Fakultät Informatik/Mathematik

Hochschule für
Technik und Wirtschaft
Dresden
University of Applied Sciences

Fachgebiet Künstliche Intelligenz

EIN ROBOTER–NACHTWÄCHTER ZUR UNTERSTÜTZUNG VON PFLEGEKRÄFTEN

Jan Brose

Studiengang: Allgemeine Informatik
Immatrikulationsjahr 2015

BACHELOR-ARBEIT

zur Erlangung des akademischen Grades

BACHELOR OF SCIENCE (B. SC.)

Gutachter

Dipl.-Inf. (FH) Frank Bahrmann

Betreuender Hochschullehrer

Prof. habil. Dr.-Ing. Hans-Joachim Böhme

Zu bearbeiten bis: 11. September 2018

In der vorliegenden Arbeit wird bei personenbezogenen Substantiven und Pronomen die Sprachform verwendet, die laut Duden „jemanden“ – eine Person – beschreibt. Aus Respekt wird der Mensch somit stets unabhängig vom biologischen/sozialen/kulturellen/. . . Geschlecht als Person angenommen.

Danksagung

Ich möchte allen Danken, die mich auf dem Weg bis hier hin und durch das Bachelorstudium unterstützt haben. Konkret möchte ich da meinen Eltern, meinem Bruder, Laura, meinen Professoren, den wissenschaftlichen Mitarbeitern, sowie der Arbeitsgruppe für künstliche Intelligenz und kognitive Robotik der HTW danken.

INHALTSVERZEICHNIS

ABBILDUNGSVERZEICHNIS

TABELLENVERZEICHNIS

1 EINLEITUNG

In Deutschland sind zur Zeit über 2,5 Millionen Menschen auf Pflege angewiesen[1], von denen gut ein Drittel in Pflegeeinrichtungen untergebracht ist. Gleichzeitig wird in den Nachrichten[2][3] und in der politischen Debatte immer wieder vom Pflegenotstand in Deutschland und von einer Überarbeitung der Pfleger aufgrund eines sehr großen Arbeitspensums und einer sehr anspruchsvollen Arbeit gesprochen.
Parallel dazu schreitet die Digitalisierung in der Gesellschaft weiter voran, sodass Maschinen, Computer und Roboter den Menschen an vielen Stellen bei seiner Arbeit unterstützen und ihm diese so erleichtern. So ergibt sich dann an einem gewissen Punkt die Frage, in wie weit technische Assistenzsysteme eingesetzt werden können, um das Pflegepersonal in Pflegeeinrichtungen entlasten zu können. Eines dieser Projekte, die sich mit dieser Frage befassen, ist die von [Lischke, Bahrmann, Hellbach & Böhme, 2017] beschriebene Roboter–Nachtwächter–Anwendung, welche den Rahmen für diese Bachelor–Arbeit bilden wird.
Diese Veröffentlichung ist Teil eines Forschungsprojektes der Arbeitsgruppe für künstliche Intelligenz und kognitive Robotik der HTW–Dresden um Prof. habil. Dr.-Ing. Hans-Joachim Böhme. Dabei werden in Kooperation mit einer lokalen Pflegeeinrichtung Anwendungsmöglichkeiten für den Einsatz von Assistenzrobotersystemen im Pflegesystem erforscht. Zu diesen Projekten gehört auch die oben erwähnte Roboter–Nachtwächter–Anwendung, welche die Pflegekräfte bei der nächtlichen Arbeit unterstützen und idealerweise auch die Sicherheit der Bewohner der Einrichtung erhöhen soll. Die aktuelle Situation wird von [Lischke et al., 2017] genauer beschrieben, während auf das Setting eingegangen wird.
So ist die Situation vor Ort, dass es einen gewissen Anteil an Patienten gibt, welche unter verschiedenen Formen von Demenz leiden. Zum Krankheitsbild der Menschen gehört unter Umständen auch die Bettflucht, beispielsweise ausgelöst durch das Verlangen, nachts einkaufen gehen zu müssen. Das ist für diese Personen mitunter sehr gefährlich, da es dauern kann, bis sie nach einem Sturz

[1] Bundesamt, 2017, S.5.
[2] Eubel und Heine, 2013.
[3] Stalinski, 2017.

gefunden werden. Das Risiko zu stürzen ist besonders hoch, wenn es den Personen gelingt, in die Treppenhäuser zu gelangen. Außerdem bestehen auch Gefahren, falls es den Personen gelingen würde, das Gebäude zu verlassen. Eine Gefahr besteht darin, dass sie sich aussperren würden, da die Türen zwar von innen aus Brandschutzgründen nicht verschlossen sein dürfen, aber von außen nicht geöffnet werden können. Dabei sind die Personen dann den Gefahren des öffentlichen Raums für merklich verwirrte Menschen ausgesetzt. In diesem Fall besteht besonders in kalten Nächten auch die Gefahr einer Unterkühlung oder von Schlimmeren.
Nun könnte man meinen, dass die Pflegekräfte dafür verantwortlich sind, damit dies nicht passieren kann. Dabei zeigt sich aber das Problem, dass es in Deutschland im Allgemeinen aktuell einen Pflegenotstand gibt, es also an Plätzen und Personal fehlt.
Deshalb ergibt sich für das Pflegepersonal in dieser speziellen Einrichtung das Problem, dass eine Pflegekraft während der Nachtschicht für 2 Stockwerke verantwortlich ist, welche aufgrund des Grundrisses des Gebäudes auch nicht komplett eingesehen werden können. Eine Karte von einem Stockwerk der Einrichtung kann man in Abbildung 1.1 sehen.

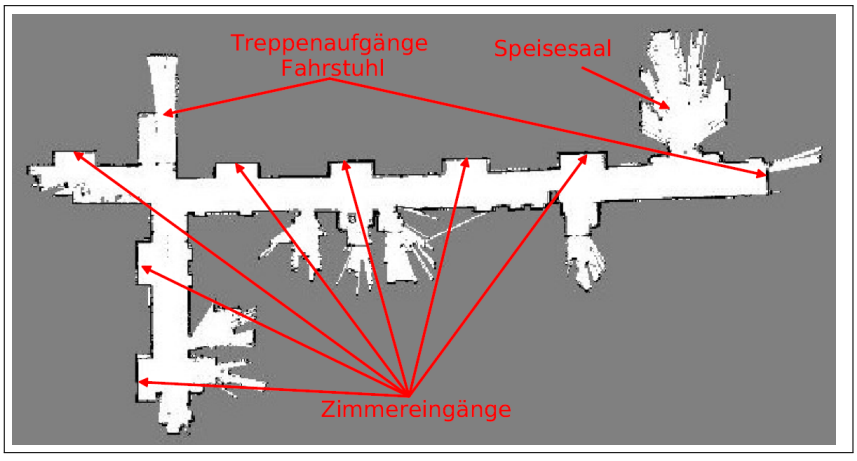

Abbildung 1.1: Karte von einem Stockwerk der Pflegeeinrichtung

Die Stockwerke verfügen über einen L–förmigen Grundriss. Auf den Gängen gibt es Einbuchtungen an den Eingängen der Zimmer, welche nicht von überall im Gang eingesehen werden können. Der Speisesaal ist vom Gang aus auch nicht einsehbar. Die Pflegekräfte müssen also immer den ganzen Flur ablaufen, dann das Stockwerk wechseln und dort auch den ganzen Flur ablaufen und das über die komplette Nachtschicht.
Es gibt also immer einen Bereich, in welchem die Pflegekraft aktuell nicht sein

kann. Dies kann auch der Fall sein, falls gerade ein Bewohner Hilfe benötigt.
Roboter könnten in solchen Situationen dadurch helfen, dass sie sich eigenständig auf einem Stockwerk bewegen können. Sie können auch auf festgelegten Routen patrouillieren. Dabei sind sie durch ihre Sensoren eigenständig in der Lage, Personen zu erkennen und die Situationen einzuschätzen, in denen sich die Personen befinden. Der Roboter verfügt auch über die Fähigkeit, mit einer Person aktiv zu kommunizieren. Er kann somit versuchen, sie zu beruhigen und ablenken. Außerdem kann er nebenbei eine Pflegekraft rufen, welche der Person weiterhelfen kann, bzw. diese wieder in ihr Zimmer bringen kann. Ein Roboter hat auch die Möglichkeit, Ausrüstung mitzuführen, wie beispielsweise einen Erste–Hilfe–Koffer, welchen das Personal dann nicht erst holen muss. Wie man bei [Hebesberger, Körtner, Pripfl, Gisinger, Hanheide et al., 2015] lesen kann, wird dies auch vom Pflegepersonal gewünscht.
Man hätte auch darüber nachdenken können, ein Kamera–System zu installieren, welches einem Roboter gegenüber aber den Nachteil hat, dass es fest verbaut wird und dabei auch nicht über die Möglichkeit verfügt, eine Person direkt zu konfrontieren. Dabei müsste in jedem Fall eine Pflegekraft eingreifen. Deshalb hat man diese Idee wieder verworfen.
Für die Umsetzung hat man sich dann entschieden, einen Service Roboter zu nutzen. Bei diesem handelt es sich um einen Scitos G5 von MetraLabs, einer Firma, die sich auf die Entwicklung von Service–Robotern spezialisiert hat. Einen der im Projekt genutzten Roboter kann man in der Abbildung 1.2 sehen.

Abbildung 1.2: Roboter „Anna Constantia“ - eine mobile Plattform vom Typ Scitos G5 hergestellt von MetraLabs

Damit der Roboter in der Lage ist, diese Aufgaben zu bewältigen und seine Funktionen zu erfüllen, benötigt er ein leistungsstarkes System im Hintergrund. Dieses lässt sich in mehrere Aufgabenbereiche unterteilen.
Diese Aufgabenbereiche sind beispielsweise die Navigation, die Personen–Erkennung und die Mensch–Maschine–Kommunikation. Dabei können sich die Bereiche aber auch überschneiden. So werden beispielsweise sowohl für die Navigation, als auch die Personen–Erkennung verarbeitete Sensordaten benötigt. Bei der Mensch–Maschine–Kommunikation wird dann wiederum mit den Ergebnissen der Personen–Erkennung gearbeitet und dabei auf die Navigation zurückgegriffen.
Dieser Rückgriff wird benötigt, damit der Roboter in die Nähe der Person gelangen kann, um mit dieser Person interagieren zu können. Nur sollte der Roboter nicht irgendwo in die Nähe der Person fahren und dort stehen bleiben. Die Position sollte dabei schon einen Sinn ergeben. Es wäre gut, wenn ein Roboter sich zum Kommunizieren im Sichtfeld einer Person positioniert. Es sollten sich auch keine Hindernisse zwischen der Person und dem Roboter befinden.
Genau diese Aufgabe des Ermittelns einer sinnvollen Position soll in dieser Arbeit angegangen werden. Dabei soll aus Daten, welche die Personen–Erkennung liefert, eine Position ermittelt werden, an welcher die Navigation den Roboter dann für die Mensch–Maschine–Kommunikation platzieren kann.

2 ZIELSTELLUNG

In dieser Arbeit soll im Rahmen eines Projektes für einen Roboter—Nachtwächter, der Pflegekräfte unterstützen soll, ein Modul entwickelt werden.
Dieses Modul soll dabei den Übergang zwischen der Personen–Erkennung und der Mensch–Maschine–Kommunikation bilden. Es soll genutzt werden können, um eine Position zu ermitteln, an welcher sich der Roboter platzieren könnte, um mit einer Person zu interagieren.
Dieses Modul soll also den Aufenthaltsort einer Person geliefert bekommen. Anhand dessen soll dann eine Position in der Nähe der Person ermittelt werden. Diese sollte so gewählt werden, dass sie vom System genutzt werden kann, um mit dem Roboter in einem sozial verträglichen Abstand eine Konversation zu starten. Dabei soll die Positionierung auch vom Verhalten der Person abhängig sein. Diese Position soll es dem Roboter also ermöglichen, sich ideal zu platzieren, um eine Person anzusprechen und damit zu binden.
Im Verlauf der Arbeit soll dieses Modul auch in den Gesamtkontext des Projektes eingeordnet werden.

3 LÖSUNGSWEG

Für die Berechnung der Position, welche der Roboter ansteuern soll, um eine Interaktion mit einer Person zu starten, soll das Modul „Communication–Position–Processor“ entwickelt werden, welches im bestehenden Roboter–Projekt nutzbar ist.
Dafür benötigt das Modul eine Schnittstelle, um die Daten für die Berechnung, vom System zu erhalten. Diese Daten sollten im Allgemeinen die Position, Blickrichtung und Bewegung der Person beschreiben.
Außerdem wird auch eine Schnittstelle benötigt, um die ermittelte Position dem System zur Verfügung zu stellen. Hier sollte das System eine Position und Ausrichtung zur Positionierung des Roboters erhalten.
Bei der Implementation dieser Schnittstellen wird dann nach Möglichkeit auf schon bestehende Elemente des Systems zurückgegriffen, welche sich dafür besonders eignen. Ein einfaches Konzept zur Veranschaulichung der Positionsermittlung ist in Abbildung 3.1 zu sehen. Dabei wird das Modul als Black–Box dargestellt, da es hier nur um eine grobe Einordnung des Systems geht. Der Communication–Position–Processor ist durch einen roten Rahmen hervorgehoben.

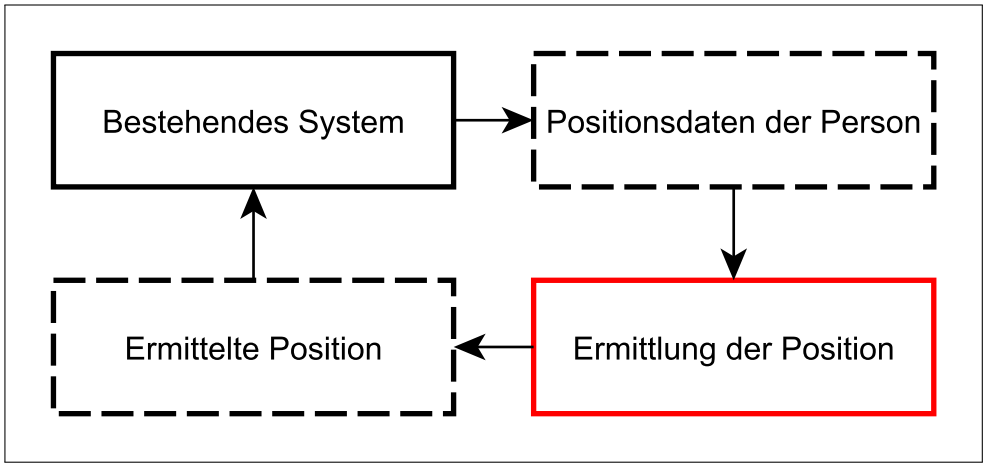

Abbildung 3.1: Grobes Konzept für die Entwicklung des Moduls.

