arg_fmt,
51                                     description=main.__doc__)
52     parser.add_argument(
53         '-v', '--verbose',
54         action='store_const',
55         const=True,
56         default=False,
57         help="displays debug information (default = False)"
58     )
59     parser.add_argument(
60         '-l', '--limb',
61         choices=['left', 'right'],
62         #required=True,
63         default='left',
64         help='the limb to run the measurements on'
65     )
66     args = parser.parse_args(rospy.myargv()[1:])
67
68     #Init
69     rospy.init_node("pen_and_paper", anonymous = True)
70     time.sleep(0.5)
71     print("Init started...")
72     arm = arm_class.Arm(args.limb, args.verbose, True)
73     print("Init finished...")
74
75     arm.set_neutral(False)
76     raw_input("Press Enter to grab pen...")
77     for i in range(3):
78         print("gripping in: {}".format(3-i))
79         time.sleep(1)
80     arm._gripper.close()
81     #high pose
82     if not arm.move_to_pose(arm_class.alter_pose_inc(pose, args.
83     verbose, posz=0.12)):
84         arm.simplefailsafe()
85         return False
86     raw_input("mark point")
```

```
85     if not arm.move_direct(pose):
86         arm.simplefailsafe()
87         return False
88     #draw point
89     if not arm.move_to_pose(arm_class.alter_pose_inc(pose, args.
90     verbose, posz=-0.01)):
91         arm.simplefailsafe()
92         return False
93     print(arm._current_pose)
94     for i in range(20):
95         if not arm.move_to_pose(arm_class.alter_pose_inc(pose, args.
96     verbose, posz=(i*0.02))):
97             arm.simplefailsafe()
98             return False
99         time.sleep(1)
100        arm.cam.update_snapshot = True
101        print(arm._current_pose)
102        raw_input("{} - ".format(i+1))
103
104    #exit strategy
105    arm.set_neutral(False)
106    raw_input("please remove pen...")
107    arm.simplefailsafe(True)
108
109if __name__ == "__main__":
110    main()
```

Listing 11: point_on_paper.py

K - control_interface.py

```
1 #!/usr/bin/env python
2 #-*- coding:utf-8 -*-
3 import argparse
4 import sys
5 import time
6 import rospy
7 import cv2 as cv
8 from copy import deepcopy
9 from geometry_msgs.msg import (
10     Pose,
11     Point,
12     Quaternion
13 )
14
15 sys.path.append("/home/user/schuermann_BA/ros_ws/src/baxter_staples/
16                 scripts/base_classes")
17
18 import detector
19 import baxter_interface
20
21 #-----Constants-----
22 paper_pose = Pose(
23     position=Point(
24         x=0.610,
25         y=0.285,
26         z=0.100
27     ),
28     orientation=Quaternion(
29         x=-0.700,
30         y=0.700,
31         z=0.000,
32         w=0.000
33     )
34 )
35
36 pick_paper_pose = Pose(
37     position=Point(
38         x=0.660,
39         y=-0.150,
40         z=-0.180
41     ),
42     orientation=Quaternion(
43         x=0.000,
44         y=0.999,
45         z=0.000,
```

```
46         w=0.000
47     )
48 )
49
50 approvement_height = -0.15
51 marking_height = -0.202
52
53 commands = {
54     'cam' : "opens the menu for camera control",
55     'arm' : "opens the menu for arm control",
56     'find' : "tries to find a staple with current pose",
57     'run' : "starts a full run from document provision to marking
      the staple",
58     'quit' : "resets all camera settings, moves arms in neutral
      poses and disables robot"
59 }
60
61 #-----Program-----
62 def mark_staple(arm, box_cnt):
63     """
64         Approaches the given contour in dependance of the arms current
        pose.
65
66         For this function to work properly please dont move the arm
        inbetween finding the contour and calling this.
67
68         Parameters:
69             arm:          The robots limb to perform the movement
70             box_cnt:       The box contour to be approached
71
72         Return:
73             True/False: False if any calculated pose is not reachable,
        else True
74         """
75
76     print("marking staple")
77     arm.cam.set_highlight(detector.draw_cnt_on_img(box_cnt, arm.cam.
