

Studienarbeit - T3100

Selbstlokalisierung eines Pepper-Roboters innerhalb eines definierten Gebäudes hier: Hauptgebäude der DHBW Stuttgart Campus Horb

Fachbereich Elektrotechnik
Fachrichtung Automation

An der Dualen Hochschule Stuttgart
Campus Horb

Eingereicht von:

Tim Schneider
Burgweg 5
72160 Horb am Neckar

Ausbildungsbetrieb:

Eisenmann SE
Tübinger Straße 81
71032 Böblingen

Studienjahrgang:

2017

Matrikelnummer:

6195068

Bearbeitungszeitraum:

09.09.2019 bis 07.01.2020

Betreuer:

Dipl. Ing. Markus Steppacher

Ehrenwörtliche Erklärung

Erklärung gemäß § 5 (3) der „Studien- und Prüfungsordnung DHBW Technik“ vom 29. September 2017.

Ich versichere hiermit, dass ich meine Studienarbeit - T3100 mit dem Thema *Selbstlokalisierung eines Pepper-Roboters innerhalb eines definierten Gebäudes* selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Horb, 12. Dezember 2019

Tim Schneider

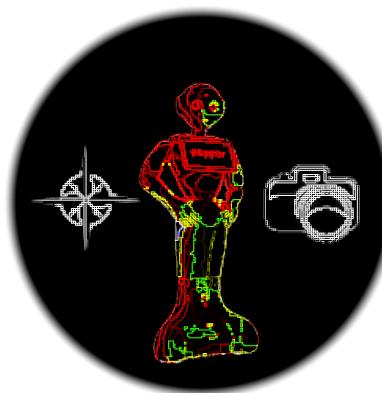
Zusammenfassung

Im Rahmen dieser Studienarbeit wird der humanoide Kommunikationsroboter Pepper als Portier programmiert. Ziel ist es dabei, dass sich Pepper in der Dualen Hochschule Baden-Würtemberg (DHBW) am Campus Horb befindet und Gästen den Weg zu bestimmten Räumen zeigt. Die Aufgabe wurde in zwei Studienarbeiten geteilt. Innerhalb dieser Arbeit wird die Lokalisierung des Roboters mit Hilfe einer Bildverarbeitung (BV) behandelt. Die Aspekte der Kartenerstellung, der Erfassung eines Zielraumes und der Bewegung des Roboters wird in [Bol19] vorgestellt.

Die Arbeit stellt zuerst die benötigte Theorie in Bezug auf die BV dar. Im weiteren Verlauf wird ein ausführlicher Blick auf die durchgeführte Programmierung gelegt. Diese wird in Python durchgeführt und wird von einem Programmiergerät auf Pepper ausgeführt.

Die erstellte Software lässt den Roboter zuerst auf eine benachbarte Tür drehen und macht danach ein Foto von dieser. Danach folgt eine BV, um ein Türschild zu extrahieren und eine Raumnummer zu finden. Diese kann daraufhin an andere Programme weitergegeben werden.

Sobald Pepper in einem Gang steht und die Software gestartet wird, kann Pepper autonom einen Raum finden. Dieses Vorgehen wird erfolgreich ausgeführt, solange die Belichtung ausreichend ist.



Logo des ProjectPepper-Entwicklerteams

Abstract

In this thesis the humanoid communication robot Pepper is to be programmed as a porter. The goal is that Pepper is located in the DHBW at Campus Horb and guides guests to persons or certain rooms. The task is divided into two study papers. Within this thesis the localization of the robot by an image processing is treated. The aspects of mapping, capturing a target room and moving the robot are presented in [Bol19].

The paper first presents the necessary theory regard to the picture processing. In the further course a detailed view is put on the accomplished programming for Pepper. This was done in Python and is currently executed remotely by a programming device on Pepper.

The created software first lets the robot rotate on an adjacent wall and then takes a photo of this door. This is followed by image processing to extract a door label and find a room number. This can be passed on to other programs.

Once Pepper is in a corridor and the software is started, a room can be found autonomously. This procedure is performed successful as long as the lighting is good.

Inhaltsverzeichnis

Formelverzeichnis	VII
Abbildungsverzeichnis	VII
Quellcodeverzeichnis	IX
1 Einleitung	1
1.1 Vorstellung der Dualen Hochschule Horb	1
1.2 Aufgaben- und Problemstellung	2
1.3 Struktur der Arbeit	3
2 Theoretische Grundlagen	4
2.1 Einführung in die Bildverarbeitung	4
2.1.1 Farbräume und deren Verwendung	4
2.1.2 Lokale Operatoren	5
2.1.3 Konturen finden	7
2.1.4 Schrifterkennung (optical character recognition)	7
2.2 Vorstellung des humanoiden Roboters Pepper	8
3 Stand der Technik	10
3.1 Programmierungsmöglichkeiten bei Pepper	10
3.2 Vorstellung der Ergebnisse vorangegangener Studienarbeiten	11
3.2.1 Elfmeterschießen mit NAO	11
3.2.2 Menschliche Schussbewegung des NAO Roboters in C/C++	12
3.2.3 Pepper - Inbetriebnahme und Entwicklung einer Portier-App	12
3.2.4 Dynamische Wegfindung für eine Portier-App am Pepper	13
4 Erkennen der Position innerhalb der DHBW anhand einer Bildauswertung	15
4.1 Strukturelemente an der DHBW	15
4.2 Vorstellung der verwendeten Bibliotheken	17
4.3 Initialbefehle für die Raumsuche	18

4.3.1	Autonomus Life Bewegungen deaktivieren und Sicherheitsabstände minimieren	18
4.3.2	Services anbinden und Einschreiben zum Videostream	19
4.4	Ausrichten des Roboters auf eine Wand	19
4.5	Aufnahme von Bildern	22
4.6	Anhand des Türschildes die Position bestimmen	23
4.7	Weitergabe der gefundenen Raumnummer	30
4.8	Grenzen des erstellten Algorithmus	31
5	Fazit	35
5.1	Zusammenfassung	35
5.2	Kritische Würdigung der Arbeit	35
5.3	Ausblick	36
Literatur		X
Anhang		XIII
Inhalt der CD	XIII	
CD	XIII	

Formelverzeichnis

2.1 Gaußfilter mit 3x3 Fenster	6
--	---

Abbildungsverzeichnis

1.1 DHBW Campus Horb	1
1.2 Aufbau des Campus Horb	2
2.1 Darstellung unterschiedlicher Farbräume	5
2.2 Beispiele für lokale Operatoren	7
2.3 Frontalansicht Pepper	8
3.1 Beispielhafte Naomarks	13
4.1 Beispielhaftes Türschild an der DHBW	16
4.2 Ablauf der vorgestellten BV anhand von vier Bildern	24
4.3 Ablauf der vorgestellten getEdges Funktion anhand von drei Bildern	27
4.4 Türschild befindet sich nicht ganz im Bild	31
4.5 Türschild zu dunkel für eine Auswertung	32
4.6 Bildfokus liegt im Raum	33
4.7 Aufnahme eines unscharfen Bildes	33
4.8 Überbeleuchtetes Türschild in der Lobby	34

Abkürzungsverzeichnis

AL	Autonomus Life
BV	Bildverarbeitung
DHBW	Duale Hochschule Baden-Würtemberg
DH	Duale Hochschule
OCR	Optical character recognition
Regex	Regular Expression
ROS	robot operating system

Quellcodeverzeichnis

4.1	Anbindung Services und Kamera Initialisierung	19
4.2	Drehung Peppers zu einer Wand	20
4.3	Anfahren der richtigen Entfernung zur Wand	21
4.4	Bewegung des Kopfes für die Aufnahme eines Bildes	22
4.5	Aufnahme eines Bildes mit Pepper	23
4.6	Vorstellung der OCR Funktion	25
4.7	Vorstellung der Rooming Funktion	26
4.8	Vorstellung der getEdges Funktion	27
4.9	Vorstellung der Perspective Transformation Funktion	28
4.10	Vorstellung der Detect Room Funktion	29
4.11	Speicherung des gefundenen Raums	30

1 Einleitung

Zu Beginn der Arbeit wird zuerst der Campus Horb der DHBW vorgestellt, dabei wird bereits auf die Struktur eingegangen. Gefolgt wird diese Vorstellung von der Beschreibung des strukturellen Aufbaus der Arbeit und einer genauen Aufgabenbeschreibung.

1.1 Vorstellung der Dualen Hochschule Horb

Die DHBW in Horb ist ein Außencampus der Dualen Hochschule (DH) Stuttgart. Seit über 25 Jahren können Studierende in Horb in unterschiedlichen technischen Fachrichtungen ihren Hochschulabschluss erlangen. Insgesamt werden am Campus rund 1000 Studierende betreut. Dabei wechseln sich die Theoriephasen im dreimonatigen Wechsel mit Praxisphasen ab. Diese können bei rund 300 Dualen Partnern (Unternehmen aus Technik und Wirtschaft) absolviert werden. Ergänzt wird das Angebot durch internationale Kooperationen und Austauschprogrammen mit anderen Hochschulen.



Abbildung 1.1: DHBW Campus Horb¹

Die Hochschule gliedert sich in ein Hauptgebäude und dem Motorenprüfstand (nummer 1 Abbildung 1.2). Im weiteren Verlauf der Arbeit wird lediglich auf das

¹ [Dua]

Erste eingegangen. Dieses gliedert sich weiter in vier Etagen mit jeweils drei Fluren, ein schematischer Aufbau ist in Abbildung 1.2 zu sehen. Von dem zentralen Treppenhaus (I) gelangt man zuerst in einen weiten offenen Bereich - der Lobby. Innerhalb dieser befindet sich auf jeder Etage eine Litfaßsäule und von dort zweigen die drei Gänge (A,B,C) nach links, rechts und vorne ab. Um zwischen den unterschiedlichen Etagen zu wechseln stehen die Treppenhäuser (I,II,III) zur Verfügung. Für mobilitätseingeschränkte Personen stehen zwei Aufzüge, an den Stellen I und III bereit.

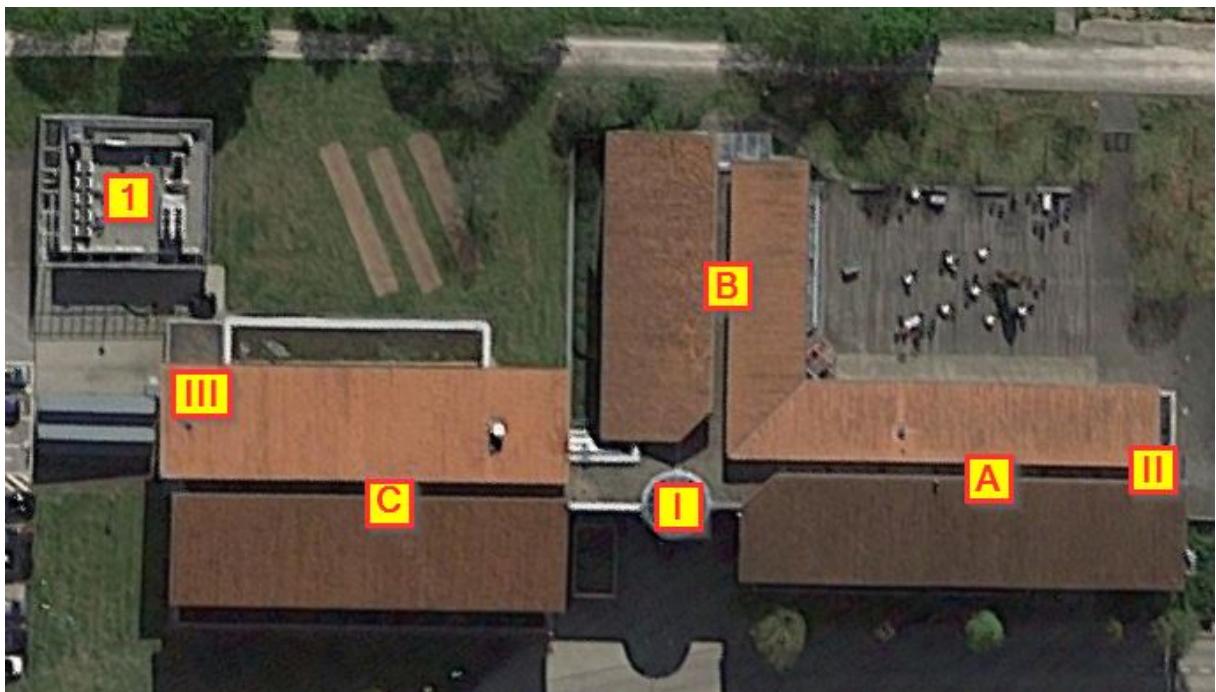


Abbildung 1.2: Aufbau des Campus Horb¹

1.2 Aufgaben- und Problemstellung

Um Besucher an der DHBW einen Weg weisen zu können, soll der Kommunikationsroboter Pepper eingesetzt werden. Dabei soll der Roboter aus einem Gespräch den gewünschten Raum erkennen und den Gast im Anschluss dorthin begleiten oder den Weg verbal beschreiben.