Da es natürlich keinen Sinn ergibt, dieses Modul direkt auf dem Roboter zu tes-

ten und zu entwickeln, wird dieses in einer simulierten Umgebung durchgeführt. Dabei kann auch die Funktionalität des Communication–Position–Processor besser, schneller und vor allem einfacher getestet werden, als mit dem Roboter selbst. So ein Vorgehen ist bei der Entwicklung von Modulen für komplexe Systeme Standard. Für die Entwicklung wird dieselbe Umgebung genutzt, welche auch bei Arbeiten an der Navigation genutzt wird.
Das Modul soll am Ende in der Umsetzung über zwei verschiedene Modi verfügen. Diese sind „Konfrontieren“ und „Folgen“ . Das Modul sollte die folgenden Anforderungen erfüllen:

- Beim Konfrontieren soll die Position im Sichtfeld der Person liegen. Der Abstand zwischen Roboter und Person soll so gewählt sein, dass eine Interaktion problemlos möglich ist.
- Beim Folgen soll die Position so gewählt sein, dass sie gemessen an der Bewegungsrichtung immer hinter der Person ist, um diese nicht zu behindern. Außerdem soll der Abstand mit steigender Geschwindigkeit größer werden, um einen längeren Bremsweg zu ermöglichen.
- Die Position soll in jedem Fall so gewählt werden, dass diese sozial verträglich ist.
- Es soll eine Trägheit vorhanden sein, welche verhindert, dass bei Sensorrauschen oder minimalen Bewegungen der Person eine neue Position ermittelt wird.
- Die gelieferte Position soll in jedem Fall Sinn ergeben. Das heißt es sollte immer eine freie Sichtlinie zwischen Person und Roboter geben. Außerdem darf die Position nicht auf Wänden oder anderen Hindernissen platziert sein.
- Der Modul soll so aufgebaut sein, dass die Parameter mit geringem Aufwand verändert und somit einfach angepasst werden können.
- Außerdem muss darauf geachtet werden, dass das Modul um weitere Modi ergänzt werden kann, falls dies nötig wird.
- Im Allgemeinen sollte das Modul so gestaltet werden, dass es so wenig wie möglich Rechenleistung verbraucht. Das ist sinnvoll, um ausreichend Rechenleistung für parallel ablaufende, sehr rechenintensive Aufgaben nutzen zu können, wie beispielsweise die von [Eisenbach, Vorndran, Sorge & Gross, 2015] beschriebene Personen–Erkennung mit ihrem großen Leistungsbedarf.

4 HAUPTTEIL

In diesem Teil wird die Umsetzung des Lösungsweges genauer betrachtet, sowie eine Einordnung in den Gesamtkontext des Systems vorgenommen. Begonnen wird dabei mit einer kurzen Analyse des aktuellen Standes bezüglich der Entwicklung und Nutzung von technischen Assistenzsystemen in der Pflege. Dem folgt eine Einordnung in den Gesamtkontext des Roboter–Nachtwächter–Projektes. Dem schließt sich eine Beschreibung des Aufbaus des Moduls an. Den Abschluss bildet eine Erklärung der einzelnen Bestandteile des Moduls.

4.1 STAND DER TECHNIK

In diesem Kapitel gibt es eine kurze Betrachtung des Standes der Dinge im Bezug auf den Einsatz von Robotern in der Pflege. Dabei werden aber auch Projekte bedacht, bei denen ähnliche Fähigkeiten benötigt werden. Außerdem wird ein Blick auf die sozialen Problemstellungen bei der Arbeit mit Robotern geworfen.
Zuerst werfen wir dabei einen Blick auf die Arbeiten der Forschungsgruppe. Diese forscht nicht nur an dem von [Lischke et al., 2017] erwähnten Roboter–Nachtwächter für Pflegeeinrichtungen, sondern auch an einem Museumsführer, welcher von [Poschmann et al., 2012] beschrieben wird. Dieser muss auch in der Lage sein, auf Personen zuzugehen und hat auch sonst viel mit dem Nachtwächter gemeinsam, sodass an beiden Projekten parallel gearbeitet wird.
Ein weiteres Projekt, welches sich mit der Pflege von Menschen beschäftigt, ist das von [Gross et al., 2011], [Kessler, Schmidt, Helsper & Gross, 2013] und [Volkhardt & Gross, 2013] beschriebene Projekt. In diesem Projekt geht es darum, einen Roboter zu entwickeln, welcher in der Lage ist, ältere Menschen die Zuhause Hilfe benötigen, zu unterstützen. Dabei muss ähnlich wie beim Roboter–Nachtwächter eine Person erkannt werden können. Außerdem sollte sich dort der Roboter auch sinnvoll zur Person positionieren, um ihr helfen zu können.
In einem ähnlichen Umfeld wie das Roboter–Nachtwächter–Projekt arbeitet auch der Assistenzroboter, welcher von [Eisenbach et al., 2015] und [Horst-Michael Gross et al., 2017] beschrieben wird. Dabei soll ein Roboter Patienten bei ihren

Übungen auf den Fluren in einem Krankenhaus begleiten. Dazu muss er in der Lage sein einer Person zu folgen, sie wieder zu erkennen, sich im Umfeld zurechtfinden und sich problemlos in einer belebten Umgebung bewegen zu können.
Natürlich muss man sich auch damit beschäftigen, wie die Akzeptanz gegenüber Robotern bei der Arbeit mit älteren Menschen ist. Dabei kann man beispielsweise den Ausführungen von [Broadbent, Stafford & MacDonald, 2009] folgen. Die Pflegekräfte, deren Arbeit durch die Roboter erleichtert werden soll, sollten dabei nicht vergessen werden. Deren Wünsche an die Assistenzsysteme werden von [Hebesberger et al., 2015] genauer untersucht.
Es ist natürlich auch nötig, sich mit kulturellen Unterschieden beim Umgang mit Robotern zu beschäftigen. Als Beispiel dafür bietet sich auch ein Vergleich zwischen dem westlichen Kulturkreis und Japan an, wie er von [Kitano, 2006] ausgeführt wird.
Natürlich gibt es auch immer ethische Probleme und Bedenken, wie sie unter anderem von [Vercelli, Rainero, Ciferri, Boido & Pirri, 2018] thematisiert werden.
Es gibt selbstverständlich noch eine große Zahl anderer Projekte, welche sich mit Robotern, ihrem Verhalten, der Mensch–Maschine–Kommunikation, ethischen Bedenken, psychologischen Auswirkungen und immer neuen Einsatzgebieten befassen.

4.2 EINORDNUNG IN DEN GESAMTKONTEXT DES PROJEKTES

Wie bereits erwähnt ist das im Rahmen dieser Arbeit entwickelte Modul ein Teil des Projektes zur Entwicklung eines Roboter–Nachtwächters für Pflegeeinrichtungen. Diese Einordnung ist allerdings ziemlich allgemein und wird an dieser Stelle konkretisiert.
In der Einleitung wird schon darauf eingegangen, dass man das Gesamtsystem in mehrere Aufgabenbereiche unterteilen kann. Dabei gibt es zum Beispiel die Navigation zur Steuerung des Roboters, die Personen–Erkennung und die Mensch–Maschine Kommunikation. Diese Aufgabenbereiche überschneiden sich an gewissen Punkten. Es gibt im System Module, welche man sowohl dem einen als auch dem anderen Aufgabenbereich zuordnen könnte.
So verhält es sich auch mit diesem Modul, welches im Rahmen der vorliegenden Arbeit entwickelt wurde. Man kann dieses problemlos der Navigation zuordnen, aber auch der Personen–Erkennung und der Mensch–Maschine–Kommunikation.

Das liegt daran, dass sich diese Aufgabenbereiche überschneiden und somit Module benötigt werden, die diesen Übergang übernehmen. Solche Module erfüllen dann eine Hybrid–Aufgabe, welche die Bereiche verbindet.
Genau das kennzeichnet auch das hier beschriebene Modul, indem es die Aufgabenbereiche der Personen–Erkennung, der Navigation und der Mensch–Maschine–Kommunikation miteinander verbindet. Um die Bewältigung dieser Hybrid–Aufgabe durch das Modul etwas genauer zu verstehen, wird dieses jetzt erst einmal in den Kontext der drei genannten Aufgabenbereiche eingeordnet.
Bei der Navigation ist diese Einordnung einfach, da dieses Modul eine Position liefert, an der sich der Roboter positionieren soll. Das Modul liefert in diesem Fall also Eingangsdaten, welche von der Navigation genutzt werden können, um den Roboter zu positionieren. Dabei hat die Navigation aber keinen sehr großen Einfluss auf die Arbeit des Moduls selbst. Durch das Modul kann aber auch auf Daten, die die Navigation nutzt, zurückgegriffen werden.
Die Einordnung in den Kontext der Mensch–Maschine–Kommunikation ist auch noch relativ einfach. In diesem Modul werden zum einen Bedingungen für die Kommunikation geschaffen, dadurch dass eine Position zum Start dieser ermittelt wird. Außerdem werden soziale Regeln beachtet, welche zur passiven und nonverbalen Kommunikation gehören. Das Modul bildet also ähnlich der Navigation eine Voraussetzung für die Schaffung der Bedingungen einer erfolgreichen Interaktion.
Die Personen–Erkennung hat den anderen beiden Aufgabenteilen gegenüber einen größeren Einfluss auf dieses Modul. Um dies besser zu verstehen, lohnt es sich, die Personen–Erkennung genauer zu betrachten. Dazu wurde Abbildung 4.1 erstellt, um die Personen–Erkennung etwas vereinfacht zu veranschaulichen. Dafür wurden die Veröffentlichungen von [Eisenbach et al., 2015] und [Volkhardt & Gross, 2013] zu Rate gezogen und Abbildungen, die diese genutzt haben, noch etwas vereinfacht. Die Erkennung von Personen wird an dieser Stelle nur soweit erklärt, wie dies für die Einordnung notwendig ist.

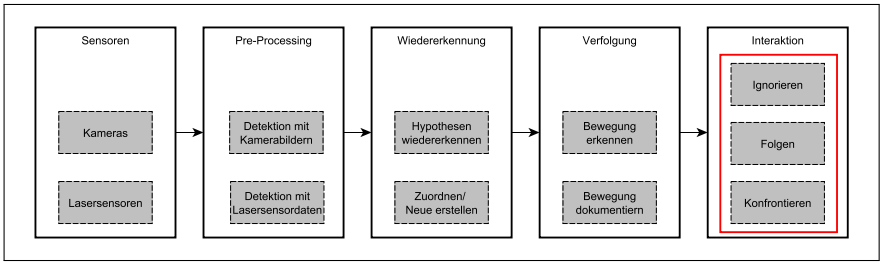

Abbildung 4.1: Veranschaulichung der Personenerkennung

Rot markiert ist in Abbildung 4.1 der Bereich dieser Arbeit im Gesamtkontext der Personen–Erkennung. Wie man in der Abbildung 4.1 sehen kann, beginnt die Erkennung von Personen mit der Auswertung der Sensordaten, die dem Roboter zur Verfügung stehen. Dies sind zum einen Lasersensoren, welche am Roboter verbaut sind und ein $360°$–Bild der Umgebung des Roboters liefern, als auch ein RGB–D–Sensor vom Typ Microsoft Kinect One.
In den Daten, die die Lasersensoren liefern, ist es möglich, nach Beinpaaren zu suchen, um ein Verdachtsmoment auf eine Person zu erhalten. Dabei sind Beinpaare dadurch erkennbar, dass die Sensoren zwei Hindernisse ähnlicher Form in räumlicher Nähe zueinander wahrnehmen. Diese Detektion kann aber auch beispielsweise von Stuhlbeinen getäuscht werden. Daher will man sich nicht nur auf diese Daten verlassen.
Deshalb werden als zweiter Lieferant für Verdachtsmomente die Daten des RGB–D–Sensors genutzt. Dabei wird auf den Bildern, die dieser liefert, nach Personen gesucht.
Im nächsten Schritt werden die Daten der beiden Sensoren zusammengefasst, falls möglich. Die resultierenden Eindrücke werden dann im nächsten Schritt versucht Verdachtsmomenten zuzuordnen, welche in vorherigen Durchläufen erkannt wurden. So können diese aktualisiert werden. Falls ein Eindruck keinem bestehenden Verdachtsmoment zugeordnet werden kann, wird dieser bewertet. Dabei soll festgestellt werden, in wie weit es sich wahrscheinlich um eine Person handelt. Wenn die Bewertung mit ausreichend hoher Wahrscheinlichkeit für eine Person spricht, wird ein neuer Verdachtsmoment mit diesen Daten erstellt, ansonsten wird er ignoriert.
Ab diesem Moment handelt es sich bei einem Verdachtsmoment um eine Person, wobei natürlich auch die Möglichkeit besteht, dass die Erkennung getäuscht wurde. Durch die vorangegangenen Schritte wird es auch möglich, die Bewegungen der Person zu verfolgen.
Zuletzt wird geklärt, wie mit der Person in der betreffenden Situation weiter verfahren werden soll. Dabei ergeben sich drei Möglichkeiten.

1. Man kann die Person ignorieren: Dies könnte man beispielsweise nutzen, um zu verhindern, dass der Roboter Pfleger in Konversationen verwickelt, wenn er sie wahrnimmt.

2. Die Person kann verfolgt werden.

3. Es kann eine Konversation gestartet werden, die Person wird also angesprochen und somit gebunden.

Dabei ist die erste Möglichkeit nicht sehr kompliziert, da dabei nicht auf die erkannte Person reagiert wird. Bei der zweiten und dritten Möglichkeit greift dann das entwickelte Modul ein und liefert für die Navigation eine Position mit der die beiden Möglichkeiten umgesetzt werden können. Dann wird die Verbindung zwischen der Personen–Erkennung, der Navigation und der Mensch–Maschine–Kommunikation geschaffen.
Hier wird auch ersichtlich, dass das Modul auf eine gut funktionierende Personen–Erkennung angewiesen ist.

