Institut für Parallele und Verteilte Systeme
Maschinelles Lernen und Robotik

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit Nr. 233

Nicht-Markov Feature Entdeckung für echte Roboter

Kai Kälberer

Studiengang:	Softwaretechnik
Prüfer/in:	Prof. Dr. rer. nat. Marc Toussaint
Betreuer/in:	M. Sc. Stefan Otte Dipl.-Phys. Robert Lieck
Beginn am:	11. Mai 2015
Beendet am:	10. November 2015
CR-Nummer:	I.2.9

Kurzfassung

In der vorliegenden Bacheloarbeit soll mit Hilfe des PULSE-Algorithmus [LT15] ein prädikatives Umgebungsmodell in einer realen Umgebung erlernt werden. Dies wird mit Hilfe des PR2 von Willow Garage, der sich in einem Raum ohne jeglicher Vorkenntnisse mit verschiedenen Knöpfen und einer verschlossenen Tür befindet, umgesetzt werden. Er sollte hierzu erlernen können, durch welchen Knopfdruck er die Tür für wie viele Zeitschritte öffnen kann.

Zu Beginn wird in das Thema eingeleitet und einige verwandte Arbeiten beschrieben. Darauf folgen die theoretischen Grundlagen sowie die Funktionsweise des Algorithmus.

Im Anschluss wird der Versuchsaufbau, der verwendete Roboter und dessen Betriebssystem sowie verschiedene verwendete Module dargestellt.

Im praktischen Teil der Arbeit wird mit Hilfe einer Simulation der realen Umgebung ein Vergleich verschiedener Strategien durchgeführt. Bei diesem wird gezeigt, dass der Algorithmus in einer echten Umgebung tatsächlich angewendet werden kann. Zugleich wird die Strategie, die die besten Ergebnisse liefert bestimmt.

Danach folgt die Umsetzung auf dem PR2. Die verschiedenen Anforderungen, die der PR2 erfüllen muss werden beschrieben und deren Umsetzung erklärt.

Eine kurze Zusammenfassung aller Ergebnisse sowie ein Ausblick über mögliche Weiterentwicklungen schließen die Arbeit ab.

Inhaltsverzeichnis

1 Einleitung **1**
- 1.1 Motivation . . . 1
- 1.2 Ziel der Arbeit . . . 2
- 1.3 Gliederung . . . 2
- 1.4 Verwandte Arbeiten . . . 3

2 Theoretischer Hintergrund **5**
- 2.1 Reinforcement Learning . . . 5
- 2.2 Markov Prozesse . . . 7
  - 2.2.1 Markov Eigenschaft . . . 7
  - 2.2.2 Markov Kette . . . 8
  - 2.2.3 Markov Decision Process . . . 8
  - 2.2.4 Partially Observable Markov Decision Process . . . 9
- 2.3 TD Learning . . . 11
- 2.4 Regularization . . . 11
- 2.5 Feature Expansion . . . 11
- 2.6 Vorstellung PULSE Algorithmus . . . 12
  - 2.6.1 Temporally extended features . . . 12
  - 2.6.2 Der Algorithmus . . . 12

3 Hintergrund zur praktischen Umsetzung **15**
- 3.1 Versuchsaufbau . . . 15
- 3.2 PR2 . . . 15
- 3.3 Robot Operating System . . . 16
  - 3.3.1 Über ROS . . . 16
  - 3.3.2 Funktionsweise . . . 17
  - 3.3.3 Paketstruktur . . . 18
- 3.4 Verwendete ROS Module . . . 19
  - 3.4.1 ar_track_alvar . . . 19
  - 3.4.2 pr2_2dnav . . . 19
  - 3.4.3 Motion Generation . . . 20

4 Simulation **21**
4.1 Simulierte Welt . . . 21
4.1.1 Strategien . . . 22
4.2 Evaluation . . . 23
4.2.1 Qualitative Analyse . . . 23
4.2.2 Quantitative Analyse . . . 26

5 Umsetzung auf dem PR2 **31**
5.1 Anforderungen . . . 31
5.2 Umsetzung . . . 31
5.2.1 Perzeption . . . 31
5.2.2 Navigation . . . 32
5.2.3 Drücken der Knöpfe . . . 32
5.2.4 Tür öffnen . . . 33
5.2.5 Ablauf . . . 34
5.2.6 Probleme mit dem PR2 . . . 34

6 Zusammenfassung und Ausblick **37**

Literaturverzeichnis **43**

1 Einleitung

1.1 Motivation

Roboter gewannen in der Vergangenheit, vor allem in der Industrie immer mehr an Bedeutung. Mit ihnen konnten viele Arbeitsschritte sehr vereinfacht und beschleunigt werden. Bisher beschränkt sich ihr Einsatz jedoch meistens auf fest definierte, immer gleich bleibende Aufgaben, sodass die Roboter nur sehr spezifisch eingesetzt werden können. Sie handeln zwar in gewisser Weise auch autonom, können ihre Aufgabe aber meist nur sehr begrenzt, je nach Sensorinformationen, variieren und erfordern trotzdem immer wieder ein Eingreifen durch Menschen.

Im Bereich der Personal Robotics ist das oben beschriebene „autonome“ Handeln jedoch meist nicht ausreichend. Hier sollen die Roboter möglichst viele Dinge autonom und komplett ohne das Eingreifen von Menschen machen können. Es ist also meist notwendig, dass die Roboter „lernen“ können um somit den Bedürfnissen der Benutzer zu entsprechen. So sollen sie zum Beispiel nicht nur in einer bestimmten Umgebung funktionieren, sondern bei jedem Endnutzer auf gleiche Weise. Somit ist es notwendig, dass sie frei in einer unbekannten Umgebung navigieren können und dabei Erfahrungen sammeln, um den verschiedenen Aufgaben, welche ihnen gestellt werden, nachkommen zu können. Außerdem müssen sie in der Lage sein Dinge in Relation setzen zu können um „komplexe“ Aufgaben zu lösen. Nehmen wir hier als Beispiel den Auftrag, dem Benutzer ein kaltes Getränk zu bringen. Dafür muss der Roboter in der Lage sein zum Kühlschrank zu navigieren, diesen zu öffnen, ein Getränk beziehungsweise das gewünschte Getränk zu erkennen, dieses zu greifen, die Tür zu schließen, mit dem Getränk zurück zum Nutzer zu navigieren und ihm dieses dann zu übergeben. Es werden für diese, für einen Menschen, triviale Aufgabe schon sehr viele Dinge von einem Roboter verlangt. Er soll sie jedoch nicht nur in einer Umgebung ausführen können, sondern sogar in vielen verschiedenen. Hierzu muss der Roboter also den Weg zum Kühlschrank erkennen, das Öffnen der Tür, das Identifizieren des gewollten Getränkes, das richtige Greifen sowie das Schließen der Tür beherrschen und den Rückweg finden.

Deshalb wird es immer wichtiger eine Möglichkeit zu finden, wie Roboter selbständig ein Weltmodell erlernen können um den Aufgaben, welche sie erfüllen müssen gerecht zu werden, um von der sehr beschränkten, durch fest programmierte Abläufe erzeugten, Einsetzbarkeit weg zu kommen und ein wirklich autonomes Handeln voranzutreiben.

1.2 Ziel der Arbeit

In dieser Arbeit soll mit Hilfe des PR2, einem Roboter, welcher mit ROS (Robot Operating System) funktioniert, eine Möglichkeit zum Erlernen eines Weltmodells getestet werden. Hierzu sollen mit Hilfe der Sensoren und Aktuatoren des PR2 zuerst Daten gesammelt werden, diese auf eine, für den PULSE-Algorithmus[LT15], geeignete Abstraktionsebene gebracht werden, um sie danach vom Algorithmus auswerten zu lassen und damit ein Weltmodell der Roboterumgebung zu finden. Des Weiteren sollen verschiedene Strategien zur Wahl der nächsten Aktion getestet werden, um die Beste dieser Strategien zu bestimmen. Dafür wird zuerst eine Simulation durchgeführt, um mit ihr die vielversprechendsten Strategien zu bestimmen, welche dann auf dem PR2 in der realen Welt getestet werden sollen. Der Versuchsaufbau ist hierbei ein relativ einfacher: Der Roboter befindet sich in einem kleinen Raum, in welchem sich einige Knöpfe und eine Tür befinden. Er soll in diesem, ihm unbekannten Raum, nun ein möglichst korrektes Weltmodell zum Öffnen der Tür mit Hilfe von PULSE[LT15] ermitteln.

1.3 Gliederung

Die Arbeit ist folgendermaßen gegliedert:

Kapitel 2 – Theoretischer Hintergrund: Hier werden zunächst theoretische Grundlagen, die zum Verständnis des PULSE-Algorithmus[LT15] nötig sind, erarbeitet und anschließend kurz die Funktion des Algorithmus beschrieben.

Kapitel 3 – Hintergrund zur praktischen Umsetzung: In diesem Kapitel werden Hintergründe für die praktische Umsetzung aufgezeigt. Es wird kurz auf das Roboter Operating System (ROS) eingegangen. Der PR2 und die verwendbaren Sensoren/Aktuatoren werden beschrieben. Außerdem werden die Wichtigsten verwendeten Pakete kurz erläutert.

Kapitel 4 – Simulation: Hier wird die Simulierte Welt vorgestellt, welche zur Simulation verwendet wird. Außerdem werden die getesteten Strategien evaluiert.

Kapitel 5 – Umsetzung auf dem PR2: In diesem Kapitel wird die praktische Umsetzung beschrieben und dokumentiert sowie die gesammelten Ergebnisse evaluiert.

Kapitel 6 – Zusammenfassung und Ausblick Dieses Kapitel fasst die Ergebnisse der Arbeit zusammen und stellt Anknüpfungspunkte vor.

1.4 Verwandte Arbeiten

Das Rahmenwerk der Partially Observable Markov Decision Processes (POMDP) wurde in der Forschung, beziehungsweise in der praktischen Anwendung der Robotik, schon oft verwendet. Ein großes Problem dabei ist in der Regel, dass für große Zustands- beziehungsweise Aktionsmengen die Rechenzeit zu lange für den Einsatz auf echten Robotern ist. Deshalb verwenden viele der Ansätze Methoden, um die Menge der Zustände/Aktionen im aktuellen Moment in Untermengen zu teilen, mit dem Ziel die Problemgröße zu verringern.

In [PMP+03] wurde dieser Ansatz beispielsweise verfolgt um einen Roboter zur Assistenz in der Krankenpflege zu programmieren. Dieser soll anhand von seinen Sensordaten während der Interaktion mit Menschen die richtigen Entscheidungen treffen, um diese möglichst zuverlässig bei alltäglichen Dingen zu unterstützen sowie Krankenpflegern zu assistieren. Da für einen Assistenzroboter viele der Aktionen in Echzeit gewählt werden müssen, wurde eine Hierarchie zum Rahmenwerk hinzugefügt. Diese soll die große Aktionsmenge in kleinere Teilmengen zerteilen, um somit für bestimmte Situationen nur ein POMDP mit kleinerem Aktionsraum lösen zu müssen. Dies führt zur schnelleren Berechenbarkeit und deshalb auch zur schnelleren Auswahl einer Aktion .

In [SV04] wurde das Lösen von POMDP's zur Roboterplanung verwendet. Hierbei wurden die „belief states“ in Teilmengen unterteilt, um das Lösen der Probleme zu beschleunigen. Der entwickelte Algorithmus wurde dann in einer simulierten Robotikumgebung getestet, in welcher er in einer ihm unbekannten Umgebung Briefe ausliefern soll.

In [VT15] wurde mit Hilfe von Reinforcement Learning in Verbindung mit POMDP's ein Planungsalgorithmus zur Objektmanipulation entwickelt. Dieser sollte ausschließlich anhand des Force Feedbacks lernen. Es wird anstatt alle möglichen Lösungen zu simulieren, so wie es bei den meisten Lösungsansätzen der POMDP's üblich ist, die zeitaufwändige Simulation durch Trajektorienoptimierung ersetzt. Zugleich wird aber auch wieder nur eine Teilmenge der POMDP's in jedem Schritt betrachtet.

Ein anderer Algorithmus, welcher auch auf typischen Robotikanwendungen getestet wurde, wird in [KHL08] beschrieben. Der sogenannte SARSOP-Algorithmus berechnet „optimally reachable belief states“, um das Lösen von POMDP's zu beschleunigen. Der Ansatz wurde auf Navigations-, Greif- und Explorationsproblemen getestet, wie sie auch in der Robotik üblich sind.

2 Theoretischer Hintergrund

2.1 Reinforcement Learning

Um Wissen zu erlangen, muss der Mensch lernen. Hierzu werden uns teilweise Sachen von Anderen gezeigt oder wir schauen Anderen zu und reproduzieren ihr Verhalten. Wenn jedoch niemand da ist um uns zu lehren, bleibt selbst Menschen meist nichts anderes übrig, als Dinge auszuprobieren um dann anhand des Beobachteten selbst Schlüsse zu ziehen, was wir durch unser Handeln verursacht haben und ob es uns weiter bringt oder nicht.