Innerhalb dieser Arbeit wird dabei auf die Bestimmung der Startposition eingegangen. Diese wird benötigt, da sich der Roboter an einem beliebigen Punkt im

¹ [Goo]

Hauptgebäude der DH befindet. Ziel soll es dabei sein, dass sich Pepper innerhalb einer kurzen Zeit lokalisiert hat. Die gesamte Erkennung soll dabei mittels der verbauten Kamera erfolgen.

Weitere Aspekte, wie die Begrüßung von Gästen oder die Wegfindung wurden parallel zu dieser Arbeit von Jonas Boller [Bol19] entwickelt.

1.3 Struktur der Arbeit

Im ersten Teil der Arbeit werden die theoretischen Grundlagen für das Verständnis gelegt. Dabei wird auf die Möglichkeiten der BV eingegangen. Weiterhin wird der verwendete Roboter Pepper vorgestellt und es werden Hinweise für einen möglichen Einsatz gegeben.

Im nächsten Abschnitt wird auf den aktuellen Stand der Technik eingegangen. Einerseits werden hier unterschiedliche Programmierungsmöglichkeiten in Bezug auf Pepper vorgestellt. Andererseits werden hier auch vorangegangene Studienarbeiten in Bezug auf humanoide Roboter am Campus Horb vorgestellt.

Darauffolgend wird die praktische Umsetzung der Programmierung beschrieben. Dabei werden zuerst grundlegende Beispiele aufgezeigt. Im Anschluss werden die einzelnen Funktionen für die Bewegungen des Roboters bei der Suche nach Türschildern vorgestellt. Abschließend werden die Verarbeitung und Erkennung der Bilder erklärt.

Abgeschlossen wird diese Arbeit mit einem Fazit, einer kritischen Bewertung und den möglichen Erweiterungsmöglichkeiten für zukünftige Studienarbeiten. Teile dieser Betrachtung werden bereits in der nächsten Arbeit bearbeitet.

2 Theoretische Grundlagen

In diesem ersten Abschnitt sollen Grundlagen in der Theorie gelegt werden. Zuerst wird eine Einführung in den Bereich der BV gegeben. Dabei wird auf unterschiedliche Themen, wie Farbräume, Filter und die Schrifterkennung eingegangen. Darauffolgend wird der verwendete Roboter Pepper kurz vorgestellt. Eine ausführliche Beschreibung und Hinweise zur Inbetriebnahme und Umgang mit dem Roboter wird in [Bol19] gegeben.

2.1 Einführung in die Bildverarbeitung

Da in der folgenden Arbeit mittels Algorithmen der Bildverarbeitung (BV) eine Lokalisierung des Roboters stattfinden soll, werden zuerst die Grundlagen in diesem Themenbereich gelegt. In diesem Abschnitt kann der sehr große und stetig wachsende Bereich jedoch nur angeschnitten werden. Für weitere Informationen sei auf die verwendete Literatur verwiesen.

Die Ursprünge der BV liegen laut [Neu05] in den 60-er Jahren des letzten Jahrhunderts. In diesen ersten Jahren wurden mittels diskreter Logik-Gatter mikroskopische Bilder ausgewertet. Seitdem hat sich die Technik stark verändert und heutzutage kann auf aufwendige Algorithmen zurückgegriffen werden, welche nachfolgend teilweise vorgestellt werden sollen. Für die Programmierung kann mit Hilfe der Bibliothek OpenCV auf alle Funktionen einfach zurückgegriffen werden.

2.1.1 Farbräume und deren Verwendung

Wenn ein digitales Bild nach der Aufnahme abgespeichert werden soll, muss zuvor bereits festgelegt sein, in welchem Format dies erfolgen soll. Im Allgemeinen kann zwischen Binär-, Grauwert- und Farbbilder unterschieden werden. Eine Transformation im späteren Verlauf zwischen den Farbräumen ist danach teilweise ebenso möglich. Diese sind jedoch entweder mit Informationsverlusten

oder mit einer Approximation des Bildes verbunden. Ein durchgeführtes Beispiel ist in Abbildung 2.1 zu sehen.

Bei Binärbildern (Abbildung 2.1b) erhält jeder Pixel einen booleschen Wert. Dadurch entsteht ein Schwarz-Weiß-Bild ohne weitere Abschwächungen. Die Datenmenge ist hierbei minimal. Diese Bilder sind dabei besonders geeignet, umk einzelne Aspekte hervorzuheben.

Bei Grauwertbildern (Abbildung 2.1c) wird jeder Pixel mittels eines Bytes dargestellt, dadurch stehen nun 256 Zustände pro Pixel zur Verfügung. Dies ermöglicht eine Abstufung zwischen schwarz und weiß, wodurch die Bilder mehr Informationen zur Verfügung stellen. Grauwertbilder haben in der BV einen hohen Stellenwert, da die Bilder ein gutes Nutzen-Aufwand Verhältnis aufweisen.

Bei den Farbbildern (Abbildung 2.1a) gibt es unterschiedliche Codierungsmöglichkeiten, welche in unterschiedlichen Bereichen eingesetzt werden. Innerhalb der BV spielen diese jedoch eine untergeordnete Rolle. In den meisten Fällen werden die gewonnenen Bilder direkt eingegraut oder in die unterschiedlichen Farbkanäle getrennt. An dieser Stelle wird nicht weiter auf Farbbilder eingegangen, da sie keine weitere Verwendung bei der späteren Programmierung spielen.

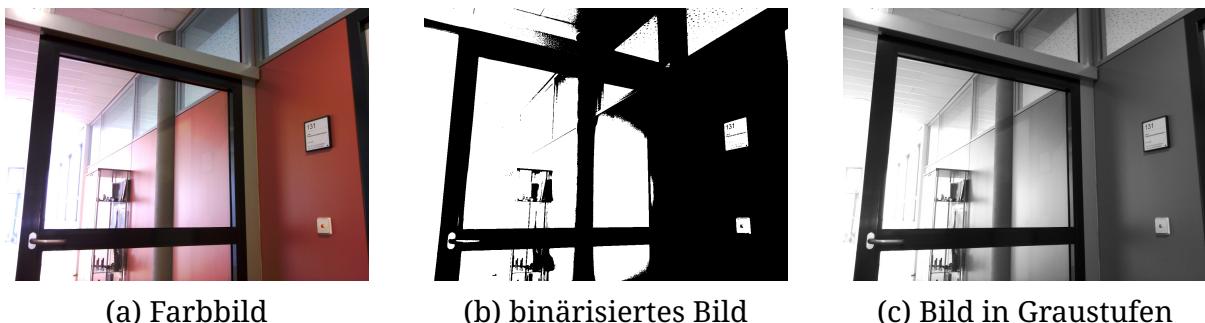


Abbildung 2.1: Darstellung unterschiedlicher Farträume

2.1.2 Lokale Operatoren

Neben einfachen arithmetischen Bildoperationen, welche zwei Bilder Pixel für Pixel verknüpfen, können lokale Operatoren ein einzelnes Bild manipulieren. Dabei arbeiten diese mathematischen Operatoren laut [Erh08] nicht nur auf einem Pixel, sondern beziehen auch die Umgebung (ein Fenster) mit ein. Das Fenster wandert dabei jeweils durch das Ursprungsbild. Die Ergebnisse der mathematischen Anwendungen müssen in einem separaten Ergebnisbild abgespeichert

werden. Bei der Benutzung von lokalen Operatoren muss darauf geachtet werden, dass die Änderung von Randpixeln nicht möglich ist und das Ergebnisbild damit kleiner wird. Weiterhin sind diese Algorithmen teilweise sehr rechenintensiv, besonders bei größeren Fenstern. Die lokalen Operatoren werden in morphologische Funktionen und Filter untergliedert. Im Folgenden soll jeweils ein Beispiel vorgestellt werden, welches im späteren Verlauf Verwendung findet. Ein Beispiel zur Veranschaulichung ist in ?? abgebildet.

Medianfilter, als Beispiel einer morphologischen Operation Morphologische Operatoren manipulieren die Form von Objekten in einem Foto, dabei beziehen sie sich immer auf die Nachbarschaft eines Pixels. Das Medianfilter (Abbildung 2.2b) ist ein Rangordnungsfilter innerhalb der morphologischen Operationen. Bei diesen werden die Grauwerte innerhalb des Fensters der Größe sortiert und bestimmte ausgewählt. Wie der Name bereits vermuten lässt, wird beim Medianfilter der Median der Grauwerte verwendet.

Dadurch können gestörte Bildpunkte oder auch Bildzeilen eliminiert werden. Weiterhin kann durch die Anwendung das Rauschen verkleinert werden. Durch diese Eigenschaft wird das Medianfilter sehr regelmäßig in der BV angewandt. In der OpenCV Bibliothek wird dieser Befehl laut [BK11] *BilateralFilter* genannt.

Gaußfilter, als Beispiel eines Filters Eine effektive Art der Glättung von Bildern ist laut [BB05] die Anwendung eines Gaußfilters. Dabei wird der Grauwert des Pixels z.B. nach Gleichung 2.1 [Erh08] berechnet. In dem Beispiel (Abbildung 2.2c) wird ein 3×3 Fenster mit den Faktoren nach dem Pascalschen Dreieck verwendet. Alternativ können auch größere Fenster oder andere Gewichtungen verwendet werden.

$$h_{ga3}(x, y) = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \quad (2.1)$$



(a) Originalbild

(b) mit Medianfilter

(c) mit Gaußfilter

Abbildung 2.2: Beispiele für lokale Operatoren

2.1.3 Konturen finden

Markante „Ereignisse“ in einem Bild, wie Kanten und Konturen, die durch lokale Veränderungen der Intensität oder Farbe zustande kommen, sind für die visuelle Wahrnehmung und Interpretation von Bildern von höchster Bedeutung. [BB05, S. 111]

Eine Kante lässt sich demnach als Änderung der Intensität an einem Ort und entlang einer Richtung beschreiben. Für die Detektion können unterschiedliche Verfahren genutzt werden, wie gradientenbasierte Operatoren oder auch besondere Filter z.B. das Kompass Filter. Ein weiteres häufig verwendetes Filter ist das Canny Filter. Dabei werden unterschiedliche gerichtete Operatoren auf unterschiedlichen Ebenen auf das Eingangsbild angewandt. Das Ergebnis wird am Ende in einem gemeinsamen Kantenbild abgespeichert. Weiterführende Informationen und eine mögliche Implementierung wird in Codeausschnitt 4.8 oder [BK11] geliefert.