4.3 ÜBERBLICK ÜBER DAS ENTWICKELTE MODUL

Ziel dieser Arbeit war es, ein Modul zu entwickeln, welches in der Lage ist eine ideale Position in der Nähe einer Person zu berechnen. Dieses sollte nicht direkt auf dem Roboter umgesetzt werden, da es einfach zu aufwendig wäre, jeden Test auf diesem durchzuführen und die Testsituationen zu provozieren, wurde auf eine simulierte Testumgebung zurückgegriffen. Dabei fiel die Wahl, wie bereits in den Ausführungen zum Lösungsweg erwähnt, auf die Testumgebung in welcher die Pfadplanung auch entwickelt wurde. Diese bietet den Vorteil, dass sich dort Positionen recht einfach visualisieren lassen. Das ist sehr nützlich, da eine Position ermittelt wird.
Innerhalb dieser Testumgebung ist es aber nicht möglich, eine Personen–Erkennung zu simulieren. Dies ist aber kein größeres Problem, da so garantiert ist, dass das Modul unabhängig von der Personen–Erkennung einsatzfähig ist.
Aus diesem Grund müssen die Daten, welche eine Person simulieren sollen, anders erzeugt werden. Dazu bietet sich ein zuvor entwickelter, manuell steuerbarer simulierter Roboter an. Die Daten, die dieser erzeugt, liegen aber noch nicht in der Form vor, die die Eingangsdaten des Moduls haben sollen. Diese müssen also noch umgeformt werden. Dafür wird dann ein Prä–Prozessor implementiert.
Die errechnete Position kann dann mit einem Plugin zur Anzeige einer Roboter–Position erfolgen. Dafür wird ein Post–Prozessor benötigt, welcher die Daten für dieses Plugin in die richtige Form bringt.
Wenn man dies beachtet, wird es möglich, ein erstes Konzept zu entwerfen, wie das Modul dann in die Testumgebung integriert werden kann. Dieses ist in Abbildung 4.2 zu sehen.
Dieses Konzept zeigt die erste Idee, welche im weiteren Verlauf noch konkreti-

siert wurde. Deshalb wurden die Datenflüsse an dieser Stelle noch nicht genauer benannt und zu Oberbegriffen zusammengefasst. Die einzelnen Arbeitsschritte werden nach ihren Aufgaben benannt. Durch einen roten Rahmen wurde innerhalb der Abbildung der Teil gekennzeichnet, der neu zu entwickeln ist.

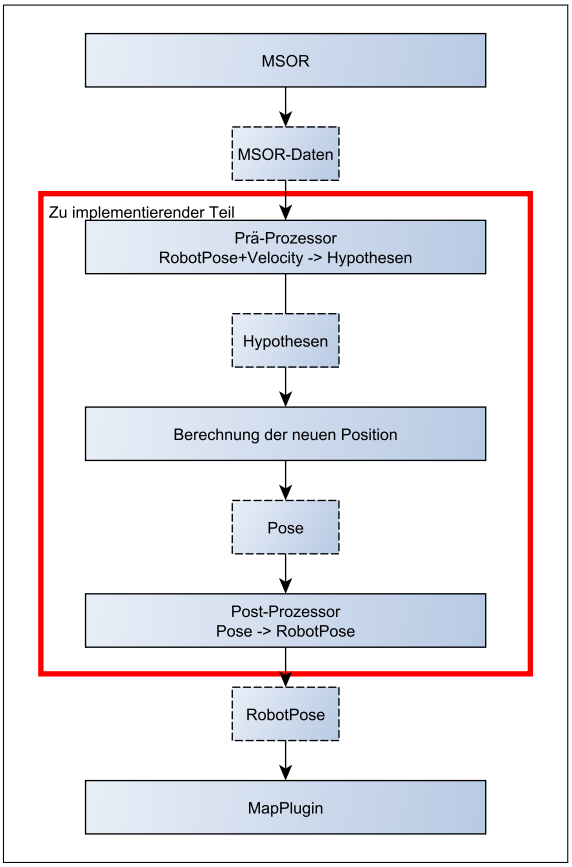

Abbildung 4.2: Ablaufdiagramm eines ersten Konzepts für die Entwicklung des Moduls

Nachdem ein erstes Konzept für die Umsetzung vorliegt, kann man damit beginnen, das Modul genauer zu planen. Dazu kann man den einzelnen Schritten Klassen zuweisen, welche diese Aufgabe im weiteren Verlauf übernehmen sollen. Außerdem kann man dann die Ressourcenflüsse konkreter definieren. Des Weiteren soll Rücksicht auf weitere Ressourcen genommen werden, welche im ersten Konzept noch vernachlässigt wurden.
Wenn man diese Schritte so umsetzt, kann man damit das Konzept konkretisieren, so dass sich dabei das Ablaufdiagramm aus Abbildung 4.3 ergibt.

In diesem Diagramm sieht man jetzt, dass abgesehen von den Daten, welche der der simulierte Hindernisroboter liefert, auch noch ein Modus benötigt wird. Es wird auch gezeigt, dass das Modul eine Karte von der Navigation bekommt, welche später für die Sicherheitsmechanismen genutzt wird.

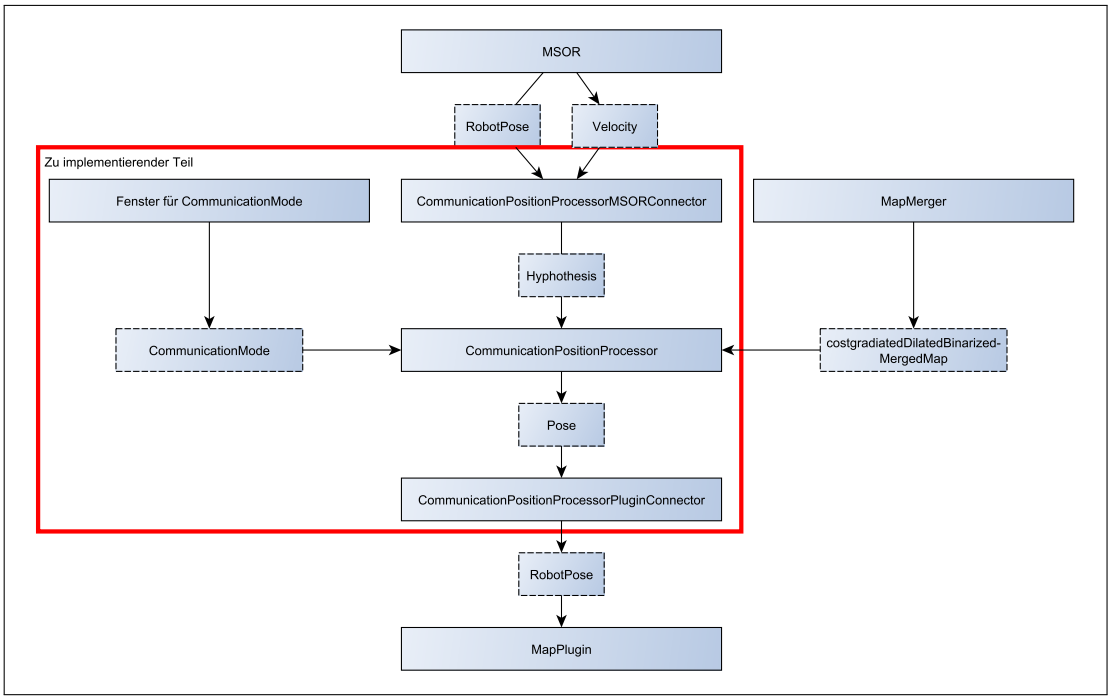

Abbildung 4.3: Ablaufdiagramm des Konzeptes mit den beteiligten Klassen

Die Funktionsweise und die Umsetzung der einzelnen in diesem Konzept gezeigten Elemente werden in den folgenden Unterkapiteln genauer erläutert. Dabei wird im Kapitel 4.4 damit begonnen zu erklären, was genau eine Hypothese ist und wie diese dann im Programm definiert ist. Im Kapitel 4.5 wird darauf eingegangen, was der simulierte Hindernisroboter genau ist und wieso die Daten, die er liefert, umgewandelt werden müssen. Es wird dabei auch erklärt, wie genau dies dann umgesetzt wird. Im darauf folgenden Kapitel 4.6 soll es dann um das eigentliche Modul gehen. Dabei wird noch speziell auf dessen Bestandteile eingegangen und diese werden genauer erläutert. Dem schließt sich im Kapitel 4.7 eine Beschreibung des Post–Processors und des genutzten Map–Plugins an. Im Kapitel 4.8 wird dann noch gezeigt, wie dieses Modul in der Testumgebung integriert wurde.

4.4 UMSETZUNG DER BESTÄTIGTEN HYPOTHESEN

In diesem Teil soll es darum gehen, was eine bestätigte Hypothese ist, was diese auszeichnet und welche Bedeutung diese Hypothesen für die Arbeit haben. Die Erläuterung zu Hypothesen und dazu, wie diese gefunden werden, sind notwen-

dig, bevor dann auf die Bestätigung eingegangen werden kann.
Bei einer Hypothese handelt es sich um einen Ort von Interesse innerhalb des wahrnehmbaren Bereichs um einen Roboter herum. Sie befindet sich also innerhalb des Bereichs, der durch Sensoren des Roboters erfasst wird. Diese Orte sind deshalb von Interesse, da an ihnen eine Person vermutet wird. Es handelt sich also um eine Art Verdachtspunkt für sich dort aufhaltende Personen. Die Verdachtspunkte werden durch die Auswertung der Sensordaten des Roboters und der Fusion dieser gefunden.
Das Finden der Verdachtsmomente erfolgt, wie bereits bei der Einordnung in den Gesamtkontext erwähnt, durch die Suche nach Beinpaaren in dem $360°$ Laserbild des Roboter–Umfeldes, als auch im Erkennen von Personen in den Bildern des RGB–D–Sensors des Roboters. Die dabei gefunden Punkte werden dann zusammengefasst und schon bestehenden Hypothesen zugeordnet, falls dies möglich ist. Dies geschieht zum einen durch das Vergleichen mit bestehenden Hypothesen, welche dann im Anschluss daran aktualisiert werden können.
Falls eine Zuordnung nicht möglich ist, wird der Eindruck bzw. die Hypothese bewertet. Dabei werden die Daten der Sensoren noch einmal genauer betrachtet und dementsprechend eine Entscheidung gefällt. Wenn diese Entscheidung positiv ist, wird der Eindruck bestätigt und aus ihm eine neue Hypothese erstellt. Diese wird fortan auch als bestätigte Hypothese bezeichnet.
Im idealen Fall werden alle Personen in der Nähe des Roboters erkannt und bestätigten Hypothesen zugeordnet. Somit würde jede Hypothese eine Person repräsentieren. Da aber kein System perfekt ist, wird in Tabelle 4.1 dargestellt welche Fälle es bei der Bewertung von Hypothesen gibt.

	Hypothese repräsentiert eine Person	Hypothese repräsentiert keine Person
Hypothese wurde bestätigt	richtig positiv	falsch positiv
Hypothese wurde nicht bestätigt	falsch negativ	richtig negativ

Tabelle 4.1: Mögliche Fälle bei der Bewertung von Hypothesen

Dabei kann man erkennen, dass die Hypothese als positiv oder negativ bewertet werden kann. Positiv heißt in diesem Fall, sie wird bestätigt, und negativ heißt, sie wird nicht bestätigt. Außerdem sieht man, dass sich eine Person hinter der Hypothese verbergen kann oder auch nicht. Daraus ergeben sich dann vier mögliche Szenarien.

1. richtig positiv: Die Hypothese wurde bestätigt und hinter ihr verbirgt sich

tatsächlich eine Person. Die Bewertung lief also fehlerlos.

2. falsch positiv: Die Hypothese wurde bestätigt, aber es verbirgt sich keine Person hinter dieser. Die Personen–Erkennung wurde also getäuscht, die Bewertung war damit fehlerhaft.

3. falsch negativ: Die Hypothese wurde nicht bestätigt, obwohl sich am Ort eine Person befindet. Die Personen–Erkennung hat diese also nicht erkannt und damit war auch die Bewertung fehlerhaft.

4. richtig negativ: Die Hypothese wurde nicht bestätigt und am Ort befindet sich auch keine Person. Die Bewertung hat wie gewünscht funktioniert.

Während die erste und die vierte Möglichkeit das anzustrebende Ideal sind, existieren noch die Fehlerfälle 2 und 3. Dabei ist der Fehlerfall 3 nichts, was für nachfolgende Module zum Problem wird, da diese Personen dann nur ignoriert werden. Fehlerfall 2 hingegen kann zum Problem werden, wenn mit einer Person interagiert werden soll, die nicht existiert. Dies ist aber für das Modul noch kein Problem, sondern erst für die nachfolgende Mensch–Maschine–Kommunikation. Wie man mit solchen Fällen verfahren kann, wird beispielsweise von [Volkhardt & Gross, 2013] genauer thematisiert. Darauf soll aber an dieser Stelle nicht genauer eingegangen werden, da es für die Arbeit des Moduls nicht von Bedeutung ist. An diesem Punkt weiß jetzt zwar die Personen–Erkennung über die bestätigten Hypothesen Bescheid, der Rest des Systems kann damit aber nicht viel anfangen. Daher werden die Hypothesen für die weitere Bearbeitung vereinfacht und vereinheitlicht. Im Zuge dessen ergeben sich dann drei Punkte, die bei einer Hypothese besonders interessant sind.

1. Die Position, an der sich die Hypothese im globalen Koordinatensystem befindet.

2. Die Bewegung der Hypothese. Dabei ist besonders interessant, in welche Richtung sich die Hypothese bewegt und mit welcher Geschwindigkeit dies geschieht.

3. Die Orientierung der Hypothese. Dabei geht es darum, zu wissen, in welche Richtung die Person schaut, welche von der Hypothese repräsentiert wird, oder wie sie ausgerichtet ist.

Um das jetzt zu implementieren lohnt ein Blick in das bestehende System. Dort existiert schon eine Verwaltungseinheit, welche in der Lage ist, die Position und

die Orientierung eines Objektes zu verwalten. Dabei handelt es sich um die $Pose$–Klasse.
In dieser wird die Position eines Objektes durch deren Echt–Welt–Koordinaten im Raum beschrieben. Dieses Koordinatensystem ergibt sich auf Basis der statischen Karte, welche der Roboter von seiner Umgebung hat und ist fest. Die Koordinaten sind also unabhängig von der Roboter–Position. Die Orientierung wird in der $Pose$–Klasse durch einen Winkel im Radianten–Maß verwaltet. Dabei repräsentiert $0\ rad$ die positive Richtung der X–Achse und $\pi\ rad$ bzw. $-\pi\ rad$ die negative Richtung der X–Achse im globalen Koordinatensystem des Roboters. Damit lassen sich also zwei interessante Teile der Hypothese schon verwalten.
Für die Bewegung, wurde dann noch ein zweidimensionaler Vektor genutzt. Dieser bildet die Geschwindigkeit in X–Richtung und Y–Richtung in Meter pro Sekunde ab. Dadurch werden sowohl Geschwindigkeit als auch Richtung der Bewegung relativ einfach und kompakt abgebildet. Wenn man diese beiden Teile jetzt zusammenfasst, ergibt sich die in Abbildung 4.4 gezeigte Klasse zur Verwaltung von Hypothesen.

