DHBW
Duale Hochschule
Baden-Württemberg

Studienarbeit - T3200

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:	09.03.2020 bis 22.06.2020
Betreuer:	Dipl. Ing. (FH/BA) 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 - T3200 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, 17. Juni 2020

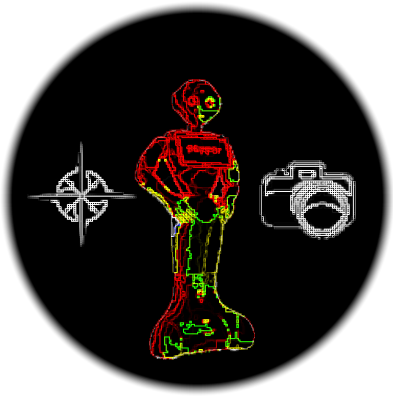
Tim Schneider

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) und das autonome Fahrstuhlfahren behandelt. Die Aspekte der Kartenerstellung, der Erfassung eines Zielraumes und der Bewegung des Roboters wird in [Bol20] vorgestellt.

Die Arbeit stellt zuerst die benötigte Theorie in Bezug auf Möglichkeiten des Fahrstuhlfahrens für Roboter dar. Im weiteren Verlauf wird ein ausführlicher Blick auf die durchgeführte Programmierung gelegt. Dabei wurden manche Funktionalitäten bereits umgesetzt und müssen zukünftig noch getestet werden. Andere Themen werden lediglich ausführlich beschrieben.

Es wurden neben der neuen Software auch Anpassungen und Verbesserungen im Vergleich zu der Vorgängerarbeit [Sch19] durchgeführt. Ein weiterer Aspekt ist die Beschreibung von Testalgorithmen, welche eine Arbeit ohne Roboter zulassen.



Logo des ProjectPepper-Entwicklerteams

Abstract

In this thesis the humanoid communication robot Pepper is 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 split into two thesis. Within this thesis the localization of the robot with the help of an image processing BV and the autonomous elevator driving is treated. The aspects of mapping, the detection of a target area and the movement of the robot are presented in [Bol20].

This thesis firstly presents the required theory regarding possibilities of navigantion of robots in elevators. Furthermore a detailed look at the programming implemented is given. Thereby some functionalities have already beeing implemented and have to be tested in the future. Other topics are only described in detail and not programmed right now.

In addition to the new software, adaptations and improvements were made compared to the previous work [Sch19]. Another aspect is the description of test algorithms which allow for working without a robot.

Inhaltsverzeichnis

Abbildungsverzeichnis

Quellcodeverzeichnis

Abkürzungsverzeichnis

BV	Bildverarbeitung
DHBW	Duale Hochschule Baden-Würtemberg
DH	Duale Hochschule
HSV	Farbraum, in dem hue (Farbwert), saturation (Farbsättigung) und value (Dunkelstufe) eine Farbe definieren
OCR	Optical character recognition
SBBZ	Sonderpädagogisches Bildungs- und Beratungszentrum

1 Einleitung

Als Einleitung in diese Studienarbeit wird die Aufgabenstellung und der strukturelle Aufbau der Arbeit vorgestellt.

1.1 Aufgaben- und Problemstellung

Um Besucher an der Duale Hochschule Baden-Würtemberg einen Weg weisen zu können, soll der Kommunikationsroboter Pepper eingesetzt werden. Dabei ist das Ziel, dass der Roboter aus einem Gespräch den gewünschten Raum erkennen und den Gast im Anschluss dorthin begleiten oder den Weg verbal beschreiben kann.

Innerhalb dieser Arbeit wird dabei auf das eigenständige Fahren des Roboters mit einem Fahrstuhl eingegangen. Dabei werden unterschiedliche Punkte, wie das Rufen des Fahrstuhls, dem Anwählen der richtigen Etage und die notwendigen Bewegungen thematisiert. Weiterhin werden noch offene Punkte aus der vorangegangenen Studienarbeit [Sch19] und die Erfahrungen aus einem Feldversuch an einer Schule behandelt.

Weitere Aspekte, wie der Übergang zwischen unterschiedlichen Karten und weitere Benutzungshinweise in Bezug auf Pepper, wurden parallel zu dieser Arbeit von Jonas Boller [Bol20] entwickelt.

1.2 Struktur der Arbeit

Im ersten Teil der Arbeit werden die theoretischen Grundlagen für das Verständnis gelegt. Dabei werden unterschiedliche Möglichkeiten für das autonome Fahrstuhlfahren von Robotern vorgestellt.

Im nächsten Abschnitt wird auf den aktuellen Stand der Studienarbeit zu Beginn des Semesters eingegangen. Einerseits werden hier die letzten Studienarbeiten kurz beschrieben, um einen besseren Einstieg in diese Arbeit zu haben. Danach werden noch die Erfahrungen aus dem Besuch des Sonderpädagogischen

Bildungs- und Beratungszentrums (SBBZ) Sehen gegeben. In diesem Bereich werden vorrangig unterschiedliche Störungen Peppers beschrieben. Eine umfassendere Darstellung zu der Feldstudie ist in [Bol20] zu finden.

Darauffolgend werden drei praktische Themen behandelt. Zuerst wird anhand von Test- und Simulationssoftware die Suche nach Raumnummernschildern verbessert und ein erster Ansatz für ein Auffinden des Fahrstuhlrufknopfes behandelt. Im weiteren Verlauf wird die Lokalisierung Peppers grundsätzlich verändert und neu beschrieben. Diese Aspekte konnten dabei lediglich theoretisch ausgearbeitet werden, da die DHBW aufgrund der Corona-Pandemie geschlossen war. Im letzten Abschnitt zu der praktischen Ausarbeitung gehört die ausführliche Beschreibung, wie Pepper Fahrstuhl fahren kann. Hier werden die groben Konzepte und Programmbeispiele gegeben.

Abgeschlossen wird diese Studienarbeit mit einem Fazit, einer kritischen Bewertung und den möglichen Erweiterungen zukünftige Studienarbeiten.

2 Theoretische Grundlagen

In diesem ersten Abschnitt sollen die theoretischen Grundlagen für das weitere Verständnis der Arbeit aufgezeigt werden. Hierbei wird analysiert, inwiefern Roboter eigenständig Fahrstuhl fahren können.

Um mit einem Roboter autonom Fahrstuhl zu fahren, gibt es zwei Möglichkeiten. Auf der einen Seite kann eine bidirektionale Schnittstelle zur Kommunikation zwischen Roboter und Aufzug eingesetzt werden. Andererseits ist es, mittels einer BV und einer Betätigungsvorrichtung die an einem Fahrstuhl vorhandenen Knöpfe drückt, möglich .

Wie in [Rot20] beschrieben, arbeiten aktuell mehrere Firmen (u.a. Thyssenkrupp Elevators) an der Entwicklung von Schnittstellen zwischen Aufzügen und Robotern. Mittels dieser können damit neue Aufgabengebiete erschlossen werden. Als Beispiele können hier Putzroboter oder auch Serviceroboter in Firmen- oder Hotelgebäuden genannt werden. Da der Fahrstuhl an der DHBW keine solche Schnittstelle besitzt, kann diese einfache Variante leider nicht genutzt werden.