2.1.4 Schrifterkennung (optical character recognition)

Um aus einem Bild oder einer Datei Schrift herauszulesen, können Optical character recognition (OCR)-Verfahren genutzt werden. Die folgende Vorstellung ist in Anlehnung an [Cha13] geschrieben. In den meisten Fällen wird mittels eines „Matrix Matching“ Algorithmus gearbeitet. Dabei wird ein Bild in ein engmaschiges Netz zerlegt. Die daraus resultierenden schwarzen und weißen Punkte werden folgend mit bereits bekannten Anordnungen verglichen. Wenn eine

ausreichend große Anzahl an Übereinstimmungen vorliegt, wird der Buchstabe erkannt.

Die bekannteste Umsetzung dieses Algorithmus wird durch *tesseract* umgesetzt. Diese Software wurde zuerst von HP entwickelt und steht aktuell als open-source-Lösung von Google zur Verfügung. Der Algorithmus hat dabei schon angelernte Trainingssätze für verschiedene Sprachen. Diese können jedoch bei Bedarf auch noch erweitert werden, siehe [CAP19]. Eine Umsetzung in Python erfolgt durch die Bibliothek *pytesseract*.

2.2 Vorstellung des humanoiden Roboters Pepper



Abbildung 2.3: Frontalansicht Pepper¹

Der humanoide Kommunikationsroboter Pepper (siehe Abbildung 2.3) der Firma SoftbankRobotics ist laut eigenen Angaben ([aldc]) 1,20 m groß und wiegt 28 kg. Pepper kann mit dem eingebauten 795 Wh Akku bis zu 12 Stunden im Betrieb sein. Ausgestattet ist der Roboter mit mehreren Mikrofonen, einem Tablet, drei Kameras, mehreren Druck- und Berührungssensoren, zwei Sonaren und sechs Lasersensoren.

Die Kommunikation mit dem Entwickler kann wahlweise über Wi-Fi IEEE 802.11 (2.4GHz/5GHz) oder Ethernet (1000 base T) erfolgen. Personen kann Pepper mit-

¹ [Tec]

tels vier Direktionsmikrofone wahrnehmen, verständigen kann sich der Roboter mittels zwei Lautsprechern und 20 Motoren. Dadurch können Geschwindigkeiten bis zu $3,5 \frac{\text{km}}{\text{h}}$ erreicht werden. Weitere Informationen sind [Bol19] oder [aldc] zu entnehmen. Um eine Unterhaltung mit Pepper zu starten muss sich der Roboter im Autonomous Life (AL) befinden. Dieser Modus erlaubt eine autarke Kommunikation, mittels Mimik und Gestik.

3 Stand der Technik

In dem folgenden Abschnitt wird der aktuelle Stand der Technik dargestellt. Dabei wird zuerst auf die unterschiedlichen Programmierungsmöglichkeiten bei Pepper eingegangen. Durch diese Analyse kann eine für diese Studienarbeit optimale Programmiersprache und -umgebung ausgesucht werden. Im weiteren Verlauf wird auf bereits vorangegangene Studienarbeiten mit den NAO und Pepper Robotern vorgestellt. Durch das Verständnis und spätere Adaption dieses Wissens kann auf erfolgreiche Ansätze aufgebaut werden.

3.1 Programmierungsmöglichkeiten bei Pepper

Um Pepper zu programmieren, sollte bereits vor Beginn des Projekts eine Programmiersprache ausgewählt werden. Im Folgenden sollen dafür drei Möglichkeiten knapp vorgestellt werden.

Anfänger sollten erste Schritte dabei laut [alda] in der pythonbasierten Umgebung Choregraphe erlernen. In der grafischen Oberfläche können einfach Programmblöcke mittels Drag-and-Drop gezogen werden. Intern werden diese in Python umgesetzt und auf den Roboter übertragen. Größere Projekte lassen sich auf Grund der Übersichtlichkeit hier jedoch nicht umsetzen.

Um den „Umweg“ über Choregraphe zu sparen, kann Pepper auch direkt in Python 2.7 programmiert werden. Eine neuere Version wird seitens SoftbankRobotics jedoch nicht unterstützt und zukünftig auch nicht unterstützt werden (Stand 11/2019). Alle Funktionalitäten aus Choregraphe können direkt in Python umgesetzt werden, die Dokumentation ist ebenso mit einigen Programmbeispielen versehen.

Alternativ kann Pepper über Android und Java programmiert werden. Dieser Weg wird aktuell von SoftbankRobotics empfohlen und wird damit auch weiterhin unterstützt werden. Für zukünftige Arbeiten ist ein Wechsel dorthin überlegenswert. Alternativ dazu kann Pepper auch mittels C++ oder mit einem alternativen Betriebssystem z.B. robot operating system (ROS) programmiert werden.

Nach Vorlage der Aufgabenstellung wird die Programmierung in dieser Arbeit mit Python erfolgen. Das Betriebssystem NAOqi bleibt damit ebenso unverändert, da eine synchrone Verwendung vorgesehen ist.

3.2 Vorstellung der Ergebnisse vorangegangener Studienarbeiten

Ab dem Jahr 2014 wurden im Bereich der Informatik bereits mehrere Studienarbeiten im Bereich der Robotik mit NAO und Pepper geschrieben. In den folgenden Abschnitten werden diese kurz zusammengefasst. Die ersten beiden Arbeiten im Jahr 2014/2015 haben sich mit dem Elfmeterschießen mit NAO beschäftigt. In der nachfolgenden Arbeit wurde die Software von Python und Choregraphe auf C++ umgezogen und entsprechende Verbesserungen eingefügt.

Die letzten beiden Arbeiten befassen sich nun mit Pepper, dabei wurde jeweils eine Portier-App erstellt. Die Grundlagen wurden im Jahr 2017/2018 gelegt, wobei die Programmierung in Python erfolgte. Im folgenden Jahr wurde die Software zuerst nach C++ umgezogen und anschließend noch verbessert und erweitert.

3.2.1 Elfmeterschießen mit NAO

Die folgenden Informationen beruhen auf den Studienarbeiten [Bri15], [Vur15]. Im ersten Jahr haben sich zwei Studienarbeiten mit dem Elfmeterschießen mit NAO beschäftigt. Eine hat sich mit der Bewegung des Roboters und die andere mit der Erkennung eines Balles und des Tores beschäftigt. Die Modellierung der Bewegung erfolgte direkt in Choregraphe, die restliche Programmierung erfolgte in Python-Skripten. Die Schussbewegung wurde mit Hilfe einer Bezierkurve modelliert, welche laut [Mol17] nicht eingesetzt werden sollte, wie im Folgenden noch vorgestellt wird.

Die eingesetzte BV erkennt bei gegebenen Lichtverhältnissen einen roten Ball auf einem Parkettfußboden und ein gelbes Tor auf einem weißen Grund. Folgend auf eine erfolgreiche Erkennung richtet sich der Roboter auf den Ball aus und führt einen Schuss mit dem passenden Fuß aus. Die Erkennung der Objekte erfolgte mit Hilfe der OpenCV Bibliothek. Für die Tordetektion wurde zuerst ein Gelbfilter im HSV-Bereich und anschließend ein Opening durchgeführt. Danach kann ein

Rechteck über die gefundenen Konturen gelegt werden. Bei der Erkennung des Balls wurde analog vorgegangen, nur wurde hier ein Filter für rote Farben und ein Kreis statt des Rechtecks verwendet.

Die Ergebnisse der beiden Arbeiten sind kritisch zu betrachten, da einerseits die „Geschwindigkeit der Ausführung“ [Vur15] lange dauert und andererseits die Schussbewegung ebenso langsam durchgeführt wird und der Roboter keinen festen Stand während des Schusses hat. Um diese Probleme zu beseitigen, wurde im folgenden Jahr die Software verbessert. Die Ergebnisse dazu werden im folgenden Abschnitt vorgestellt.

3.2.2 Menschliche Schussbewegung des NAO Roboters in C/C++

In der Arbeit [Mol17] von Benedikt Molitor wurden die beiden vorangegangenen Studienarbeiten analysiert und eine Umsetzung in C++ durchgeführt. Weiterhin wurde die Schussbewegung des NAOs menschenähnlicher durchgeführt. Ein Schwerpunkt lag dabei ebenso bei einer höheren Stabilität des Roboters während der Bewegung.

Die Ergebnisse und Vorschläge für zukünftige Arbeiten wurden nicht weiter betrachtet, da dies die letzte Studienarbeit mit NAO war. Ab dem Folgejahr beschäftigten sich die Arbeiten mit der Erstellung und Verbesserung einer Portier App für Pepper, welche in den folgenden Kapiteln vorgestellt wird.

3.2.3 Pepper - Inbetriebnahme und Entwicklung einer Portier-App

Mit der Anschaffung des humanoiden Roboters Pepper an der DHBW Horb wurde die Arbeit an NAO eingestellt. Aufgabe der folgenden Studienarbeiten ist die Entwicklung einer Portier-App. Dabei ist das Ziel, dass der Roboter sich innerhalb der Hochschule aufhält und vorbeikommenden Personen den Weg zu allen Räumen, Personen oder besonderen Orten zeigen kann. Die erste Studienarbeit [Lab18] hat dabei eine ausführliche Beschreibung des Roboters in Bezug auf den Umgang, die Hardware und die Software gegebenen.

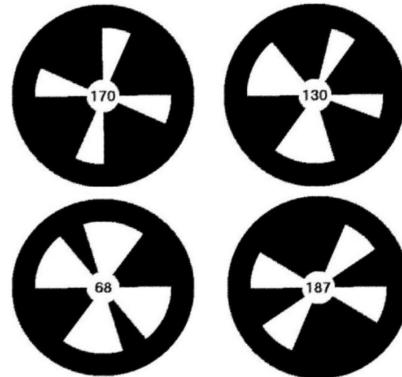


Abbildung 3.1: Beispielhafte Naomarks¹

Neben den Grundlagen konnten bereits erste Ziele in Bezug auf die Portier-App gelingen. Als erstes wurde die Implementierung von unterschiedlichen Interaktionsmodulen zur Begrüßung und Verabschiedung von Personen angelegt. In einer weiteren Entwicklung kann aus einer Unterhaltung heraus der gewünschte Raum oder die Person extrahiert werden. Weiterhin wird eine Wegbeschreibung verbal ausgegeben. Schlussendlich wird der gewünschte Raum farblich markiert auf dem Tablet von Pepper angezeigt. Die Programmierung erfolgte dabei durch Python Skripte, die Anzeige der Bilder erfolgt über Choregraphe. Um die Wegbeschreibung exakt zu erhalten, muss sich Pepper im Eingangsbereich der Hochschule befinden. Für folgende Studienarbeiten wird die flexiblere Nutzung von Pepper empfohlen. Die Wegfindung sollte also dynamisch abhängig von der Startposition durchgeführt werden. Mit diesen Aufgaben beschäftigt sich die nächste Studienarbeit von Felix Klemm.

3.2.4 Dynamische Wegfindung für eine Portier-App am Pepper

Als zweite Studienarbeit [Kle19] an dem humanoiden Roboter Pepper hat sich Felix Klemm mit der dynamischen Wegfindung und der Portierung der vorangegangenen Arbeit [Lab18] in C++ beschäftigt. Weiterhin wird auf eine einfache Positionsbestimmung mittels Naomarks beschrieben. Dieses sind schwarze Kreise mit weißen dreieckigen Fächern, wie sie in Abbildung 3.1 abgebildet sind. Bei dem Einsatz dieser Tags variieren die Winkel, wodurch eine Lokalisierung durchgeführt werden kann.

Die DHBW wird dabei in die Etagen unterteilt und anschließend findet eine Segmentierung in sieben Wegpunkte statt. Diese Wegpunkte können durch einen

¹ [aldb]

Pfadfindungssoftware angefahren werden. Die unterschiedlichen Etagen werden dabei über das zentrale Treppenhaus erreicht. Über eine verbale Ausgabe der Beschreibung mit entsprechender Unterstützung durch Armbewegungen wird der Weg zu der gewünschten Person oder Raum beschrieben.

Für zukünftige Studienarbeiten soll auf die Verwendung von Naomarks verzichtet werden, da die Ausstattung der gesamten Hochschule mit diesen landmarks nicht sinnvoll oder zielführend ist. Für die Startlokalisierung des Roboters muss damit auf ein anderes System zurückgegriffen werden. Weiterhin wird neben der verbalen Beschreibung des Weges auch das Führen von Personen direkt zu den Räumen geplant. Weiterhin ist die Integration anderer Treppenhäuser oder Fahrstühle sinnvoll. Teilweise werden diese Aufgaben in dieser Studienarbeit gelöst, andere Teilprobleme werden parallel dazu von Jonas Boller [Bol19] gelöst oder bleiben für eine spätere Bearbeitung offen.

4 Erkennen der Position innerhalb der DHBW anhand einer Bildauswertung

Um eine Lokalisierung des Roboters innerhalb der Hochschule durchführen zu können, wird eine Bildverarbeitung (BV) eingesetzt. Die Benutzung von landmarks oder auch Naomarks ist nicht möglich, da diese überall befestigt werden müssten und dies einen großen Aufwand entsprechen würde. Der Einsatz von anderen Verfahren, wie NFC, ist ebenso nicht möglich, da hierfür Einrichtungskosten entstehen würden.

In diesem Abschnitt werden zunächst unterschiedliche Strukturelemente an der Hochschule vorgestellt und mit welchen Mitteln diese erkannt werden können. Danach folgend wird die praktische Umsetzung vorgestellt. Dabei nimmt der Roboter zuerst ein Bild auf und versucht darauf mit Hilfe bekannter Strukturelemente seine Position zu identifizieren. Dabei wird auf die vorhandenen Türschilder mit einer eindeutigen Nummerierung zurückgegriffen. Sollte dies zu keinem Ergebnis führen, dreht sich Pepper, um nach anderen Türschilden zu suchen. In dem letzten Abschnitt des Kapitels wird auf aufgetretene und teilweise noch vorhandene Probleme eingegangen.

4.1 Strukturelemente an der DHBW

Um eine Lokalisierung eines Roboters anhand einer BV durchzuführen, müssen bekannte Strukturelemente vorliegend sein. Diese sollen im Folgenden vorgestellt und bewertet werden.

Das häufigste Strukturelement sind Türschilder. Ein beispielhaftes ist in Abbildung 4.1 zu sehen. Der Aufbau folgt dabei immer dem gleichen Schema. In der oberen Zeile steht in großer Schrift die Raumnummer. Diese setzt sich aus der

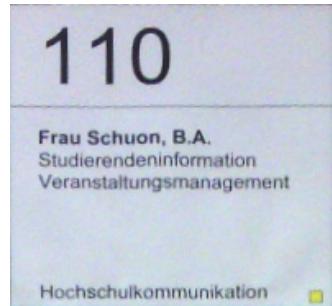


Abbildung 4.1: Beispielhaftes Türschild an der DHBW

Etagenbezeichnung und dem genauen Raum zusammen. Die Etagen werden dabei mit U,0,1 oder 2 gekennzeichnet. Die Raumbezeichnung folgt nach den Fluren, welche bereits in Abschnitt 1.1 vorgestellt wurden. Eine Ergänzung der Raumnummer kann durch nachgestellte Buchstaben erfolgen. Dies ist erforderlich, sobald ein Raum mehrere Eingänge besitzt. Die Bezeichnung a findet sich häufig. Wohingegen die Bezeichnung b lediglich einmal auftritt.

In den folgenden Zeilen steht die Funktion des Raumes und/oder die betreffenden Personen der Räume. In der oberen Ecke kann sich darüber hinaus noch ein weißes Kreuz auf grünen Grund befinden, um einen Standort eines Verbandskastens anzuzeigen. In den meisten Fällen befindet sich das Türschild in einem blauen Rahmen neben der entsprechenden Tür. In manchen Fällen ist es jedoch auch ohne Rahmen direkt auf der Tür befestigt. Die Erkennung der Türschilder bietet eine gute Möglichkeit sich innerhalb der DH zu lokalisieren. Dies lässt sich einerseits mit der hohen Anzahl an Türschildern begründen. Andererseits sind diese eindeutig und einmalig, damit lässt sich die Position genau bestimmen. Weiterhin geben sie auch eine einfache Information zu dem jeweiligen Geschoss wieder. Um die Türschilder zu erkennen, muss zuerst eines gefunden werden und anschließend mittels eines Algorithmus die Schrift erkannt und ausgelesen werden. Die beiden angesprochenen Punkte werden in den folgenden Kapiteln vorgestellt.

Neben diesen eindeutigen Strukturelementen gibt es noch weitere Erkennungspunkte für einen Roboter. Zu den anderen leicht erkennbaren Elementen gehören vorrangig große Gegenstände, welche mehrfach an der DH auftreten. Dazu gehören die Fahrstühle, die Treppenaufgänge, die Litfaßsäulen und der Bibliothekskasten im zweiten Obergeschoss. Durch die Größe der Elemente sind sie auf Bildaufnahmen leicht wiederzuerkennen. In dieser Studienarbeit wurden diese Elemente jedoch aus zeitlichen Gründen nicht für eine Lokalisierung verwendet.

Für eine zukünftige Arbeit und einer Lokalisierung im Empfangs- oder Lobbybereich der Hochschule sind diese jedoch notwendig.

4.2 Vorstellung der verwendeten Bibliotheken

Für die Erstellung, der im Folgenden vorgestellten Software, wurden unterschiedliche Bibliotheken verwendet. Einen groben Überblick darüber geben die nächsten Abschnitte.

pytesseract Die Bibliothek *pytesseract* stellt eine Verwendung von *tesseract* in Python dar. Die Schrifterkennung *tesseract* wurde dabei bereits in 2.1.4 vorgestellt. Um die Bibliothek verwenden zu können, muss ebenso *tesseract* installiert sein. In der einfachsten Funktion wird der Bibliothek ein Bild übergeben und als Rückgabe werden die gefundenen Zeichen gegeben.

OpenCV (cv2) Um die Bilder verarbeiten zu können wird die Bibliothek *cv2* verwendet, diese ermöglicht eine Verwendung von OpenCV in Python. Verwendung finden die Funktionen vorrangig bei der BV. Dabei können mit der Bibliothek Bilder zugeschnitten, farblich manipuliert oder Filter angewandt werden.

numpy Ebenso für die BV wird die Bibliothek *numpy* benötigt, diese ermöglicht den einfachen Umgang mit Arrays und vielen vereinfachten mathematischen Operatoren. Verwendung finden diese, wenn bestimmte Arraystrukturen mit bestimmten Daten vorbefüllt werden sollen, oder bei der Transformation von Bildern in Zahlenarrays.

imutils Die Bibliothek *imutils* ermöglicht eine einfachere Handhabung von OpenCV und Bildern im Allgemeinen. In der späteren Arbeit wurde es bei der Detektion von Kanten und dem Sortieren der gefundenen Konturen eingesetzt. Die Bibliothek bietet darüber hinaus noch weitere Befehle zum einfachen Schneiden, Drehen und Skalieren von Bildern an.

Damit wurden nun alle verwendeten Bibliotheken für die BV vorgestellt. Im Folgenden werden die spezifischen Bibliotheken für die erfolgreiche Programmierung von Pepper vorgestellt.

time Dank der *time* Bibliothek ist es möglich Zeiten zu messen. Dabei wird immer die aktuelle Zeit ausgelesen. Eine Zeitdifferenz kann damit einfach durch eine Subtraktion berechnet werden. Verwendet wird dies, sobald sich Pepper für eine bestimmte Zeit drehen soll, oder alternativ für die Überwachung von bestimmten Abbruchbedingungen. Die genaue Verwendung wird in Codeausschnitt 4.2 beschrieben.

qi Um auf die Funktionen von Pepper zugreifen zu können, muss die Bibliothek *qi* angebunden werden. Mit Hilfe dieser Funktionen können Service implementiert werden, auf den Speicher aus Python zugegriffen werden oder auch Motoren angesteuert werden. Die genauen Verwendungsmöglichkeiten werden im folgenden Abschnitt teilweise vorgestellt. Im weiteren Verlauf der Programmierung wird darauf auch regelmäßig zurückgegriffen.

4.3 Initialbefehle für die Raumsuche

Um die Funktionalität des Programmes zu gewährleisten, müssen manche Bedingungen erfüllt sein. Diese sollen im Folgenden vorgestellt werden.

4.3.1 Autonomus Life Bewegungen deaktivieren und Sicherheitsabstände minimieren

Pepper besitzt, wie bereits beschrieben, einen Autonomus Life (AL)-Modus. Dieser wird bei der folgenden Programmierung teilweise deaktiviert, um ein erfolgreiches Ausführen des eigentlichen Programms zu ermöglichen. Wenn dieser Modus nicht deaktiviert wird, sucht Pepper eigenständig nach Kontakt zu Menschen. Dies führt dazu, dass sich der Kopf und eventuell auch die Räder bewegen. Diese Bewegungen würden in der späteren Arbeit zu Problemen führen, da beispielsweise für die Aufnahme eines Bildes der Roboter stillstehen muss. Weiterhin wird so auch sichergestellt, dass der Roboter keine Gespräche beginnt, während er nach einer Raumnummer sucht.

Des Weiteren werden die Sicherheitsabstände auf 15 cm verkürzt. Durch diesen Eingriff kann sich Pepper näher an Objekte bewegen und an diesen auch besser drehen. Durch die schmalen Gänge an der DH und dem eventuellen Aufenthalt

von Personen bei Pepper, ist dies zwingend erforderlich. Dabei muss jedoch beachtet werden, dass Pepper nun auf Wände hinzu beschleunigen kann und erst kurz vor der Wand abremst. Dieses Verhalten kann von Menschen als Zusammenstoß mit der Wand interpretiert werden. Zu diesem Zustand kann es jedoch nicht kommen, da die Geschwindigkeiten hinreichend klein sind. Nach Beendigung des Programms müssen die Sicherheitsabstände zwangsweise wieder auf ihren Initialwert gesetzt werden.

4.3.2 Services anbinden und Einschreiben zum Videostream

Um auf bestimmte Funktionen zuzugreifen, müssen zuerst bestimmte Services angelegt werden. Dadurch kann im späteren Verlauf der Programmierung auf diese Funktionen zugegriffen werden. Die Instanziierung der benötigten Services wird vollständig in Codeausschnitt 4.1 beschrieben. Für eine bessere Darstellung werden alle vorgestellten Codeausschnitte mit vereinfachten und ausgewählten Kommentaren in dieser Arbeit dargestellt, die vollständige Fassung befindet sich im Anhang.

In der letzten Zeile wird ebenso die Kamera initialisiert und ein Video Client angelegt, mit diesem ist es möglich Bilder zu empfangen. Die Übergabeparameter der Funktion sind die Auflösung, der Farbraum und die Bildfrequenz (fps). Eine Auflistung der unterschiedlichen Parameter ist dabei [alda] zu entnehmen.

```
1 video_service = session.service("ALVideoDevice")
2 autonomous_service = session.service("ALAutonomousLife")
3 speech_service = session.service("ALTextToSpeech")
4 motion_service = session.service("ALMotion")
5 memory_service = session.service("ALMemory")
6 resolution = 3      # VGA
7 colorSpace = 11    # RGB
8 videoClient = video_service.subscribe("python_client", resolution,
                                         colorSpace, 5)
```

Codeausschnitt 4.1: Anbindung Services und Kamera Initialisierung

4.4 Ausrichten des Roboters auf eine Wand

Für die Erkennung von Türschildern und eine entsprechende Ermittlung der Raumnummer für eine Lokalisierung ist es notwendig, dass Pepper sich zu einer

Wand innerhalb der DH ausrichtet, dafür muss sich der Roboter innerhalb eines Ganges befinden. Die entwickelten Algorithmen sollen im Folgenden vorgestellt werden.