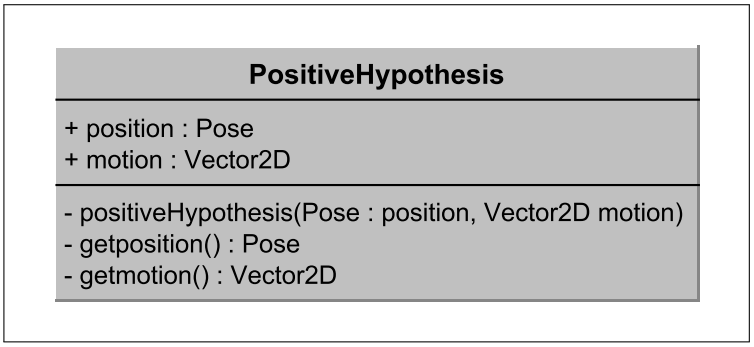

Abbildung 4.4: Aufbau der Klasse zur Beschreibung einer bestätigten Hypothese

Die Übergabe von Hypothesen innerhalb des Systems erfolgt mit Hilfe des Container–Systems. Es wird dazu ein $StateContainer$ vom Typ $PositiveHypothesis$ erstellt. In diesem werden die Werte immer gemeinsam mit einem Zeitstempel gespeichert. Dadurch benötigt diese Klasse auch keine Funktionen, um die Werte nachträglich zu verändern, sondern nur um sie zu lesen. So ist die Aktualität der Daten gewährleistet und Zugriffskonflikte werden verhindert.

4.5 MSOR UND PRE–PROCESSOR

In diesem Teil der Arbeit soll es um den Manual Steering Obstacle Robot(MSOR) und die Verarbeitung der von ihm gelieferten Daten im Rahmen des Projektes gehen. Der MSOR ist ein steuerbares Hindernis, welches ursprünglich implementiert wurde, um in der Testumgebung gewisse Problemsituationen, die sich bei der Verwaltung von Karten in einer dynamischen Umgebung ergeben, besser testen zu können. So konnte man ihn beispielsweise nutzen, um die Parameter beim Fusionieren von statischen und Sensorkarten besser einzustellen. Dieses Objekt besteht natürlich nur für die Testumgebung, welche zum Testen der Navigation existiert. Der Name des MSOR ergibt sich daraus, dass er eine manuell steuerbare Weiterentwicklung des „Obstacale Robot“ ist, welcher sich relativ zufällig durch die Testumgebung bewegt hat. Er wird als Roboter bezeichnet, da er mit demselben Grundgerüst simuliert wird wie der eigentliche Roboter auch.
Der MSOR verfügt über drei Schnittstellen, von denen eine ein Eingang ist und zwei Ausgänge sind. Im Eingang erhält er einen Container, der ihm $Velocity$–Werte liefert. Diese Werte sind zum einen eine Geschwindigkeit, mit der sich der Roboter vorwärts bzw. rückwärts bewegen sollte, als auch eine Rotationsgeschwindigkeit, welche angibt, wie er sich um die eigene Achse drehen soll. Diese Werte werden von einem Plugin erzeugt, welches über ein mit der Maus bedienbares Steuerkreuz verfügt. Alternativ kann in diesem aber auch eine externe Quelle zur Steuerung zugelassen werden. Dies kann dann beispielsweise ein Gamepad sein. Die Startposition des MSOR wird bei seiner Initialisierung festgelegt.
Der eine Ausgang ist ein Container, in welchem die tatsächlichen $Velocity$–Werte gespeichert werden. Dies ist notwendig, da die Eingaben nicht immer genau so umgesetzt werden können. Es kann beispielsweise passieren, dass der MSOR vor eine Wand fährt. Dann ist die tatsächliche Geschwindigkeit 0, aber die Eingabe kann anders lauten.
Die zweite Ausgabe ist ein Container, in welchem die Position des MSOR verwaltet wird. Diese Positionsangabe erfolgt im $RobotPose$–Format. Dabei handelt es sich um eine Erweiterung des $Pose$–Formates um einen Radius für die Größe des Roboters und einen Odometer–Wert, in welchem die zurückgelegte Distanz gespeichert ist.
Der MSOR soll jetzt also genutzt werden, um die Daten einer Hypothese zu erzeugen und diese simulieren zu können. Dabei besteht aber noch das Problem, dass der MSOR die Daten in einem anderem Format liefert, als dies für die Hy-

pothesen vereinbart ist. Um dieses Problem zu lösen, wird es nötig, einen Prä–Prozessor zu implementieren, welcher die beiden vom MSOR gelieferten Datenströme fusioniert und in die richtige Form bringt.
Bei den Positionswerten ist dabei keine Berechnung erforderlich, da diese in beiden Formaten als $Pose$ verwaltet werden. Sie können also einfach übernommen werden. Bei der Geschwindigkeit sieht die Sache allerdings anders aus. Dort ist für die Hypothesen vereinbart, Geschwindigkeiten nur als zweidimensionalen Vektor anzugeben. Daher kann die Rotationsgeschwindigkeit vernachlässigt werden, da sie in diesem Vektor nicht dargestellt werden kann. So wird also nur die Geschwindigkeit der Fortbewegung beachtet und da diese in Richtung der Orientierung des MSOR erfolgt, ergibt sich die Formel 4.1 für die Umformung der Geschwindigkeiten.

$$\vec{v} = \begin{pmatrix} v * cos(\alpha) \\ v * sin(\alpha) \end{pmatrix} \tag{4.1}$$

Dabei repräsentiert $\vec{v}$ den zu errechnenden Vektor mit der Geschwindigkeit, v ist dabei der in der $Velocity$ gespeicherte Wert für die Geschwindigkeit und α repräsentiert die Orientierung des MSOR.
Um die Aktualität der Hypothesen garantieren zu können, muss dieser Prä–Prozessor, immer, wenn ein neuer Wert in den Container mit der $RobotPose$ oder $Velocity$ geschrieben wird, diese Berechnung durchführen und das Ergebnis in den Container für die Hypothesen schreiben. Zur Umsetzung dieser Garantie existieren schon ableitbare Klassen, welche in anderen Teilen des Projektes zu solchen Zwecken genutzt werden.
Da wären zum einen die $StateObserver$, welche immer reagieren, wenn ein neuer Wert in den von ihnen überwachten Container geschrieben wird. Bei diesen kann es aber zu Problemen kommen, wenn dort umfangreichere Berechnungen durchgeführt werden und sie können immer nur einen Container auf einmal überwachen. Daher eignen sie sich für diesen Teil nicht.
Dafür existieren aber die $SyncProcessor$–Klassen, welche die Mutterklassen für Module bilden sollen, die bis zu vier Container überwachen können und bei Änderungen größere Berechnungen durchführen können. In diesem Fall wird ein $SyncProcessor2$ genutzt, da dieser es ermöglicht, die beiden Container mit der $Velocity$ und der $RobotPose$ gleichzeitig zu überwachen. Die Berechnungen werden dann in der geerbten $onData$–Methode vorgenommen, welche den aktuellsten Wert aus den beiden Containern gleich mitliefert. Das Ergebnis der Berechnungen wird dann in einer $PositiveHypothesis$ zusammengefasst und in den Container für die Hypothesen geschrieben.

Auf das Diagramm für die Klasse dieses Prä–Prozessors kann an dieser Stelle verzichtet werden. Diese Klasse enthält nur zwei Membervariablen, den Container für die Hypothesen und den Time–Provider. Der Provider sorgt dafür, dass im ganzen System mit einer einheitlichen Zeit gearbeitet wird. Bei den Methoden gibt es einen Konstruktor, welchem die Container mit der $RobotPose$ und $Velocity$, der Container für die Hypothesen und der Time–Provider übergeben werden. Die andere enthaltene Methode ist die oben beschriebene geerbte $onData$–Methode.

4.6 COMMUNICATION–POSITION–PROCESSOR

Dieses Kapitel handelt von dem Teil des entwickelten Moduls, das die Berechnungen übernehmen und die Position zur Platzierung des Roboters ermitteln soll. Da dieses Modul das Herzstück dieser Arbeit bildet, wird es ausführlicher behandelt. Dafür wird in den folgenden Unterkapiteln genauer auf die wichtigsten Aspekte dieses Moduls eingegangen.
Begonnen wird mit einem genaueren Blick auf die Eingangsdaten des Moduls und auf die Herkunft derer. Außerdem werden dabei die Ausgangsdaten noch einmal etwas genauer erklärt. Danach werden die Berechnungen erläutert, welche im Rahmen des Moduls durchgeführt werden müssen. Dabei wird auch auf die Parameter dieser Berechnungen eingegangen und wie sich die gewählten Werte dafür ergeben. Danach werden die Sicherheitsmechanismen, die es gibt, damit immer ein sinnvoller Punkt ermittelt werden kann, erläutert. Dabei wird genauer auf eine Wanddetektion und eine Suche nach Alternativpunkten eingegangen. Zuletzt erfolgt noch ein Blick wie die vorher beschriebenen Teilschritte zusammengesetzt wurden und weshalb es so realisiert wurde.

4.6.1 EINGANGSDATEN UND AUSGANGSDATEN

An dieser Stelle werden die Eingangs- und Ausgangsdaten des Moduls genauer thematisiert. Dazu lohnt es sich, zuerst einen genaueren Blick auf das im gesamten Roboter–Projekt genutzte Container–System zu werfen. Dieses wird auch im entwickelten Modul wieder genutzt, um die Übergabe und Verwaltung der Daten zu übernehmen.
Bei dem Container–System wird der aktuelle Stand der Daten immer mit dem aktuellen Zeitstempel gespeichert. Dadurch wird erreicht, dass man immer direkt weiß, was der aktuellste Wert innerhalb des Containers ist. Außerdem werden Zugriffsprobleme vermieden, da ein neuer Wert mit einem neuem Zeitstempel

eingetragen wird ohne den der ursprünglichen Wert zu verändern.
Bei der Initialisierung eines Containers muss diesem noch ein Typ mitgegeben werden, welcher sicherstellt, dass nur Daten eines bestimmten Formates in ihm gespeichert werden können.
Es gibt noch spezialisierte Versionen von Containern, welche sich auf bestimmte Arten von Daten konzentrieren und für diese Vorteile bieten. In dieser Arbeit werden hauptsächlich $StateContainer$ genutzt. Diese eignen sich gut um kleine Datenstrukturen zu verwalten.
Es gibt aber auch noch den $GridMapContainer$, welcher gesondert zum Verwalten von Karten existiert. Außerdem gibt es noch ableitbare Interfaces dieser Grund–Arten der Container in Form von Providern und Changern. Diese erweitern die Grundarten derart, dass die Provider für Eingangsdaten gedacht sind und die Changer für Ausgangsdaten. Für den $StateContainer$ wären dies der $StateProvider$ und der $StateChanger$.
In Kapitel 4.3 wurden im Konzept schon alle Eingangsdaten genannt, welche für dieses Modul benötigt werden. Dabei gibt es die Hypothesen, welche schon häufiger thematisiert wurden. Diese werden in einem $StateContainer$ des Typs $PositiveHypothesis$ verwaltet und in der Testumgebung vom MSOR und dem Prä–Prozessor erzeugt. Wenn das Modul an seinem Bestimmungsort integriert ist, erhält es die Daten von der Personen–Erkennung.
Außerdem benötigt das Modul noch den Modus, in dem es agieren soll. Dieser wird mithilfe eines Enums $CommunicationMode$ definiert. Dort gibt es einen Wert für den Modus „Folgen“ und einen Wert für den Modus „Konfrontieren“. Die Umsetzung in einem Enum bietet sich an, da ein solches sehr einfach zu erweitern ist und somit auch ideal dafür ist ggf. später noch einen weiteren Modus hinzuzufügen.
Im Roboter erfolgt die Bestimmung des Modus dann innerhalb der Personen–Erkennung, wo entschieden werden soll, wie mit der Hypothese verfahren werden soll. In der Testumgebung wird dies aktuell über ein einfaches Plugin gesteuert. Dieses besitzt nur eine $Combobox$, über welche einer der beiden Modi ausgewählt werden kann. Die Auswahl wird dann in den $StateContainer$ vom Typ $CommunicationMode$ geschrieben und kann so vom Modul genutzt werden.
Zuletzt benötigt das Modul noch eine aktuelle Karte von der Umgebung des Roboters. Die Karte wird nicht für die Berechnungen an sich benötigt. Sie wird für die Sicherheitsmechanismen genutzt, in welchen dann geprüft wird, ob der errechnete Punkt gültig ist und ggf. ein Alternativpunkt gesucht wird.
Für die Übergabe der Karte wird dann der bereits genannte $GridMapContainer$

genutzt. Die verwendete Karte wird dabei von der lokalen Navigation geliefert, in welcher die lokalen Laserkarten und die statische Karte fusioniert werden und für die weitere Nutzung noch aufgearbeitet werden.
In der lokalen Navigation gibt es nun aber eine Vielzahl von Karten. Deshalb geht es um die Entscheidung, welche Karte am sinnvollsten für dieses Modul ist. Dabei ist die Entscheidung dann auf die dilatierte, binarisierte Version der Karte mit Kostengradienten gefallen. Wie diese Karte in der Testumgebung visualisiert wird, ist in Abbildung 4.5 zu sehen.
Diese Karte entsteht dadurch, dass zuerst die lokalen Karten mit der statischen Karte fusioniert werden. Wie genau das Erstellen der lokalen und statischen Karte funktioniert und was dabei abläuft, kann man in den Veröffentlichungen von [Bahrmann, Hellbach, Keil & Böhme, 2014] und [Bahrmann, Hellbach & Böhme, 2016] genauer nachlesen. Danach wird über diese Karte eine Kostengradienten–Funktion laufen gelassen, bei welcher dann jeder Zelle anhand ihrer Belegtheitswahrscheinlichkeit ein Wert mit einem Kostengradienten zugeteilt wird. Dieser ist umso größer, je höher die Belegtheitswahrscheinlichkeit ist. Im nachfolgenden Schritt wird die Karte dilatiert. Das heißt, dass die Wände und Hindernisse, also Stellen mit hohem Kostengradienten, aufgebläht werden. So haben jetzt auch Zellen direkt neben Hindernissen einen erhöhten Kostengradienten, welcher aber mit Abstand zur eigentlichen blockierten Zelle abnimmt. Im nächsten Schritt wurde die Karte dann binarisiert. Das heißt, es wurde für jede Zelle anhand des Kostengradienten entschieden, ob diese belegt ist oder ob sie frei ist.