Im Prinzip verfolgt „Reinforcement Learning“ genau diesen Ansatz:

Dem Lernenden wird nicht explizit gesagt, was er als nächstes tun soll. Viel mehr soll er durch sein Handeln probieren, mit welchen Aktionen er einen möglichst hohen „Gewinn“ erzielen kann. Er soll also herausfinden, durch welche Aktion er seinen Reward maximieren kann. Dieser Reward wird meist als eine Zahl dargestellt, wobei zum Beispiel 1 einen geringen Reward darstellt und 10 eventuell das Maximum. Auch negative Rewards sind als „Bestrafung“ für schlechtes Handeln möglich. Da dieser Reward auf lange Sicht maximiert werden soll, ist es nicht immer sinnvoll in jedem Schritt die Aktion zu wählen, welche den momentan besten Reward verspricht. Der langfristige Reward muss im Auge behalten werden.

Beim „Reinforcement Learning“ steht nicht die Lernmethode im Vordergrund, sondern viel mehr das Finden des Lernproblems. Die zentrale Idee ist also, die wichtigsten Aspekte des Problems zu finden und anhand von diesen, mit Hilfe der Umgebung, ein Ziel zu erreichen. Dazu muss der Lernende seine Umgebung wahrnehmen, Aktionen ausführen und Ziele in der Umgebung haben. In diesem konkreten Beispiel muss der PR2 mit Hilfe seiner Sensoren den Raum wahrnehmen, mit seinen Aktuatoren Knöpfe drücken, Türen öffnen und als Ziel die Aktionen finden, welche die Tür entriegeln.

Ein zentraler Aspekt im „Reinforcement Learning“ ist das Abwägen zwischen „exploration“, also etwa erforschen und „exploitation“, also etwa ausnutzen. Da der Lernende zwar auf der einen Seite einen möglichst hohen Reward erhalten möchte, welchen er durch das Ausführen von „bewährten“ Aktionen erhalten kann, er aber um Aktionen mit hohem Reward zu finden auch Aktionen austesten muss, die er noch nie benutzt hat. Er muss ausnutzen was er weiß und trotzdem immer wieder erforschen, ob es evenutell noch bessere Aktionen gibt um seinen Reward im Ganzen zu maximieren. [SB98]

Elemente des Reinforcement Learning

Policy Die Policy beschreibt die Strategie, in welcher der Lernende seine Aktionen auswählt. Dies kann sowohl eine Berechnung, eine gleichbleibende Reihenfolge und eine Tabelle sein, oder auch einfach nur per Zufall gewählt werden. Oft sind sie jedoch stochastisch und wählen Aktionen basierend auf Annahmen, die am Wahrscheinlichsten passieren werden und den größten Reward erbringen. Die policy wird durch π dargestellt.

Reward Funktion Das langfristige Maximieren der erhaltenen Rewards beschreibt prinzipiell das Ziel im Reinforcement Learning. Die Reward Funktion legt für jeden Zustand fest, wie viel er zum Erfüllen des jeweiligen Ziels beiträgt. Deshalb muss sie jedem Zustand einen numerischen Wert zuweisen, der den Nutzen des Zustandes repräsentiert. Die Reward Funktion $R : S \times A \to \mathbb{R}$ weißt einer im Zustand s gewählten Aktion a einen numerischen Reward zu. Sie stellt dabei nur den direkt erhaltenen Reward dar.

Value Funktion Im Gegensatz zur Reward Funktion soll die Value Funktion den langfristigen Gewinn approximieren. Grob gesagt wird durch sie also der nach dem Ausführen einer Aktion zu erwartende Gesamtgewinn dargestellt. Dieser langfristige Reward ist natürlich von der Policy abhängig, da diese bestimmt welche Aktionen ausgeführt werden sollen.

$$\begin{aligned} V^{\pi}(s) &= E_{\pi}\{r_0 + \gamma r_1 + \gamma^2 r_2 + ... | s_o = s\} \\ &= R(s, \pi(s)) + \gamma \textstyle\sum_{s'} P(s'|s, \pi(s)) V^{\pi}(s') \end{aligned}$$

Die sogenannte Value-Funktion V stellt also für die Pocliy π und den Startzustand s den erwarteten Reward dar. Sollen zusätzlich noch die gewählten Aktionen berücksichtigt werden, so kann die state-action value - Funktion, oder auch Q-Funktion genannt, verwendet werden. Diese repräsentiert dann den erwarteten Reward aus dem Startzustand s, wenn als erstes die Aktion a gewählt wird.

$$Q^{\pi}(s, a) = R(s, a) + \gamma \textstyle\sum_{s'} P(s'|s, a) Q^{\pi}(s', \pi(s'))$$

Wenn hier die optimale Funktion gefunden wurde, kann eine optimale Policy mit Hilfe von: $\pi^*(s) = argmax_a Q^*(s, a)$ gefunden werden, wobei $Q^*(s, a)$ je nach Funktion mit $V^*(s)$ substituiert werden kann.

Die Value Funktionen beruhen auf dem Optimalitätsprinzip von Bellman, welches besagt, dass jede Teillösung einer optimalen Lösung auch optimal sein muss.

Modell Im Reinforcement Learning wird zwischen model-free und model-based Varianten unterschieden, da entweder ohne ein Modell, nur durch trial-and-error, ein Reinforcement Problem gelöst werden kann. Genauso gut kann aber auch ein Modell verwendet werden das probiert, den nächsten Zustand und seinen Reward vorherzusagen und anhand von dieser Information die nächste Aktion zu wählen. Somit kann mit einem Modell eine bessere Planung betrieben werden.

[SB98]

2.2 Markov Prozesse

2.2.1 Markov Eigenschaft

Die Markov Eigenschaft bezeichnet eine „erinnerungslose“ Form eines stochastischen Prozesses beziehungsweise eines Zustandsübergangs. Das heißt, wenn jeder zukünftige Zustand des Prozesses immer nur von dem direkt vorhergehenden Zustand abhängig ist und nicht von der Folge der Zustände in der Vergangenheit, so erfüllt er die Eigenschaft, dass die konditionelle Wahrscheinlichkeitsverteilung der zukünftigen Zustände nur vom momentanen Zustand abhängig ist.

Formal geschrieben ist die Wahrscheinlichkeitsverteilung:

$$Pr\{s_{t+1} = s', r_{t+1} = r | s_t, a_t, r_t, s_{t-1}, a_{t-1}, ..., r_1, s_0, a_0\}$$

Wenn die Markov Eingeschaft erfüllt ist, reicht es jedoch nur die Abhängigkeit von dem direkt vorhergehenden Zustand und der gewählten Aktion zu betrachten:

$$Pr\{s_{t+1} = s', r_{t+1} = r | s_t, a_t\}$$

Andersherum gesagt, wenn beide Formeln das gleiche Ergebnis für einen Zustand liefern, besitzt dieser die Markov Eigenschaft, wenn dies für alle Zustände zutrifft, besitzt der ganze Prozess die Markov Eigenschaft.

Alle Prozesse beziehungsweise Features, die diese Eigenschaft nicht erfüllen sind somit Nicht-Markov. [SB98]

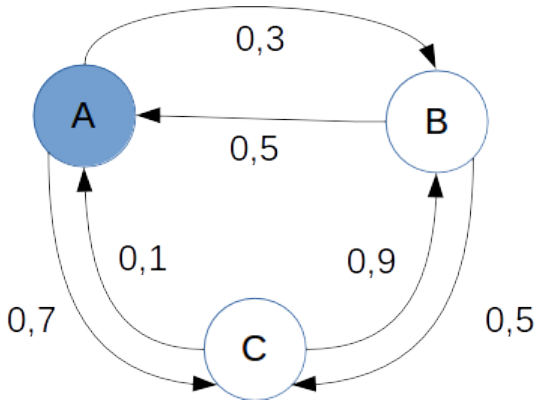

Abbildung 2.1: Beispiel Markov Kette mit drei Zuständen

2.2.2 Markov Kette

Eine Markov Kette beschreibt einen Prozess, der zufällige Zustandsübergänge in einem Zustandsraum beschreibt, welche die Markov Eigenschaft erfüllen. In Abbildung 2.1 ist eine einfache Markov Kette mit drei Zuständen visualisiert. Jeder Zustand hat dabei zwei Übergänge in einen Folgezustand, welcher mit Wahrscheinlichkeiten angegeben ist. Die Übergangswahrscheinlichkeiten sind alle nur vom momentanen Zustand abhängig und erfüllen somit die Markov Eigenschaft. Die Markov Kette wird dabei durch den Zustandsraum {A,B,C}, die Übergangswahrscheinlichkeiten, die an den Übergängen stehen, und dem Startzustand A definiert. Des Weiteren muss für Markov Ketten gewährleistet sein, dass es immer einen nächsten Zustand gibt und der Prozess nicht terminiert.

Eine Markov Kette, die nur vom vorhergehenden Zustand abhängt wird first-order genannt. Man kann das Modell zu einer höheren Ordnung erweitern, also M^{th}-order, sodass nicht nur der direkt vorherige Zustand Einfluss auf den Momentanen hat, sondern die M Vorhergehenden. Dies macht nur bis zu einem gewissen Grad Sinn, da die Menge der Parameter dabei exponentiell mit M wächst.

2.2.3 Markov Decision Process

Die oben genannte Markov Kette kann zu einem mächtigeren Rahmenwerk erweitert werden: dem Markov Decision Process (MDP). Dieser Prozess findet vor allem im Reinforcement Learning häufige Verwendung. Zusätzlich zum Zustandsraum, den Übergangswahrscheinlichkeiten und dem Startzustand, wird der MDP noch durch den Aktionsraum, eine Rewardfunktion und einen Discountfaktor beschrieben. Im Gegensatz zur Markov Kette ist der Prozess nicht vollständig zufällig, sondern wird zusätzlich von Entscheidungen durch das Wählen von verschiedenen Aktionen beeinflusst, wie es ein Lernender im Reinforcement Learning tun

würde. Da jedoch in einer unbekannten, beziehungsweise zumindest partiell unbekannten, Umgebung nicht immer fest steht was nach dem Ausführen einer Aktion passiert, besteht weiterhin eine Zufallskomponente, die nach Wahl einer Aktion den Folgezustand beeinflusst. Die Rewardfunktion soll bestimmen, ob eine Aktion „gut“ oder „schlecht“ ist, während der Discountfaktor den Unterschied der Relevanz zwischen direkten und zukünftigen Rewards repräsentiert.

Formal ist ein MDP also ein 5-Tupel $(S, A, P_a(s, s'), R_a(s, s'), \gamma)$ mit:

- S als endlichen Zustandsraum
- A als endlichen Aktionsraum
- $P_a(s, s') = Pr\{s_{t+1} = s' | s_t = s, a_t = a\}$ also der Übergangswahrscheinlichkeit von s nach s’ nachdem Aktion a gewählt wurde
- $R_a(s, s') = E\{r_{t+1} | s_t = s, a_t = a, s_{t+1} = s'\}$ als den erwarteten Reward, nach Wahl von a in Zustand s mit dem Folgezustand s’
- $\gamma \in \{0, 1\}$ als Discountfaktor

[SB98]

Das Hauptproblem von MDP’s stellt oft das Finden einer Policy dar, um Aktionen in Abhängikeit zum aktuellen Zustand auszuwählen. Ziel ist dann, mit Hilfe dieser Policy $\pi(s)$ Aktionen möglichst so zu wählen, dass langfristig gesehen der größte Reward erzielt wird. Für einen unendlichen Horizont also:

$$max \sum_{t=0}^{\infty} \gamma^t R_{a_t}(s_t, s_{t+1})$$

In Abbildung 2.2 ist ein beispielhafter MDP dargestellt. Im Gegensatz zu Abbildung 2.1 sind die Übergänge nicht mehr nur zufällig, sondern auch abhängig von der Wahl der Aktionen. Außerdem kommen zusätzlich die Rewards hinzu, die in der Abbildung in grün dargestellt sind.

2.2.4 Partially Observable Markov Decision Process

Es ist offensichtlich, dass die „reale“ Welt sehr viel komplexer ist als nur von einem direkt vorhergehenden Gesamtzustand abhängig zu sein, sondern auch von vielen Aktionen die in der Vergangenheit ausgeführt wurden.

Wie schon beschrieben, soll im vorliegenden konkreten Fall ein Roboter herausfinden, wie er eine Tür öffnen kann. Hierzu muss er Aktionen ausführen, die ihn in einen nächsten Zustand führen, der laut dem MDP unabhängig von allen Aktionen davor wäre. Dies würde auch

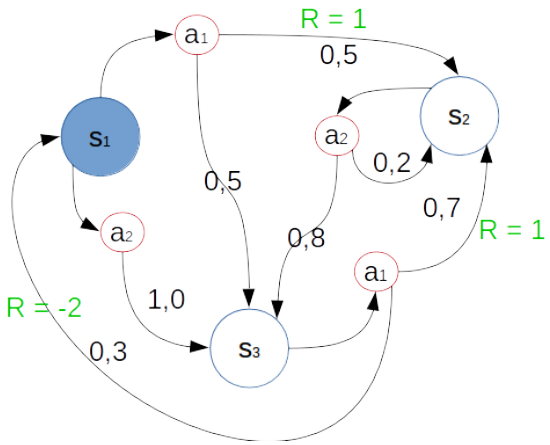

Abbildung 2.2: Beispiel Markov Decision Process mit 3 Zuständen und 2 Aktionen

zutreffen, wenn der Roboter nur einen Knopf drücken müsste und dann direkt die Tür öffnen könnte. Sobald er jedoch eine Kombination von Knöpfen drücken muss oder ein Knopf die Tür für mehr als einen Zeitschritt offen hält, ist die Markov Eigenschaft nicht mehr erfüllt. Deshalb reicht das Modell der MDP's dafür nicht aus.