        _snapshot))
78     time.sleep(2)
79     point_distance = detector.distance_to_point(box_cnt[0])
80     markingpose = arm_class.alter_pose_inc(arm._current_pose, posx=
        point_distance[0], posy=point_distance[1])
81     markingpose = arm_class.alter_pose_abs(markingpose, arm._verbose,
        posz=marking_height)
82     print("current pose:\n{}\nmarkingpose:\n{}\n".format(arm.
        _current_pose, markingpose))
83     if not arm.move_direct(markingpose):
84         return False
85     retreat = arm_class.alter_pose_inc(arm._current_pose, posz=0.05)
86     if not arm.move_to_pose(retreat):
```

```
85         return False
86     return True
87
88 def approve_matches(arm, matches):
89     """
90     Approaches the given matches in dependency to the arms current
91     pose and approves them in a close-up view.
92     If a match gets confirmed it will be marked and the program stops.
93
94     For this function to work properly please dont move the arm
95     inbetween finding the matches and calling this.
96
97     Parameters:
98         arm:          The robots limb to perform the movements
99         matches:      A list of box contours to be approved
100    Return:
101        True/False: False if any calculated pose is not reachable,
102    else True
103    """
104
105    prove_poses = list()
106    contour_img = deepcopy(arm.cam._snapshot)
107    for m in matches:
108        arm.cam.set_highlight(detector.draw_cnt_on_img(m[1],
109        contour_img))
110        time.sleep(1)
111        staple_distance = arm.cam.distance_to_point(m[1][0])
112        prove_poses.append(arm_class.alter_pose_inc(arm._current_pose,
113        posx=staple_distance[0], posy=staple_distance[1]+0.01))
114    for p in prove_poses:
115        if not arm.move_to_pose(arm_class.alter_pose_abs(p, arm.
116        _verbose, posz=approvement_height + 0.1)):
117            return False
118        if not arm.move_precise(arm_class.alter_pose_abs(p, arm.
119        _verbose, posz=approvement_height)):
120            return False
121        if not arm.cam.windowed:
122            window(arm)
123        else:
124            arm.cam.update_snapshot = True
125            time.sleep(1)
126            small_roi(arm)
127            time.sleep(1)
128            success, found_match, cnt_img = detector.detect_staple(
129            deepcopy(arm.cam._snapshot))
130            if success:
131                arm.cam.set_highlight(cnt_img)
132                print("found it")
133                if not mark_staple(arm, found_match):
```

```
125             return False
126         return True
127     else:
128         if not arm.move_to_pose(arm_class.alter_pose_inc(arm.
129         _current_pose, arm._verbose, posz=0.1)):
130             return False
131     print("no staple found")
132
133
134 def window(arm):
135     """
136     Toggles between normal mode (640x400) and windowed mode (320x200).
137     To check the current mode please check arm.cam.windowed.
138
139     Parameters:
140         arm:    The robots limb whose camera mode shall be toggled
141     """
142     if not arm.cam.windowed:
143         arm.cam.controller.resolution = (320, 200)
144         arm.cam.controller.window = (600, 200)
145         arm.cam.windowed = True
146         arm.cam.update_snapshot = True
147     else:
148         arm.cam.controller.window = (320, 200)
149         arm.cam.controller.resolution = (640, 400)
150         arm.cam.windowed = False
151         arm.cam.update_snapshot = True
152
153 def small_roi(arm):
154     """
155     Masks the current snapshot so that it only displays a field around
156     the action point.
157
158     Parameters:
159         arm:    The robots limb whose snapshot shall be masked
160     """
161     arm.cam._snapshot = detector.mask_window(deepcopy(arm.cam.
162     _snapshot), arm.cam.get_action_point())
163
164 def find_staple(arm):
165     """
166     The complete walkthrough to find a staple on a provided document.
167
168     Moves the arm to a pose to overview the complete workplate. Then
169     detects the document and possible staples on it.
170     Checks the possibilities and marks a match, if found.
171
172     Parameters:
173         arm:    The robots limb to execute the walkthrough
```

```
169     """
170 
171     if arm.cam.windowed:
172         window(arm)
173 
174     arm.move_to_pose(paper_pose)
175     time.sleep(1)
176 
177     arm.cam.update_snapshot = True
178     time.sleep(0.2)
179 
180     paper_success, only_rim, paper_cnt = detector.detect_paper(
181         deepcopy(arm.cam._snapshot))
182 
183     if paper_success:
184         arm.cam.set_highlight(only_rim)
185         time.sleep(2)
186 
187         staple_success, matches = detector.detect_staple(only_rim)
188 
189         if staple_success:
190             approve_matches(arm=arm, matches=matches)
191 
192     else:
193         print("No document detectable. Please rearrange it on the
194         workplate")
195 
196 def full_run(left):
197     """
198 
199     The complete walkthrough with document provision, staple detection
200     and staple marking.