Als Alternative steht die Verwendung einer BV zur Debatte. Dabei werden mittels einer Kamera am Roboter die Knöpfe am Fahrstuhl detektiert. Hierbei können Mustererkennung nach [Yin08] Verwendung finden. Der richtige Knopf muss danach mittels eines Aktors betätigt werden. Das Verhalten ähnelt dabei dem Vorgehen des Menschen im gesamten Ablauf. Im Rahmen einer Bachelorarbeit an der Fachhochschule Südwestfalen in Iserlohn (vgl. [Ven16]) wurde dieser Ansatz mit einem mobilen Industrieroboter bereits umgesetzt. Eine theoretische Beschreibung der Aufgabenstellung kann in dem Paper [Kan+10] nachgelesen werden. In Anlehnung daran wird in Kapitel 6 ein Vorschlag für eine Umsetzung bei Pepper vorgestellt.

3 Aktueller Stand

Nach der Vorstellung der theoretischen Themen werden im Folgenden einerseits die Ergebnisse der vorangegangenen Studienarbeiten zu dieser Thematik aufgezeigt, um einen besseren Einstieg zu erhalten. Andererseits werden die Erkenntnisse aus einem Besuch mit Pepper an einer Schule für unteranderem seheingeschränkte Kinder beschrieben.

3.1 Zusammenfassung vorangegangener Studienarbeiten

In den vorangegangenen zwei Studienarbeiten [Sch19] und [Bol19] wurden bereits erste Vorgehensweisen für dieses Thema vorgestellt. Um einen schnelleren Einstieg in die aktuellen Bereiche zu geben, werden diese Arbeiten hier noch kurz dargestellt.

Jonas Boller hat sich in seiner Arbeit [Bol19] mit der autonomen Bewegung Peppers zwischen zwei Punkten innerhalb der Duale Hochschule (DH) beschäftigt. Dabei wurden unterschiedliche Karten aufgenommen und die betreffenden Räume eingetragen. Zum Ende war damit ein automatisches Anfahren der Punkte innerhalb einer Karte möglich. Außerdem wurde die Kommunikation mit Menschen einprogrammiert. Somit kann Pepper aus einem Gespräch einen Zielraum extrahieren und diese Information an die benötigten Programme weitergeben. Weiterhin wurden unterschiedliche Gesprächssequenzen für eine natürliche Unterhaltung integriert.

In der Arbeit [Sch19] wurde dem gegenüber anhand einer BV die Startposition ermittelt. Dafür muss Pepper in einem Gang der DHBW in Horb aufgestellt werden und danach wird im erfolgreichen Fall eine Raumnummer in Sichtweite mittels einer Optical character recognition (OCR) erkannt. Mithilfe dieser Information kann eine nachfolgende Bewegung zu einem Wunschraum erfolgen. Eine genaue Beschreibung des Algorithmus und aktuelle Anpassungen werden in Kapitel 5 vorgestellt.

3.2 Erkenntnisse aus dem Besuch der Schule Heiligenbronn

Um die ersten Programme mit der späteren Zielgruppe zu testen, wurde ein Besuch an der SBBZ in Heiligenbronn im März organisiert. An der Schule werden, unter anderem seheingeschränkte und höreingeschränkte Kinder der Jahrgangsstufen eins bis neun, in mehreren Klassen betreut. Bei unterschiedlichen Aufgabenstellungen wurde versucht, wie eine Interaktion erfolgen kann und wie Pepper und die Kinder darauf reagieren. Dabei haben sich mehrere Punkte kristallisiert, welche im Folgenden vorgestellt werden. Bei manchen Problemen können gleich Lösungsvorschläge mit angegeben werden, welche teilweise bereits erprobt wurden.

Zu Beginn des Besuchs waren die meisten Probanden sehr begeistert von Pepper, dabei haben sich alle Kinder getraut in eine Interaktion mit Pepper zu treten. Besonders auffällig war dabei, dass sich „Technikbegeisterte“ eher zurückgehalten haben. Weiterhin wurde das Interesse mit einer stärkeren Einschränkung im Sehbereich größer. Ein blindes Kind hatte demnach das meiste Interesse an Pepper.

> Kind: Hallo Pepper, wie heißt du?
>
> Pepper: Hallo Mensch, ich bin Pepper.

Nach dem die ersten Berührungsängste abgebaut waren und sich alle gegenseitig vorgestellt hatten (siehe oben), wurden die ersten Gehversuche unternommen. Bei diesem Versuch hat sich ein blindes Kind an Peppers Schultern festgehalten und konnte somit einen bestimmten Weg abfahren. Bei den Bewegungen sollte hierbei darauf geachtet werden, dass die Hände nicht im Nackenbereich eingeklemmt werden können. Weiterhin ist darauf zu achten, dass Peppers Sicherheitsabstand nach hinten klein genug eingestellt ist, da sonst ein Stillstand zu erwarten ist. Insgesamt kann hier von einem Erfolg gesprochen werden, das Kind konnte von Pepper geführt werden, solange keine zu großen Drehungen mit der Bewegung verbunden waren. Softwareseitig sollte hierbei dringend geachtet werden, dass der Kopf sich nicht bewegt und die hinteren Sicherheitssensoren ignoriert werden.

Bei den weiteren Versuchen sind regelmäßig Probleme aufgetreten, welche unterschiedliche Ursachen haben. Am häufigsten hat Pepper keine Reaktion mehr gezeigt, wenn sich mehr als fünf Kinder im Sichtbereich befanden. Pepper hat

den Kopf dann meist zwischen den Kindern hin und herbewegt und niemanden mehr fokussiert. Ein ähnliches Phänomen ist aufgetreten, wenn mehrere Probanden gleichzeitig in eine Konversation einsteigen wollten. Um eine Verständigung sicher ausführen zu können, sollten sich demnach nicht mehr als zwei Personen vor Pepper befinden und lediglich einer sollte davon die Kommunikation übernehmen. Im schlechtesten Fall kann diese Überreizung zu einem Anhalten des Programms führen. Um aus diesem Zustand wieder herauszukommen ist ein harter Neustart (Drücken des Brustknopfes für mehr als 10 Sekunden) notwendig. Dabei kann es passieren, dass nicht alle Funktionalitäten richtig geladen werden, weshalb Pepper noch einmal ordnungsgemäß neugestartet werden sollte. Dieses Vorgehen ist ebenso empfehlenswert bei plötzlich auftauchenden Problemen mit sämtlichen Rädern oder Sensoren.

4 Erstellung von Testsoftware

Mit der Fertigstellung der vorangegangenen T3100 wurde ein Algorithmus zur Erkennung von Raumnummern für die DHBW entwickelt. Um die Modifikationen am Programmcode vergleichen zu können, wurde eine Auswertungssoftware vergleichbar mit einer Software in the Loop erstellt. Mit dieser werden jeweils alle bereits aufgenommen Bilder auf bestimmte Merkmale untersucht. Dabei werden zwei Parameter aufgenommen, einerseits die Empfindlichkeit und andererseits die Ausführungszeit. Damit lässt sich final eine objektive Bewertung im Vergleich zwischen den Programmierständen abgeben. Ein leicht abgewandelter Algorithmus wurde später für die Detektion des Fahrstuhlknopfes verwendet. Im Folgenden wird zunächst die Grundstruktur und später die beiden unterschiedlichen Programmteile vorgestellt.