Partially Observable Markov Decision Processes, kurz POMDP, erweitern das MDP Modell so, dass es für diese Zwecke anwendbar ist. Die zugrunde liegende Struktur der MDP's bleibt die Gleiche, es werden jedoch zusätzlich, zu den bereits beschriebenen Attributen, Observationen hinzugefügt. Anstatt direkt den momentanen Zustand überwachen zu können, erhält man „nur" eine Beobachtung, also einen Hinweis, in welchem Zustand man sich befinden. Diese Beobachtung kann wieder probabilistisch sein, was dazu führt, dass zusätzlich auch noch ein Observationsmodell erzeugt werden muss. Dieses repräsentiert die Wahrscheinlichkeit, in welchem Zustand man sich nach einer Observation befindet. Es müssen hierbei nicht alle ausgeführten Aktionen gespeichert werden. Eine Wahrscheinlichkeitsverteilung über alle möglichen Zustände zu haben, die mit jeder neuen Aktion und Observation aktualisiert wird, ist ausreichend, da in dieser Verteilung alle bisher gesammelten Erkenntnisse enthalten sind.

Formal wird das 5-Tupel der MDP's zu einem 7-Tupel $(S, A, P, R, \Omega, O, \gamma)$ erweitert:

- S, A , P, R und γ sind gleich zu den MDP's
- Ω ist die Menge der Observationen
- $O : S \times A \to \Omega$ ist die Menge der bedingten Observationswahrscheinlichkeit

[KLC98]

2.3 TD Learning

Temporal Difference Learning beschreibt eine Methode um die Value-Funktion basierend auf bisherigen Abschätzungen und neuen Beobachtungen zu aktualisieren, um so eine bessere Vorhersage über die Values zu erhalten. Für einen Zustand s_t in Zeitschritt t soll $V(s_t)$ basierend auf den nachfolgenden Ereignissen angepasst werden. Monte Carlo Methoden warten hierbei auf das endgültige Ergebnis dieser Ereignisse. Im Gegensatz hierzu wird beim TD Learning immer direkt im nächsten Zeitschritt die Schätzung mit einer neueren Schätzung aktualisiert und nicht auf wirkliche Ergebnisse gewartet. $V(s_t)$ wird direkt mit dem erhaltenen Reward und der Schätzung von V des Folgezustands aktualisiert:

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + V(s_{t+1} - V(s)]$$

Dabei beschreibt die Lernrate α, inwieweit die Schätzung von $V(s)$ mit jeder neuen Schätzung angepasst wird. [SB98]

2.4 Regularization

Oft sind die beim Lernen gesammelten Daten sehr gestreut und führen dazu, dass es zu einem „Overfitting“ kommt, dass das Modell die Features sozusagen auswendig lernt und deshalb nur genau diese erkennen kann. Somit findet keine Generalisierung der Trainingsdaten statt. Deshalb wird eine sogenannte „Regularization“ auf die Daten angewendet, um das „Overfitting“ soweit wie möglich zu reduzieren und ein möglichst aussagekräftiges Modell der Daten zu erhalten, welches auch ähnliche Features als richtig erkennen kann. Mit der Regularisierung kann die Komplexität des gewünschten Modells bestimmt werden, um eine möglichst gute Vorhersage zu erhalten. Dabei muss die Regularisierung so erfolgen, dass das Modell auf der einen Seite nicht overfitted, da es zu komplex ist, beziehungsweise underfitted, da es zu simpel ist. Um dies zu erreichen, wird zur Zielfunktion ein Regularisierungsterm hinzugefügt. Zum Beispiel für L_1-Regularization die Summe der Beträge aller Koeffizienten $\lambda \sum_{j=1}^{k} |\beta_j|$, wobei lambda den Regularisierungsparameter bestimmt und beta die Koeffizienten. Mit der L_1-Regularization findet außerdem eine Vorauswahl der Features statt, da viele der Features auf null abgebildet werden, sodass diese nicht weiter betrachtet werden müssen. Deshalb entsteht nach der Regularisierung ein spärlicheres Featureset.

2.5 Feature Expansion

Der Ansatz der Feature Expansion besteht darin, vorhandene Features mit Basisfeatures, welche meist erst einmal gefunden werden müssen, zu kombinieren, um so mehr Features, die aussagekräftig sein könnten zu erhalten. Es sollen durch Konjunktion verschiedener Features

„mächtigere“ Features erzeugt werden. Mit diesen zusätzlichen Features kann dann eine bessere Approximation der Value-Funktion erfolgen, um so zu einem besseren Endergebnis der ganzen Vorhersage zu kommen. Des Weiteren werden die erzeugten Features, welche ein bestimmtes Gewicht erhalten in das endgültige Featureset aufgenommen. Es können dabei jedoch auch viele unnötige Features entstehen, welche dann die Rechenzeit unnötig verlängern oder die Vorhersage negativ beeinflussen.

2.6 Vorstellung PULSE Algorithmus

Der PULSE Algortihmus[LT15] soll oben beschriebene POMDP mit verzögerten Kausalitäten lösen können. Hierzu verwendet er eine Kombination der Ansätze aus Feature Expansion und L_1-Regularisation im TD-Learning.

2.6.1 Temporally extended features

Der Algorithmus verwendet eine Menge $\mathcal{T}$ von „Temporally extended features“ (im Folgenden TEF), um POMDP's zu lösen. $\mathcal{T}$ beschreibt alle möglichen Abbildungen der Historie $(\mathcal{A} \times \mathcal{O} \times \mathcal{R})^*$, also Aktion-Observation-Reward Tripeln, auf reelle Zahlen $\mathbb{R}$. Der Algorithmus verwendet in jeder Iteration aber nur eine Teilmenge von $\mathcal{T}$, welche aus einer Funktion $\mathcal{N}^+ : \mathcal{P}(\mathcal{T}) \to \mathcal{P}(\mathcal{T})$, für welche $\mathcal{P}(\mathcal{T})$ die Potenzmenge von $\mathcal{T}$ beschreibt, generiert wird. Das heißt für eine gegebene Menge an Features $\mathcal{F}$ ist $\mathcal{N}^+(\mathcal{F})$ eine Menge von Feature Kandidaten, welche dazu verwendet werden, um die kleinste Teilmenge $\mathcal{T}_{\mathcal{N}^+} \subseteq \mathcal{T}$ die unter $\mathcal{T}$ abgeschlossen ist, zu erforschen.

Um zeitlich verzögerte Kausalitäten zu beschreiben werden Basisfeatures $\mathcal{B}$, die aus den einzigartigen Aktionen, Observationen und Rewards für jeden Zeitschritt bestehen, miteinander verlinkt, um daraus neue Features zu erzeugen. $\mathcal{N}^+$ wird so definiert, dass es aus allen möglichen Konjunktionen von bereits vorhandenen Features und den Basisfeatures, die neuen Feature Kandidaten erzeugt. Um nicht zu weit in die Vergangenheit zu gehen wird des Weiteren $\mathcal{N}^+$ so modifiziert, dass es immer nur die Basisfeatures aus einem vorausgehenden Schritt verwendet und maximal zu einem Zeithorizont $t_{min} = -k$ zurückgeht. [LT15]

2.6.2 Der Algorithmus

Der Algorithmus verwendet die oben beschriebenen TEF's, die mit Hilfe von $\mathcal{N}^+$ erzeugt wurden, um die bereits vorhandene Menge an Features zu erweitern und diese ebenfalls zu überprüfen. Mit Hilfe von L_1-regularization in der Zielfunktion werden alle überflüssig erzeugten Features dann wiederum gelöscht. Dies führt dazu, dass die Menge der Features im Algorithmus sozusagen „pulsiert“, da in jeder Iteration zuerst neue Features hinzugefügt

Algorithmus 2.1 PULSE Algorithmus aus [LT15]

procedure MAIN($\mathcal{N}^+$, $\mathcal{O}$, $\mathcal{D}$)
  Initialize :$\mathcal{F} \leftarrow \emptyset, \Theta \leftarrow \emptyset$
  repeat
    GROW_FEATURE_SET($\mathcal{F}, \Theta, \mathcal{N}^+$)
    $\Theta \leftarrow argmin_\Theta \mathcal{O}(\mathcal{F}, \Theta, \mathcal{D})$ // Hier werden die Gewichte der Features optimiert
    SHRINK_FEATURE_SET($\mathcal{F}, \Theta$)
  until $\mathcal{O}$, $\mathcal{F}$, Θ do not change
  return $\mathcal{F}, \Theta$
end procedure
procedure GROW_FEATURE_SET($\mathcal{F}, \Theta, \mathcal{N}^+$)
  Initialize :$\mathcal{F}^+ \leftarrow \mathcal{N}^+(\mathcal{F})$
  for all $f \in \mathcal{F}^+$ **do**
    if $f \notin \mathcal{F}$ **then**
      $\Theta_f \leftarrow 0$
    end if
  end for
  $\mathcal{F} \leftarrow \mathcal{F} \cup \mathcal{F}^+$
end procedure
procedure SHRINK_FEATURE_SET($\mathcal{F}, \Theta$)
  for all $f \in \mathcal{F}$ **do**
    if $\Theta_f is 0$ **then**
      $\mathcal{F} \leftarrow \mathcal{F} \setminus f$
    end if
  end for
end procedure

werden, direkt im Anschluss jedoch auch alle unnötig erzeugten Features wieder gelöscht werden. Durch wiederholtes Anwenden führen $\mathcal{N}^+$ und die Zielfunktion dazu, dass ein Optimum erreicht wird. Im Algorithmus 2.1 ist der Ablauf dargestellt. Hierbei bezeichnen $\mathcal{O}$ die Zielfunktion, Θ die Gewichte der Features und $\mathcal{D}$ die Daten, welche an den Algorithmus übergeben werden sollen. Der Algorithmus kann in einer model-based und einer model-free Variante verwendet werden. In vorliegendem Fall wird die model-based Variante angewandt, die TEF's mit conditional random fields(CRF) verbindet. Hierbei beschreiben die CRF's die Wahrscheinlichkeiten für die nächsten Observationen und Rewards in Abhängigkeit vom aktuellen Zustand und der gewählten Aktion.[LT15]

3 Hintergrund zur praktischen Umsetzung

3.1 Versuchsaufbau

Der Versuchsaufbau ist relativ einfach. Der Roboter befindet sich in einem Raum ohne jegliche Vorkenntnisse von diesem. In ihm befinden sich zudem einige Knöpfe, die zur Vereinfachung der Perzeption durch Alvar Marker dargestellt sind. Jeder dieser Alvar Marker hat eine eindeutige ID, mit welcher der Roboter die „Knöpfe“ identifizieren kann. Des Weiteren gibt es eine Tür, welche zu Beginn verschlossen ist. Da es zu aufwendig wäre einen wirklichen Mechanismus dafür zu entwickeln, wird die Tür zu gehalten, bis der Roboter den Knopf, der die Tür entriegelt, drückt. Nun soll der Roboter für eine bestimmte Zeit Daten sammeln, um so später möglichst genau ermitteln zu können, durch welchen Knopfdruck/welche Kombination von verschiedenen Knöpfen er die Tür öffnen kann.

Da der Roboter verschieden lange zur Ausführung der Aktionen benötigt, wird jedes Ausführen einer Aktion als ein Zeitschritt gelten, unabhängig davon, wie lange der Roboter tatsächlich benötigt.

3.2 PR2

Der PR2 ist ein Forschungs- und Entwicklungsroboter, der mit dem Robot Operating System (ROS) betrieben wird. Er wurde speziell zur Verwendung in der Personal Robotic entwickelt. Hierzu besitzt der PR2(siehe Abbildung 3.1) eine mobile Basis mit vier steuerbaren Rädern, um sich in einer natürlichen menschlichen Umgebung mit einer Geschwindigkeit von etwa 1m/s frei bewegen zu können. Damit er mit dieser Basis sicher zu seinem Ziel gelangen kann ohne an Hindernissen zu scheitern beziehungsweise mit Menschen zu kollidieren, besitzt der PR2 neben der Positionsbestimmung über Odometrie anhand seines Antriebs sowohl an seiner Basis als auch an seinem Oberkörper jeweils einen Laserscanner, um Hindernisse zu erkennen und zu meiden. Am Oberkörper des PR2 befinden sich zwei Arme mit jeweils sieben Gelenken, was zu einem hohen Bewegungsradius der Arme führt. Am Ende der Arme befinden sich als Endeffektoren Gripper die sich drehen lassen, um so immer aus dem richtigen Winkel greifen zu können. An den Endeffektoren sind zusätzlich noch

Abbildung 3.1: Links der Vesuchsaufbau: Die untere Marker Reihe als Knöpfe und ein Marker an der Türklinke.
Rechts: Der PR2 von Willow Garage.