201 
202     Parameters:
203         left:    The arm which shall execute the detection cycle
204 
205     Return:
206         True/False: False if any movement is not executable, else
207         True
208 
209     """
210 
211     right = arm_class.Arm('right', left._verbose)
212     left.set_neutral(False)
213     right.set_neutral(False)
214 
215     #supply document
216     if not right.pick(pick_paper_pose):
217         right.simplefailsafe()
218 
219         return False
220 
221     if not right.place_paper():
222 
223         return False
224 
225     right.set_neutral()
226 
227     #detect and mark staple
228     find_staple(left)
229 
230     left.set_neutral(False)
231 
232     return True
233 
234 
235 def main():
236     # Argument Parsing
```

```
213     arg_fmt = argparse.RawDescriptionHelpFormatter
214     parser = argparse.ArgumentParser(formatter_class=arg_fmt,
215                                     description=main.__doc__)
216     parser.add_argument(
217         '-v', '--verbose',
218         action='store_const',
219         const=True,
220         default=False,
221         help="displays debug information (default = False)"
222     )
223     args = parser.parse_args(rospy.myargv()[1:])
224
225     #Init
226     rospy.init_node("lut_interface", anonymous = True)
227     time.sleep(0.5)
228     print("--- Ctrl-D stops the program ---")
229     print("Init started...")
230     arm = arm_class.Arm('left', args.verbose, cam_wanted=True)
231     print("Init finished...")
232
233     while(True):
234         cmd = raw_input("Enter command\n")
235         if cmd == "cam":
236             cmd_param = raw_input("possible options: update, write,
237 window, exposure, gain, roi")
238             if cmd_param == 'update':
239                 arm.cam.update_snapshot = True
240             elif cmd_param == "write":
241                 cv.imwrite("/home/user/schuermann_BA/ros_ws/src/
242 baxter_staples/cv_images/ rename_me.jpg", arm.cam._img)
243             elif cmd_param == "window":
244                 window(arm)
245             elif cmd_param == "exposure":
246                 cmd_param = raw_input("Enter value (0-100): ")
247                 arm.cam.controller.exposure = int(cmd_param)
248             elif cmd_param == "gain":
249                 cmd_param = raw_input("Enter value (0-79): ")
250                 arm.cam.controller.gain = int(cmd_param)
251             elif cmd_param == "roi":
252                 arm.cam._snapshot = detector.mask_window(deepcopy(arm.
253 cam._snapshot), arm.cam.get_action_point())
254             else:
255                 print("no such command")
256         elif cmd == "arm":
257             cmd_param = raw_input("possible options: get, go, rotate,
258 neutral, approach, retreat, pen")
259             if cmd_param == "get":
260                 print(arm._current_pose)
```

```
256         elif cmd_param == "go":
257             arm.set_neutral()
258             arm.move_to_pose(arm_class.alter_pose_inc(deepcopy(
259                 paper_pose), posz=0.1))
260             arm.move_precise(paper_pose)
261         elif cmd_param == "rotate":
262             cmd_param = raw_input("enter rotation: ")
263             arm.move_to_pose(arm_class.alter_pose_inc(arm.
264                 _current_pose, args.verbose, orx=float(cmd_param)+arm.
265                 _current_pose.orientation.x))
266         elif cmd_param == "neutral":
267             arm.set_neutral()
268         elif cmd_param == "approach":
269             arm.move_direct(arm_class.alter_pose_abs(arm.
270                 _current_pose, posz=-0.1))
271         elif cmd_param == "retreat":
272             arm.move_to_pose(arm_class.alter_pose_inc(arm.
273                 _current_pose, arm._verbose, posz=0.15))
274         elif cmd_param == "pen":
275             raw_input("Press Enter to grab pen...")