4.1 Vorstellung der Grundstruktur

Die Grundstruktur der Analysesoftware lässt sich in mehrere Bereiche unterteilen, welche im Folgenden vorgestellt werden.

Hilfsordner leeren Um die Ergebnisse besser nachvollziehen zu können, werden unterschiedliche Ordner mit Aufnahmen aus der DH gefüllt. Diese werden während der Ausführung der Testsoftware nach erfolgreicher und missglückter Auswertung sortiert. Weiterhin werden teilweise temporäre Daten, wie angewandte Filter, ebenso mit abgespeichert. Um jeweils lediglich die aktuellsten Bilder zu haben, werden zu Beginn der Auswertung alle betreffenden Ordner geleert, siehe Codeausschnitt 4.1.

```
deletingFolders = ['.\mis','.\mnewcropped','.\mnewedges','.\msuc','.\mlobby
    '] #Hilfsordner für missglückte, abgeschnittene, verschärfte Kanten,
    erfolgreiche und Lobbybilder

for folders in deletingFolders: #In allen ausgewählten Ordnern
    for the_file in os.listdir(folders):    #über alle Dateien
        file_path = os.path.join(folders, the_file)
        try:
            if os.path.isfile(file_path):
                os.unlink(file_path)    #Bild löschen
        except Exception as e:
            print(e)
```

Codeausschnitt 4.1: Leeren der Hilfsordner

Initialisierung Im nächsten Schritt kann mit der Initialisierung begonnen werden, dafür wird der Erfolgszäher auf null festgesetzt und alle zu untersuchenden Bilder werden in einer Liste abgespeichert. Des Weiteren wird eine Textdatei geöffnet, um eine Loggingdatei zu erstellen. In dieser werden alle erzielten Ergebnisse (Welche Bilder waren erfolgreich? Welche Bilder waren nicht erfolgreich? Welche Schritte der BV wurden bei welchem Bild ausgeführt?) fortlaufend abgespeichert. Diese Daten werden jeweils zu Beginn eines Durchlaufs überschrieben, so dass lediglich die aktuellsten Daten vorliegend sind. Bevor ein neuer Testdurchlauf durchgeführt wird, sollte bei Interesse ein Blick in die Logdatei stattfinden und dieses bei Bedarf gesichert werden. Die beschriebenen Vorgänge sind in Codeausschnitt 4.2 ersichtlich.

```
from PepperOCR import * #für die BV notwendig
font = cv2.FONT_HERSHEY_SIMPLEX #Ergebnis in das Bild schreiben

success = 0 #Anzahl der erfolgreich erkannten Bilder
log = open('log.txt', 'w') #Log starten und den alten überschreiben
images = os.listdir('.\img')    #alle Bilder einlesen und in Liste
    speichern
pictures = len(images)  #Anzahl der Bilder zwischenspeichern
print 'Bearbeitung laeuft...'
start_time = time.time()    #Aktuelle Zeit als Startzeit festlegen
```

Codeausschnitt 4.2: Initialisierung der Analyse

Grundstruktur Nach der erfolgreichen Initialisierung der Software kann mit der eigentlichen Problembehandlung begonnen werden, dafür wird über alle gespeicherten Bilder iteriert. Dieses Vorgehen ist in Codeausschnitt 4.3 zu sehen. Die folgende Ermittlung der Raumnummer und die Überprüfung auf einen Fahrstuhl findet in den folgenden beiden Unterkapiteln (Abschnitt 4.2 und Abschnitt 4.3) statt. Darüber hinaus wird in dem Codeschnipsel die weitere grundlegende Struktur aufgezeigt. Die genannten Funktionalitäten werden im weiteren Verlauf dieses Kapitels noch näher beschrieben.

```
for pic in range(pictures):
    path = os.path.join('.\img',images[pic])
    img = cv2.imread(path)
    try: #Durchführung der Analyse wie später beschrieben
    except: #Standardwerte bei Fehler setzen
    #Log schreiben
    if valid:
        success +=1 #Erfolgszähler bei Erfolg um eins erhöhen
        #Text mit weiteren Informationen auf das Bild schreiben
    else:
        #Text mit weiteren Informationen auf das Bild schreiben
#Bild abspeichern
end_time = time.time()  #aktuelle Zeit als Endzeit speichern
```

Codeausschnitt 4.3: Grundstruktur der Analyse

Bilder speichern Für eine bessere Verständlichkeit darüber, wie viele Bilder erfolgreich überprüft wurden und an welchen Parametern eventuell Änderungen notwendig sind, werden neben der Anzahl der Treffer noch die bearbeiteten Bilder abgespeichert. Dies geschieht in Codeausschnitt 4.4. In den ersten beiden Zeilen des Programmausschnitts wird dabei zuerst die gefundene Raumnummer auf das Bild geschrieben und darauffolgend wird es unter einem sprechenden Namen, bestehend aus „suc“ und dem ursprünglichen Dateinamen abgespeichert. Der Dateipfad wird aus diesem String und dem Ordnerstring zusammengebaut.

```
string = str(room) #Bildbeschriftung anlegen
cv2.putText(img, string, (10,450), font, 3, (0, 255, 0), 2, cv2LINE_AA)
    #Text auf das Bild schreiben
string = 'ratio'+images[pic]    #Dateipfad anlegen
path = os.path.join('.\msuc',string)    #vollständigen Dateipfad anlegen
cv2.imwrite(path,img)   #Bild speichern
```

Codeausschnitt 4.4: Abspeichern der Hilfsbilder

4.2 Erweiterung für Raumnummernsuche

Um in allen Bildern nach Raumnummern zu suchen, wird die eben beschriebene Grundstruktur verwendet und in Codeausschnitt 4.3 wird im try-case Block die Funktion *DetectRoom(img)* aus der PepperOCR Bibliothek aufgerufen. Weitere Veränderungen betreffen die abzuspeichernden Bilder. Hierbei wird einerseits eine Kopie des Originals in den entsprechenden Ordnern für erfolgreiche und missglückte Bilder abgelegt. Andererseits werden alle Bilder der Zwischenschritte aus der Funktion ebenso in den entsprechenden Ordnern abgespeichert, dies sieht ausschnittsweise wie in Codeausschnitt 4.5 aus. Mithilfe dieser Bilder lassen sich Fehlerquellen und falsche Parameter leicht bestimmen. Ein Beispiel für eine Anwendung dieser Analysesoftware und eine entsprechende Änderung der Funktion findet in Kapitel 5 statt.

```
1 string = 'edges'+images[pic]
2 path = os.path.join('.\edges',string)
3 cv2.imwrite(path,Kanten)
4 string = 'warp'+images[pic]
5 path = os.path.join('.\warp',string)
6 cv2.imwrite(path,warp)
```

Codeausschnitt 4.5: Abspeichern der Zwischenbilder

4.3 Erweiterung für Aufzugssuche

Für die Aufzugsuche kann das Grundgerüst vollständig übernommen werden. Es muss lediglich in Codeausschnitt 4.3 im try-case Block die entsprechende Funktion *isElevator(img)* aufgerufen werden. Die Funktionsweise dieser Funktion wird im späteren Verlauf in Abschnitt 6.1 ausführlich vorgestellt. Da dabei keine zusätzliche Bilder erzeugt werden, genügt das sortierte Speichern der Kopien in den entsprechenden Ordnern für Bilder mit und ohne Fahrstuhl. Inwiefern diese Daten für die Erstellung der zukünftigen Software sinnvoll ist, zeigt sich an den weiteren Erklärungen in Kapitel 6.