„Force Torque Sensoren“ angebracht, welche man verwenden kann, um ab einem gewissen Threshold eine Bewegung zu stoppen. Am Kopf des PR2, welcher sich um 350° drehen und um 115° neigen lässt, befinden sich verschiedene Kameras sowie eine zusätzliche Microsoft Kinect. Durch diese können Pointclouds sowie normale Bildstreams zur Erkennung von Objekte verwendet werden. Auch an den beiden Armen ist jeweils, direkt hinter dem Gripper, eine Kamera angebracht. Um alle Aktuatoren und Sensoren anzusteuern und verwenden zu können, sind in der Basis des PR2 zwei Onboard Server verbaut. [PR215]

3.3 Robot Operating System

3.3.1 Über ROS

ROS ist ein für Roboter entwickeltes Framework, welches es ermöglicht, robuste Software für Roboter zu schreiben. In ihm werden die benötigten Konventionen beschrieben. Des Weiteren beinhaltet es eine Sammlung von vielen verschiedenen Tools, beispielsweise zur Visualisierung von Sensordaten, einer Simulation des Roboters zum Testen der entwickelten Software und vielen Weiteren. Außerdem beinhaltet es eine große Sammlung an Paketen die verwendet werden können. Die für diese Arbeit relevanten Pakete werden im Teil Verwendete ROS Module vorgestellt. Zu den direkt von ROS gestellten Paketen kommen außerdem noch

viele Pakete, die im großen Ecosystem von ROS zu finden sind, da jeder seine Pakete zur freien Verfügung stellen kann. [ros15a]

3.3.2 Funktionsweise

Eine der wichtigsten Komponenten von ROS ist das Message Passing System. Dieses ermöglicht die Implementierung der Software mit Hilfe von verschiedenen Knoten umzusetzen. Hierbei hat durch das Message Passing System jeder Knoten ein klares Interface und die Funktionen sind in einzelne Module getrennt. Die Knoten können entweder kontinuierlich über Publisher und Subscriber miteinander kommunizieren und Daten senden beziehungsweise empfangen oder mit Hilfe von Services und Service Proxies Remote Procedure Calls ausführen, um einzelne Funktionen nur nach einem bestimmten Aufruf auszuführen. Erleichtert wird dies durch viele verfügbare „Standart Robot Messages“, die die einfache Kommunikation innerhalb des Ecosystems ermöglichen.[ros15b]

Die Kommunikation zwischen den Knoten erfolgt über eine direkte TCP/IP-basierte Verbindung. Sie wird jedoch erst ermöglicht, wenn ein sogenannter „ROS Master“ verfügbar ist. Dieser agiert als eine Art Namensserver:

- Ein Knoten, der Daten publishen will, benachrichtigt den Master Knoten, welcher die Information des Knotens abspeichert. Es wird jedoch noch nicht begonnen Daten zu publishen.
- Will ein Knoten auf eine Topic subscriben, fragt er beim Master Knoten an ob die gewünschte Topic vorhanden ist.
- Sobald der Master Knoten für eine vorhandene Topic eine Anfrage erhält, benachrichtigt er sowohl Publisher als auch Subscriber und es wird eine direkte Verbindung zwischen den Beiden aufgebaut.
- Der Publisher sendet Messages an den Subscriber.

Ohne den Master Knoten können keine anderen Knoten laufen. Deshalb muss dieser immer zuerst gestartet werden.[ros15d]

Für jeden Publisher wird eine fest definierte Message festgelegt. Analog dazu muss für jeden Service eine .srv-Datei definiert sein, welche die jeweiligen request- und response-messages enthält. Außerdem muss zum erfolgreichen Subscriben auf eine gepublishte Topic gewährleistet sein, dass der Message-Typ des Subscribers mit dem Publisher übereinstimmt, da ansonsten keine Nachrichten empfangen werden können. Selbes gilt auch für Services, wo die jeweiligen .srv-Typen übereinstimmen müssen.[ros15c]

In Schaubild 3.3 sind beispielhaft drei Knoten dargestellt. Knoten A published eine fest definierte Message als Topic. Knoten B und C können nun auf diese Topic subscriben um die

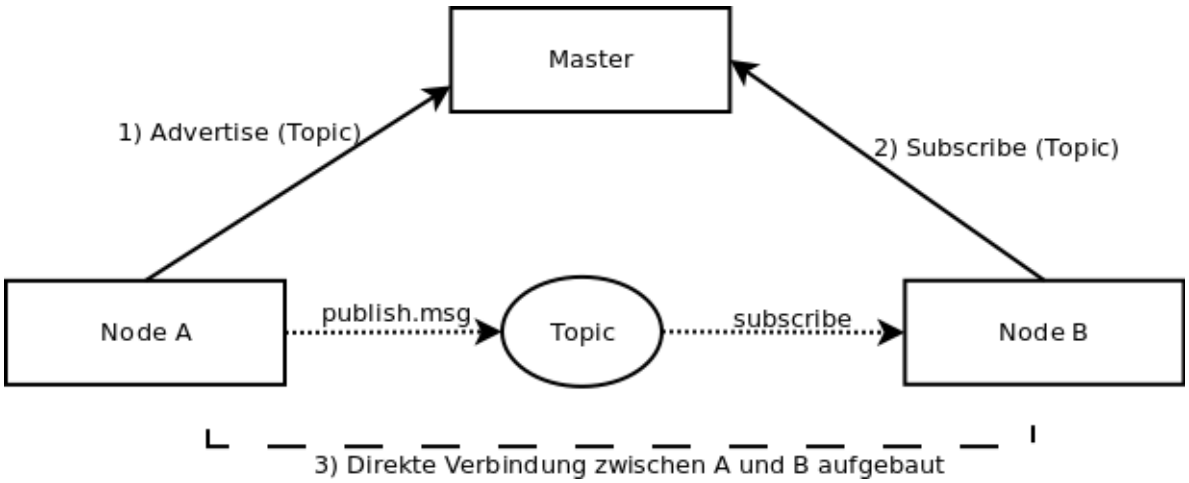

Abbildung 3.2: Beispiel Verbindungsaufbau zwischen zwei Knoten

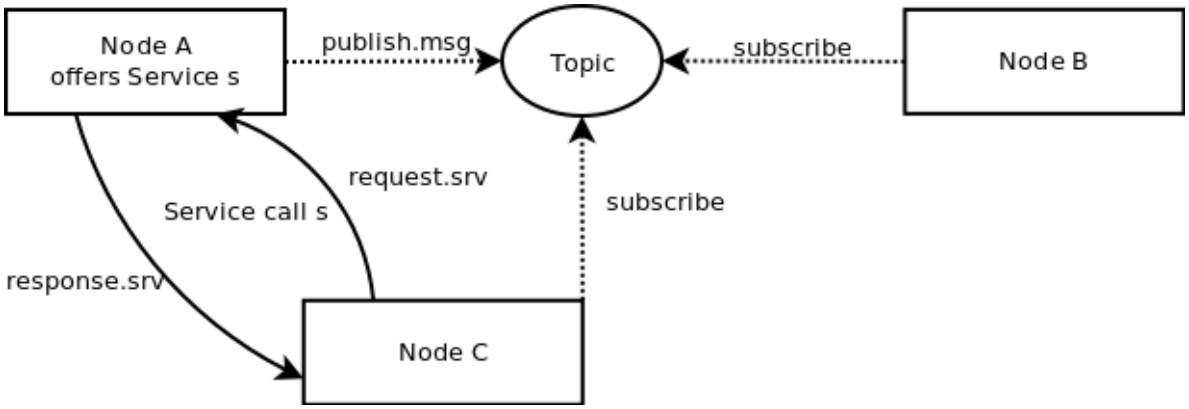

Abbildung 3.3: Funktionsweise des ROS Message Passing Systems

Informationen, die Knoten A sendet zu empfangen. Außerdem bietet A einen Service s an, welcher über einen Remote Procedure Call aufgerufen werden kann. Im Schaubild sendet C einen solchen Aufruf mit den entsprechenden Messages an A und erhält nach Ausführung der Methode die jeweilige Response-Message zurück.

3.3.3 Paketstruktur

An oberster Stelle der ROS Paketstruktur steht ein Workspace, in welchem die einzelnen Komponenten zusammengeführt werden. In diesem Workspace können die einzelnen Packa-

ges abgelegt werden. Packages enthalten die Software für die einzelnen Module. In ihnen befinden sich die verschiedenen Knoten, die ROS-abhängigen Bibliotheken, Datensätze, Ressourcen und Konfigurationsdateien. Des Weiteren werden die Message- und Service-files im Package abgelegt. Über den Workspace können alle in ihm befindlichen Packages auf einmal kompiliert werden. Hierbei werden auch die Messages und Services so erzeugt, dass sie dann verwendbar gemacht werden.[ros15c]

3.4 Verwendete ROS Module

3.4.1 ar_track_alvar

Das Paket ar_track_alvar ist ein Wrapper für die Software Bibliothek „ALVAR“, welche dazu gemacht wurde, um virtuelle und augmented Reality Anwendungen zu erstellen. Es beinhaltet eine Vielzahl an Funktionen. Für die Anwendung am PR2 ist aber hauptsächlich die Markererkennung wichtig. Hierfür bietet „ALVAR“ die Möglichkeit eine sehr genaue Marker Pose zurückzugeben. Mit ar_track_alvar kann genau diese Funktion auf dem PR2 oder mit jeder anderen ROS-fähigen Kamera genutzt werden. Hierzu muss lediglich ein .launch-file gestartet werden. Wenn der Knoten des Paketes gestartet ist, wird ein Publisher gestartet, der für jeden erkannten Marker dessen Position und Orientierung sowie seine ID und andere Daten versendet. Von dieser Topic können dann wiederum die Daten ausgelesen werden, um eine genaue Position des Markers zu erhalten und zum Beispiel an diese zu navigieren.[art15] [Fin15]

3.4.2 pr2_2dnav

Pr2_2dnav ist ein Paket, welches es dem PR2 ermöglicht autonom durch eine Umgebung zu navigieren. Hierzu kann das Paket mit oder ohne SLAM gestartet werden. SLAM steht für „Simultaneous Localization and Mapping“ und beschreibt ein Verfahren, mit welchem ein mobiler Roboter, wie der PR2, während der Navigation eine Karte seiner Umgebung erstellen und sich dann in dieser lokalisieren kann. Wird das Paket ohne SLAM gestartet muss ihm eine Karte der Umgebung übergeben werden.

Zur Navigation wird der sogenannte Navigation Stack verwendet. Über diesen kann dem Roboter eine Position innerhalb der Karte übergeben werden, um dorthin zu navigieren. Hierzu wird anhand der Karte ein globaler Plan erstellt, der den kürzesten Weg zum Ziel beschreibt. Zusätzlich wird noch ein lokaler Plan anhand der direkten Sensordaten des Roboters erstellt, um zum Beispiel Hindernisse, die nicht auf der Karte erkennbar sind, zu umfahren. Sollte das Hindernis nicht direkt umfahren werden können, wird über den globalen Plan ein anderer Weg gesucht.[2dn15]

3.4.3 Motion Generation

Um Bewegungen des Roboters zu erzeugen wurde ein am MLR Institut entwickelter low-level real-time Controller verwendet. Dieser lässt sich über ein high-level Action Interface, welches in Python implementiert ist und dem Benutzer ermöglicht *Actions* auszuführen, verwenden. Davon bietet es Verschiedene, welche angepasst werden können, um die gewünschten Aktionen auszuführen. Abgeleitet werden alle *Actions* von einer *Activity*. Einige dieser abgeleiteten *Activities* sind schon gegeben, wie etwa eine *ReachActivity*, von welcher wiederum eine *reach_marker*-Funktion erzeugt werden kann. Außerdem können auch eigene *Activites* erzeugt werden. Des Weiteren gibt es die Möglichkeit mehrere Aktionen zu einem Plan zusammenzufassen. Hierzu gibt es einen „with“-Teil, in dem Aktionen, die dauerhaft bis zum Beendigen des Planes ausgeführt werden enthalten sind und einen „plan“-Teil, in welchem sich die nacheinander auszuführenden Aktionen befinden. Im „plan“-Teil können außerdem Tupel erzeugt werden, die gleichzeitig ausgeführt werden. Nach Beendigung der letzten Aktion im „plan“-Teil wird der ganze Plan, also auch der „with“-Teil, beendet.

4 Simulation

4.1 Simulierte Welt

Die „Simulierte Welt“ wurde in Python implementiert, um den PULSE-Algorithmus[LT15] zu testen und die Perfomanz verschiedener Strategien, zur Wahl der Aktionen, zu vergleichen. Sie soll möglichst die realen Bedingungen simulieren. In der „Simulierten Welt“ können deshalb äquivalent zum realen Versuchsaufbau mehrere „Knöpfe“ und „Türen“ erzeugt werden. Diese werden dann zufällig miteinander verlinkt, sodass jedem Knopf entweder eine zu öffnende Türe zugewiesen wird oder auch keine. Außerdem wird zufällig bestimmt für wie viele Zeitschritte die Tür geöffnet bleibt.