276             for i in range(5):
277                 print("gripping in: {}".format(5-i))
278                 time.sleep(1)
279                 arm._gripper.close()
280             else:
281                 print("no such command")
282         elif cmd == "find":
283             if arm.cam.windowed:
284                 window(arm)
285                 find_staple(arm)
286                 arm.set_neutral(False)
287             elif cmd == "run":
288                 full_run(arm)
289             elif cmd == "quit":
290                 print("resetting camera settings")
291                 arm.cam.controller.resolution = (640, 400)
292                 arm.cam.controller.exposure = arm.cam.controller.
293                 CONTROL_AUTO
294                 arm.cam.controller.gain = arm.cam.controller.CONTROL_AUTO
295                 arm.cam.controller.white_balance_red = arm.cam.controller.
296                 CONTROL_AUTO
297                 arm.cam.controller.white_balance_green = arm.cam.controller.
298                 .CONTROL_AUTO
299                 arm.cam.controller.white_balance_blue = arm.cam.controller.
300                 CONTROL_AUTO
301                 arm.move_to_pose(arm_class.alter_pose_inc(arm.
302                     _current_pose, posz=0.1))
303                 arm.simplefailsafe()
```

```
294         break
295     else:
296         print("no such command\navailable commands:")
297         for c in commands.items():
298             print("{} - {}".format(c[0], c[1]))
299
300
301 if __name__ == "__main__":
302     main()
```

Listing 12: control_interface.py

K - Diagramme der Abschlussmessungen

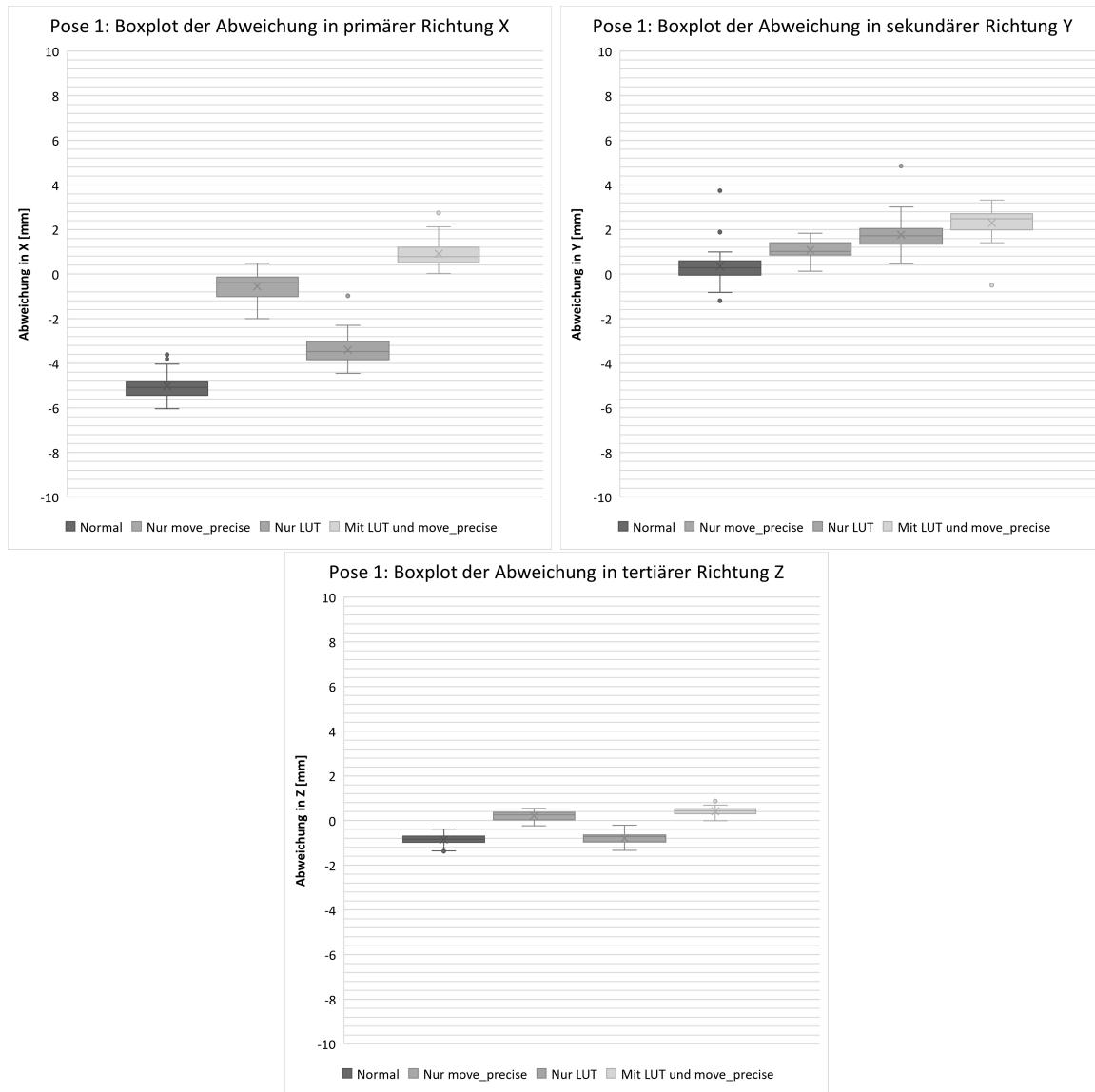


Abbildung 33: Erhöhung der Positioniergenauigkeit: Pose 1

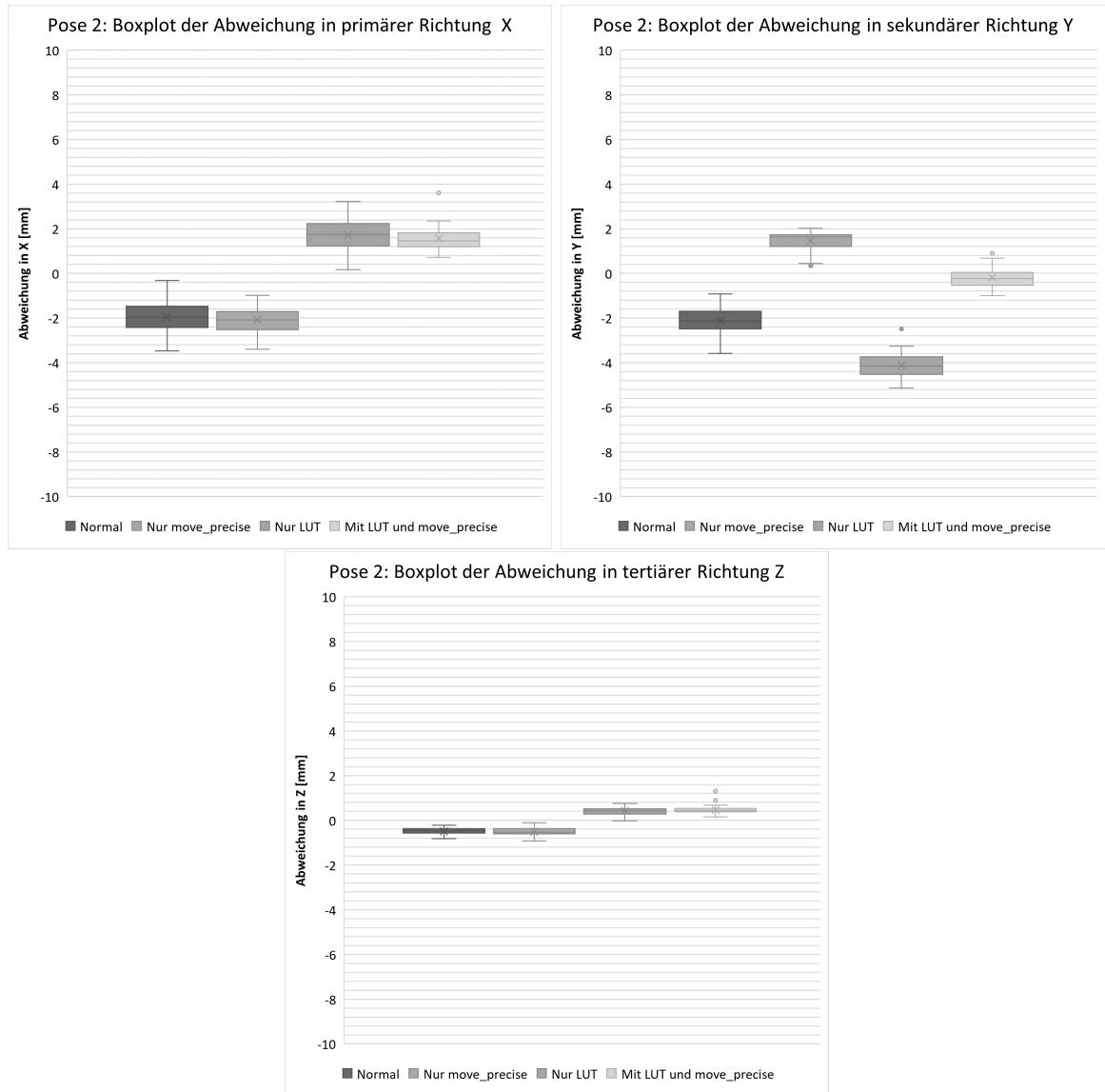


Abbildung 34: Erhöhung der Positioniergenauigkeit: Pose 2

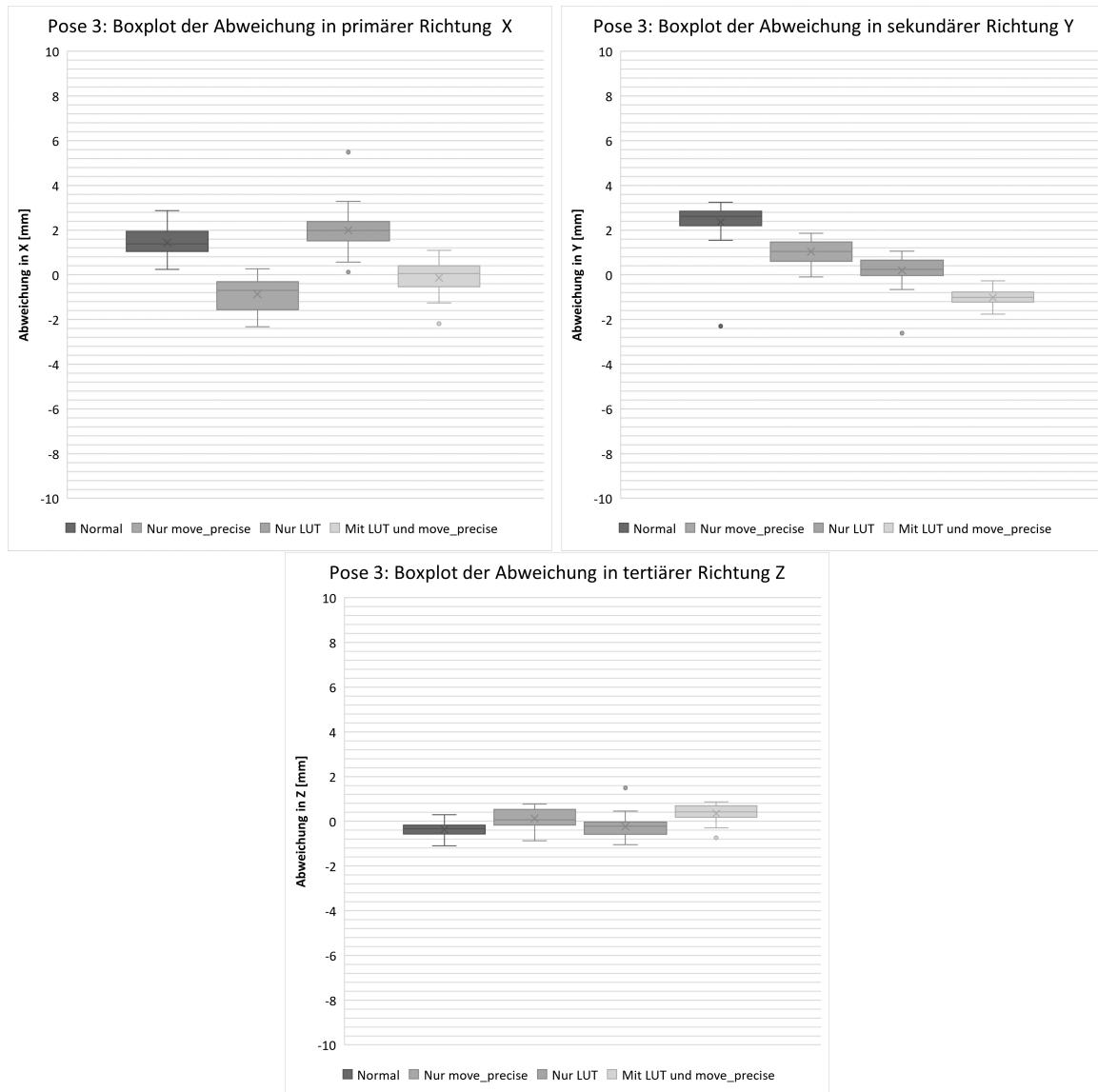


Abbildung 35: Erhöhung der Positioniergenauigkeit: Pose 3

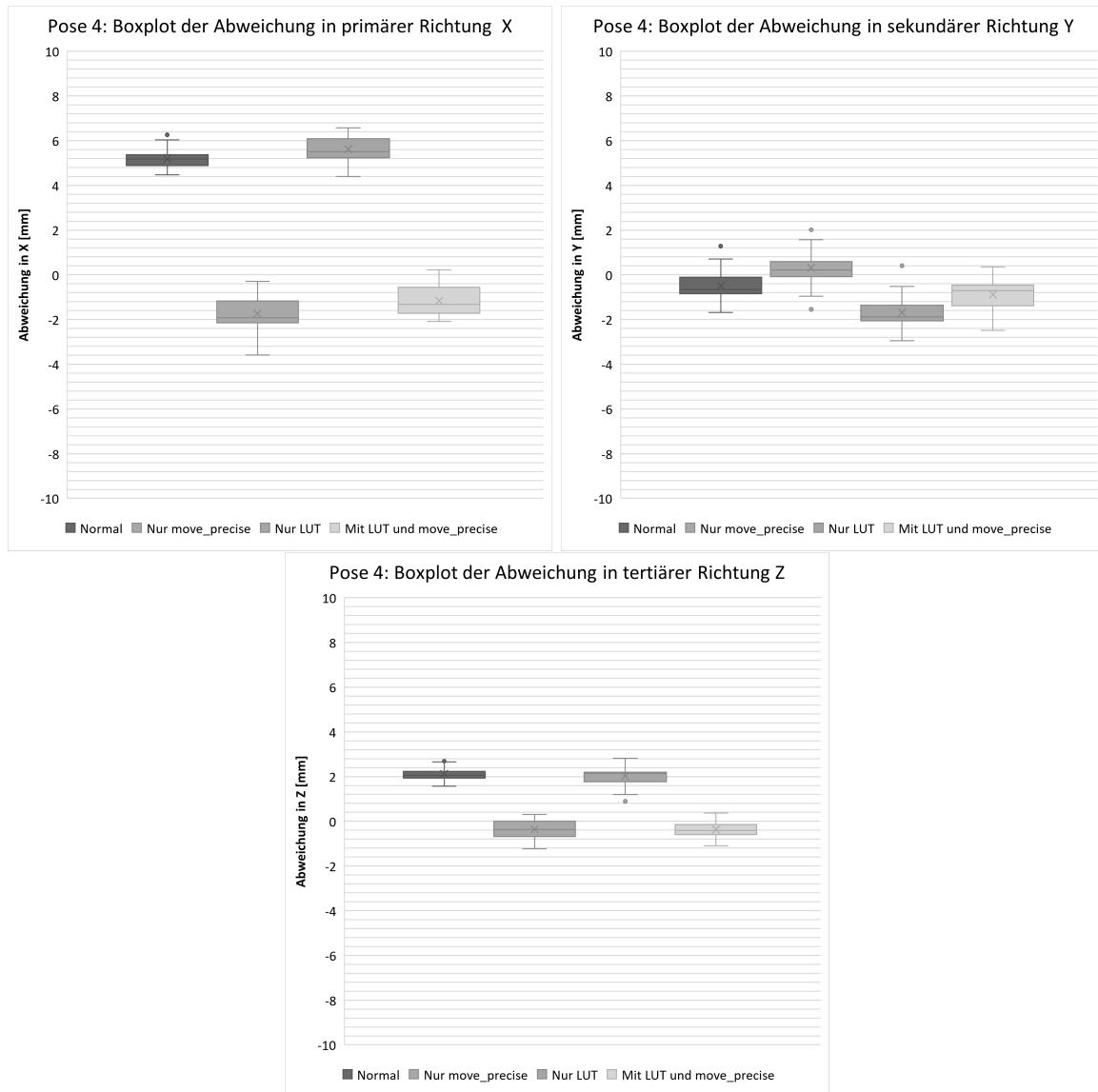


Abbildung 36: Erhöhung der Positioniergenauigkeit: Pose 4

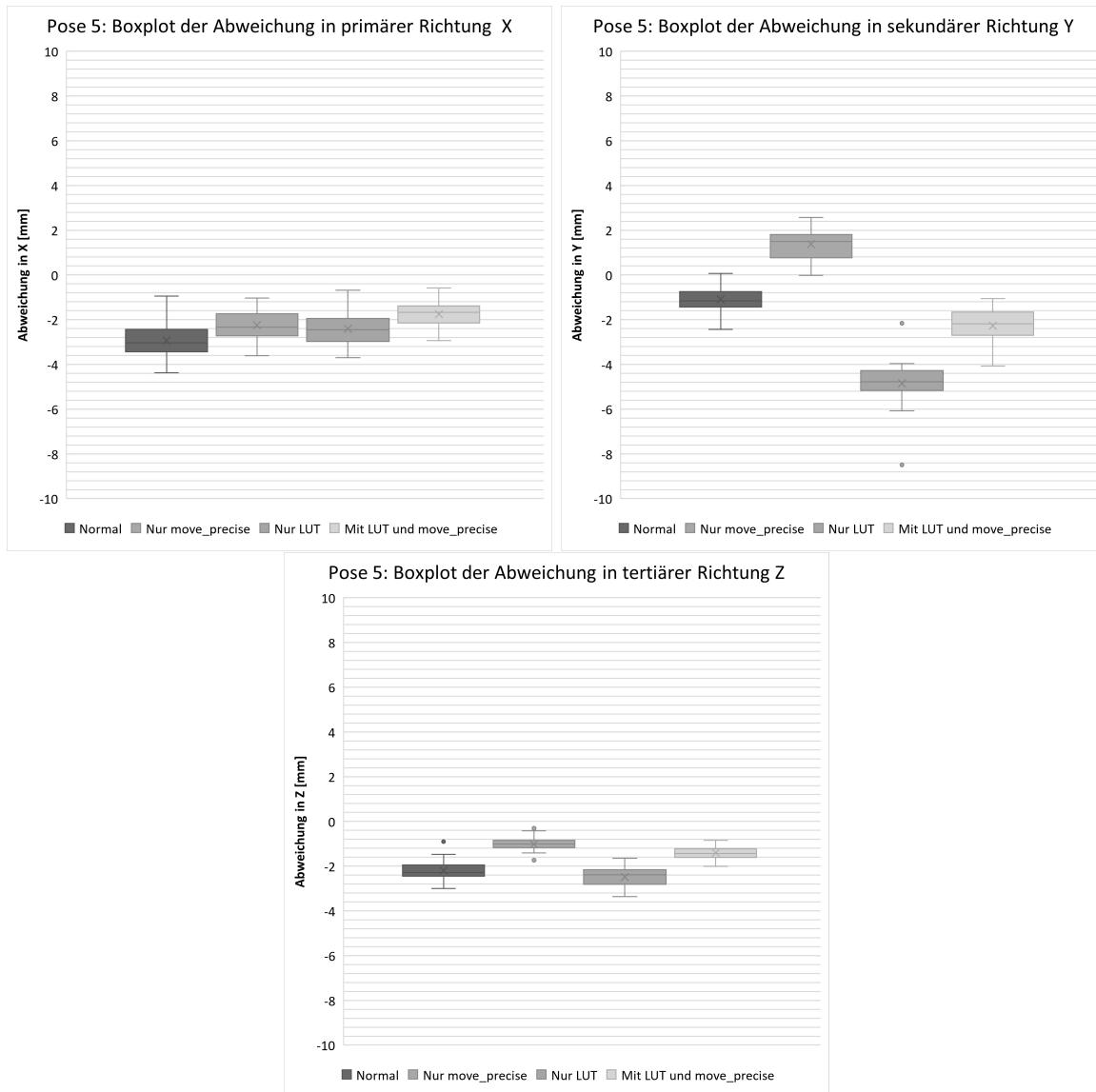


Abbildung 37: Erhöhung der Positioniergenauigkeit: Pose 5