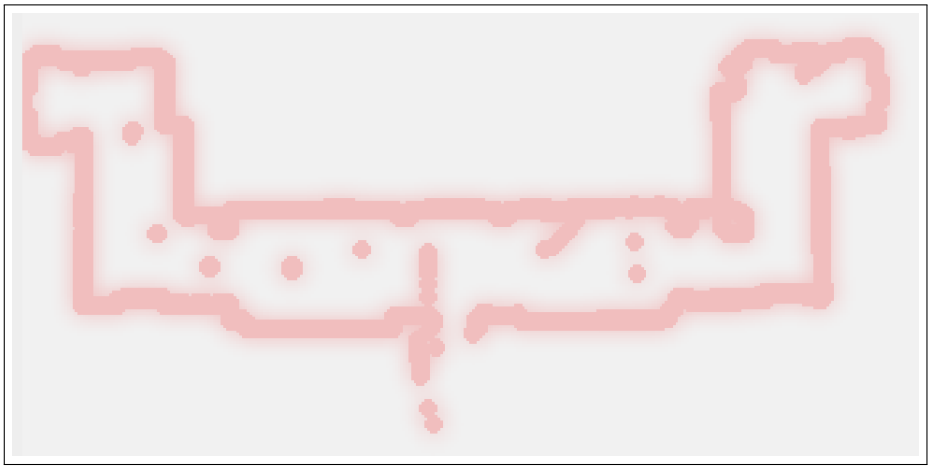

Abbildung 4.5: Dilatierte, binarisierte, Karte mit Kostengradienten in der Testumgebung (Rotfärbung eines Pixels symbolisiert seinen Kostengradienten)

Diese Karte bietet sich gut an, da hier alle Wände enthalten sein sollten. Da in

diesem Modul eine Position berechnet werden soll, an welcher sich der Roboter platzieren soll, ist vor allem die Dilatation sehr nützlich, da somit auch keine Position gefunden werden kann, an der kein Platz für den Roboter wäre. Dies hat aber auch den Nachteil, dass in ihr durch die Dilatation auch freie Bereiche als belegt markiert werden.
Die Ausgangsdaten dieses Moduls bestehen aus der Position und der Orientierung des Roboters. Beide sind, wie bereits erwähnt, im $Pose$ Format zusammengefasst. Für die Ausgabe wird also nur ein $StateContainer$ des Typs $Pose$ benötigt, in welchen dann immer der errechnete Wert geschrieben wird.

4.6.2 HERLEITUNG DER BERECHNUNGEN UND DER PARAMETER

In diesem Teil wird genauer darauf eingegangen, welche Berechnungen durchgeführt werden, um die Position und die Orientierung zu bestimmen. Außerdem wird darauf eingegangen, welche Parameter existieren und wie sich die für diese festgelegte Werte ergeben. Die Parameter wurden dann in der Umsetzung so eingestellt, dass sie auch über die Konfigurationsdatei des Roboters verändert werden können.
In den folgenden drei Unterabschnitten wird auf die einzelnen Berechnungen der Position in den beiden Modi und der Orientierung genauer eingegangen.

POSITION KONFRONTIEREN

Die Position beim Konfrontieren soll so gewählt sein, dass sich der Roboter immer im Sichtfeld der Person positioniert. Daher ist sie sowohl von seiner Orientierung als auch seiner bisherigen Position abhängig. Beim Konfrontieren soll die Entfernung zur Person geschwindigkeitsunabhängig sein. Anhand dieser Festlegungen ergibt sich dann die Formel 4.2 für die Berechnung der Position. Dabei wird die Position als zweidimensionaler Vektor dargestellt.

$$\vec{pos} = \begin{pmatrix} xhypo + cos(\alpha) * dist \\ yhypo + sin(\alpha) * dist \end{pmatrix} \tag{4.2}$$

Hierbei ist $\vec{pos}$ die zu errechnende Position, $xhypo$ und $yhypo$ bilden dabei die Position der Hypothese, α ist der Winkel, welcher die Orientierung der Hypothese widerspiegelt und $dist$ ist die Distanz, welche sich zwischen dem Roboter und der Hypothese befinden soll. Diese Distanz ist also bei der Berechnung der Posi-

tion beim Konfrontieren der einzige Parameter, welcher nicht von vorhergehenden Schritten geliefert wird.
Zur Festlegung des Wertes für die Distanz, in welcher sich der Roboter dann im Abstand zu einer Person positionieren soll, bedarf es einiger Überlegungen. Dabei sollte zum einen beachtet werden, dass der Roboter nah genug an der Person stehen sollte, damit diese weiß, dass er mit ihr kommunizieren möchte und ggf. auch Interaktionen mit dem Roboter und seinem Bildschirm möglich sind. Auf der anderen Seite sollte der Roboter aber auch nicht so nah an der Person stehen, dass diese sich unwohl fühlt.
Um dafür eine geeignete Entfernung zu finden, lohnt sich ein Blick auf die in der Veröffentlichung [Kessler et al., 2013] gezeigte Tabelle. Diese basiert auf der von [Hall, 1966] getroffenen Einteilung von Persönlichkeitsbereichen im Umkreis einer Person.
Aus den bisherigen Erfahrungen ist zu berücksichtigen, dass in einem ersten Test in der Einrichtung der Roboter von den Bewohnern durchaus freundschaftlich aufgenommen und nach dessen Ende sogar vermisst wurde.
Daher würde es sich anbieten, den Roboter zwischen 45 cm und 1,2 Metern von der Hypothese entfernt zu platzieren. Allerdings ist es auch möglich, dass in der Einrichtung Menschen unterwegs sind, die noch keinen Kontakt zum Roboter hatten oder diesem nicht so freundschaftlich gesinnt sind. Für diese würde sich dann ein Bereich zwischen 1,2 Metern und 3,6 Metern anbieten, in welchem nicht freundschaftliche Konversationen stattfinden.
Damit der Roboter aber nicht unterscheiden muss, welche Einstellung eine Person zu ihm hat, wird die Distanz nicht variabel gestaltet. Aus diesem Grund wurde entschieden, immer eine Distanz von 1,2 Metern zu wählen. Das liegt genau auf der Grenze zwischen den Bereichen für eine Konversation mit einem Freund oder einem Fremden. Außerdem bietet die Distanz den Vorteil, dass sie nah genug ist, um der Person das Gefühl zu geben, angesprochen zu werden. Auch ist die Distanz noch gering genug, um beispielsweise den Bildschirm des Roboters mit in die Konversation einbinden zu können.
Die Berechnung liefert mit dieser Parameter–Wahl jetzt immer eine Position, die im Sichtfeld der Person liegt und sich genau 1,2 Meter von dieser entfernt befindet.
Nun kann es aber sein, dass genau diese errechnete Stelle ungültig ist. Das wäre der Fall, wenn sie in einem Bereich liegt an dem sich schon ein Hindernis befindet oder für die Hypothese gesehen hinter einer Wand. Um dies zu verhindern, gibt es die in Abschnitt 4.6.3 beschriebene Sicherheitsmechanismen, welche dann

Alternativpositionen liefern.

POSITION FOLGEN

Die Position beim Folgen sollte so gewählt sein, dass der Roboter sich idealerweise immer hinter der Person befindet. Dabei ist hinter der Person in diesem Fall so zu definieren, dass hier die Bewegungsrichtung genutzt wird. Der Roboter sollte sich also immer auf der entgegengesetzten Seite befinden, in die sich die Person bewegt.
Außerdem soll der Abstand bei höheren Geschwindigkeiten größer gewählt sein, als bei geringeren Geschwindigkeiten. Dies soll dem Roboter im Falle eines Sturzes der Person einen längeren Reaktionszeitraum und Bremsweg einräumen.
Zuerst muss daher in diesem Fall die Bewegungsrichtung ermittelt werden. Das erfolgt dadurch, dass man den Winkel zwischen einem Vektor, der die positive Richtung der X–Achse repräsentiert und dem Vektor der Geschwindigkeit ermittelt. Falls sich die Hypothese nicht bewegt, wird die Orientierung genutzt.
Danach ergibt sich die in Formel 4.3 folgende Berechnung für die Position des Roboters im Modus „Folgen“.

$$\vec{pos} = \begin{pmatrix} xhypo \\ yhypo \end{pmatrix} + \begin{pmatrix} cos(\beta + 180°) \\ sin(\beta + 180°) \end{pmatrix} * dist + \begin{pmatrix} |\frac{motx}{maxspeed}| \\ |\frac{moty}{maxspeed}| \end{pmatrix} \tag{4.3}$$

Dabei sind $\vec{pos}$ die Position, $xhypo$ und $yhypo$ die Position der Hypothese, β der Winkel für die Bewegungsrichtung, $motx$ und $moty$ die Bewegung und $maxspeed$ die Maximalgeschwindigkeit des Roboters. Die Berechnung hat eine gewisse Ähnlichkeit mit der beim Konfrontieren im ersten Teil, besitzt aber noch einen Teil, der die Geschwindigkeit mit berücksichtigt.
Letzterer Teil ist so gewählt, dass die Position um bis zu 1 Meter zurückgesetzt wird, wenn der Roboter mit Höchstgeschwindigkeit fährt. Dabei ist die Höchstgeschwindigkeit der genutzten Roboter 1 m/s. Man kann den Parameter aber auch niedriger setzen, um den Abstand bei Geschwindigkeiten, die größer als $maxspeed$ sind, zu vergrößern. Damit ist $maxspeed$ also einer der beiden Parameter.
Der zweite Parameter ist in diesem Fall $dist$, welcher wie beim Konfrontieren den grundlegenden Abstand bildet. Nun musste wieder überlegt werden, wie man die Werte sinnvoll setzen sollte. Dafür bot es sich an, die Veröffentlichung [Horst-Michael Gross et al., 2017] zu betrachten.
In dieser wird mit einem ähnlichen Roboter von dem gleichen Hersteller gearbei-

tet. Außerdem ist auch das Umfeld ein ähnliches, da es gewisse Gemeinsamkeiten im Einsatzgebiet gibt. Dort werden Patienten auf Krankenhausfluren begleitet, während es beim Roboter–Nachtwächter darum geht, eine Person auf dem Flur in einer Pflegeeinrichtung zu verfolgen. Diese Veröffentlichung ist auch von daher praktisch, da sie von einem praktischen Test berichtet, bei dem sich gewisse nutzbare Werte, für die Distanz zwischen der Person und dem Ziel, eingestellt haben. Dabei wurde dort im Rahmen des Projektes ein Abstand zwischen 1,4 Metern bei langsamen Personen und 2,5 Metern bei schnelleren Personen ermittelt.
Darauf basierend ist die Entscheidung gefallen, $dist$ mit 1,4 m zu belegen. $max\text{-}speed$ wird mit 1 m/s gewählt.
So können die Distanzen zwischen Position und Person beim Folgen also zwischen 1,4 m und 2,4 m schwanken. Die Positionen werden wie beim Konfrontieren natürlich noch auf die Verfügbarkeit kontrolliert, bevor die Orientierung berechnet wird.

WINKEL DER ORIENTIERUNG

Die Orientierung ist ein Winkel, welcher angibt, wie sich der Roboter ausrichten soll. Bei den Anforderungen im Lösungsweg wird bereits erwähnt, dass der Roboter immer zur Person hin ausgerichtet sein soll. Der Winkel für die Orientierung wird auf dieselbe Weise angegeben, wie dies bei der Orientierung der Hypothese erfolgt. Daher ist die Berechnung relativ einfach. Dazu wird ein Vektor zwischen der ermittelten Position und der Postion der Hypothese gebildet. Danach wird noch ein Vektor genutzt, bei welchem der X–Wert positiv und der Y–Wert 0 ist. Im Anschluss daran wird der Winkel zwischen beiden Vektoren errechnet. Dieses Prinzip wird in der Abbildung 4.6 noch einmal veranschaulicht. Der errechnete Winkel bildet dann die Orientierung des Roboters ab.

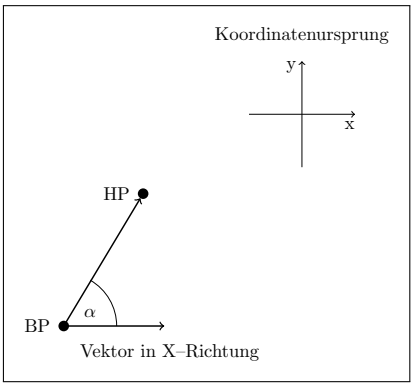

Abbildung 4.6: Veranschaulichung der Berechnung der Orientierung des Roboters.

In dieser Abbildung wird die berechnete Position durch BP und die Position der Hypothese durch HP gezeigt. Diese sind durch einen Vektor verbunden. Von der berechneten Position geht dann noch ein Vektor in positiver X–Richtung ab. Zwischen diesen beiden Vektoren wird dann der Winkel α gesucht. Dieser Winkel entspricht der Orientierung.

4.6.3 SICHERHEITSMECHANISMEN

Um garantieren zu können, dass das Modul immer eine gültige Position liefert, werden noch Sicherheitsmechanismen benötigt. Diese sollen dann überprüfen, ob der errechnete Punkt gültig ist. Damit eine errechnete Position gültig ist, muss sie zwei Bedingungen erfüllen:

- Die errechnete Position darf nicht auf einer Wand oder einem anderem Hindernis liegen.
- Zwischen der errechneten Position und der Position der Hypothese darf sich kein Hindernis befinden.

Die erste Bedingung ist relativ einfach zu prüfen. Dafür benötigt man die aktuelle Instanz der bei den Eingangsdaten beschrieben Karte aus dem Container. Diese bietet dann die Möglichkeit zu überprüfen, ob die Position auf der Karte frei ist.
Für die zweite Bedingung wurde im Rahmen des Projektes eine Wanddetektion entwickelt. Die Funktionsweise wird in einem gesonderten Unterkapitel beschrieben. Die Wanddetektion liefert dann einen Wert, der die Existenz eines Hindernisses zwischen beiden Punkten bestätigt oder ablehnt.
Wenn eine der beiden Bedingungen nicht erfüllt werden kann, muss ein Alternativpunkt ermittelt werden. Dieser sollte den selben Abstand vom Ziel haben, wie die errechnete Position und den kleinstmöglichen Abstand zu der errechneten Position. Wie die Suche nach dem Alternativpunkt genau funktioniert, wird in einem weiteren Unterkapitel beschrieben.

WANDDETEKTION

Die Wanddetektion selbst ist nicht sehr komplex. Dabei handelt es sich um ein Modul, welches einen Karten–Container bei der Initialisierung bekommt. Außerdem verfügt es noch über eine Methode, welche zwei Punkte geliefert bekommt und einen $boolean$–Wert liefert, der angibt, ob eine Wand gefunden wurde oder nicht.

Innerhalb der Methode wird dann die aktuellste Instanz der Karte geladen. Mit Hilfe des von [Bresenham, 1965] entwickelten Algorithmus zum Zeichnen von Linien wird dann überprüft, ob sich zwischen beiden Punkten ein belegter Bereich befindet. Dabei muss noch berücksichtigt werden, dass die Hypothese selbst dafür sorgt, dass ein Bereich um sie herum belegt ist. So wird um die Position, welche die Hypothese repräsentieren soll, noch ein Radius gewählt, in welchem nicht geprüft wird. Dieser wurde so gewählt, dass er auch eine Wand feststellen kann, vor welcher die Hypothese direkt steht. Ein Beispiel für die Funktionsweise dieser Erkennung ist in Abbildung 4.7 zu sehen.

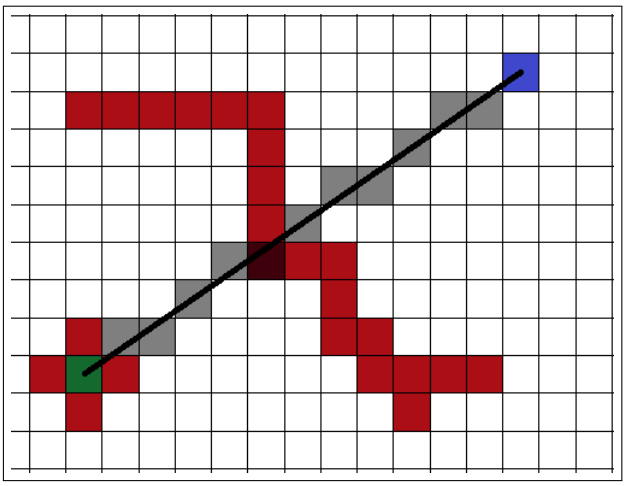

Abbildung 4.7: Veranschaulichung zur Funktionsweise der Wanddetektion

In der Abbildung ist ein vereinfachtes Beispiel zu sehen, welches die Funktionsweise der Wanddetektion erklären soll. Zu sehen ist die durch Gitterlinien dargestellte GridMap . Das grüne Feld ist die Position der Hypothese, das blaue Feld die errechnete Position. Rote Felder sind blockierte Felder. Graue Felder zeigen die Gerade, die der Algorithmus finden würde. Das dunkle Feld in der Mitte markiert den Punkt der die Wanddetektion auslösen würde. Man erkennt in der Abbildung den Bereich um die Hypothese, welcher auch blockiert ist durch ihren eigenen Umfang. Dieser wird im Algorithmus dann ignoriert. In der Praxis sind die Wände innerhalb der Karte weiter aufgebläht. Darauf wurde aber verzichtet, da die Abbildung nur das Prinzip zeigen soll. Bei der dunkelrot markierten Stelle würde die Wanddetektion stoppen und melden, dass sich ein Hindernis zwischen den beiden Punkten befindet.

ALTERNATIVPUNKTE

Die Suche nach Alternativpunkten wird notwendig, wenn der errechnete Punkt ungültig ist. Dafür wird dann die aktuellste Version der Karte genutzt. Auf dieser wird ein rechteckiger Bereich, in dessen Mitte die Position der Hypothese liegt, nach einer Alternative abgesucht. Dafür kann eine Funktion, die von den Karten angeboten wird, genutzt werden.
Dazu muss man den Bereich definieren. Die Ecken werden dadurch bestimmt, dass ihre X- und Y–Werte jeweils um den Abstand zwischen Hypothese und berechneter Position vom Mittelpunkt abweichen.
Bei der Suche wird dann die Position ermittelt die folgende Bedingungen erfüllt:

- Die Zelle der GridMap an der Position ist als frei markiert.
- Der Abstand zwischen dem Punkt und der Hypothese ist genau so groß wie bei der errechneten Position.
- Zwischen der Hypothese und der errechneten Position befindet sich keine Wand.
- Die Position hat den kleinstmöglichen Abstand zur berechneten Position.

In Abbildung A.4 im Anhang sind diese Suche auch noch einmal skizziert.
Falls keine Position diese Bedingungen erfüllt, was sehr unwahrscheinlich ist, gibt es noch eine Funktion, die garantieren soll, dass in jedem Fall eine Position geliefert werden kann. Dafür wird im Suchbereich die Position mit dem geringsten Kostengradienten genutzt. Der vorherige Teil wurde aber so konstruiert, dass es sehr unwahrscheinlich ist, dass diese zweite Suche genutzt werden muss.

4.6.4 AUFBAU DER KLASSE UND UMSETZUNG

Das Modul muss ähnlich wie der Prä–Prozessor auch einen Eingang überwachen. Deshalb gilt es wieder zu klären, ob ein $StateObserver$ oder ein $SyncProcessor$ zu nutzen ist. In dieser Klasse finden unter Umständen sehr umfangreiche Berechnungen statt. Daher wird wieder auf einen $SyncProcessor$ zurückgegriffen. Da nur ein Eingang zu überwachen ist, wird ein $SyncProcessor1$ genutzt.
Dieses Modul benötigt die bei den Eingangs- und Ausgangsdaten beschriebenen Container. Sie werden im Konstruktor bei der Initialisierung mit übergeben.
Außerdem verfügt die abgeleitete Klasse auch wieder über eine $onData$–Methode. In dieser muss dann jedes Mal beim Eintreffen der Daten geprüft werden, ob

Berechnungen notwendig sind. Wenn dem so ist, müssen die beschriebenen Berechnungen durchgeführt werden und die Sicherheitsmechanismen beachtet werden. Die Vorgehensweise kann in vier grundlegende Schritte eingeteilt werden.
Zuerst muss geschaut werden, ob es überhaupt notwendig ist, eine neue Position zu berechnen. Dazu wird eine Funktion genutzt, welche prüft, ob sich seit dem letztem Durchlauf die Position der Hypothese bzw. der Status des Moduls geändert haben. Für den Fall, dass die Veränderungen geringer als ein einstellbarer Schwellwert sind, sind keine Berechnungen notwendig und die nachfolgenden Schritte entfallen. Die Parameter wurden in der Implementation so gewählt, dass die neue Position um 15 cm von der vorherigen abweichen kann. Die Orientierung darf um $5°$ variieren. Diese Parameter wurden durch Tests in der simulierten Umgebung ermittelt.
Falls eine neue Berechnung notwendig ist, weil sich die Position, die Orientierung oder der Modus stärker geändert haben, werden die nachfolgenden drei Schritte durchgeführt.
Dabei wird zuerst eine neue Position ermittelt. Diese wird zuerst wie beschrieben berechnet. Das ermittelte Ergebnis wird dann, wie in den Sicherheitsmechanismen beschrieben, überprüft. Es wird geprüft, ob die Position auf einem Hindernis liegt oder sich ein Hindernis zwischen Hypothese und Position befindet. Wenn die Position ungültig ist, wird wie beschrieben eine Alternativposition ermittelt.
Nachdem die Position feststeht, wird die Orientierung berechnet. Dies geschieht wie oben beschrieben. Es gibt lediglich eine Sicherheitsmechanik davor, um einen Nullvektor bei der Berechnung zu vermeiden. Diese ist nur vorhanden, da bei anderer Parametereinstellung auch solche Ergebnisse möglich wären. Der Nullvektor tritt auf, falls die ermittelte Position und die Hypothesen–Position übereinstimmen. Dann wird die Orientierung der Hypothese übernommen.
Im letzten Schritt werden jetzt die Position und die Orientierung zu einer $Pose$ zusammengefasst und in den dafür vorgesehenen Container geschrieben.
In Abbildung A.3 ist der Ablauf innerhalb der $onData$–Methode noch einmal veranschaulicht.

4.7 POST–PROCESSOR UND KARTEN–PLUGIN

In diesem Teil sollen der Post–Prozessor und das Karten–Plugin genauer betrachtet werden. Dazu ist es sinnvoll zuerst das Karten–Plugin zu betrachten, da daraus die Notwendigkeit des Post–Prozessors erkennbar wird.

Dieses Karten–Plugin existiert schon in der Testumgebung. Es wird dazu genutzt um die Position des Roboters, die Zielposition der Pfadplanung und die Position des Manual Steering Obstacle Robot zu visualisieren. Dafür benötigt das Plugin lediglich einen $StateContainer$ vom Typ $RobotPose$. Diese $RobotPose$ ist wie beim Prä–Prozessor beschrieben eine Erweiterung der $Pose$–Klasse um einen Radius und einen Odometer–Wert. Der Radius wird im Plugin dann genutzt, um die Größe des Roboters in der Testumgebung darzustellen. Wie dieses Plugin im Einsatz aussieht, kann man in Abbildung 4.8 sehen. Diese zeigt durch den Kreis die Position und durch den eingezeichneten Radius die Orientierung des Roboters an.

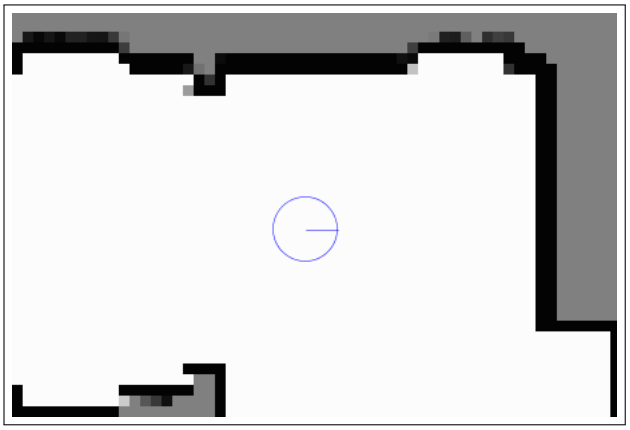

Abbildung 4.8: Vom $RobotPosePlugin$ visualisierte $RobotPose$.

Der $CommunicationPositionProcessor$ liefert nun aber nur eine $Pose$. Dadurch wird der Post–Processor notwendig, welcher diese $Pose$ in eine $RobotPose$ umwandelt. Dazu müssen der $Pose$ ein Radius und ein Odometerwert beigefügt werden.
Da die $RobotPose$ nur zur Visualisierung genutzt wird, kann der Odometerwert ignoriert werden. Er wird einfach dauerhaft mit 0 belegt. Der Radius wird auf 0,3 m festgelegt. Das ist etwas kleiner als der des simulierten Roboters, wurde aber gewählt, um diese auch in der Größe unterscheiden zu können.
Da in dem Post–Prozessor eigentlich keine Berechnungen stattfinden und in diesem nur Daten aus einem Container in einen anderen geschrieben werden müssen, kann man einen $StateObserver$ nutzen.
Die Klasse besitzt also nur zwei Container, je einen für die $Pose$ und die $RobotPose$. Bei der Initialisierung durch den Konstruktor müssen ihr diese mit übergeben werden. Im Konstruktor muss diese Instanz dann dem $StateContainer$ mit der $Pose$ als Observer zugeordnet werden.
Da diese Klasse vom $StateObserver$ abgeleitet wird, enthält sie eine $stateChanged$–

Methode, welche jedes mal ausgelöst wird, wenn ein neuer Wert in den Container geschrieben wird. In dieser Methode werden dann eine neue $RobotPose$ aus der $Pose$ und den oben festgelegten Werten erstellt und diese in den dafür bestimmten Container geschrieben.

4.8 INTEGRATION IN DIE TESTUMGEBUNG

Die zuvor beschrieben Module können nun in die Testumgebung integriert werden. Dazu müssen ein paar Dinge beachtet werden.
Konkret müssen in der Testumgebung, wie sie für Pfadplanung genutzt wird, ein paar Änderungen vorgenommen werden, bzw. Dinge hinzugefügt werden.
Diese Änderungen erfolgen in zwei verschiedenen Klassen. Zum einen müssen in der $BasicDynamicNavigationRobot$ folgende Anpassungen vorgenommen werden:

- Falls noch kein MSOR existiert, muss die $populateEnviroment$–Methode genutzt werden, um einen hinzuzufügen.
- Es müssen $StateContainer$ von den Typen $PositiveHypothesis$, $Pose$, $RobotPose$ und $CommunicationMode$ initialisiert werden.
- Es müssen der Prä–Prozessor, das eigentliche Modul und der Post–Prozessor initialisiert werden.
- Es werden noch zwei get–Methoden für die $StateContainer$ mit dem $CommunicationMode$ und der $RobotPose$ benötigt.

In der $StartBDNRobot$ müssen dann noch die beiden Plugins hinzugefügt werden. Das sind zum einen das Plugin, über welches der $CommunicationMode$ eingestellt werden kann, als auch das Plugin, welches die RobotPose anzeigt. Die Container bekommen beide Plugins über die zuvor erstellten get–Methoden.
Danach ist das Modul in der Testumgebung nutzbar. Wie dies genau aussieht, kann man in Abbildung A.1 im Anhang sehen. Die blaue $RobotPose$ ist die ermittelte Position, die grüne der MSOR. Links sind die Plugins zur Steuerung des MSOR und zum Einstellen des Modus.
Die Integration in den Roboter erfolgt unter Nutzung der Container für die Hypothese, den Modus, die Karte und die Position als Schnittstellen zum Modul.

5 ERGEBNISSE

An diesem Punkt sollen die Ergebnisse dieser Arbeit betrachtet und ausgewertet werden. Dazu wird zuerst ein Überblick über das Erreichte gegeben.
Als Ergebnis wurde ein Modul entwickelt, welches in der Lage ist, anhand einer ihm gelieferten Position einer Hypothese die für diese Situation ideale Position zu ermitteln, um den Roboter für eine Interaktion zu positionieren. Dazu wurden theoretische Grundlagen geschaffen, was bei der Berechnung der Position von Bedeutung ist und diese an das geplante Einsatzszenario angepasst. Es wurden Sicherheitsmechanismen integriert, welche dafür sorgen, dass immer eine Position gefunden werden kann. Das Modul wurde so konstruiert, dass es über klar definierte Schnittstellen im bestehenden Roboter–Projekt integriert werden kann. Außerdem wurde auch ein Gerüst geschaffen, mit welchem es möglich ist, das Modul in einer simulierten Umgebung zu integrieren.
Das Modul wurde so konstruiert, dass es die im Lösungsweg genannten Bedingungen erfüllt. Der Nachweis dafür wird durch die Betrachtung und Auswertung der Test–Ergebnisse erbracht.
Um die Ergebnisse besser veranschaulichen zu können, werden dabei die Daten der Bewegung und die der errechneten Position dokumentiert. Mit Hilfe dieser Daten wurden dann Diagramme erstellt, in denen der Verlauf der Positionen dokumentiert ist. Bei den ersten drei Diagrammen wurde die Orientierung an ausgewählten Punkten durch Vektoren veranschaulicht, damit es möglich ist, zu erkennen, dass die errechnete Position immer auf die der Hypothese orientiert ist. Im Hintergrund ist immer die Karte zu sehen, auf welcher die Tests durchgeführt wurden. Im Ablauf kann es dadurch, dass der zeitliche Ablauf nicht mit dargestellt werden kann, so wirken, als ob die Position des Ziels und die errechnete Position sich überschneiden, da sich die Bahne kreuzen. Dies wirkt aber nur so, da die Punkte zu unterschiedlichen Zeiten erreicht werden.
Beim Konfrontieren sollte die Position immer im Sichtfeld der Person liegen und gleichzeitig nah genug an ihr, um eine Kommunikation zu ermöglichen. Aufgrund der implementierten Trägheit liegt der Abstand zwischen Person und Hypothese immer zwischen 1,05 und 1,35 Metern. Dies ist nach [Hall, 1966] eine Distanz, in

welcher Kommunikationen stattfinden können und welche sozial verträglich ist. Um dies zu verdeutlichen, wurden zwei Tests vorgenommen. Dabei konnte gezeigt werden, dass die Berechnung wie gewünscht funktioniert. Die Ergebnisse kann man in Abbildung 5.1 betrachten. Links dargestellt ist eine Fahrt mit dem Manual Steering Obstacle Robot im Modus „Konfrontieren“. Die Fahrt begann an der Position $(-12, 0)$. Rechts ist das Drehen des MSOR um die eigene Achse und die errechnete Position dazu dokumentiert. Der MSOR ist bei $(-12, 0)$ gestartet.