In Abbildung 4.1 ist das UML-Diagramm zur „Simulierten Welt“ visualisiert. Um eine Observation zu starten, muss zuerst ein Objekt der Klasse Observation erzeugt werden. Außerdem muss ein Environment erzeugt werden, welchem übergeben wird, wie viele Knöpfe und Türen sich in der Simulation befinden sollen. Anhand von diesen Zahlen werden dann die Knöpfe und Türen generiert. Dabei wird den Knöpfen zufällig eine oder auch keine Tür zugewiesen, welche sie dann durch Aufruf der Methode „push_button()“ für eine ihnen ebenfalls zufällig zugewiesen Zeitraum öffnen. Außerdem wird ein Objekt der Klasse Strategy erzeugt, dem die Liste der Knöpfe und der Türen übergeben wird. Zum Starten der eigentlichen Observation muss die Methode „observe“ mit einem Parameter, welcher die zu benutzende Strategie und die Menge der zu observierenden Zeitschritte festlegt, aufgerufen werden. Nun wird für den festgelegten Zeitraum in jedem Schritt, je nach Strategie, entweder ein Knopf gedrückt oder eine Tür geöffnet. Es wird dabei ein 3-Tupel der ausgeführten Aktion, der resultierenden Beobachtung und dem Reward, welcher nur durch Öffnen der Tür erhalten wird, in die Liste der gesammelten Daten gespeichert. Da die Observationen in beschriebenem Fall irrelevant sind, wird für diese immer eine Null in die gesammelten Daten geschrieben.

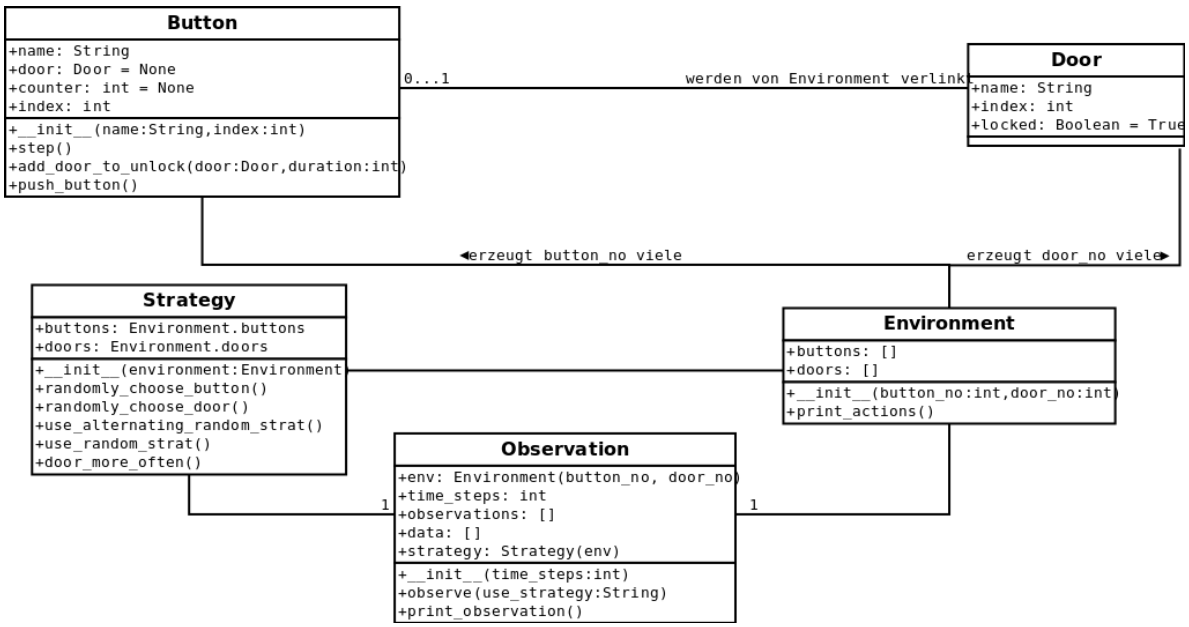

Abbildung 4.1: UML Diagramm der Simulierten Welt

4.1.1 Strategien

Folgende Strategien wurden zum Testen implementiert:

Random In jedem Schritt soll eine zufällige Aktion gewählt werden. Es ist dabei egal, ob es sich um das Drücken eines Knopfes oder das Öffnen der Tür handelt.

AlterningRandom Hierbei wird immer abwechselnd ein Knopf gedrückt und dann probiert eine Tür zu öffnen.

TwoButtons Es werden immer zwei Knöpfe gedrückt und dann probiert eine Tür zu öffnen.

NoDoubleUsage Die Aktionen werden wie bei der Random-Strategie gewählt, jedoch wird der gedrückte Knopf aus der Liste gelöscht, um zuerst alle anderen Knöpfe zu drücken. Ist kein Knopf mehr verfügbar, werden wieder alle zur Liste hinzugefügt.

DoorMoreOften Hierbei werden die Aktionen ebenfalls zufällig gewählt. Jedoch ist die Chance, die Tür zu probieren, zehnmal so wahrscheinlich als Knöpfe zu drücken.

4.2 Evaluation

Zur Evaluation des Algorithmus, in Verbindung mit den fünf zuvor genannten Strategien zum Sammeln der Testdaten, wurden qualitative und quantitative Tests durchgeführt. Mit deren Ergebnissen soll überprüft werden, ob der Algorithmus überhaupt für eine solche Anwendung geeignet beziehungsweise sinnvoll ist. Zugleich soll die „beste“ Strategie bestimmt werden, welche dann auf dem PR2 zu verwenden ist.

4.2.1 Qualitative Analyse

Zur qualitativen Auswertung wurden für die verschiedenen Ansätze jeweils Testdaten in der gleichen Größe erzeugt, welche dann an den Algorithmus übergeben wurden, um mit Hilfe von ihnen die Features zu erlernen. Im Anschluss wurde für jede Strategie überprüft ob die relevanten Features vorhanden sind und wie schwer diese gewichtet sind. Zugleich sollen die Ergebnisse der einzelnen Strategien miteinander verglichen werden, um so ein Fazit ziehen zu können.

Erläuterung der Features Die Features sind immer gleich aufgebaut:

idx weight basis-features

Hierbei beschreibt die idx eine eindeutige Kennung des jeweiligen Features. Das Gewicht bezeichnet die Aussagekraft des Features: Je höher desto wahrscheinlicher. Die Basis-Features zeigen wie die jeweiligen Features zusammen gesetzt sind. Sind die Gewichte negativ, ist ein Feature unwahrscheinlich. Alle Features treten immer paarweise mit positivem und negativem Gewicht auf.

Als Beispiel für diesen Fall:

idx	weight	basis-features
13	-11.1503	$(1, -7) \wedge r(0, 0)$
14	11.1503	$(1, -7) \wedge r(1, 0)$

Hierbei bedeutet Feature 13 in etwa: „Es ist eher unwahrscheinlich, dass wenn Aktion eins vor sieben Zeitschritten gewählt wurde, kein Reward erhalten wurde."

Feature 14 hingegen wäre andersherum gesagt: „Es ist eher wahrscheinlich, dass wenn Aktion eins vor sieben Zeitschritten gewählt wurde, ein Reward erhalten wurde."

Außerdem ist zu beachten, dass für den hier betrachteten Fall paarweise Features ausreichend sind, da es nur einen direkten temporalen Zusammenhang zwischen Knopf und Tür gibt. Des Weiteren kann auf die Observationen verzichtet werden, da diese gleichbedeutend mit den Rewards sind. Geht die Tür auf? → Ja — Reward = 1 → Nein — Reward = 0.

Beim Vergleich der Features reicht es nur die Features mit positiven Gewichten zu betrachten, da die Aussage beider Features die Gleiche ist.

Beispiel Auswertung In Abbildung 4.2 ist ein beispielhafter Vergleich der gelernten Featuresets zu sehen. Für diesen Vergleich wurde für jede Strategie ein Trainingsdatensatz der Größe 1000 erzeugt und anschließend die resultierenden Features ausgegeben und gefiltert, um alle Features, welche ein zu geringes Gewicht haben, aus der Auswertung auszuschließen.

Wie dieses Beispiel zeigt, wurden die relevanten Features in unterschiedlichen Farben hervorgehoben. „Wahre" Features sind dunkelgrün markiert, Features die richtig sind aber zu einem zu späten Zeitschritt entdeckt wurden sind hellgrün, während falsche Features rot sind. Die gelbe Markierung beschreibt die notwendige Aktion, also das Öffnen der Tür zum Zeitschritt null, da ansonsten kein Reward erhalten werden kann. Wie zu sehen ist, hat die TwoButton Strategie in diesem Fall am Schlechtesten abgeschnitten, da nur ein falsches Feature gelernt wurde. Danach kommt die AlterningRandom Strategie, die eines der Features zu spät entdeckt hat und ein gänzlich falsches beinhaltet. Die Random policy hat zwar keine falschen Features, aber auch hier wurde eines der Features zu spät entdeckt. Die NoDoubleUsage und DoorMoreOften Strategien haben die Features richtig entdeckt. Jedoch benötigt die DoorMoreOften Strategie dazu weniger Features, was bedeutet, dass diese das beste Ergebnis erzielt hat.

idx	|weight|	weight	basis-features...
4	175.518	175.518	a(3, -9) ^ r(1, 0)
5	107.968	107.968	a(4, -9) ^ r(0, 0)
8	253.673	253.673	a(4, -8) ^ r(1, 0)
10	794.683	794.683	a(2, -7) ^ r(1, 0)
11	103.724	103.724	a(3, -7) ^ r(0, 0)
13	108.663	108.663	a(4, -7) ^ r(0, 0)
16	254.506	254.506	a(4, -6) ^ r(1, 0)
18	498.929	498.929	a(1, -5) ^ r(1, 0)
20	461.954	461.954	a(2, -5) ^ r(1, 0)
21	546.607	546.607	a(3, -5) ^ r(0, 0)
23	110.713	110.713	a(4, -5) ^ r(0, 0)
26	255.882	255.882	a(4, -4) ^ r(1, 0)
28	472.117	472.117	a(1, -3) ^ r(1, 0)
30	385.976	385.976	a(2, -3) ^ r(1, 0)
31	539.839	539.839	a(3, -3) ^ r(0, 0)
33	115.623	115.623	a(4, -3) ^ r(0, 0)
38	259.561	259.561	a(4, -2) ^ r(1, 0)
40	49.052	49.052	a(1, -1) ^ r(1, 0)
42	50.503	50.503	a(2, -1) ^ r(1, 0)
43	55.378	55.378	a(3, -1) ^ r(0, 0)
45	14.181	14.181	a(4, -1) ^ r(0, 0)
47	26.824	26.824	a(2, 0) ^ r(0, 0)
50	312.999	312.999	a(4, 0) ^ r(1, 0)

idx	|weight|	weight	basis-features...
5	368.441	368.441	a(1, -8) ^ r(0, 0)
8	60.585	60.585	a(2, -8) ^ r(1, 0)
9	272.426	272.426	a(3, -8) ^ r(0, 0)
11	280.622	280.622	a(4, -8) ^ r(0, 0)
14	111.926	111.926	a(2, -7) ^ r(1, 0)
19	379.774	379.774	a(1, -6) ^ r(0, 0)
22	743.126	743.126	a(2, -6) ^ r(1, 0)
23	382.123	382.123	a(3, -6) ^ r(0, 0)
25	343.457	343.457	a(4, -6) ^ r(0, 0)
30	876.228	876.228	a(2, -5) ^ r(1, 0)
31	800.787	800.787	a(3, -5) ^ r(0, 0)
33	828.296	828.296	a(4, -5) ^ r(0, 0)
38	331.918	331.918	a(2, -4) ^ r(1, 0)
39	573.558	573.558	a(3, -4) ^ r(0, 0)
41	637.845	637.845	a(4, -4) ^ r(0, 0)
44	12.975	12.975	a(1, -3) ^ r(1, 0)
46	314.688	314.688	a(2, -3) ^ r(1, 0)
47	626.518	626.518	a(3, -3) ^ r(0, 0)
49	595.496	595.496	a(4, -3) ^ r(0, 0)
52	361.023	361.023	a(1, -2) ^ r(1, 0)
55	571.257	571.257	a(3, -2) ^ r(0, 0)
57	581.863	581.863	a(4, -2) ^ r(0, 0)
60	820.597	820.597	a(1, -1) ^ r(1, 0)
63	768.303	768.303	a(3, -1) ^ r(0, 0)
65	779.052	779.052	a(4, -1) ^ r(0, 0)
67	120.939	120.939	a(1, 0) ^ r(0, 0)
69	681.908	681.908	a(2, 0) ^ r(0, 0)
72	470.337	470.337	a(4, 0) ^ r(1, 0)