5 Lokalisierung von Pepper - Verbesserungen

Nach der Vorstellung der Testsoftware befasst sich das nächste Kapitel mit der konkreten Lokalisierung von Pepper in der DHBW in Horb. Die Grundlagen für den Softwarecode wurden in [Sch19] erklärt. Diese sind in großen Teilen immer noch gültig. An manchen Stellen wurden jedoch Veränderungen vorgenommen, welche im Folgenden erklärt werden. Dabei wurden sowohl die Bewegungsabläufe, als auch die BV und entsprechenden Auswertung der Raumnummern aus den Bilddaten verbessert. Des Weiteren wurde ein Algorithmus implementiert, mit dem es möglich ist, in einer Lobby zu starten und eigenständig einen Flur mit Türschildern aufzusuchen. Dieser Teil konnte jedoch nur in ausgewählten Sequenzen getestet werden und sollte vor dem Einsatz unbedingt überprüft werden.

5.1 Bewegungsabläufe

Für eine erfolgreiche Lokalisierung von Pepper an der DHBW in Horb werden unterschiedliche Bewegungen durchgeführt. Zuerst wird überprüft, ob sich der Roboter in einem Flur oder in der Lobby befindet. Sollte der zweite Fall vorliegen, fährt Pepper in einen Gang. Nach einer Ausrichtung zu einer Wand werden nacheinander unterschiedliche Positionen für eine Bildaufnahme angefahren. Die Bewegungen wurden in zwei Funktionen untergliedert. Dabei wurde *FindFloor()* neu entwickelt und die vorhandene Funktion *Movement()* wurde abgewandelt. Eine Vorstellung dieser beiden erfolgt in den nächsten Abschnitten.

5.1.1 Vorstellung der Funktion FindFloor

In *FindFloor()* wird überprüft ob sich Pepper in einem Gang befindet. Wenn dies nicht der Fall ist, wird der Nächste angefahren. Dieses Vorgehen wurde ausgewählt, da sich in der Lobby lediglich eine geringe Anzahl an Türschildern befinden. Die Information, dass sich Pepper in der Lobby befindet, ist ebenso nicht

ausreichend, da der in [Bol20] vorgestellte Algorithmus eine exakte Startposition benötigt. Der Aufbau gliedert sich dabei in mehrere Teilbereiche, welche im Folgenden vorgestellt werden sollen.

Initialiserungen Um die Kommunikation zwischen dem auszuführendem Pythonskript und Pepper sicherzustellen, werden in einem ersten Schritt die benötigten Dienste angebunden. Eine genauere Beschreibung zu dem Vorgehen wurde in [Sch19] dargestellt. Die benötigten Befehle sind in Codeausschnitt 5.1 abgebildet. Des Weiteren werden die lokalen Variablen *insideLobby* und *floorCounter* initialisiert. Die Erste wird benötigt, um einen Abbruch der Funktion zu erreichen, sobald sich Pepper in einem Flur befindet. Die zweite Variable wird genutzt, um zu zählen, wie oft Pepper bereits versucht hat, einen Flur zu erreichen. Nach einer bestimmten - noch festzulegenden - Anzahl an Versuchen beendet Pepper die Suche nach einem Gang erfolglos und sollte umgesetzt werden. Ein neuer Durchlauf ist wiederum auch möglich, da Pepper ja bereits eine andere Startsituation aufweist.

```
1    autonomous_service = session.service("ALAutonomousLife")
2    speech_service = session.service("ALTextToSpeech")
3    motion_service = session.service("ALMotion")
4    memory_session = session.service("ALMemory")
5    tts = session.service("ALTextToSpeech")
6    insideLobby = True
7    floorCounter = 0
```

Codeausschnitt 5.1: Initialisierung FindFloor

Hauptschleife und lokale Variablen Sobald die Funktion gestartet wird und die Initialisierung der Dienste abgeschlossen ist, kann der Hauptteil beginnen. Diese Schleife wird solange ausgeführt, wie sich Pepper in der Lobby oder an einem unbekannten Ort befindet. Weiterhin werden die lokalen temporären Variablen angelegt. Um die Entfernung nach hinten und vorne abzuspeichern, wird ein leeres Array *sonars* angelegt. Außerdem wird die minimale und maximale Entfernung, mit der Information des jeweiligen Indexes, abgespeichert. In einer Variablen kann bestimmt werden mit wie vielen Schritten sich Pepper für eine Datenaufnahme drehen soll. Aus dieser Angabe kann der benötigte Drehwinkel berechnet werden. Zu beachten ist, dass experimentell ein Vollkreis mit 6,4 Radiant bestimmt wurde. Dies entspricht wiederum $\pi = 3,2$. Dieser Wert wurde von

dem Hersteller Aldebaran fest hinterlegt. Der ungewöhnliche Wert hat Auswirkungen auf alle Drehungen und sollte während der Entwicklung beachtet werden. Der abgebildete Programmcode (Codeausschnitt 5.2) zeigt das beschriebene Vorgehen.

```
    while insideLobby and floorCounter < 10: #solange sich Pepper in einer
        Lobby befindet
        floorCounter += 1   #Anzahl Fluchtversuche
        takingSonars = 0
        minSonar = 1000; minSonarIndex = 100
        maxSonar = 0; maxSonarIndex = 100
        turnResolution = 8  #Anzahl Schritte pro Messzyklus für Halbkreis
        turnRadiant = 3.2/turnResolution    #PI ist 3,2!!
        sonars = [] #Array um gemessene Werte zu speichern
```

Codeausschnitt 5.2: Variablen für FindFloor

Halbkreisdrehung mit Bestimmung der Extremwerte Als ersten eigentlichen Schritt in der Funktion *FindFloor* dreht sich Pepper in einem Halbkreis. Dabei werden n-mal, je nach eingestellter Variable, die Entfernung nach vorne und hinten gemessen. Die Summe aus den beiden Messwerten wird in einem Array abgespeichert. Während der Laufzeit wird weiterhin die minimale und maximale Entfernung bestimmt. Diese Information wird für eine spätere Auswertung benötigt. Der Ausschnitt ist in Codeausschnitt 5.3 zu finden. Die Abarbeitung erfolgt dabei in einer Schleife, bis der Halbkreis abgefahren wurde.

```
    while takingSonars < turnResolution:    #drehe im Halbkreis (mit
        turnResolution Schritten) und nehme Sonarwerte auf
            #Abstände messen
            frontDistance = memory_session.getData("Device/SubDeviceList/
                Platform/Front/Sonar/Sensor/Value")
            backDistance =memory_session.getData("Device/SubDeviceList/
                Platform/Back/Sonar/Sensor/Value")
            distanceSonar = frontDistance + backDistance #Gesamtabstand
            printsay("Aufnahme "+ str(takingSonars), onlyPrint = True)
            sonars.append(distanceSonar) #Abstand zum Array hinzufügen
            if distanceSonar < minSonar:minSonar = distanceSonar;
                minSonarIndex = takingSonars
            if distanceSonar > maxSonar:maxSonar = distanceSonar;
                maxSonarIndex = takingSonars
            motion_service.moveTo(0.0,0.0,turnRadiant) #nächster Punkt
            takingSonars += 1
```