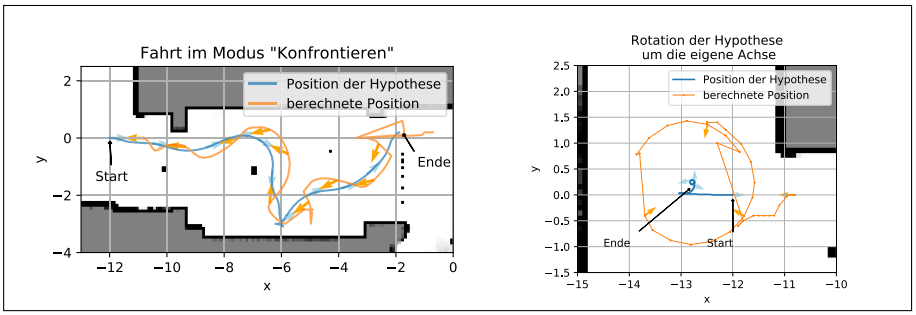

Abbildung 5.1: Dokumentation von Fahrten mit dem MSOR im Modus „Konfrontieren“

Es wurde zuerst eine Bewegung des Ziels durch die Karte simuliert. Dabei war der Modus „Konfrontieren“ aktiviert, was heißt, dass die Position immer im Sichtfeld der Person liegen sollte. Das positive Ergebnis kann man auf der linken Seite anhand des ähnlichen, aber leicht versetzten Verlaufs der beiden Linien erkennen. Größere Sprünge bei der Position können dabei auf das Eingreifen der Sicherheitsmechanismen zurückgeführt werden.
Auf der rechten Seite der Abbildung sieht man, wie die errechnete Position sich um den Roboter dreht. Dazu hat sich dieser um die eigene Achse gedreht, so dass diese kreisförmige Anordnung der errechneten Positionen entstanden ist.
Beim „Folgen“ sollte der Roboter immer hinter dem Ziel platziert sein und der Abstand bei größeren Geschwindigkeiten größer werden. Aufgrund der Trägheit kann der Abstand bei den getroffenen Einstellungen, abhängig von der Geschwindigkeit, bei 1,25 m bis 2,55 m liegen. Wobei der Abstand auch noch größer werden kann, wenn die eingestellte Maximalgeschwindigkeit von der Hypothese überschritten wird. Der Minimal–Abstand wird erreicht, wenn sich die Hypothese nicht bewegt. Außerdem soll die Position immer auf der entgegengesetzten Seite zur Bewegung der Hypothese liegen. Wie dies konkret aussieht, kann man in Abbildung 5.2 sehen.
Hier ist eine Fahrt des MSOR und die errechnete Position im Modus „Folgen“ zu sehen. Dabei sind die Sprünge in der Positionierung gut zu erkennen, wenn sich das Ziel rückwärts bewegt. Der Startpunkt liegt bei $(-12, 0)$. Dabei bewegt

sich das Ziel einmal quer durch den Flur und auch auf Hindernisse zu. Es wurde dennoch in jeder Situation eine Position gefunden. Die Sprünge in der errechneten Position lassen sich wieder mit dem Eingreifen der Sicherheitsmechanismen erklären, welche notwendig werden, wenn eine Position ungültig ist. Im Bereich zwischen den X–Koordinaten von -14 bis -12 ist auch ein Problem erkennbar, welches später noch einmal genauer thematisiert wird.

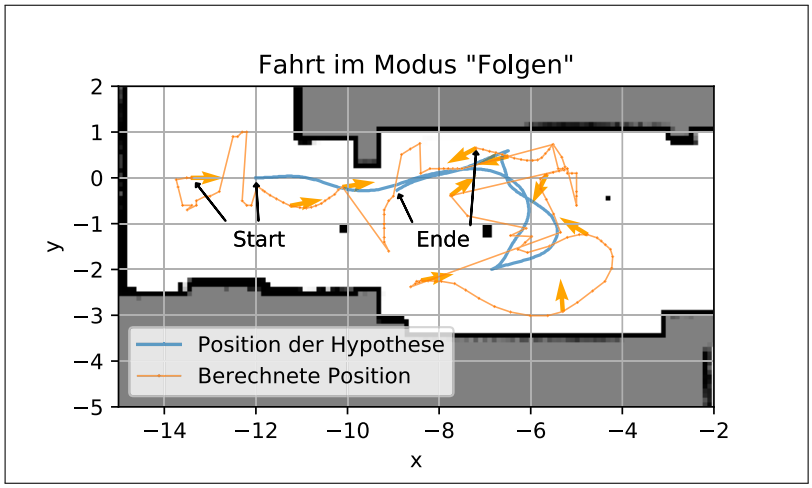

Abbildung 5.2: Dokumentation einer Fahrt mit dem MSOR im Modus „Folgen“

Es sollte auch möglich sein, den Modus während des Ablaufs zu wechseln. Dabei erkennt man dann einen Sprung bei der Position, da die berechnete Position ab dem Zeitpunkt andere Bedingungen erfüllen muss. Wie genau das aussieht, kann man in Abbildung 5.3 sehen. Links ist der Wechsel von „Konfrontieren“ zu „Folgen“ dargestellt – rechts der entsprechende Wechsel von „Folgen“ zu „Konfrontieren“.

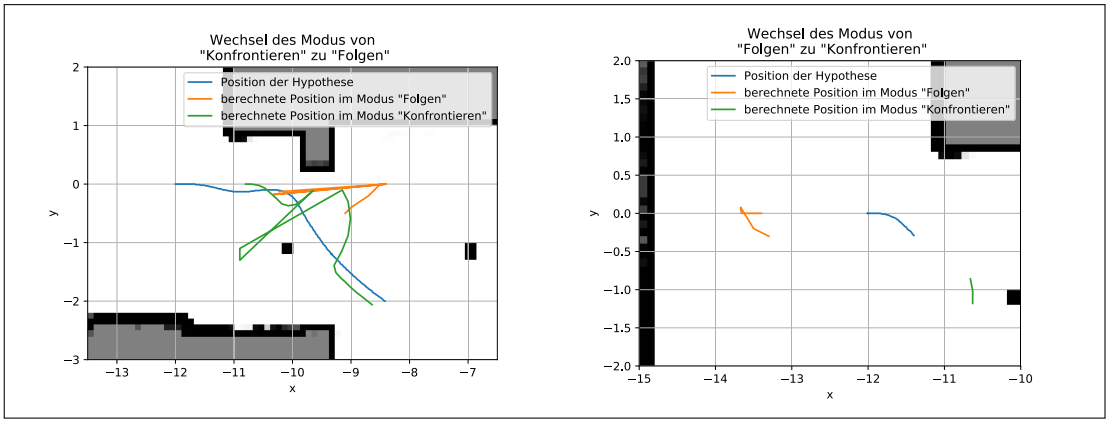

Abbildung 5.3: Dokumentation von Fahrten mit dem MSOR mit Wechsel zwischen den Modi

In den bisherigen Diagrammen wurde schon öfter von sogenannten Sprüngen

berichtet, wenn die Alternativpunktsuche einsetzt. Daher wurde noch einmal extra in einem Test Alternativpunktberechnungen provoziert, um besser zeigen zu können, was damit gemeint ist. Die Visualisierung dieser Daten ist in Abbildung 5.4 zu sehen. Hier wurden gezielt Alternativpunktberechnungen provoziert, um zeigen zu können, wie sich die Position verändert. Die resultierenden Sprünge sind dann mit Zahlen markiert.

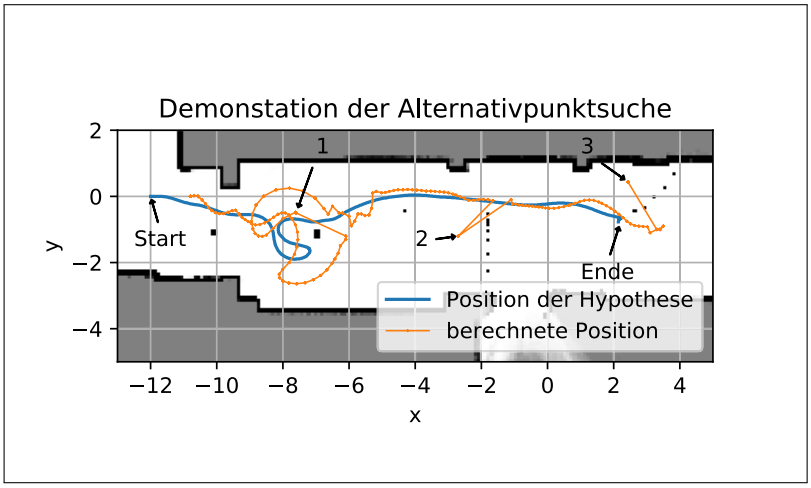

Abbildung 5.4: Demonstration der Alternativpunktsuche

Dabei werden die größeren Abweichungen dadurch hervorgerufen, dass sichergestellt werden soll, dass sich der Roboter dort platzieren kann. Aus diesem Grund wird eine Karte genutzt, in welcher die Hindernisse weiter aufgebläht werden. Deswegen kann auch schon ein Alternativpunkt berechnet werden müssen, wenn die Position noch nicht genau auf einem Hindernis liegt. Dies sieht man gut an der mit 2 markierten Position.
Bei den in Abbildung 5.2 visualisierten Daten wurde erwähnt, dass dabei ein Problem sichtbar wird. Dieses Problem hat aber nichts mit dem Modul an sich, sondern mit der Karte zu tun, die es nutzt. Konkret kann es beim Fusionieren der lokalen Sensorkarten und der statischen Karte dazu kommen, dass ein sich bewegendes Objekt einen Schleier hinter sich herzieht, welcher noch als blockiert gilt, obwohl der Bereich wieder frei ist. Dies kann an den Grenzen der lokalen Karten passieren, also im Bereich der gerade noch so vom Sensor erfasst wird, wenn der Wert in der lokalen Karte nicht sicher genug ist, um zu sagen, dass das Feld frei ist. Dadurch kann dann, wenn sich die Hypothese in diesem Bereich bewegt, die Alternativpunktsuche häufiger als nötig ausgelöst werden. Weitere Testdaten, die dieses Problem zeigen, kann man in Abbildung A.2 im Anhang sehen.
Um zu zeigen, dass das Modul auch der letzten Anforderung, nämlich der eines

möglichst geringen Rechenaufwandes, gerecht wird, wurden auf einer längeren Fahrt alle Daten dokumentiert, also ob wegen der Trägheit eingegriffen wurde oder Alternativpunkte gesucht werden mussten.
Dazu wurde auf der Fahrt bei jedem Aufruf der $onData$–Methode, also bei jedem neuem Stand im Hypothesen–Container, dokumentiert, in wieweit Berechnungen notwendig waren. Das Ergebnis dieses Tests kann man in Abbildung 5.5 sehen. Links ist der Pfad dargestellt, der in diesem Test befahren wurde. Dabei wurden die Punkte, an denen der Modus gewechselt wurde, markiert. Rechts ist das Diagramm angegeben, das die Häufigkeit der verschiedenen Berechnungsstufen verdeutlicht.

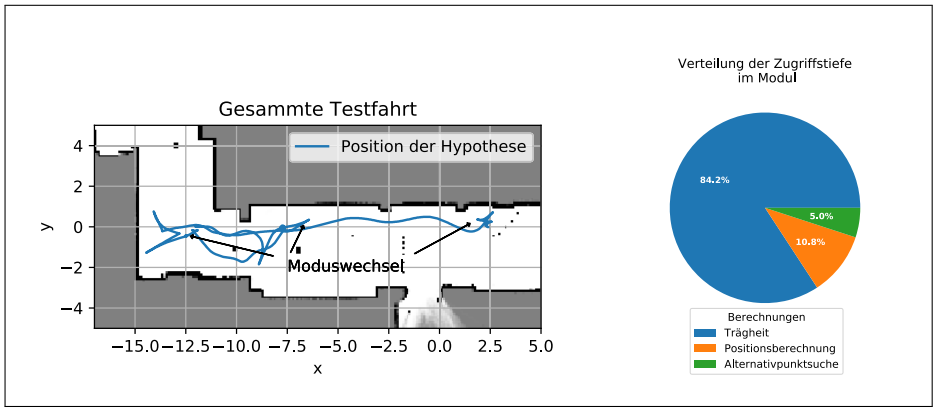

Abbildung 5.5: Verteilung der Zugriffstiefe im Modul während einer Testfahrt

Dabei wurden in den Hypothesen–Container insgesamt 2 610 Werte eingetragen. Diese wurden dann vom Modul verarbeitet. Es wurde bei 2 198 Fällen festgestellt, dass keine weiteren Berechnungen notwendig sind, da die Abweichung von der vorherigen Position nicht groß genug ist. In 281 Fällen hat eine Neuberechnung der Position genügt und in 131 Fällen war es notwendig einen Alternativpunkt zu suchen. Es war nicht einmal notwendig, auf die zweite Suche für den „worst case“ zurückzugreifen, was notwendig wird, wenn die Alternativpunktsuche nicht erfolgreich war. Somit kann man ableiten, dass das Modul nur Berechnungen tätigt, wenn diese notwendig sind. Das ergibt sich daraus, dass zum einen genug Punkte geliefert werden, um dem Ziel zu folgen und gleichzeitig aber sehr viele Berechnungen eingespart werden, welche nicht notwendig sind.
Diese Werte können natürlich von Durchlauf zu Durchlauf variieren, je nachdem wie sich die Hypothese bewegt und die Karte geschaffen ist.