idx	|weight|	weight	basis-features...
2	687.154	687.154	a(4, -9) ^ r(1, 0)
3	385.032	385.032	a(1, -8) ^ r(0, 0)
5	123.855	123.855	a(4, -8) ^ r(0, 0)
7	39.501	39.501	a(1, -7) ^ r(0, 0)
9	123.997	123.997	a(4, -7) ^ r(0, 0)
12	685.095	685.095	a(4, -6) ^ r(1, 0)
13	638.667	638.667	a(3, -5) ^ r(0, 0)
15	123.527	123.527	a(4, -5) ^ r(0, 0)
17	614.867	614.867	a(3, -4) ^ r(0, 0)
19	135.062	135.062	a(4, -4) ^ r(0, 0)
24	719.108	719.108	a(4, -3) ^ r(1, 0)
27	640.356	640.356	a(3, -2) ^ r(0, 0)
29	14.593	14.593	a(4, -2) ^ r(0, 0)
31	653.244	653.244	a(3, -1) ^ r(0, 0)
33	160.743	160.743	a(4, -1) ^ r(0, 0)
35	281.625	281.625	a(1, 0) ^ r(0, 0)
37	283.337	283.337	a(2, 0) ^ r(0, 0)
42	106.311	106.311	a(4, 0) ^ r(1, 0)
45	137.233	137.233	r(0, 0)

idx	|weight|	weight	basis-features...
1	150.888	150.888	a(1, -9) ^ r(0, 0)
3	290.621	290.621	a(2, -9) ^ r(0, 0)
6	126.997	126.997	a(2, -8) ^ r(1, 0)
9	300.871	300.871	a(4, -8) ^ r(0, 0)
11	299.915	299.915	a(1, -7) ^ r(0, 0)
14	982.083	982.083	a(2, -7) ^ r(1, 0)
15	236.125	236.125	a(4, -7) ^ r(0, 0)
20	166.012	166.012	a(2, -6) ^ r(1, 0)
22	165.088	165.088	a(4, -6) ^ r(1, 0)
24	155.621	155.621	a(1, -5) ^ r(1, 0)
26	75.991	75.991	a(2, -5) ^ r(1, 0)
27	957.405	957.405	a(3, -5) ^ r(0, 0)
29	567.849	567.849	a(4, -5) ^ r(0, 0)
32	162.717	162.717	a(1, -4) ^ r(1, 0)
34	955.406	955.406	a(2, -4) ^ r(1, 0)
35	100.196	100.196	a(3, -4) ^ r(0, 0)
37	712.336	712.336	a(4, -4) ^ r(0, 0)
40	417.572	417.572	a(1, -3) ^ r(1, 0)
42	605.334	605.334	a(2, -3) ^ r(1, 0)
43	101.675	101.675	a(3, -3) ^ r(0, 0)
45	644.131	644.131	a(4, -3) ^ r(0, 0)
48	476.057	476.057	a(1, -2) ^ r(1, 0)
50	321.473	321.473	a(2, -2) ^ r(1, 0)
51	665.086	665.086	a(3, -2) ^ r(0, 0)
53	789.445	789.445	a(4, -2) ^ r(0, 0)
56	541.222	541.222	a(1, -1) ^ r(1, 0)
58	22.891	22.891	a(2, -1) ^ r(1, 0)
59	684.835	684.835	a(3, -1) ^ r(0, 0)
61	678.663	678.663	a(4, -1) ^ r(0, 0)
63	66.864	66.864	a(1, 0) ^ r(0, 0)
65	103.697	103.697	a(2, 0) ^ r(0, 0)
67	214.827	214.827	a(3, 0) ^ r(0, 0)
70	455.756	455.756	a(4, 0) ^ r(1, 0)

idx	|weight|	weight	basis-features...
1	121.537	121.537	a(1, -9) ^ r(0, 0)
6	103.803	103.803	a(2, -8) ^ r(1, 0)
7	107.763	107.763	a(4, -8) ^ r(0, 0)
10	106.964	106.964	a(2, -7) ^ r(1, 0)
11	172.571	172.571	a(4, -7) ^ r(0, 0)
14	956.717	956.717	a(2, -6) ^ r(1, 0)
15	223.709	223.709	a(4, -6) ^ r(0, 0)
18	973.379	973.379	a(1, -5) ^ r(1, 0)
20	934.227	934.227	a(2, -5) ^ r(1, 0)
21	44.127	44.127	a(3, -5) ^ r(0, 0)
23	433.684	433.684	a(4, -5) ^ r(0, 0)
26	959.968	959.968	a(1, -4) ^ r(1, 0)
28	943.018	943.018	a(2, -4) ^ r(1, 0)
29	631.436	631.436	a(3, -4) ^ r(0, 0)
31	392.637	392.637	a(4, -4) ^ r(0, 0)
34	798.916	798.916	a(1, -3) ^ r(1, 0)
36	838.813	838.813	a(2, -3) ^ r(1, 0)
37	608.937	608.937	a(3, -3) ^ r(0, 0)
39	837.387	837.387	a(4, -3) ^ r(0, 0)
42	821.036	821.036	a(1, -2) ^ r(1, 0)
44	794.305	794.305	a(2, -2) ^ r(1, 0)
45	26.912	26.912	a(3, -2) ^ r(0, 0)
47	623.291	623.291	a(4, -2) ^ r(0, 0)
50	119.933	119.933	a(1, -1) ^ r(1, 0)
52	893.649	893.649	a(2, -1) ^ r(1, 0)
53	466.194	466.194	a(3, -1) ^ r(0, 0)
55	672.689	672.689	a(4, -1) ^ r(0, 0)
58	309.665	309.665	a(4, 0) ^ r(1, 0)
59	399.798	399.798	r(0, 0)

Abbildung 4.2: Die gefilterten gelernten Featuresets der verschiedenen Strategien, v.l.n.r.: AlterningRandom, Random, TwoButton, NoDoubleUsage und DoorMoreOften. Dunkelgrün sind die wirklich benötigten Features, rot die falsch gelernten Features, hellgrün richtig gelernte Features, welche aber zu einem zu späten Zeitpunkt gefunden wurden und gelb die notwendige Aktion um einen Reward zu erhalten. In diesem Beispiel war die Umgebung wie folgt: Knopf 2 öffnet die Tür für acht Zeitschritte und Knopf 1 für fünf Zeitschritte.

Fazit Bei dem qualitativen Vergleich hat sich gezeigt, dass mit manchen der anfänglich bedachten Strategien die wirklichen Aktionen nicht oder nur bedingt in den Features zu finden waren. Dies begründet sich dadurch, dass das Öffnen der Tür relativ selten probiert wird. Bei der Random Strategie wird die Tür beispielsweise durchschnittlich nur jedes vierte Mal ausgewählt, das bedeutet, dass wenn zuerst alle Knöpfe gedrückt werden die Tür auf jeden Fall geöffnet werden kann. Dies führt dazu, dass die „wirklichen" Features zum Teil gar nicht gelernt werden können, da der Zusammenhang nicht aus den Trainingsdaten hervorgeht. Trotzdem waren bei der gewählten Trainingsdatengröße von 1000 die Ergebnisse bei allen Policies relativ gut. Am Schlechtesten schnitten AlterningRandom und TwoButtons ab, da bei diesen Strategien immer nur in geraden Schritten beziehungsweise in jedem dritten Schritt die Tür geöffnet wird, somit können manche Features also nie getestet werden. Wenn Knopf 2 die Tür beispielsweise für zwei Zeitschritte öffnet, wird das Feature mit AlterningRandom nie ermittelt werden können, da im ungeraden Zeitschritt der Knopf gedrückt wird und im Geraden die Tür geöffnet wird. Dies bedeutet, dass nur eine temporale Abhängigkeit von einem Zeitschritt gezeigt werden kann. Die gelernten Featuresets der anderen Strategien enthielten die Ergebnisse meist, am Zuverlässigsten war jedoch die Strategie DoorMoreOften.

4.2.2 Quantitative Analyse

Zum quantitativen Vergleich sollten die Durchschnittswahrscheinlichkeiten („mean likelihoods") der mit Hilfe der Strategien erzeugten Modelle verglichen werden. Dazu wurden für jede Strategie in mehreren Iterationen jeweils Trainingsdaten der Größen 25, 50, 100, 250, 500 und 1000 gesammelt und innerhalb jeder Iteration für jede der Größen ein Modell gelernt. Mit diesem Modell wurde dann jeweils zehnmal die mean-likelihood auf einem Testdatensatz der Größe 1000, welcher zufällig erstellt wird, berechnet und von diesen zehn Durchläufen der Durchschnitt bestimmt. Danach wurden die Daten abgespeichert, um diese zum Erstellen verschiedener Plots zu benutzen.

Algorithmus 4.1 Pseudocode zur Veranschaulichung des Datensammelprozess

```
generate environment
for all policies do
    for set_size in [25, 50, 100, 250, 500, 1000] do
        generate traindata with policy and set_size
        setup model of PULSE
        fit data
        tune the weights
        for n in 1...10 do
            generate testdata
            likelihood += mean_likelihood_of_model
        end for
        mean = likelihood/10
        append run_no, set_size, mean and the policy to file
    end for
end for
```

Vergleich der Strategien

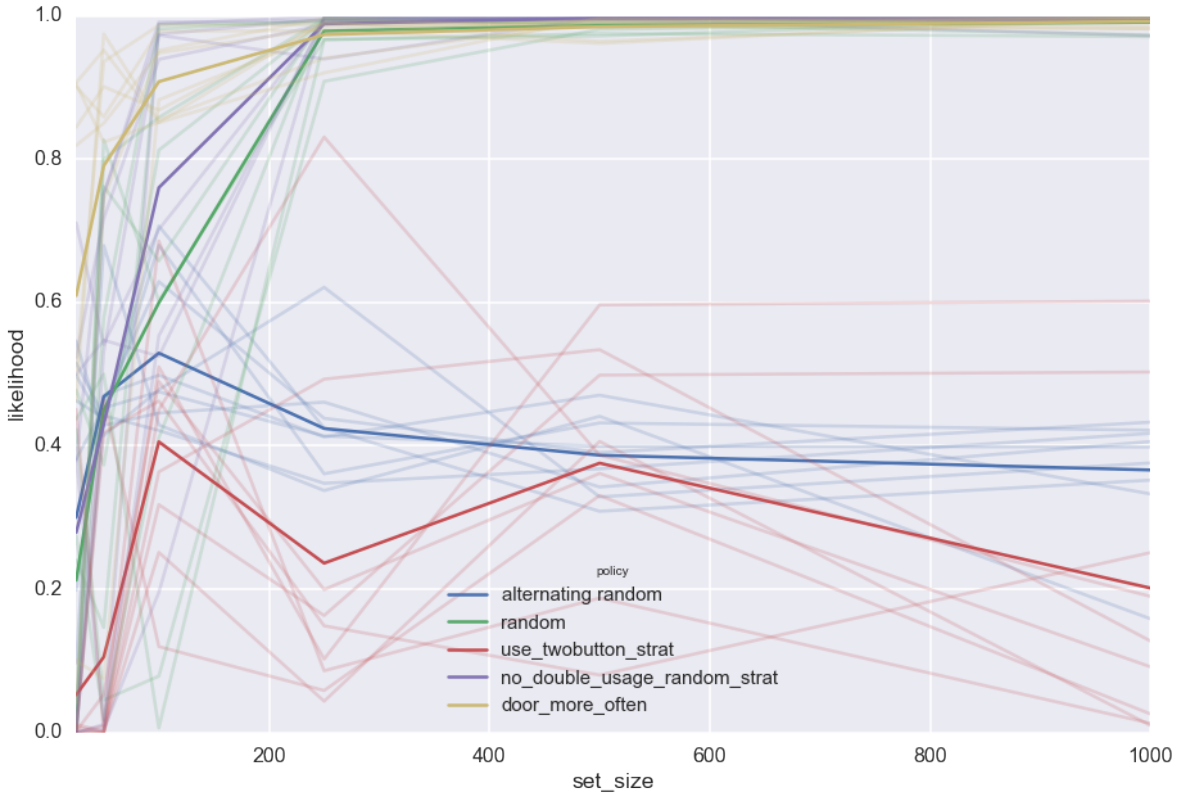

Abbildung 4.3: Plot mit einer Umgebung in der eine der Aktionen die Tür für 5 Zeitschritte öffnet.

Wie Abbildung 4.3 visualisiert, kommt die DoorMoreOften-Strategie mit kleineren Featuresets schon zu einem guten Ergebnis. Dies begründet sich dadurch, dass bei dieser Strategie zehnmal so oft die Tür geöffnet als ein Knopf gedrückt wird. Somit kann viel genauer bestimmt werden, wie lange eine Tür nach Drücken eines Knopfes wirklich geöffnet ist.

Die Strategien NoDoubleUsage und Random kommen mit einem Datensatz der groß genug ist auch zu einem sehr guten Ergebnis.

Lediglich AlterningRandom sowie die TwoButtonStrat führen zu weniger guten Ergebnissen, was auf die schon bereits beschriebenen Eigenschaften dieser Strategien zurückzuführen ist.

Das teilweise vorkommende Abfallen der Werte bei größeren Datensätze erklärt sich dadurch, dass bei den kleinen Datensätze die benötigten Features gar nicht vorhanden sind und die Vorhersage somit zu einem besseren Wert kommen kann, als wenn mehrere Features

vorhanden sind. Außerdem ist zu beachten, dass teilweise auch die Qualität der zufällig ermittelten Trainings- sowie Testdaten zu schlechteren Ergebnissen führen kann.