Codeausschnitt 5.3: Halbkreisdrehung mit Bestimmung der Extremwerte

Bestimmung des Ortes Der nächste und letzte Abschnitt dieser Funktion überprüft, ob sich der Roboter in einer Lobby oder in einem Flur befindet. Dieser Absatz ist sehr kritisch zu betrachten, da keine ausreichend große Datenlage oder ausreichend viele Testläufe vorliegen! Anhand der vorhandenen Daten sollte sich der Roboter in einem Gang befinden, wenn eine durchschnittliche Entfernung kleiner als 4 Meter und die minimale Entfernung kleiner als zwei Meter beträgt. Sollte dies nicht der Fall sein und sich Pepper in einer Lobby befinden, dreht sich der Roboter zu dem Punkt mit der maximalen Entfernung. Darauffolgend wird die größere Entfernung vorwärts angefahren.

Für das übergeordnete Programm wird eine Null oder eine Eins, jeweils für ein erfolgreiches oder missglücktes Auffinden der Lobby, gesendet. Fehlercodes werden in diesem Entwicklungsstadium nicht eingesetzt, da bis jetzt keine Fehlerfälle bekannt sind. Der beschriebene Codeausschnitt ist in 5.4 aufgezeigt.

```
1    if sum(sonars)/len(sonars) < 4.0 and min(sonars) < 2.0:
2            printsay("Aufnahme beendet, bin in einem Gang, drehe mich zum
                 Minimalpunkt"+str(minSonarIndex))
3            motion_service.moveTo(0.0,0.0,(minSonarIndex*turnRadiant))  #
                 zur Position mit minimalen Abstand drehen
4            insideLobby = False
5            return 0
6        else:
7            printsay("Aufnahme beendet, bin in einer Lobby, drehe mich zum
                 Maximalpunkt"+str(maxSonarIndex))
8            motion_service.moveTo(0.0,0.0,(maxSonarIndex*turnRadiant)) #an
                 Position mit maximaler Entfernung drehen
9            frontDistance = memory_session.getData("Device/SubDeviceList/
                 Platform/Front/Sonar/Sensor/Value")
10           backDistance =memory_session.getData("Device/SubDeviceList/
                 Platform/Back/Sonar/Sensor/Value")
11           if backDistance > frontDistance: #Bei Bedarf noch um 180 Grad
                 drehen
12               motion_service.moveTo(0.0,0.0,3.2)
13               frontDistance = backDistance
14           printsay("Bewege mich um"+str(round(frontDistance,2))+"nach
                 vorne")
15           motion_service.moveTo(frontDistance,0.0,0.0) #Bewegung in einen
                  Gang(weiteste gemessene Entfernung)
16   return 1    #keinen Gang gefunden
```

Codeausschnitt 5.4: Bestimmung des Ortes von Pepper

5.1.2 Veränderungen der Movement Funktion

Die Funktion *Movement* wurde bereits in [Sch19] vorgestellt und wurde im Rahmen dieser Arbeit an die zuvor beschriebenen Änderungen angepasst. Im Folgenden werden die Unterschiede zwischen den beiden Versionen aufgezeigt. Auf unveränderte Abschnitte wird nicht weiter eingegangen. Die Einteilung erfolgt dabei nach der vorherigen Arbeit.

Drehung zur Wand Die Funktion beginnt mit einer Drehung zu einer Wand, um ein Bild mit einem Türschild aufzunehmen. Dieser Teil wurde vollständig verändert. Da diese Funktion nun nach der *FindFloor* Funktion aufgerufen wird, ist Pepper bereits zu einer minimalen Entfernung zu einer Wand ausgerichtet. Dadurch kann nun eine definierte Drehung für die weiteren Aufnahmen verwendet werden. Die nun vorgestellte Lösung konnte noch nicht an der DHBW getestet werden. Eine globales Array *rotatings* stellt die benötigten Drehwinkel zur Verfügung. Insgesamt dreht sich Pepper um $\frac{19}{10}\pi$. Dabei ist wieder zu beachten, dass $\pi = 3,2$ gilt. Die Aufnahmen verteilen sich demnach nach dem Muster welches in Codeausschnitt 5.5 im Programmkommentar vorgegeben ist.

```
1    rotatings = [0.0, 0.32, 2.56, 0.32, 0.32, 2.56] #Drehwerte für die
         Bewegung, wenn Pepper mit einer minimalen Entfernung zu einer Wand
         steht, werden diese sechs Werte nacheinander angefahren. Insgesamt
         wird eine knappe Drehung durchgeführt. Berechnungsgrundlage ist,
         dass Pi 3,2 ist (experimentelle Bestimmung...). Konstrukt konnte
         nicht an der DH getestet werden!
2    rotatingCounter = 0
3    # Bis zum nächsten Bildpunkt drehen, laut Def von rotatings
4    #Bildpunkte sind:
5
6    #          GANG
7    #
8    # W    x6       5x    W
9    # A   x1          4x   A
10   # N    x2       3x    N
11   # D                   D
12   #          GANG
13
14   motion_service.moveTo(0.0,0.0,rotatings[rotatingCounter])
15   rotatingCounter += 1
16   printsay("Naechster Punkt für eine Bildaufnahme erreicht")
```

Codeausschnitt 5.5: Veränderung der Drehung

Entfernung zur Wand anfahren In diesem Bereich wurden keine Veränderungen vorgenommen. Eine Beschreibung ist in [Sch19] zu finden.

Steuersignale Wie bereits zuvor beschrieben, dreht sich Pepper nicht zu einer Wand mit einer minimalen Entfernung. Demnach kann das Steuersignal 5 entfallen. Sonst wurden keine Änderungen an den Übergabeparametern vorgenommen.

5.2 Erkennung von Türschildern

Neben den Bewegungsabläufen muss für die erfolgreiche Lokalisierung Peppers in dem Gebäude der DHBW in Horb die Erkennung der Türschilder und eine entsprechende Extraktion der Raumnummer erfolgen. In [Sch19] wurde dafür bereits ein Algorithmus vorgestellt. Dieser hat sich bis auf einen Teil als sinnvoll etabliert. Im Folgenden werden die Änderungen an der *DetectRoom* Funktion vorgestellt. Mit Hilfe dieser konnten zeitliche Einsparungen bei der Testausführung generiert werden. Eine Auswirkung im Einsatz bei Pepper konnte bislang nicht erprobt werden.

DetectRoom Die *DetectRoom* Funktion wurde um eine selektive Auswahl in Anlehnung an [Ros] ergänzt. Mit dieser ist es möglich lediglich geometrisch passende Formen auf eine Raumnummer zu untersuchen. Dadurch kann bei Bildern ohne einem erkennbaren Türschild schneller die Untersuchung abgebrochen werden, was einer zeitlichen Einsparung entspricht. Die konkrete Anwendung ist in Codeausschnitt 5.6 zu sehen. Sobald eine Kontur mit vier Punkten gefunden wurde, wird das Verhältnis von der Breite zur Höhe berechnet. Da die Türschilder quadratisch sind, sollte dieser Wert bei ca. eins liegen. In einem weiteren Verfahren wird die Konvektivität berechnet. Damit ist eine Abschätzung auf die Rechtwinkligkeit der Kontur möglich. Eine ausführliche Beschreibung zu dem Vorgehen ist in [YKR08] aus dem Band [Yin08] zu finden. Bei einem Quadrat handelt es sich um den Faktor eins, was den Maximalwert darstellt. Als letztes können die eben berechneten Werte mit den Vorgabewerten verglichen werden und erst wenn alle Bedingungen erfüllt sind, wird eine Überprüfung auf eine Raumnummer durchgeführt.

```python
if len(approx) == 4:
    #Verhältnis Breite zu Höhe bestimmen
    (x, y, w, h) = cv2.boundingRect(approx)
    aspectRatio = w / float(h)

    #Verhältnis aus konvexer zu realer Fläche berechnen
    area = cv2.contourArea(c)
    hullArea = cv2.contourArea(cv2.convexHull(c))
    solidity = area / float(hullArea)

    #Vergleich mit den vorgegebenen Vergleichswerten
    keepDims = w > 40 and h > 40
    keepSolidity = solidity > 0.9
    keepAspectRatio = aspectRatio >= 0.6 and aspectRatio <= 1.2

    #Sicherstellen, dass alle Bedingungen erfüllt sind
    if keepDims and keepSolidity and keepAspectRatio:

        #Anwendung des bekannten Algorithmus
```

Codeausschnitt 5.6: Änderung an der DetectRoom Funktion

Zeitliche Verbesserungen Um die unterschiedlichen Funktionen zu testen und eine Bewertung über die Effektivität abzugeben, wurde ein Versuch, mit der bereits in Abschnitt 4.2 vorgestellten Software, durchgeführt. Das Ergebnis ist in Abbildung 5.1 ersichtlich. Dabei wurden zwei unterschiedliche Mengen an Bildern verwendet. Bei der ersten Messung wurden ausschließlich Bilder aus Gängen verwendet. Bei der Zweiten wurden auch andere Bilder, wie vom Aufzug, Böden oder Wänden verwendet. Unter realen Bedingungen sollten diese jedoch nicht aufgenommen werden. Demzufolge ist die erste Messreihe die aussagekräftigere. Bei beiden Versuchen lässt sich jedoch feststellen, dass der neuere Algorithmus schneller arbeitet. Begründen lässt sich das, da nun weniger Befehle durchgeführt werden müssen und zu kleine oder zu große Rechtecke ignoriert werden. Insgesamt betrachtet konnten die Verbesserungen eine Verbesserung um mindestens 40 % erzielen. Wenn mehr uninteressante Bilder in der Menge enthalten sind, können sogar Einsparungen in Höhe von 60 % erreicht werden.

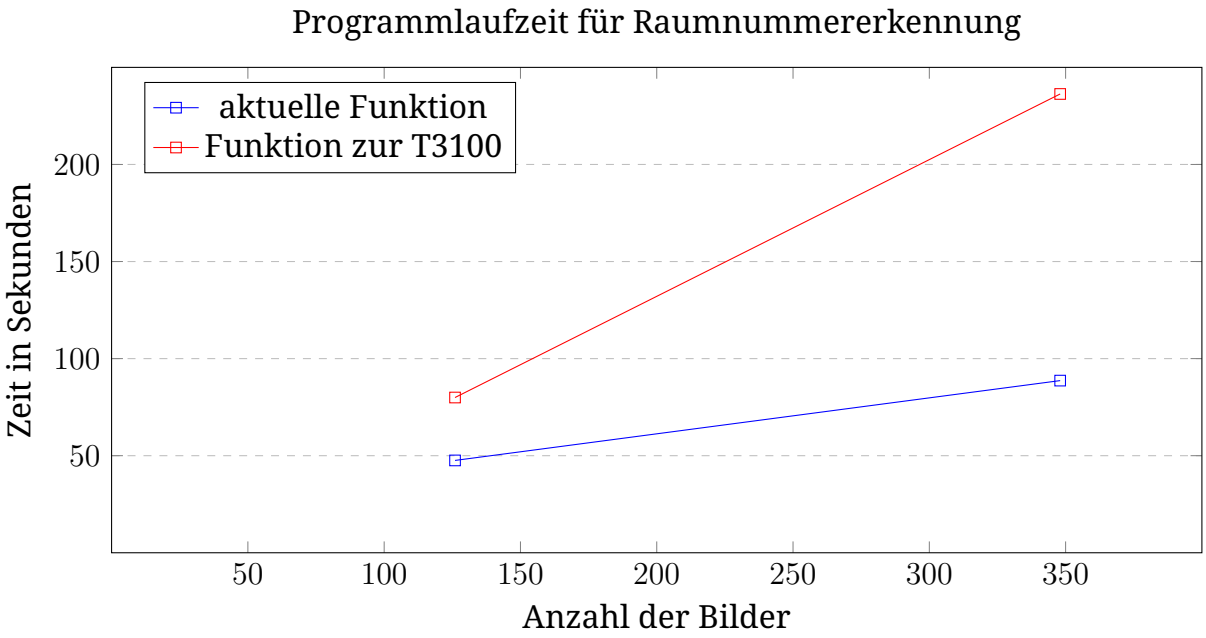


Abbildung 5.1: Darstellung der Programmlaufzeiten für die Nummernsuche anhand zweier Programmstände

6 Pepper fährt mit dem Fahrstuhl

Nachdem Pepper die Bestimmung der Startposition durchführen kann und auch Räume autonom anfahren kann (vgl. [Bol20]), soll ein weiterer Aspekt für einen Portier betrachtet werden. Da der Roboter Besucher zu bestimmten Räumen führen soll und anschließend wieder zurück an die Startposition fahren soll, muss Pepper eventuell eigenständig Fahrstuhl fahren. Solange Personen geführt werden, kann darauf noch teilweise verzichtet werden. Alle vorgestellten Methoden und Ansätze konnten nicht an der DHBW getestet werden. Der Aufbau der Software erfolgt, wie in Kapitel 2 angesprochen, nach dem in [Kan+10] vorgestelltem Ansatz.

6.1 Suchen eines Anforderungsknopfes an einem Fahrstuhl

Um Fahrstuhl zu fahren, muss als Erstes der Fahrstuhl gefunden und der Rufknopf betätigt werden. In [Bol20] wird vorgestellt, inwiefern Pepper bei einer Navigation über mehrere Stockwerke bis zu einem bestimmten Übergabepunkt am Aufzug fährt. Ab dieser Stelle soll eine BV eingesetzt werden, um den Rufknopf zu finden. Für diesen ersten Abschnitt konnte bereits die Software (siehe Codeausschnitt 6.1) erstellt werden und mittels der in Abschnitt 4.3 vorgestellten Testsoftware überprüft werden. Für die Auswertung wurden Testbilder von einer anderen Aufnahmequelle verwendet, weshalb die Parameter sicherlich noch an Pepper angepasst werden müssen. Die Software arbeitet die aufgenommenen Bilder innerhalb von vier Schritten ab. An einem Beispiel ist dies in Abbildung 6.1 zu sehen. Das Bild wird zunächst in den HSV-Farbraum transformiert, da hier eine Auswertung von silberner Farbe einfacher ist (vgl. Abbildung 6.1b). Danach kann mit den vorgegebenen Werten für Silber eine Binarisierung durchgeführt werden. Diese ist in Abbildung 6.1c zu sehen. Hier ist gut erkennbar, dass der Rufknopf eine gute Struktur aufweist. Anhand dieser Eigenschaft lässt er sich in einem letzten Schritt identifizieren. Dafür wurde, der in Abschnitt 5.2 bereits für die Detektion von Türschildern benutzte Algorithmus, mit anderen Parametern

verwendet. Ein Ergebnis mit dem eingezeichneten Ziel ist in Abbildung 6.1d ersichtlich. Der vorgestellte Algorithmus muss bei Bedarf eventuell mehrfach mit unterschiedlichen Blickwinkeln durchgeführt werden, um den Fahrstuhl sicher zu detektieren.

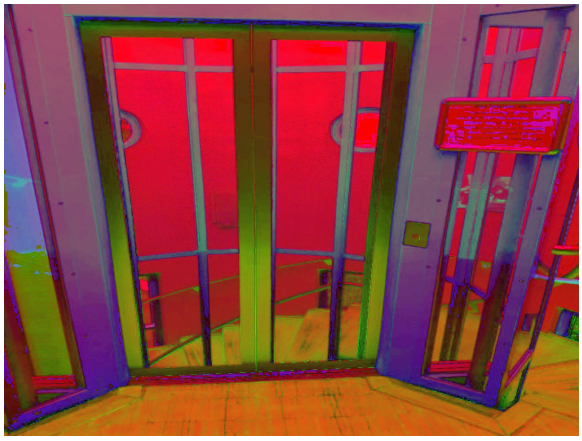
```
valid = False
#Grenzwerte für silber sind experimentell ermittelt, kann bei Pepper anders
     ausfallen
silverMin = np.array([4, 30, 50],np.uint8)
silverMax = np.array([30, 130, 150],np.uint8)
#Umwandlung in den HSV Farbraum, Silber kann besser ermittelt werden
hsv_img = cv2.cvtColor(img,cv2.COLOR_BGR2HSV)
#binarisiertes Bild mit dem Silberfilter anwenden
silver = cv2.inRange(hsv_img, silverMin, silverMax)
cnts = cv2.findContours(silver.copy(), cv2.RETR_EXTERNAL,cv2.
    CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts)
#Anwendung des bekannten Algorithmus zur Detektion von Vierecken mit
    bestimmten Abmessungen
```

Codeausschnitt 6.1: Finden des Fahrstuhlrufknopfes

6.2 Drücken des gefundenen Anforderungsknopfes

Als Ergebnis der vorgestellten Funktion (Abschnitt 6.1) liegen demnach die Koordinaten für den Rufknopf und die Größe im Bild vor. Mit Hilfe dieser Informationen kann die relative Position zu Pepper berechnet werden und mit Hilfe einer Ultraschallmessung die Entfernung. Durch diese Parameter sollte es nun möglich sein, Pepper langsam auf den Anforderungsknopf fahren zu lassen und regelmäßig die neue Bewegung zu berechnen. Sobald sich der Roboter nah genug am Fahrstuhl befindet, kann die Hand in die Nähe des Knopfes bewegt werden. Dieses Vorgehen benötigt ein ausführliches Erproben an der DHBW um ein möglichst gutes Ergebnis zu erzielen. Mit demselben Vorgehen, sollte ebenso ein Betätigen des ersten Knopfes möglich sein. Solange die Übergabeposition in diesen Programmaufruf genau genug ist, kann auch über einen festen Bewegungsablauf nachgedacht werden. Weiterhin sollte überlegt werden inwiefern es sinnvoll ist, bevor Pepper einen Aufzug ruft, zu überprüfen, ob dieser zufällig gerade geöffnet bereitsteht. Eine Alternative ist das lautstarke, freundliche Fragen von Passanten, ob diese einen Aufzug rufen könnten. Diese Variante sollte allerdings

(a) Ausgangsbild



(b) Ausgangsbild im HSV Farbraum

(c) Silbergefiltertes Bild

(d) markierter Knopf

Abbildung 6.1: Auffinden des silbernen Knopfes

nur im Ausnahmefall eingesetzt werden, wenn sich zum Beispiel kein Knopf finden lässt.

6.3 Befahren eines Fahrstuhls

Direkt nachdem der Aufzug gerufen wurde, sollte sich Pepper vor dem Fahrstuhl positionieren. Dies kann mit Hilfe eines festen Weges geschehen, da der Roboter zuvor jeweils an einer festen Position stehen muss. Im besten Fall reicht hierfür eine leichte Linksdrehung aus. Mittels einer Messung der vorderen Entfernung zu einem Hindernis kann abgeschätzt werden, wann der Fahrstuhl offen ist. Sobald dies der Fall ist, muss Pepper schnellst möglich hineinfahren. Dies ist wichtig, da die Türen sich nach einer kurzen Zeit wieder schließen. Außerdem sackt

der Aufzug bei einer Belastung ein Stück nach unten, um ein Verklemmen der Räder zu verhindern, sollte die Bewegung ebenfalls schnell erfolgen.

6.4 Zieletagenknopf suchen und drücken

Pepper sieht nach dem Einfahren ein Bild welches Abbildung 6.2a ähnlich sein wird. Aus dieser Perspektive muss sich der Roboter zunächst um 90 ° nach links drehen. Hier sollte es wie in Abbildung 6.2b aussehen.

(a) Sicht nach dem Einfahren in den Fahrstuhl

(b) Sicht nach einer Drehung nach links auf die Zielknöpfe

Abbildung 6.2: Blick in den Fahrstuhl aus zwei Perspektiven

An dieser Stelle ist die Verwendung von Mitteln der BV schwer anzuwenden, da auf dem silbernen Tableau die silbernen Knöpfe nur sehr schwer erkannt werden können. Stattdessen sollte an dieser Stelle mit festen Koordinaten gearbeitet werden. Mittel der BV sollten aber trotzdem ergänzend hinzugezogen werden, da sich Pepper sicherlich nicht immer an der exakten Position befindet. An dieser Stelle ist viel Erproben an der DHBW notwendig.

6.5 Ausfahren aus dem Fahrstuhl

Um den Fahrstuhl auf der richtigen Etage zu verlassen, muss sich Pepper noch einmal nach links drehen und kann sobald die gemessene Entfernung nach vorne größer geworden ist, wieder ausfahren. Dabei gilt wieder, dass dies schnell

erfolgen sollte. Einerseits um nicht zwischen den Türen eingeklemmt zu werden und andererseits, um den Spalt zwischen Aufzug und Etage möglichst gut zu überwinden.

6.6 Weitere Hinweise

Abschließend zum Thema Fahrstuhl fahren soll noch auf Schwierigkeiten bei der Umsetzung eingegangen werden. Die gesamte Darstellung beruhte auf der Annahme, dass es keine Eingriffe von Menschen in die Abarbeitung gab. Dies kann jedoch als Portier nicht ausgeschlossen werden. Es sollte also überlegt werden, inwiefern Menschen im und vor dem Aufzug zu Problemen führen können. Alternativ können diese natürlich auch direkt um Hilfe gebeten werden.