6 AUSBLICK

In diesem Kapitel geht es darum, wie die Inhalte dieser Arbeit in Zukunft weiterentwickelt und genutzt werden können.
Eine der ersten Aufgaben die realisiert werden müssten, wäre das Modul an seinem Bestimmungsort im Roboter zu integrieren. Danach kann man testen, inwiefern noch Parameter des Moduls für Praxiseinsätze angepasst werden müssen. Die Positionen, die dieses Modul dann liefert, können von der Navigation des Roboters genutzt werden, um sich einer Hypothese zu nähern.
Außerdem kann noch geprüft werden, ob das Modul auch für andere Situationen nutzbar ist. Nach jetzigem Stand wurde es nur so entwickelt, dass es in einem Bereich, welcher vom Roboter wahrgenommen werden kann, fehlerfrei funktioniert. Dies hängt damit zusammen, dass der Roboter nur Hypothesen ansteuern kann, die er wahrnimmt. Nun gibt es aber die Überlegung dem Roboter eine Rufen–Funktion zu geben, über welche man ihn zu seiner eigenen Position dirigieren kann. Dieses Rufen funktioniert beispielsweise über ein mobiles Client–Gerät, welches mit dem Roboter verbunden ist und von den Pflegekräften genutzt wird. In diesem Fall könnte man beispielsweise auch das Modul nutzen, um eine Zielposition für den Roboter zu ermitteln.
Auch können im Zusammenhang mit dem Projekt entwickelte Elemente weiterführend an anderen Stellen genutzt werden. So kann man die Wanddetektion überall verwenden um Hindernisse zwischen zwei Punkten zu suchen.
Ferner steht auch ein Praxistest des Roboters in der Einrichtung an. In diesem Test kann das Modul dann auch genutzt werden, um Personen auf den Fluren anzusteuern.
Im Anschluss daran kann noch getestet werden das Modul in anderen Projekten der Forschungsgruppe zu nutzen. Dafür würde sich beispielsweise das Projekt mit dem Roboter–Museumsführer anbieten, welcher von [Poschmann et al., 2012] beschrieben wird.

LITERATUR

Bahrmann, F., Hellbach, S. & Böhme, H.-J. (2016). A Fuzzy-based Adaptive Environment Model for Indoor Robot Localization. In *Telehealth and Assistive Technology / 847: Intelligent Systems and Robotics*.

Bahrmann, F., Hellbach, S., Keil, S. & Böhme, H.-J. (2014). Understanding Dynamic Environments with Fuzzy Perception. In *International Conference on Neural Information Processing* (S. 553–562). Springer.

Bresenham, J. E. (1965). Algorithm for computer control of a digital plotter. *IBM Systems journal*, *4*(1), 25–30.

Broadbent, E., Stafford, R. & MacDonald, B. (2009). Acceptance of healthcare robots for the older population: Review and future directions. *International journal of social robotics*, *1*(4), 319.

Bundesamt, S. (2017). *2017 Pflegestatistik 2015 Pflege im Rahmen der Pflegeversicherung Deutschlandergebnisse*. Statistisches Bundesamt. Zugriff unter https://www.destatis.de/DE/Publikationen/Thematisch/Gesundheit/Pflege/PflegeDeutschlandergebnisse5224001159004.pdf?__blob=publicationFile

Eisenbach, M., Vorndran, A., Sorge, S. & Gross, H.-M. (2015). User Recognition for Guiding and Following People with a Mobile Robot in a Clinical Environment. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)* (S. 3600–3607). IEEE.

Eubel, C. & Heine, H. (2013). Pflegenotstand in Deutschland: Auf Personalsuche in Asien. *Online: http://www. tagesspiegel. de/politik/pflegenotstand-in–deutschland-auf-personalsuche-in-asien/8011390. html*.

Gross, H.-M., Schroeter, C., Mueller, S., Volkhardt, M., Einhorn, E., Bley, A., … Merten, M. (2011). I'll keep an eye on you: Home robot companion for elderly people with cognitive impairment. In *Systems, Man, and Cybernetics (SMC), 2011 IEEE International Conference on* (S. 2481–2488). IEEE.

Gross, H.-M. [Horst-Michael], Meyer, S., Scheidig, A., Eisenbach, M., Mueller, S., Trinh, T. Q., … Fricke, C. (2017). Mobile robot companion for walking training of stroke patients in clinical post-stroke rehabilitation. In *Robotics*

and Automation (ICRA), 2017 IEEE International Conference on (S. 1028–1035). IEEE.

Hall, E. T. (1966). *The hidden dimension*. Garden City, NY: Doubleday.

Hebesberger, D., Körtner, T., Pripfl, J., Gisinger, C., Hanheide, M. et al. (2015). What do staff in eldercare want a robot for? An assessment of potential tasks and user requirements for a long-term deployment.

Kessler, J., Schmidt, M., Helsper, S. & Gross, H.-M. (2013). I'm Still Watching You: Update on Observing a Person in a Home Environment. In *Europ. Conf. on Mobile Robots (ECMR)*.

Kitano, N. (2006). A comparative analysis: Social acceptance of robots between the West and Japan. *EURON Atelier on Roboethics*.

Lischke, F., Bahrmann, F., Hellbach, S. & Böhme, H.-J. (2017). RoNiSCo: Robotic Night Shift Companion. *Machine Learning Reports*, (03), 26–35.

Poschmann, P., Donner, M., Bahrmann, F., Rudolph, M., Fonfara, J., Hellbach, S. & Böhme, H.-J. (2012). Wizard of Oz revisited: Researching on a tour guide robot while being faced with the public. In *RO-MAN, 2012 IEEE* (S. 701–706). IEEE.

Stalinski, S. (2017). Pflegenotstand in Deutschland Überlastet ausgebrannt - und weg. *Online: https://www.tagesschau.de/inland/pflege-notstand-101.html*.

Vercelli, A., Rainero, I., Ciferri, L., Boido, M. & Pirri, F. (2018). Robots in Elderly Care. *DigitCult-Scientific Journal on Digital Cultures*, *2*(2), 37–50.

Volkhardt, M. & Gross, H.-M. [Horst-Michael]. (2013). Finding People in Home Environments with a Mobile Robot. In *Europ. Conf. on Mobile Robots (ECMR)*.

ANLAGEN

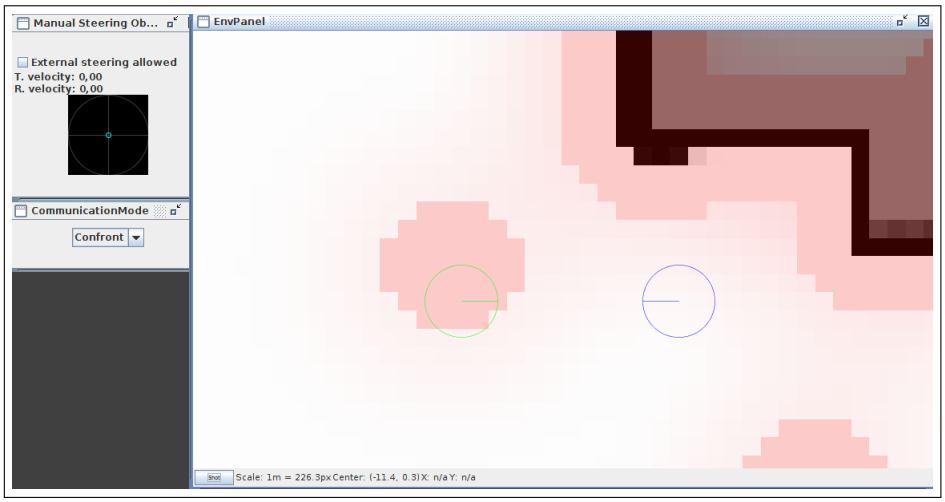

Abbildung A.1: Testumgebung mit dem integrierten Modul.

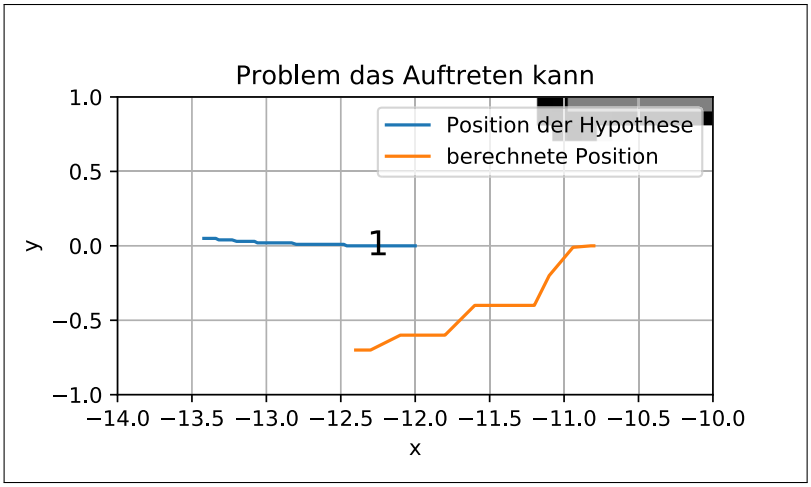

Abbildung A.2: Alternativpunktberechnung, die durch Wahrnehmungsprobleme des Roboters ausgelöst werden

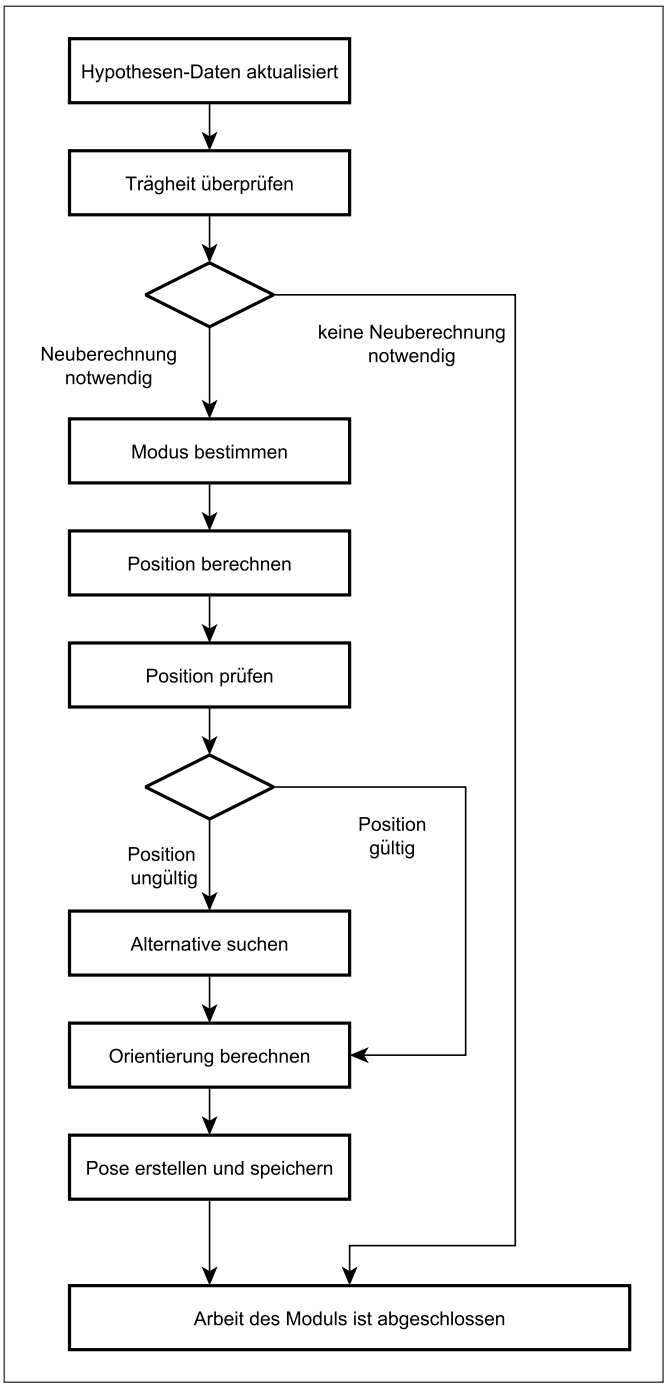

Abbildung A.3: Ablauf der Berechnung der Position für den Roboter.

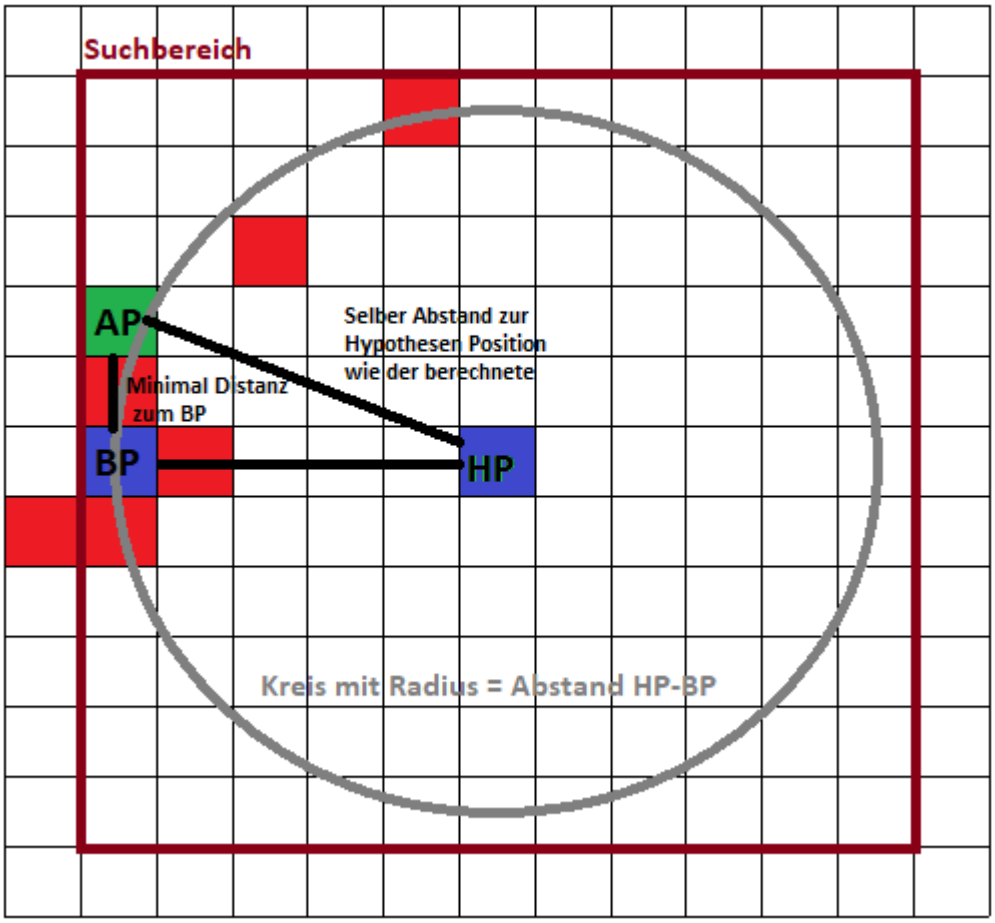

Abbildung A.4: Skizze der Alternativpunktsuche

THESEN

- Es ist möglich eine Position zur Platzierung eines Roboters in Bezug auf eine Position dynamisch zu ermitteln.
- Es ist möglich eine sozialverträgliche Position zu ermitteln, an welcher sich ein Roboter in Bezug auf eine Person positionieren sollte.

SELBSTSTÄNDIGKEITSERKLÄRUNG

Ich versichere hiermit, dass ich die Bachelor-Arbeit mit dem Titel

„Ein Roboter–Nachtwächter zur Unterstützung von Pflegekräften“

selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Dresden,

Jan Brose