Drehung zur Wand Der entwickelte Algorithmus liegt als eigenständige Funktion vor und wird nach der Initialisierungsphase aufgerufen. Pepper dreht sich dabei zunächst solange um die eigene Achse, bis eine bestimmte Distanz nach vorne unterschritten wird oder alternativ eine Kontrollzeit abgelaufen ist. Sollte zweiteres passieren, wurde keine Wand gefunden und es kann geschlussfolgert werden, dass Pepper sich nicht in einem Gang befindet. Die Programmierung sieht wie in Codeausschnitt 4.2 gezeigt aus. Die Drehung wird in Zeile 5 aktiviert und erst wieder nach der while-Schleife in Zeile 16 deaktiviert.

```

1 # aktuelle Sonarwerte aus Speicher auslese
2 frontDistance = memory_session.getData("Device/SubDeviceList/Platform/
    Front/Sonar/Sensor/Value")
3 backDistance = memory_session.getData("Device/SubDeviceList/Platform/
    Back/Sonar/Sensor/Value")
4 # Drehung aktivieren
5 motion_service.move(0,0,2)
6 # Timer starten, notwendig für Laufzeitüberwachung
7 startTime = time.time()
8 actualTime = time.time()
9 # lasse Pepper solange drehen, wie der vordere Abstand zu groß ist und
    die Zeitüberwachung i.O. ist
10 while frontDistance > distanceSearch and actualTime-startTime <
    timeSecurity:
11     frontDistance = memory_session.getData("Device/SubDeviceList/
        Platform/Front/Sonar/Sensor/Value")
12     backDistance = memory_session.getData("Device/SubDeviceList/
        Platform/Back/Sonar/Sensor/Value")
13     actualTime = time.time()
14     pass
15 # deaktiviere die Bewegung, bremse jedoch nicht abrupt ab
16 motion_service.move(0,0,0)

```

Codeausschnitt 4.2: Drehung Peppers zu einer Wand

Für die Drehung wurde eine Winkelgeschwindigkeit von $2 \frac{\text{rad}}{\text{s}}$ ausgewählt. Mit Hilfe des *move* können für die x- und y-Achse und dem Winkel θ jeweils Geschwindigkeiten vorgegeben werden. Pepper dreht sich mit diesen Geschwindigkeiten, bis ein neuer Befehl kommt. Um die Bewegung wieder zu stoppen kann entweder ein *stopMove* verwendet werden, oder die Geschwindigkeit in jede Richtung auf

null gesetzt werden. Letztere Möglichkeit wird verwendet, da erstere zu einer unnatürlichen Bewegung und einem sehr abrupten Verhalten führt. Während der Schleife werden die aktuellen Entfernungswerte von den Sonarsensoren aus dem Speicher eingelesen. Weiterhin wird die aktuelle Zeit neu gespeichert.

Entfernung zur Wand anfahren Da sich Pepper nach diesem ersten Schritt in Blickrichtung zu einer Wand befindet, kann in einem nächsten Vorgang die Entfernung zu der Wand eingestellt werden. Dafür bewegt sich der Roboter je nach vorliegender Entfernung nach vorne oder nach hinten. Der entsprechende Programmcode ist in Codeausschnitt 4.3 zu sehen.

```

1  # auf den gewünschten Abstand an die Wand heran fahren
2  if frontDistance > distanceWall:
3      # langsame Bewegung nach vorne, Abstand wird kleiner bis der gewünschte
4      # Wert erreicht ist
5      motion_service.move(0.2,0,0)
6      while frontDistance > distanceWall:
7          frontDistance = memory_session.getData("Device/SubDeviceList/
8              Platform/Front/Sonar/Sensor/Value")
9          pass
10     else:
11         # langsame Bewegung nach hinten, Abstand wird größer bis der gewü
12         # nschte Wert erreicht ist, Sicherheitsabstand nach hinten wird
13         # ebenso überprüft
14         motion_service.move(-0.2,0,0)
15         while frontDistance < distanceWall and backDistance >
16             distanceSecurity:
17             frontDistance = memory_session.getData("Device/SubDeviceList/
18                 Platform/Front/Sonar/Sensor/Value")
19             backDistance =memory_session.getData("Device/SubDeviceList/
20                 Platform/Back/Sonar/Sensor/Value")
21             pass
22         motion_service.move(0,0,0)
23         pass

```

Codeausschnitt 4.3: Anfahren der richtigen Entfernung zur Wand

Dabei wird zuerst überprüft, ob sich Pepper nach vorne oder nach hinten bewegen muss. Die Geschwindigkeit beträgt dabei jeweils $0,2\frac{m}{s}$. Sobald die gewünschte Entfernung zur Wand hergestellt wurde oder der Mindestabstand nach hinten unterschritten wurde, rollt Pepper langsam aus. Der entsprechende Befehl setzt dazu, wie bereits beschrieben, die Geschwindigkeit in alle Richtungen auf null herab.

Steuersignale der Funktion Bei der Abhandlung der beschriebenen Programme können Fehler auftreten, die auch an die übergeordnete Funktion übertragen werden müssen. Sollte keine Wand gefunden werden, der Roboter sich also mehr als 15 Sekunden im Kreis drehen, wird das Programm sofort abgebrochen und eine 5 übermittelt. In diesem Fall sollte der Roboter an einer anderen Position mit der Lokalisierung starten. Dabei gilt es zu beachten, dass Pepper mit diesem Programm lediglich in den Gängen der DH arbeiten kann.

Der nächste Fehler kann erzeugt werden, wenn Pepper bei einer Rückwärtsbewegung hinten einen zu geringen Abstand ermittelt. Dies kann passieren, sobald sich Menschen hinter Pepper aufhalten oder andere Gegenstände, wie Schränke sich im Gang befinden. Die Warnung kann in den meisten Fällen ignoriert werden, hilft jedoch beim Debuggen. Sobald der Abstand nach vorne jedoch zu gering ist, kann die Aufnahme eines guten Bildes für die weitere Verarbeitung nicht garantiert werden. In diesem Fall wird eine 6 übertragen. Sollte alles nach Plan laufen, wird als Bestätigung eine 0 an das Oberprogramm übermittelt.

4.5 Aufnahme von Bildern

Sobald sich Pepper auf eine Wand ausgerichtet hat und in einer angemessenen Entfernung befindet, kann ein Bild aufgenommen werden. Als ideal hat sich dabei eine Entfernung von 1,5 m herausgestellt. Das aufgenommene Bild kann in einem späteren Schritt, siehe Abschnitt 4.6 analysiert werden, um eine Raumnummer zu identifizieren.

Um ein nicht verwackeltes Bild aufzunehmen, muss als erstes der Kopf in eine stabile Lage gebracht werden und nach vorne ausgerichtet werden. Diese Bewegung sollte vor jeder Aufnahme eines Bildes erfolgen, da durch die sonstigen Drehbewegungen der Kopf nach Hinten verschoben wird. Der entsprechende Codeausschnitt zur Bewegung des Kopfes ist in Codeausschnitt 4.4 ersichtlich. Die Kopfbewegung wird dabei durch die beiden Gelenke (Head yaw und Head pitch) im Kopf durchgeführt. Dafür werden die absoluten Winkel der Funktion *setAngles* übergeben. Als letztes muss noch die Winkelgeschwindigkeit für die Bewegung übergeben werden.

```
1 motion_service.setStiffnesses("Head", 1.0)
2 motion_service.setAngles(["HeadYaw", "HeadPitch"], [0.0, -0.3], 0.1)
```

Codeausschnitt 4.4: Bewegung des Kopfes für die Aufnahme eines Bildes

Die eigentliche Bildaufnahme kann, wie in Codeausschnitt 4.5 beschrieben, durchgeführt werden. Dafür muss zuerst eine Wartezeit eingelegt werden, um alle Schwingungen im Roboter auszugleichen. Danach kann mit Hilfe des Video Services ein Bild aufgenommen werden. Das empfangende Array enthält als erstes die Breite, gefolgt von der Höhe des Bildes. Auf Arrayplatz sieben befindet sich das eigentliche Bild in Form eines Bytearrays. Die Erzeugung einer Bilddatei erfolgt teilweise fehlerhaft, da der Service nicht erreichbar ist. Sollte dies der Fall sein, wird lediglich ein mit *Null* gefülltes Array zurückliefert. Da das Problem erst später auffällt, wurde dieser Codebereich mit einem *try-except* versehen

```

1 # vor Aufnahme des Bildes warten, damit sich der Körper einschwingt.
2   VideoService notwendigerweise wieder schließen
3 time.sleep(2.0)
4 naoImage = video_service.getImageRemote(videoClient)
5 video_service.unsubscribe(videoClient)
6 #Bild aus Array erzeugen
7 imageWidth = naoImage[0]
8 imageHeight = naoImage[1]
9 # wenn der Bildservice nicht erreichbar ist, Fehler abfangen
10 try:
11     array = naoImage[6]
12     image_string = str(bytarray(array))
13     pass
14 except:
15     print "Bildaufnahme nicht erfolgreich! Service nicht erreichbar!
16       Neustart des Roboters notwendig!"
17     return str(5)
18     pass

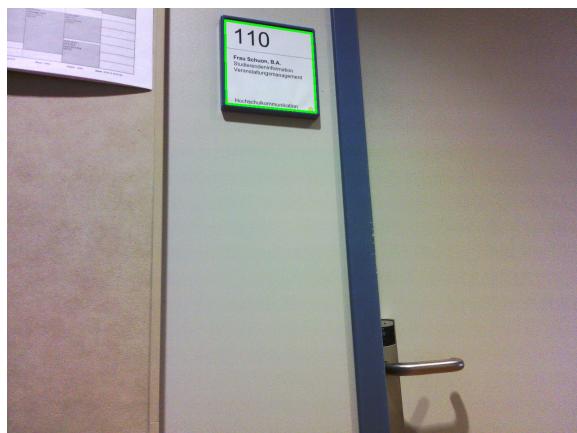
```

Codeausschnitt 4.5: Aufnahme eines Bildes mit Pepper

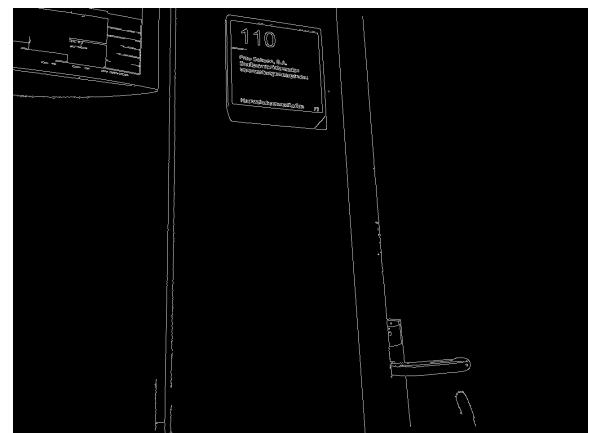
4.6 Anhand des Türschildes die Position bestimmen

Mit den bisherigen Codeausschnitten kann sich Pepper eine Wand suchen, an dieser ausrichten und ein Bild aufnehmen. In diesem Abschnitt wird nun die Verarbeitung der Bilder und der entsprechenden Ermittlung einer Raumnummer aufgezeigt. Dabei werden zuerst die Hilfsfunktionen für die Verarbeitung vorgestellt und im letzten Abschnitt können alle Funktionen zu einer Wirkseinheit zusammengefasst werden.

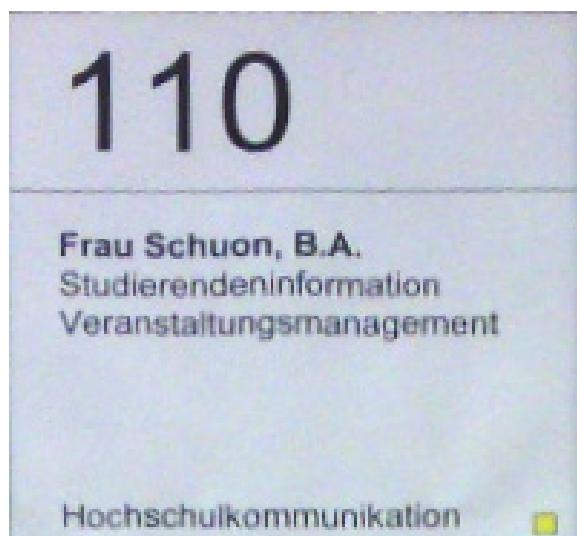
Der grobe Ablauf wird bereits im Folgenden vorgestellt und ist in der nachfolgenden Abbildung 4.2 zu sehen. In einem aufgenommenen Bild wird zuerst nach Kanten gesucht. Dieses Verfahren soll Aufschluss auf das Türschild geben, da dieses eine harte Kante zu der Umgebung hat. Aus den gesammelten Kanten können Vierecke gebildet werden. Mit diesen können in einem nächsten Schritt die Bilder zugeschnitten werden. Um den Text besser verarbeiten zu können, sollten diese danach wieder ausgerichtet werden. Das Ergebnis ist in Abbildung 4.2d ersichtlich. Mit diesem Schritt ist die BV abgeschlossen. Nachfolgend wird mit Hilfe, der bereits vorgestellten *tesseract* Bibliothek für Python der Text auf dem Bild extrahiert. Final kann nun noch in den gefundenen Texten ein Raum gesucht werden.



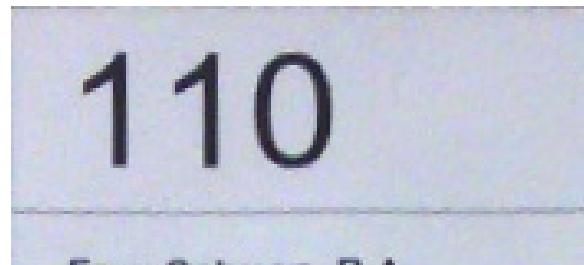
(a) Aufgenommenes Bild mit eingezeichneten Kanten



(b) Canny Algorithmus zur Detektierung von Kanten



(c) Transformiertes Türschild



(d) zugeschnittenes Türschild lediglich mit Raumnummer

Abbildung 4.2: Ablauf der vorgestellten BV anhand von vier Bildern

Vorstellung der OCR Funktion Als erste Funktion soll die eigentliche Schrifterkennung vorgestellt werden. Dabei muss zuerst der Installationspfad hinterlegt werden. Danach kann mittels der Funktion in Zeile 4, die OCR durchgeführt werden. Der empfangene Text wird ohne weitere Verarbeitung an die aufrufende Funktion zurückgegeben.

```
1 pytesseract.pytesseract.tesseract_cmd = r'C:\Users\...\07_Tesseract-OCR\n\tesseract'
2
3 def Ocr(img):
4     text = pytesseract.image_to_string(img)
5     return text
```

Codeausschnitt 4.6: Vorstellung der OCR Funktion

Rooming Die Funktion Rooming führt einen Algorithmus für die eben vorgestellte Methode für die Schrifterkennung aus und verarbeitet den empfangenen Text weiter. Das Ziel der Funktion (siehe Codeausschnitt 4.7) ist dabei, eine Raumnummer in dem Text zu finden. Bei der Erkennung der Buchstaben aus den Bildern werden teilweise Einsen als Vieren erkannt. Dieses Problem ist ebenso an den notwendigen Stellen gelöst. Die Regular Expression (Regex) in Zeile 4 stellt dabei das Herzstück der Funktion dar und sucht nach einem mindestens drei Zeichen langen String. Dieser muss auf die Beschreibung einer Raumnummer passen.

In den noch folgenden Zeilen wird die Problematik mit der Falscherkennung der vieren umgangen und die möglichen falschen Zahlen werden durch Einsen ersetzt.

```

1 def Rooming(img):
2     text = Ocr(img)
3     #erstes Zeichen Zahl 0-2,4 oder uU, gefolgt von 0-4, danach Zahl und
4     # beliebige Anzahl an Leerzeichen, null oder ein a und null oder ein
5     # b
6     splitted = re.findall("[0-2,4,u,U][0-4][0-9] *a{0,1}b{0,1}", text)
7     room = 0
8     valid = False
9     for number in range(len(splitted)):
10        room = splitted[number]
11        if room[0] == '4':
12            room = '1' + room[1:]
13        if room[1] == '4':
14            first = room[0]
15            room = first + '1' + room[2:]
16        valid = True
17        break
18    return splitted, room, valid

```

Codeausschnitt 4.7: Vorstellung der Rooming Funktion

getEdges Nach der Aufnahme eines Bildes soll dieses zuerst auf ein mögliches Türschild zugeschnitten werden. Dies hat den Vorteil, dass die Erkennungsrate durch den OCR höher ist. Dafür erhält die Funktion *getEdges* ein Bild und gibt nach der Größe sortiert acht zusammenhängende Kanten zurück. Der nun beschriebene Quellcode ist in Codeausschnitt 4.8 und einer Serie an Beispielbildern, an denen die Funktionalität klargestellt werden soll, in Abbildung 4.3 zu sehen.

Die vorgestellte Methode ist in Anlehnung an [Ros] geschrieben. Das Eingangsbild wird zuerst eingegraut(4.3a), um einerseits weniger Bildinformationen zu haben und andererseits, weil die nachfolgenden Befehle, lediglich diese Bilder verarbeiten können. Danach kann in Zeile 4 der bereits in 2.1.3 beschriebene Canny-Filter angewandt werden. Das Ergebnis dieser Bildmanipulation ist in 4.3c zu sehen. Nach diesem Schritt können Konturen gesucht werden. Die gefundenen Kanten, werden in der darauffolgenden Codezeile noch sortiert und auf eine Gesamtanzahl von acht Konturen begrenzt. Da das Türschild auf den Bildern groß aufgenommen werden soll, ist dies ausreichend. In der nächsten beschriebenen Funktion kann nun mit diesen Kanten, eine geeignete Drehmatrix erstellt werden. Mit dieser kann danach das Türschild entsprechend ausgerichtet werden.



(a) Änderung der Bittiefe auf 8-Bit

(b) Durchführung einer Glättung

(c) Durchführung Canny Transformation

Abbildung 4.3: Ablauf der vorgestellten getEdges Funktion anhand von drei Bildern

```

1 def getEdgesFromImage(orig):
2     img_gray = cv2.cvtColor(orig, cv2.COLOR_BGR2GRAY) #eingrauen
3     gray = cv2.bilateralFilter(img_gray, 11, 17, 17) #Kanten glätten in
4         grauen Bild
5     edged = cv2.Canny(gray, 30, 200) #Canny Filter anwenden, schwarzer
6         Hintergrund mit weißen Kanten
7     #Konturen finden, nach der Größe sortieren und nur 8 behalten
8     cnts = cv2.findContours(edged.copy(), cv2.RETR_TREE, cv2.
9         CHAIN_APPROX_SIMPLE)
10    cnts = imutils.grab_contours(cnts)
11    cnts = sorted(cnts, key = cv2.contourArea, reverse = True)[:8]
12    return cnts, edged

```

Codeausschnitt 4.8: Vorstellung der getEdges Funktion

Perspektivische Transformation Wie bereits eingeleitet, berechnet diese Funktion eine Drehmatrix um ein mögliches Türschild orthogonal auszurichten. Dies ist notwendig da Pepper auch von einer schrägen Position Türschilder aufnehmen kann und diese dann nicht von der Schrifterkennung verarbeitet werden können. Der Programmcode ist ebenfalls in Anlehnung an [Ros] geschrieben. Auf die mathematischen Besonderheiten wird dabei hier nicht eingegangen, sondern es wird lediglich die Vorgehensweise beschrieben.

Der Quellcode für das Verständnis ist in Codeausschnitt 4.9. In einem ersten Schritt werden die Eckpunkte der gefundenen Konturen sortiert und abgespeichert. Dabei liegt an Stelle null des Arrays der obere linke Punkt und die anderen Punkte folgen im Uhrzeigersinn. Dies geschieht in den Zeilen 6 bis 10. In den folgenden Zeilen wird die maximale Breite und Höhe des Vierecks berechnet, da dies für die Berechnung der Drehmatrix benötigt wird. Als letzter Schritt kann diese nun in Zeile 26 berechnet werden.

```

1 def PerspectiveTransformation(screenCnt):
2     pts = screenCnt.reshape(4, 2) #Array kopieren
3     rect = np.zeros((4, 2), dtype = "float32")
4     #Bestimmung der einzelnen Eckpunkte und Sortierung im Uhrzeigersinn
5     s = pts.sum(axis = 1)
6     rect[0] = pts[np.argmin(s)]
7     rect[2] = pts[np.argmax(s)]
8     diff = np.diff(pts, axis = 1)
9     rect[1] = pts[np.argmin(diff)]
10    rect[3] = pts[np.argmax(diff)]
11    (tl, tr, br, bl) = rect
12    #maximale Höhe und Breite des Vierecks bestimmen
13    widthA = np.sqrt(((br[0] - bl[0]) ** 2) + ((br[1] - bl[1]) ** 2))
14    widthB = np.sqrt(((tr[0] - tl[0]) ** 2) + ((tr[1] - tl[1]) ** 2))
15    heightA = np.sqrt(((tr[0] - br[0]) ** 2) + ((tr[1] - br[1]) ** 2))
16    heightB = np.sqrt(((tl[0] - bl[0]) ** 2) + ((tl[1] - bl[1]) ** 2))
17    maxWidth = max(int(widthA), int(widthB))
18    maxHeight = max(int(heightA), int(heightB))
19    # Zielkoordinaten bilden für eine top-down Sicht
20    dst = np.array([
21        [0, 0],
22        [maxWidth - 1, 0],
23        [maxWidth - 1, maxHeight - 1],
24        [0, maxHeight - 1]], dtype = "float32")
25    # Transformationsmatrix berechnen
26    M = cv2.getPerspectiveTransform(rect, dst)
27    return M, maxWidth, maxHeight

```

Codeausschnitt 4.9: Vorstellung der Perspective Transformation Funktion

Detect Room In dem abschließenden Abschnitt werden die zuvor vorgestellten Funktionen in einer Wirkeinheit verkettet. Dafür wird nach dem Empfang und dem Sichern des Eingangsbildes dieses zuerst auf Kanten untersucht. Sollten diese gefunden werden, kann über alle iteriert werden. Sobald vier Kanten zu einer Kontur gehören, können diese Punkte weiter untersucht werden.

Darauffolgend kann die bereits beschriebene Drehmatrix für die perspektivische Transformation berechnet werden. Im nächsten Schritt (Zeile 19) kann diese dann auf das Ausgangsbild angewandt werden. Danach wird das nun entstandene Bild noch weiter zugeschnitten, um ein möglichst gutes Ergebnis bei der darauffolgenden Schrifterkennung zu erzielen. Einerseits wird dazu ein eingegrautes Bild verwendet. Sollte dies nicht erfolgreich sein, wird der OCR Algorith-

mus noch auf ein Bild mit angewandtem Gaußfilter durchgeführt. Durch diesen Einsatz können manche zuvor nicht erkannte Türschilder noch ausgelesen werden.

```

1 def DetectRoom(img):
2     orig = img.copy()
3     cnts, edged = getEdgesFromImage(orig) # Kanten im Bild ermitteln
4
5     screenCnt = None
6     room = 0
7     splitted = ''
8     valid = False
9     warp = orig.copy()
10    # über alle Kanten iterieren
11    for c in cnts:
12        peri = cv2.arcLength(c, True)
13        approx = cv2.approxPolyDP(c, 0.03 * peri, True)
14        # sobald die Kontur vier Punkte hat, ist es ein Viereck, könnte also
15        # das gesuchte Türschild sein
16        if len(approx) == 4:
17            screenCnt = approx
18
19        M, maxWidth, maxHeight = PerspectiveTransformation(screenCnt) # 
20        # Drehmatrix zu dem gegebenen Viereck ermitteln
21        warp = cv2.warpPerspective(orig, M, (maxWidth, maxHeight)) #
22        # Drehmatrix auf die Konturen auf das Bild anwenden und
23        # darauf zuschneiden
24        width, height = warp.shape[:2] # Höhe und Breite des Tür-
25        # schildes ermitteln
26        cropped = warp[0:int(height*0.4),0:int(width*0.95)] # auf
27        # kleineren Bereich zuschneiden
28        img_gray = cv2.cvtColor(cropped, cv2.COLOR_BGR2GRAY)
29        gaus = cv2.adaptiveThreshold(img_gray, 255, cv2.
30            ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 91, 12)
31        # je nach Bildqualität ist das zugeschnittene Bild oder das
32        # verarbeitete Bild nach der Anwendung des Gausfilters besser
33        splitted, room, valid = Rooming(cropped)
34        if not valid:
35            splitted, room, valid = Rooming(gaus)
36        if valid:
37            break
38
39    return room, valid, images

```

Codeausschnitt 4.10: Vorstellung der Detect Room Funktion

4.7 Weitergabe der gefundenen Raumnummer

Um die gefundene Raumnummer auch in anderen Programmen nutzen zu können, wird diese in den Speicher von Pepper geschrieben, dies erfolgt durch die abgebildete Codezeile (4.11). Durch diesen finalen Schritt können auch andere Programme auf die erzeugte Information zurückgreifen. Somit kann Pepper autonom in einem Gang nach einem Türschild suchen, dieses identifizieren, die Raumnummer extrahieren und abspeichern.

```
1 memory_session.insertData("detectedRoom", room)
```

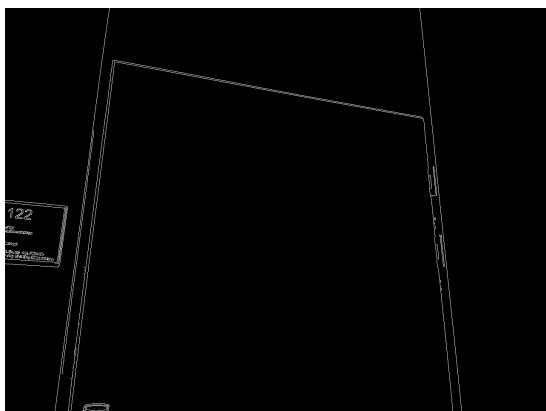
Codeausschnitt 4.11: Speicherung des gefundenen Raums

4.8 Grenzen des erstellten Algorithmus

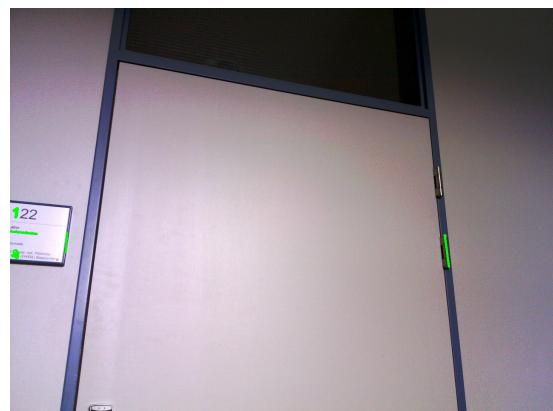
Bei der Erstellung des Programms und während der Testphase haben sich Probleme ergeben, welche im Folgenden vorgestellt werden sollen. Diese konnten nicht volumnäßig softwareseitig behoben werden. Der Großteil bezieht sich dabei auf eine nicht ausreichende Bildqualität. Die angesprochenen Punkte werden dabei jeweils mit einem beispielhaften Aufnahmefoto und der Auswertung nach der vorgestellten Software (siehe Kapitel 4) dargestellt.

Der Türrahmen ist nicht vollständig auf dem Bild Wenn ein Türschild nur teilweise auf dem Bild ersichtlich ist (Abbildung 4.4), kann keine Raumnummer identifiziert werden. Die Funktion `getEdges` kann aus diesem Bild lediglich die äußereren Kanten detektieren. Dies hat zur Folge, dass das Vieleck nicht aus vier Kanten besteht, welche weiterhin keine durchgehende Verbindung aufweisen. Dadurch kann der weitere Algorithmus ebenso nicht erfolgreich durchgeführt werden.

Ein angewandter Lösungsansatz ist die wiederholende Ausrichtung des Kopfes, um ein Aufschaukeln und ein damit einhergehendes vertikales Verschieben des Türschildes im Aufnahmebereich zu verhindern. Ein seitliches Abschneiden des Türschildes wird durch die Aufnahme mehrerer Bilder aus unterschiedlichen Blickwinkeln umgangen.



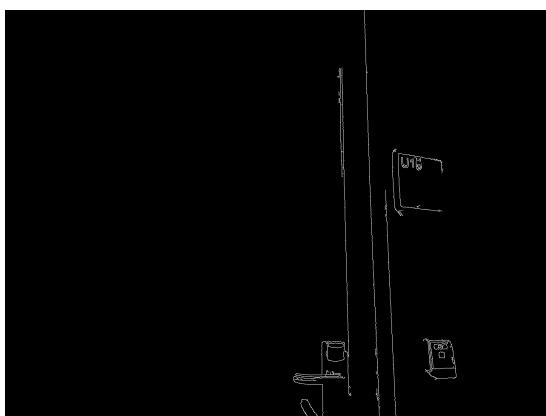
(a) Anwendung des Canny Algorithmus



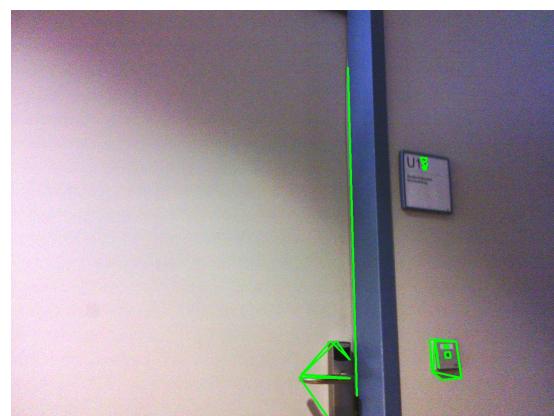
(b) erkannte Flächen in grün

Abbildung 4.4: Türschild befindet sich nicht ganz im Bild

Das Bild ist zu dunkel Ein weiteres Problem tritt auf, sobald der Gang schlecht ausgeleuchtet ist. Dies kann auftreten, wenn das Ganglicht nicht eingeschaltet ist oder der Kontrast zu gering gewählt ist. Letzteres tritt vor allem im Untergeschoss der DH regelmäßig auf. Ein Beispielhaftes Bild dazu ist in Abbildung 4.5 zu sehen. Der Unterschied zwischen einem ausreichend beleuchteten Bild und einer Fehlaufnahme, welche nicht sinnvoll weiterverarbeitet werden kann, sind dabei sehr gering und gehen abhängig vom Blickwinkel bei gleicher Beleuchtung ineinander über.



(a) Anwendung des Canny Algorithmus



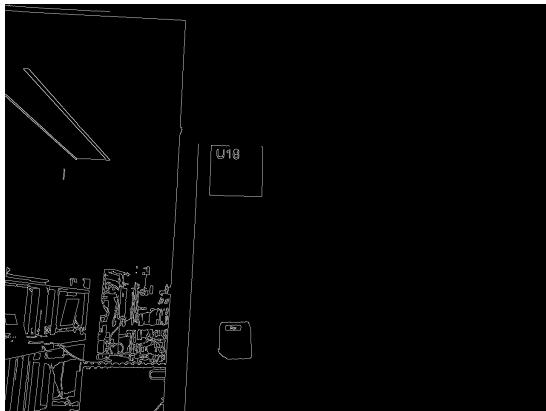
(b) erkannte Flächen in grün

Abbildung 4.5: Türschild zu dunkel für eine Auswertung

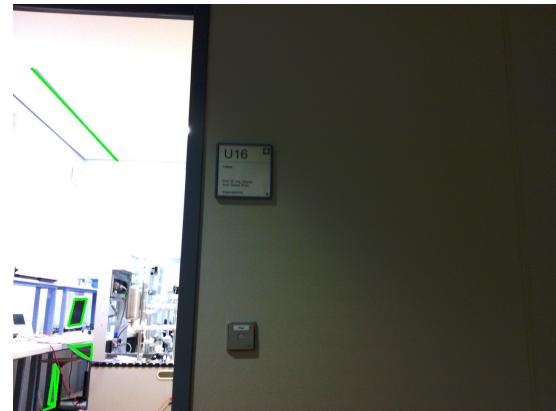
Eine Lösung dafür kann erreicht werden, indem zuerst das Licht eingeschaltet wird. Softwareseitig kann über die Implementierung eines Algorithmus für eine Überprüfung der Helligkeit im Bild nachgedacht werden.

Gegenlicht scheint aus einem Raum Ein weiteres Problem kann auftreten, sobald eine Raumtür geöffnet ist und der Messpunkt für die automatische Belichtungsanpassung (Fokus) im Inneren des Raumes liegt. Dadurch ist der „unwichtige“ Teil des Bildes komplett überbelichtet. Der „wichtigere“ Teil des Bildes - das Türschild ist hingegen sehr dunkel und kann nicht ordnungsgemäß verarbeitet werden. Ein Beispiel von diesem Phänomen ist in Abbildung 4.6 ersichtlich.

Eine Lösung für dieses Problem konnte bis jetzt nicht umgesetzt werden. Es empfiehlt sich daher, Raumtüren während des Programmablaufs geschlossen zu halten. In weiteren Arbeiten ist es ebenso möglich diesen Fehler abzufangen. Dies könnte erfolgen, indem ein Histogramm erstellt und dieses ausgewertet wird. Durch diese Information kann eine Helligkeitsanpassung durchgeführt werden. Die daraus entstehenden zusätzlichen Berechnungen - während der Programmalaufzeit des Programms, sollten dabei nicht vernachlässigt werden.



(a) Anwendung des Canny Algorithmus

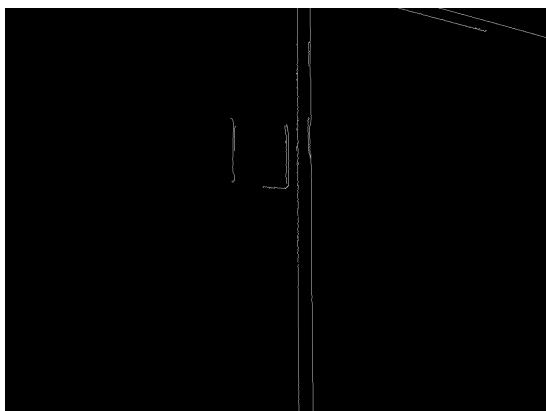


(b) erkannte Flächen in grün

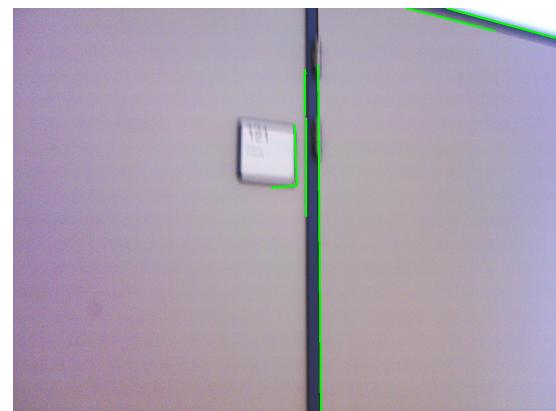
Abbildung 4.6: Bildfokus liegt im Raum

Das Bild verwackelt Ein anderes aufgetretenes Problem ist, dass Pepper Bilder verwackelt aufnimmt. Dieses Phänomen tritt besonders auf, wenn sich Gliedmaßen insbesondere der Kopf direkt vor der Aufnahme eines neuen Bildes bewegen. Da der Roboter sich auf jede durchgeführte Bewegung wieder in eine ruhige Ausgangslage versetzen muss.