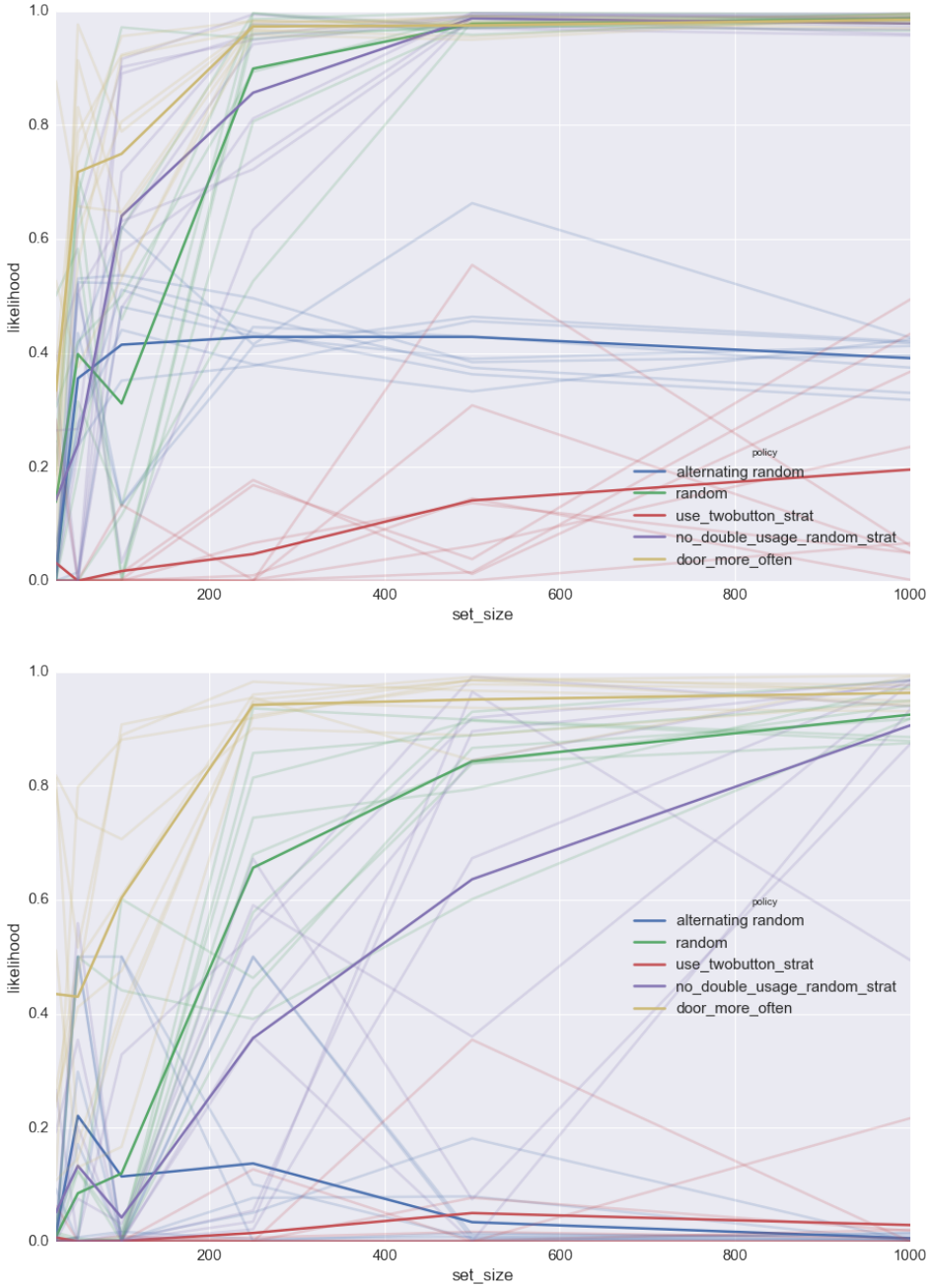

Abbildung 4.4: Weitere Abläufe mit anderen Umgebungen: Oben je ein Knopf für 5 und 2 Zeitschritte. Unten für je 5 und 7 Zeitschritte.

5 Umsetzung auf dem PR2

In diesem Kapitel wird beschrieben wie das bisher erarbeitete auf dem PR2 umgesetzt wurde.

5.1 Anforderungen

Um die beschriebene Aufgabe bewältigen zu können muss der PR2 zu Folgendem in der Lage sein:

1. Erkennen der Knöpfe sowie der Türklinke
2. Erreichen der Knöpfe und Tür
3. Drücken der Knöpfe
4. Öffnen der Tür
5. Validierung seiner Aktionen

5.2 Umsetzung

5.2.1 Perzeption

Für die Perzeption wurde das in Abschnitt 3.4 vorgestellte ar_track_alvar-Paket verwendet. Wie bereits beschrieben, wurden zur Erleichterung des Aufbaus statt Knöpfen lediglich Marker verwendet. Dies lässt sich dadurch begründen, da die Erkennung der Marker im Vergleich zu anderen Perzeptionsmethoden außerordentlich gut funktioniert und somit für einen Ablauf, der so robust wie möglich funktionieren soll, die beste Möglichkeit darstellt. Des Weiteren wird die Position der Marker sehr genau bestimmt, wodurch es lediglich geringe Abweichungen gibt. Zur Erkennung der Türklinke wurde ein kleiner Marker an dieser angebracht.

5.2.2 Navigation

Eigentlich sollte für die Navigation das in Abschnitt 3.4 beschriebene Paket pr2_2dnav verwendet werden, da dieses eine Hinderniserkennung sowie einen lokalen und globalen Planer zur Verfügung stellt. Somit kann die Navigation im Raum selbst mit vorhandenen Hindernissen erfolgen. Leider gab es Probleme bei der Verwendung mit pr2_2dnav und dem verwendeten Controller. Bei diesem werden für jedes nicht angesteuerte Gelenk, die Motoren deaktiviert. Dies führt beispielswiese dazu, dass die Arme sich lose bewegen. Eigentlich stellt dies für die tatsächliche Navigation kein Problem dar, da nur die Räder angesteuert werden müssen wenn die Arme, aufgrund dessen, dass sie sich lose bewegen, jedoch in das Sichtfeld des Laserscanners kommen, so erkennt der PR2 diese als mögliches Hindernis, was wiederum dazu führt, dass er beginnt einen Plan zu erstellen um dieses „Hindernis“ zu umfahren. Offensichtlicherweise ist es ihm aber unmöglich seine eigenen Arme zu umfahren, was dazu führt, dass sich der PR2 um sich selbst dreht. Dieses Problem lässt sich auch nicht durch ansteuern der Armgelenke über das Controllerinterface lösen, da der Controller dann alle anderen Gelenke blockiert. Dies führt dazu, dass pr2_2dnav die Räder nicht mehr ansteuern kann und der Roboter auf der Stelle stehen bleibt.

Daher wurde auf die Hinderniserkennung verzichtet und für die Navigation direkt das Actioninterface verwendet. Dies ist jedoch kein großes Problem, da der PR2 zur Sammlung der Daten für den Algorithmus, keine großen Strecken zurücklegen muss. Er muss lediglich zu einer Startposition, dann entweder zu einem Knopf oder der Tür und anschließend wieder zurück zur Startposition navigieren. Die Startposition ist dabei eine fest definierte Stelle, die für jeden Durchlauf gleich ist.

Das Navigieren zu den Knöpfen beziehungsweise der Tür wird anhand der Positionsdaten, die das ar_track_alvar-Paket zur Verfügung stellt, durchgeführt. Der PR2 positioniert sich dadurch direkt vor dem entsprechenden Marker in einem Abstand von einem halben Meter.

5.2.3 Drücken der Knöpfe

Nachdem sich der PR2 von seiner Startposition zu dem entsprechenden Knopf bewegt hat und in seiner Grundposition einen halben Meter vor diesem steht, beginnt er den Knopf zu drücken. Hierzu wird über den verwendeten Controller eine Aktivität gestartet, bei welcher der PR2 seinen rechten Endeffektor relativ zu der Markerposition an der Wand ausrichtet und diesen dann mit der Spitze des Endeffektors berührt.

Wie bereits beschrieben, befindet sich an der Spitze ein sogenannter Force-Torque-Sensor, welcher beim Überschreiten eines Thresholdes ein Signal zurück gibt. Eigentlich sollte dieser verwendet werden, um beim Erhalten des Signals die Aktion des Knopfdrucks zu beenden und zu speichern, dass dieser erfolgreich war. Falls das Signal nicht empfangen wird, so wird nichts gespeichert und die nächste Aktion beginnt.

Da es Probleme mit dem Sensor gab, wurde entschieden auf das Validieren der Aktion zu verzichten und davon auszugehen, dass der Knopfdruck immer erfolgreich ist.

Nach dem Ausführen des Knopfdrucks nimmt der Roboter wieder die Grundposition ein und bewegt sich in dieser zurück zur Startposition.

5.2.4 Tür öffnen

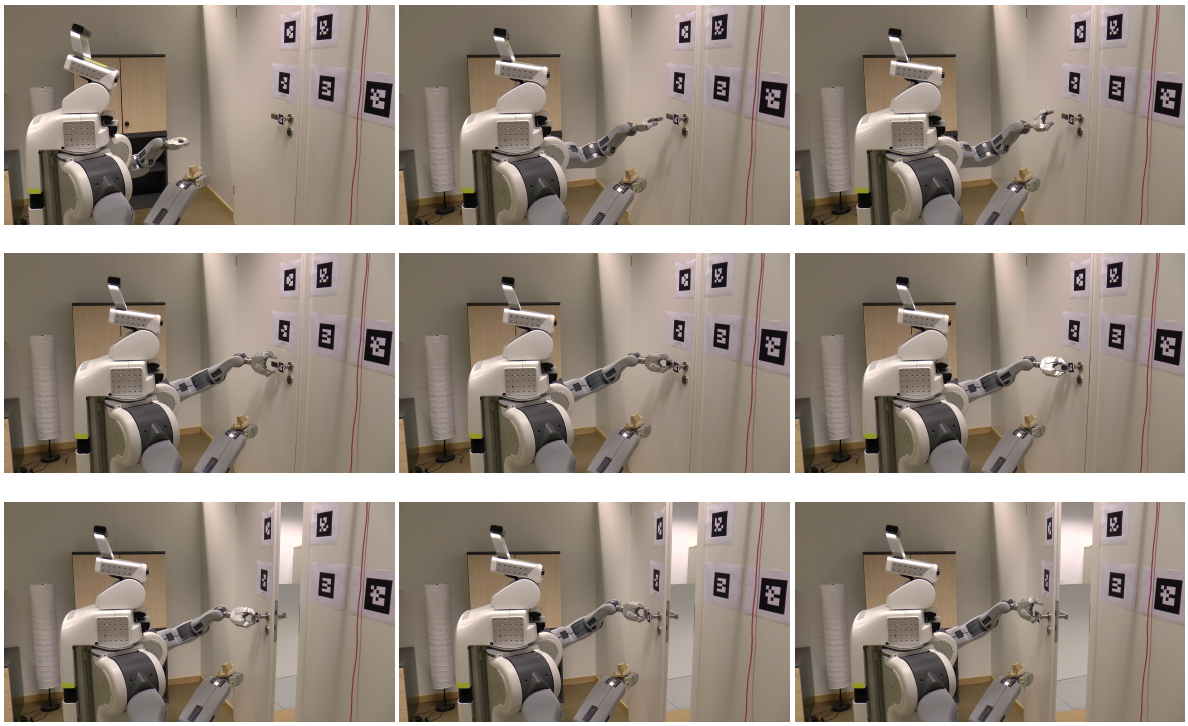

Abbildung 5.1: Ablauf des Öffnens der Tür

Analog zum Knopfdruck befindet sich der PR2 vor dem Öffnen der Tür in seiner Grundposition einen halben Meter von der Türklinke entfernt (wie im ersten Bild zu sehen). Um diese zu betätigen wird eine Folge von verschiedenen Aktionen ausgeführt. Der PR2 richtet seinen linken Gripper mit einem Offset von 10 cm nach links und 5 cm nach vorn zur Tür aus und öffnet den Gripper dabei. Dann richtet er ihn so aus, dass er die Klinke fast gerade von vorn greifen kann und bewegt diesen dann 5cm nach vorn und schließt den Gripper wieder. Im Normalfall, hat er die Klinke dann fest umschlossen. Um nun die Tür zu öffnen wird der Gripper viermal, immer abwechselnd um 10° gedreht und um einige Zentimeter nach unten bewegt. Danach „zieht“ der PR2 seinen Arm zu sich heran. Da die Tür keinen Schließmechanismus hat, wird diese von Hand zugehalten. Dazu wird die Klinke nach oben gedrückt, sodass sich diese, bei verschlossener Tür, nicht nach unten bewegen lässt. Der PR2 kann einen Kraftsensor in seinem Grippergelenk verwenden, um bei einem zu großen

Kraftaufwand seine Aktion abzubrechen. Sollte dieser Abbruch erfolgen, wird die Aktion mit einem Reward von 0 gespeichert, ansonsten mit 1. Nachdem die Tür geöffnet wurde oder ein Abbruch erfolgt ist, wird der Gripper wieder geöffnet und der Arm nach hinten bewegt.