Ein weiteres Problem ist, dass Pepper sicherstellen muss, dass die erreichte Etage mit der gewünschten Etage übereinstimmt. Dies lässt sich am einfachsten Umsetzen, indem die Fahrzeit gemessen wird und anhand mehrere Testwerte verglichen wird. Daraus resultierend stellt sich auch die Frage, wie Pepper auf einem eventuell nicht gewollten Stockwerk sich weiter verhält. Eine Möglichkeit ist hier selbstverständlich eine Lokalisierung durchzuführen. Diese dauert aktuell in der Lobby jedoch lange. Alternativ kann noch einmal die gewünschte Etage im Fahrstuhl angewählt werden. Wenn sich der Fahrstuhl nicht bewegt (Messung und Auswertung des Gyroskops) ist das gewünschte Stockwerk erreicht, solange sich nichts in der Lichtschranke befindet.

Ein weiteres Hindernis stellen die Öffnungszeiten der DHBW dar. Solange der Publikumsverkehr zugelassen ist, ist eine Erarbeitung und Erprobung von Software in den Gängen und im Treppenhaus nur sehr schwer möglich. Viele Studierende halten gerne aus Neugier an und möchten gerne Pepper in Aktion erleben, wodurch eine konzentrierte Arbeitsumgebung nur schwer erreichbar ist. Die Alternative dazu besteht aus den Morgen- und Abendstunden, wenn nur wenige Personen sich am Campus aufhalten. Dies ist jedoch nur nach einer bürokratisch schwierigen Ausnahmeregelung zugelassen. Die Beantragung lohnt sich aber in jedem Fall. Eine weitere Möglichkeit ist die Nutzung des Kellers. Da sich dort meistens nicht viele Personen aufhalten ist ein Arbeiten dort auch tagsüber gut mit dem Fahrstuhl möglich.

7 Fazit

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

7.1 Zusammenfassung

In der vorgestellten Arbeit wurde betrachtet inwiefern der humanoide Roboter Pepper als Portier eingesetzt werden kann und entsprechende Software wurde vorgestellt. Im Rahmen der Lokalisierung konnten Verbesserungen in Bezug auf die Ausführungszeit bei konstanter Ausführungsgenauigkeit erzielt werden. Weiterhin wurden Ansätze gegeben, inwiefern der Roboter autonom Fahrstuhl fahren kann. In vielen Abschnitten konnten bislang lediglich theoretische Aspekte betrachtet werden. Diese und auch die bereits umgesetzten konnten lediglich in sehr geringer Ausprägung vor Ort mit Pepper erprobt werden. Ein Einsatz sollte unter diesen Umständen daher noch nicht geschehen.

7.2 Kritische Würdigung der Arbeit

Es konnten keine Testdurchläufe in der DHBW durchgeführt werden, weshalb sämtliche Software mit Vorsicht zu betrachten ist. Darüber hinaus wurden alle Herausforderungen bereits in den jeweiligen Kapiteln beschreiben. Diese hängen teilweise mit dem nichtgeplanten Eingreifen von Menschen zusammen und erfordern eine gesonderte Analyse. Die Laufzeitverbesserungen der Lokalisierung konnten in einer Simulation deutlich gezeigt werden.

7.3 Ausblick

Als letztes soll noch ein Blick auf mögliche Erweiterungen dieser Studienarbeit geworfen werden. Dabei lässt das Thema der BV noch Ergänzungen zu. Als Erstes sollte die vorgestellte Software getestet und in den erforderlichen Teilen noch umgesetzt werden. Dabei werden sich wahrscheinlich noch Probleme ergeben, die bisher nicht erkannt wurden.

In einem weiteren Schritt sollten diese und die Arbeit [Bol20] zusammengeführt werden. Dies würde es erlauben den Portier vollumfänglich einzusetzen. Weitere Themen können die Ausweitung auf den anderen Fahrstuhl oder auch die gezielte Erkennung von Strukturelementen sein. Dabei könnte die Lobby schneller erkannt werden und eine Positionsbestimmung zusätzlich zu den Räumen wäre möglich. Dies hätte ebenso den Vorteil, dass mehr Punkte zur Lokalisierung bereitstünden und die Qualität der Position sich verbessern würde.

Literatur

[Bol19] Jonas Boller. »Fortbewegung eines Pepper-Roboters innerhalb eines definierten Gebäudes: am Beispiel des Hauptgebäudes der DHBW Stuttgart Campus Horb«. Studienarbeit - T3100. Horb: DHBW, 2019.

[Bol20] Jonas Boller. »Fortbewegung eines Pepper-Roboters innerhalb eines definierten Gebäudes: am Beispiel des Hauptgebäudes der DHBW Stuttgart Campus Horb«. Studienarbeit - T3200. Horb: DHBW, 2020.

[Kan+10] Jeong-Gwan Kang u. a. »Recognition and path planning strategy for autonomous navigation in the elevator environment«. In: *International Journal of Control, Automation and Systems* 8.4 (2010), S. 808–821. ISSN: 1598-6446. DOI: 10.1007/s12555-010-0413-3. URL: https://doi.org/10.1007/s12555-010-0413-3.

[Ros] Adrian Rosebrock. *Practical Python and OpenCV*. 4. Aufl. URL: https://www.pyimagesearch.com/2015/05/04/target-acquired-finding-targets-in-drone-and-quadcopter-video-streams-using-python-and-opencv/ (besucht am 12. 05. 2020).

[Rot20] Franz W. Rother. *Smarte Roboter machen Putzkolonnen überflüssig: Sie arbeiten rund um die Uhr, kennen keine Tarifverträge – und können inzwischen sogar Aufzug fahren. Mitsubishi Electric treibt die Entwicklung mit einem neuen Steuerungssystem.* 2020. URL: https://edison.media/roboter-ersetzen-die-putzkolonnen/25201986/ (besucht am 12. 05. 2020).

[Sch19] Tim Schneider. »Selbstlokalisierung eines Pepper-Roboters innerhalb eines definierten Gebäudes: hier: Hauptgebäude der DHBW Stuttgart Campus Horb«. Studienarbeit - T3100. Horb: DHBW, 2019.

[Ven16] Martin Venhaus. *Klara vermittelt Industriekontakte: Fachhochschule Suedwestfalen und VW AG wollen kooperieren*. Iserlohn, 2016. URL: https://www4.fh-swf.de/de/home/ueber_uns/pressemitteilungen/pressearchiv/index.php?pressInfoId=2664 (besucht am 12. 05. 2020).

[Yin08] Peng-Yeng Yin, Hrsg. *Pattern Recognition*. IN-TECH, 2008. ISBN: 978-953-307-014-8. DOI: 10.5772/149.

[YKR08] Mingqiang Yang, Kidiyo Kpalma und Joseph Ronsin. »A Survey of Shape Feature Extraction Techniques«. In: *Pattern Recognition*. Hrsg. von Peng-Yeng Yin. IN-TECH, 2008, S. 43–90. ISBN: 978-953-307-014-8. URL: https://hal.archives-ouvertes.fr/hal-00446037.