Dieses Problem konnte teilweise gelöst werden, in dem Wartezeiten zwischen der Bewegung und Aufnahme des Bildes eingelegt wurden. Trotz dieser Lösung kann sich der Kopf noch minimal bewegen, da die Ablaufzeit des Programms trotzdem kurz gehalten werden sollte, um eine möglichst schnelle Lokalisierung zu ermöglichen. Ein falsch aufgenommenes Bild ist in Abbildung 4.7 zu sehen.



(a) Anwendung des Canny Algorithmus



(b) erkannte Flächen in grün

Abbildung 4.7: Aufnahme eines unscharfen Bildes

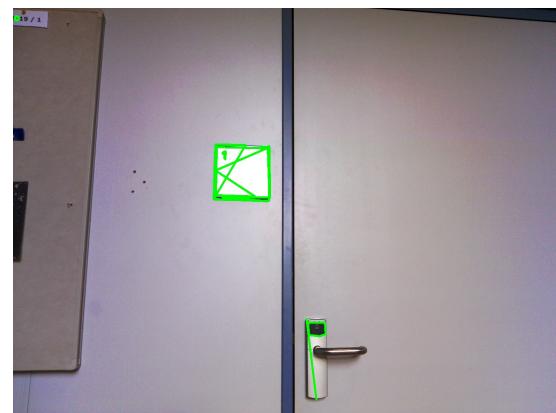
Überbeleuchtung in der Lobby Bei der Aufnahme von Bildern im Bereich der Lobby muss durch das einfallende Licht von außen damit gerechnet werden,

dass die Türschilder stark überbelichtet sind. Ein Beispiel für diesen Vorfall ist in Abbildung 4.8 zu sehen. Durch die Abdeckung der Türschilder mit Plexiglas ist eine Reflexion in diesen Bereichen besonders groß.

Eine Behebung des Problems auf softwareseitiger Basis ist nur sehr schwer möglich. Um dem zu umgehen, kann sich eine Person in die Strahlungsrichtung stellen und so verhindern, dass das Türschild zu hell wird. Alternativ kann nach anderen Strukturelementen gesucht werden.



(a) Anwendung des Canny Algorithmus



(b) erkannte Flächen in grün

Abbildung 4.8: Überbeleuchtetes Türschild in der Lobby

WLAN Verbindung verloren Da bis zum jetzigen Zeitpunkt alle Programme remote von einem Laptop durchgeführt wurden, kann es zu Programmabbrüchen kommen, sobald die WLAN Verbindung zu Pepper abbricht. Da alle Befehle vom Programmiergerät gesendet werden und Pepper keine Verarbeitung von Befehlen durchführt, kann dies besonders bei Bewegungen gefährlich werden. Wie bereits in Abschnitt 4.4 beschrieben, werden die Bewegungen gestartet und erst bei dem Erreichen eines bestimmten Punktes wieder abgeschaltet. Wenn nun nach dem Starten die Verbindung zum Roboter abbricht, bewegt sich dieser weiter. Da für eine bessere Bewegung die Sicherheitsabstände minimiert wurde, kann es zu Unfällen kommen, da die Abbremsung nicht mehr schnell genug erfolgen kann.

Dieses Problem kann gelöst werden, sobald das Programm lokal auf Pepper ausgeführt wird. Dieses Vorgehen sollte auch für eine zukünftige Anwendung durchgeführt werden.

5 Fazit

Nach der Vorstellung der praktisch umgesetzten Ergebnisse soll abschließend in dieser Arbeit ein Fazit gezogen werden. Dabei wird zuerst eine Zusammenfassung der Erfolge aufgezeigt. Danach folgend wird eine kritische Reflexion durchgeführt, bevor final ein Ausblick gegeben wird.

5.1 Zusammenfassung

In der vorgestellten Arbeit konnte der Roboter Pepper mittels Python programmiert werden, um eine Lokalisierung durchzuführen. Wenn sich Pepper in einem Gang der DHBW Horb befindet und eine gute Ausleuchtung vorhanden ist, dreht sich Pepper zunächst zu einer Wand. Danach bewegt sich der Roboter in einen idealen erprobten Abstand und bewegt den Kopf auf die Höhe der Türschilder. Darauffolgend wird ein Bild aufgenommen und mittels eines entwickelten Algorithmus ausgewertet. Dabei wird das Bild innerhalb eines bestimmten Bereichs auf die Anwesenheit einer Raumnummer untersucht. Sollte ein gültiger Raum gefunden werden, kann diese Information weitergegeben werden. Dadurch konnte eine initiale Lokalisierung durchgeführt werden.

5.2 Kritische Würdigung der Arbeit

Bereits in Abschnitt 4.8 wurden die bekannten Grenzen des erstellten Algorithmus vorgestellt. Weiterhin ist die Länge der Ausführungszeit von bis zu fünf Minuten optimierungsbedürftig. Ein weiteres Problem kann sich ergeben, sobald Menschen in der Nähe von Pepper sind. Diese werden bei den Bewegungen des Roboters ebenso wie starre Wände wahrgenommen und können die Messabstände zu diesen festen Elementen verändern. Dadurch kann die Lokalisierung fehl-schlagen, da die vorhandenen Türschilder nicht genau aufgenommen werden können.

Neben den dargestellten Grenzen und Schwachstellen, ist das Programm einsatzbereit und liefert aussagekräftige Ergebnisse.

5.3 Ausblick

Für eine zukünftige Verwendung und Verbesserung der Arbeit können einerseits die bereits bekannten Probleme und Grenzen (Abschnitt 4.8) behoben werden, andererseits ist auch die Implementierung neuer Funktionalitäten möglich. Als erstes sollte dabei auch die Lokalisierung der Lobby in den gesamten Algorithmus implementiert werden. Andererseits ist über die Verwendung eines Videostreams von Pepper nachzudenken, da dadurch eine kontinuierliche Auswertung des Bildmaterials möglich wäre.

Um den Roboter auf unterschiedlichen Stockwerken einsetzen zu können, muss dieser den Höhenunterschied mit einem Fahrstuhl überwinden. Für diese Anwendung könnte ebenso ein Algorithmus mit einer Bildverarbeitung geschrieben werden. Abschließend sollten die entwickelten Algorithmen auch lokal auf Pepper ausgeführt werden und nicht wie bis jetzt lediglich von einem Programmiergerät ausgeführt und berechnet werden.

Literatur

- [alda] aldebaran. *2D Cameras: Parameters*. Hrsg. von aldebaran. URL: http://doc.aldebaran.com/2-4/family/pepper_technical/video_pep.html (besucht am 12.11.2019).
- [aldb] aldebaran. *ALLandMarkDetection*. Hrsg. von aldebaran. URL: <http://doc.aldebaran.com/2-1/naoqi/vision/allandmarkdetection.html> (besucht am 09.12.2019).
- [aldc] aldebaran. *Pepper - Developer Guide: Technical overview: Dimensions*. Hrsg. von aldebaran. URL: http://doc.aldebaran.com/2-4/family/pepper_technical/dimensions_pep.html (besucht am 16.11.2019).
- [BB05] Wilhelm Burger und Mark James Burge. *Digitale Bildverarbeitung: Eine Einführung mit Java und ImageJ; mit 16 Tabellen*. eXamen.press. Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg, 2005. ISBN: 978-3-540-21465-6. DOI: [10.1007/3-540-27653-X](https://doi.org/10.1007/3-540-27653-X). URL: <http://dx.doi.org/10.1007/3-540-27653-X>.
- [BK11] Gary R. Bradski und Adrian Kaehler. *Learning OpenCV: Computer vision with the OpenCV library*. 1. ed., [Nachdr.] Software that sees. Beijing: O'Reilly, 2011. ISBN: 9780596516130.
- [Bol19] Jonas Boller. »Fortbewegung eines Pepper-Roboters innerhalb eines definierten Gebäudes: am Beispiel des Hauptgebäudes der DHBW Stuttgart Campus Horb«. Studienarbeit. Horb: DHBW, 2019.
- [Bri15] Mario Brinkhoff. »Elfmeterschießen mit dem Aldebaran NAO«. Studienarbeit. Horb: DHBW, 2015.
- [CAP19] Christian Clausner, Apostolos Antonacopoulos und Stefan Pletschacher. »Efficient and effective OCR engine training«. In: *International Journal on Document Analysis and Recognition (IJDAR)* (2019). ISSN: 1433-2825. DOI: [10.1007/s10032-019-00347-8](https://doi.org/10.1007/s10032-019-00347-8).

- [Cha13] Tom Chatfield. *50 Schlüsselideen Digitale Kultur*. Heidelberg: Springer Spektrum, 2013. ISBN: 978-3-8274-3063-2. DOI: 10.1007/978-3-8274-3064-9. URL: <http://dx.doi.org/10.1007/978-3-8274-3064-9>.
- [Dua] Duale Hochschule Baden-Württemberg Stuttgart Campus Horb. *Kurzvorstellung des Campus Horb*. Hrsg. von DHBW. URL: <https://www.dhbw-stuttgart.de/horb/themen/hochschule/> (besucht am 13.11.2019).
- [Erh08] Angelika Erhardt. *Einführung in die Digitale Bildverarbeitung: Grundlagen, Systeme und Anwendungen*. Wiesbaden: Vieweg+Teubner / GWV Fachverlage GmbH Wiesbaden, 2008. ISBN: 978-3-519-00478-3. DOI: 10.1007/978-3-8348-9518-9. URL: <http://dx.doi.org/10.1007/978-3-8348-9518-9>.
- [Goo] Google Maps. *DHBW Maps*. Hrsg. von Google Maps. URL: <https://www.google.de/maps/place/DHBW+Stuttgart+Campus+Horb/> (besucht am 13.11.2019).
- [Kle19] Felix Klemm. »Weiterentwicklung der Portier-App am Pepper«. Studienarbeit. Horb: DHBW, 2019.
- [Lab18] Yvette Labastille. »Pepper – Inbetriebnahme und Entwicklung einer Portier-App«. Studienarbeit. Horb: DHBW, 2018.
- [Mol17] Benedikt Molitor. »Menschliche Schussbewegung des NAO Roboters in C/C++«. Studienarbeit. Horb: DHBW, 2017.
- [Neu05] Burkhard Neumann. *Bildverarbeitung für Einsteiger: Programmbeispiele mit Mathcad ; mit 45 Tabellen*. Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg, 2005. ISBN: 9783540218883. DOI: 10.1007/b138063. URL: <http://dx.doi.org/10.1007/b138063>.
- [Ros] Adrian Rosebrock. *Practical Python and OpenCV*. 4. Aufl. URL: <https://www.pyimagesearch.com/category/building-a-pokedex/> (besucht am 05.11.2019).
- [Tec] Technik LPE. *Pepper*. Hrsg. von Technik LPE. URL: <https://techniklpeshop.de/pepper/123-pepper.html>.

- [Vur15] Fatih Vural. »Bildverarbeitung und Objekterkennung anhand von NAO (Roboter)«. Studienarbeit. Horb: DHBW, 2015.

Anhang

Inhalt der CD

- Studienarbeit - T3100 L^AT_EX Dateien inklusive Bilder
- Studienarbeit - T3100 PDF
- Erstellter Quellcode
- Erstellte Bilder während der Testphasen

CD