Da der PR2 kurz vor Abschluss der Arbeit einen Defekt hatte, konnte der Kraftsensor nicht mehr in Gang gebracht werden, weshalb dieser Punkt zum Abschluss der Arbeit noch offen steht.

Im Anschluss wird die Grundposition erneut eingenommen und wieder zur Startposition navigiert. Falls die Tür geöffnet wurde, muss diese für den weiteren Verlauf des Datensammelns wieder manuell geschlossen werden.

5.2.5 Ablauf

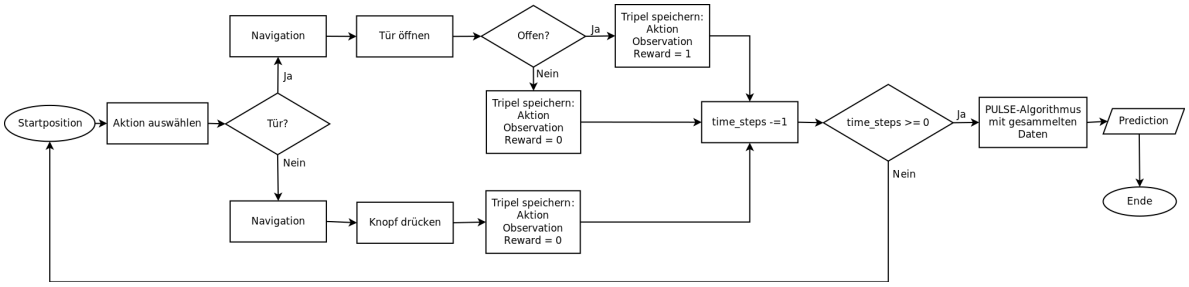

Abbildung 5.2: Flussdiagramm zum Ablauf des Datensammelprozesses

In Abbildung 5.2 ist der Ablauf des Datensammelprozesses zu sehen. Der Roboter startet aus der Startposition und wählt dann anhand einer der zuvor definierten Strategien (vgl. Kapitel 4) eine Aktion aus. Diese Aktion wird dann, wie bereits beschrieben, ausgeführt. Nach dem Ausführen werden das jeweilige Aktion-Observation-Reward-Tripel $(A \times O \times R)$ abgespeichert. Der ganze Ablauf wird für k-time-steps, die zu Beginn des Ablaufs festgelegt werden, durchgeführt. Nachdem k-Aktionen ausgeführt wurden, wird der Ablauf beendet und die Daten an den PULSE-Algorithmus übergeben, um mit Hilfe von diesem eine Vorhersage über das Weltmodell zu erhalten.

5.2.6 Probleme mit dem PR2

Der grundsätzliche Ablauf des Ausführens von Aktionen funktioniert zum Zeitpunkt der Abgabe dieser Arbeit relativ robust. Leider gab es jedoch Probleme mit den Force-Torque-Sensoren zum Verifizieren der Aktionen.

Deshalb wurde schon relativ frühzeitig auf das Verifizieren des Knopfdruckes verzichtet. Es wäre zwar für einen realitätsnäheren Ablauf gut, spielt aber für den hier definierten keine Rolle, da keine Rewards für das Drücken der Knöpfe verteilt werden. Es reicht somit aus die Aktion einfach als Solche zu speichern.

Beim Öffnen der Tür wäre diese Verifikation wichtig gewesen. Daher wurden hierbei verschiedene Ansätze getestet, um auch ohne die Sensoren auszukommen. Da der PR2 aber in den letzten Wochen vor Abschluss dieser Arbeit nicht mehr funktionstüchtig war, konnten weder die Sensoren in Gang gebracht werden, noch konnten andere Ansätze getestet und umgesetzt werden. Somit bleibt der Punkt zur Verifikation der Aktionen auch nach Abschluss dieser Arbeit offen.

Unglücklicherweise wurden, als der PR2 noch funktionstüchtig war, lediglich kleinere Funktionstests durchgeführt, was bedeutet, dass der Algorithmus nie auf mit dem PR2 gesammelten Daten ausgeführt wurde. Diese Experimente sollten erst durchgeführt werden, wenn entweder klar gewesen wäre, dass die Verifikation nicht mehr umsetzbar ist oder diese funktionieren würde.

Da der PR2 jedoch zu diesem Zeitpunkt bereits nicht mehr verwendbar war, konnten die wirklichen Tests und Experimente in Verbindung mit dem Algorithmus nicht mehr ausgeführt werden.

6 Zusammenfassung und Ausblick

Zu Beginn dieser Arbeit wurde die Relevanz und das Ziel dieser Arbeit beschrieben sowie ähnliche Ansätze im Kontext der Robotik beleuchtet. Anschließend wurde zunächst der nötige theoretische Hintergrund geschaffen, um die Funktionsweise des angewendeten Algorithmus verstehen zu können. Hierbei wurde Basiswissen zum Thema Reinforcement Learning und Markov Prozesse vorgestellt und zuletzt die Funktionsweise des PULSE Algorithmus beschrieben.

Nachdem der theoretische Hintergrund dargelegt wurde, wurde Hintergrundwissen zur praktischen Umsetzung erläutert. Der PR2 und das Robot Operating System (ROS) wurden vorgestellt und Funktionsweisen erklärt. Zudem wurden die relevanten, zusätzlich verwendeten Pakete vorgestellt.

Im praktischen Teil der Bachelorarbeit wurde zuerst die entwickelte Simulationsumgebung beschrieben, mit Hilfe welcher sowohl Trainings- als auch Testdaten zur Überprüfung des PULSE-Algorithmus mit verschiedenen Strategien und Datensätzen für die verwendete Umgebung gesammelt wurden. Mit diesen wurde sowohl ein qualitativer als auch quantitativer Vergleich der verschiedenen Ansätze vollzogen und aufgezeigt, inwiefern sich diese zur Verwendung eignen.

Im letzten Kapitel der Arbeit wurden zunächst die Anforderungen, die der entwickelte Ablauf auf dem PR2 erfüllen muss aufgezeigt und anhand von diesen die praktische Umsetzung auf dem PR2 beschrieben. Wie am Ende des Kapitels beschrieben, konnten aufgrund verschiedener Probleme nicht alle Anforderungen erfüllt werden und wegen eines Defekts am PR2 auch keine praktischen Testdurchläufe durchgeführt werden. Trotzdem konnte mit Hilfe der Simulation aufgezeigt werden, dass der PULSE-Algorithmus zum Lösen des definierten Problems geeignet ist und auch eine relativ gute Perfomanz mit sich bringt. Auch der Ablauf auf dem PR2 konnte umgesetzt werden, lediglich die offenen Punkte zur Validierung der Daten sind offen geblieben. Des Weiteren stehen auch noch wirkliche Testläufe aus, jedoch konnte in kleineren Tests schon aufgezeigt werden, dass die Funktionsweise des PR2 zum Ausführen der Aktionen gegeben ist, was bedeutet, dass auch die praktischen Durchläufe zum Sammeln von Trainingsdaten kein Problem darstellen dürften.

Ausblick

Mit funktionierenden Force-Torque-Sensoren wäre der PR2 in der Lage seine Aktionen zu validieren. Somit wäre der nächste Schritt, diese Sensoren in Gang zu bringen um die Daten wirklich autonom sammeln zu können.

Ein weiterer Schritt wäre die Verwendung vom beschriebenen pr2_2dnav Paket in Verbindung mit dem MLR Controller umzusetzen, um eine robuste Navigation gewährleisten zu können und es dem PR2 zu ermöglichen, sich autonom in einer unbekannten Umgebung bewegen zu können. Wenn diese Schritte erreicht sind, wäre die nächste sinnvolle Weiterentwicklung komplett auf Marker zu verzichten und die Perzeption mit Hilfe von anderen Paketen, die zum Beispiel Knöpfe und die Türklinke auf eine andere Art erkennen können, umzusetzen. Sollte dies funktionieren, könnte der PR2 in komplett unbekannten Umgebungen eingesetzt werden und es wäre denkbar komplexere Aufgaben zu testen. Bei diesen Aufgaben könnte jedoch der Rechenaufwand ein Problem darstellen, sodass das Lernen sehr lange dauern könnte.

Außerdem stellt sich die Frage, ob es sinnvoll ist ein Weltmodell mit Hilfe von POMDPs und Reinforcement Learning zu erlernen oder ob es für die meisten Anwendungen nicht sinnvoller ist dem Roboter das „Wissen“ direkt mit auf den Weg zu geben, da der Aufwand die Daten zu sammeln und auszuwerten enorm sein kann. Es gibt außerdem viele einschränkende Faktoren, wie zum Beispiel ungenaue Sensordaten und lange Rechenzeiten. Des Weiteren müssen die Aktionen vorher fest definiert werden, sodass auch deren Abhängigkeit definiert werden kann. Dies sollte kein Problem darstellen, da die meisten Roboter in kleinen Domänen agieren.

Lediglich wenn die Aufgabe des Roboters zum Beispiel das Verlassen eines Labyrinths, welches jedes Mal eine andere Form annimmt, ist, wäre der hier aufgezeigte Anwendungsfall eine sinnvolle Methode diese Art von Aufgaben zu lösen.

Abbildungsverzeichnis

2.1 Beispiel Markov Kette mit drei Zuständen 8
2.2 Beispiel Markov Decision Process mit 3 Zuständen und 2 Aktionen 10

3.1 Versuchsaufbau und PR2 16
3.2 Beispiel Verbindungsaufbau zwischen ROS Knoten 18
3.3 Funktionsweise des ROS Message Passing Systems 18

4.1 UML Diagramm der Simulierten Welt 22
4.2 Qualitative Auswertung 25
4.3 Qualitativer Vergleich 28
4.4 Qualitativer Vergleich II 29

5.1 Ablauf des Öffnens der Tür 33
5.2 Flussdiagramm zum Ablauf des Datensammelprozesses 34

Verzeichnis der Algorithmen

2.1 PULSE Algorithmus aus [LT15] 13

4.1 Pseudocode zur Veranschaulichung des Datensammelprozess 27

Literaturverzeichnis

[2dn15] PR2_2dnav package ROS Wiki, 2015. URL http://wiki.ros.org/pr2_2dnav. (Zitiert auf Seite 19)

[art15] AR Track Alvar on ROS Wiki, 2015. URL http://wiki.ros.org/ar_track_alvar. (Zitiert auf Seite 19)

[Fin15] V. T. R. C. of Finland. ALVAR package description, 2015. URL http://virtual.vtt.fi/virtual/proj2/multimedia/alvar/index.html. (Zitiert auf Seite 19)

[KHL08] H. Kurniawati, D. Hsu, W. S. Lee. SARSOP: Efficient Point-Based POMDP Planning by Approximating Optimally Reachable Belief Spaces. 2008. (Zitiert auf Seite 3)

[KLC98] L. P. Kaelbling, M. L. Littman, A. R. Cassandra. Planning and acting in partially observable stochastic domains. *Artificial Intelligence 101 (1998) 99–134*, 1998. (Zitiert auf Seite 10)

[LT15] R. Lieck, M. Touissant. Discovering temporally extended features for reinforcement learning in domains with delayed causalities. 2015. (Zitiert auf den Seiten 0, 2, 12, 13, 21 und 41)

[PMP+03] J. Pineau, M. Montemerlo, M. Pollack, N. Roy, S. Thrun. Towards robotic assistants in nursing homes: Challenges and results. *Robotics and Autonomous Systems 42 (2003) 271–281*, 2003. (Zitiert auf Seite 3)

[PR215] Willow Garage PR2 Overview and Tech specs, 2015. URL https://www.willowgarage.com/pages/pr2/overview. (Zitiert auf Seite 16)

[ros15a] About ROS page, 2015. URL http://www.ros.org/about-ros/. (Zitiert auf Seite 17)

[ros15b] ROS core components, 2015. URL http://www.ros.org/core-components/. (Zitiert auf Seite 17)

[ros15c] ROS Wiki: Concepts, 2015. URL http://wiki.ros.org/ROS/Concepts. (Zitiert auf den Seiten 17 und 19)

[ros15d] ROS Wiki: Master, 2015. URL http://wiki.ros.org/Master. (Zitiert auf Seite 17)

[SB98] R. S. Sutton, A. G. Barto. *Reinforcement Learning: An Introduction.* The MIT Press Cambridge, Massachusetts London, England, 1998. (Zitiert auf den Seiten 5, 7, 9 und 11)

[SV04] M. T. J. Spaan, N. Vlassis. A point-based POMDP algorithm for robot planning. *Proceedings of the 2004 IEEE lnternational Conference on Robotics & Automation New Orleans. LA Aprll2004*, 2004. (Zitiert auf Seite 3)

[VT15] N. A. Vien, M. Toussaint. POMDP Manipulation via Trajectory Optimization. 2015. (Zitiert auf Seite 3)

Alle URLs wurden zuletzt am 08. 11. 2015 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift