Konsistenzprüfung für robotische Wahrnehmung auf Basis von Simulation und logischer Schlussfolgerung

Bachelorarbeit

Andreas Romero Früh

Universität Bremen

Artificial Intelligence

Bachelorarbeit

Konsistenzprüfung für robotische Wahrnehmung auf Basis von Simulation und logischer Schlussfolgerung

Andreas Romero Früh

16. Juli 2015

Gutachter: Prof. Michael Beetz Ph.D.
2. Gutachter: Dr. Karsten Hölscher
Betreuerin: Gayane Kazhoyan

Konsistenzprüfung für robotische Wahrnehmung
auf Basis von Simulation und logischer Schlussfolgerung
© 2015 Andreas Romero Früh

Bachelorarbeit, eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Duales Studium Informatik*
im Fachbereich 03: Mathematik/Informatik,
Arbeitsgruppe Künstliche Intelligenz
der Universität Bremen

Gutachter: Prof. Michael Beetz Ph.D.
2. Gutachter: Dr. Karsten Hölscher
Betreuerin: Gayane Kazhoyan
Bindung: OCE Druck- und Kopierservice der
Staats- und Universitätsbibliothek Bremen

Kurzfassung

Ein im menschlichen Alltag eingesetzter autonomer mobiler Haushaltsroboter hat viele verschiedene komplexe alltägliche Manipulationsaufgaben durchzuführen. Ein Haushalt ist im Gegensatz zu einer Fabrikhalle, in der ein Industrieroboter eingesetzt wird, eine durch eine hohe Dynamik geprägte Umgebung. Bedingt durch die menschliche Präsenz ändert sich der Zustand der Welt kontinuierlich in einer nicht deterministischen Art und Weise. Um in einem solchen Umfeld seine Aufgaben zufriedenstellend durchführen zu können, muss ein Roboter auf Änderungen in der Umgebung reagieren und sein Verhalten anpassen können. Wichtige Bestandteile für die korrekte Wahrnehmung und Modellierung seiner Umgebung sind die Perzeption und die Wissensverarbeitung, welche in klassischen Robotersystemen voneinander getrennt werden. Die Aufgabe von klassischen Perzeptionssystemen besteht oft darin, die einzelnen Objekte, die für die aktuell vom Roboter durchzuführende Manipulationsaufgabe von Belang sind, aus den zu einer Szene gehörenden Sensordaten zu segmentieren und zu klassifizieren. Bereits erfasste Daten aus der Vergangenheit werden nicht weiter beachtet. Im Rahmen dieser Arbeit wird eine Komponente für das Perzeptionssystem RoboSherlock realisiert, die Perzeption und Reasoning miteinander kombiniert. Die Ergebnisse der verschiedenen in RoboSherlock implementierten Perzeptionsalgorithmen werden durch die Verwendung einer physikbasierten Simulation und durch logisches Schlussfolgern auf Konsistenz geprüft und gegebenenfalls korrigiert. Daten aus vergangenen Iterationen werden in den Reasoning-Prozess miteinbezogen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Lesehinweise

1. In dieser Arbeit werden viele fachspezifische Begriffe und Abkürzungen verwendet. Begriffe, die nicht im Text erläutert werden, sind im Glossar auf Seite 81 gelistet; eine Liste der verwendeten Abkürzungen findet sich auf Seite 79.

2. Weiterhin werden vielfach Anthropomorphismen wie z. B. „*Der Roboter weiß ...*“, „*Der Roboter denkt ...*“ oder „*Der Roboter erkennt ...*“ verwendet. Dabei handelt es sich um metaphorische Sprache, die in der Robotik häufig eingesetzt wird und die Lesbarkeit der Texte erhöht.

3. In dieser Arbeit wurde bei der Bildung des Plurals auf das generische Maskulinum zurückgegriffen.

4. Folgende Formatierungen wurden zur Hervorhebung im Text verwendet:

   - Fremdwörter, welche im weiteren Text erläutert werden, werden bei ihrer ersten Verwendung *kursiv* geschrieben.

   - Abkürzungen und Begriffe, die im Glossar erläutert werden oder in der Liste der Abkürzungen erscheinen, erscheinen in dieser Farbe.

   - `Schreibmaschinenschrift` wird für Quelltext, Variablennamen, Pfadangaben und Links verwendet.

1. Einleitung

Ein im menschlichen Alltag eingesetzter autonomer Haushaltsroboter kann vielseitig verwendet werden. Er kann bei der Bewältigung des Haushalts helfen und Reinigungsaufgaben übernehmen, er kann beim Tischdecken, beim Aufräumen oder beim Abwaschen helfen – die Möglichkeiten sind unbegrenzt. In der Industrie eingesetzte Roboter müssen häufig klar definierte Arbeiten mit fixen Parametern verrichten. Das Umfeld, in dem solche Roboter eingesetzt werden, ist in der Regel vom Menschen isoliert und keinen unvorhersehbaren Einflüssen unterlegen. Im Gegensatz zu einem autonomen Roboter muss ein Industrieroboter nicht mit plötzlichen Hindernissen oder Änderungen in seinem Umfeld rechnen. Seine sich wiederholende Aufgabe führt er stets auf die selbe Art und Weise durch.

Ein autonomer Haushaltsroboter würde mit einem solchen Vorgehen in einem von Menschen bewohnten Haushalt zum Scheitern verurteilt sein, da das menschliche Umfeld durch eine hohe Dynamik geprägt ist. Gegenstände werden z. B. durch menschliche Aktionen bewegt, befinden sich nicht an ihrem richtigen Platz oder sind dem Roboter ein Hindernis bei seinen Bewegungen. Daher ist es wichtig, dass ein Haushaltsroboter seine Umgebung fortwährend wahrnimmt, um Änderungen zu erkennen und darauf reagieren zu können.

Für das Erfassen der Umgebung stehen verschiedene Sensoren wie z. B. Laserscanner, RGB-Kameras und Tiefenkameras zur Verfügung. Die von diesen Sensoren erhaltenen Daten müssen vom Roboter interpretiert werden, um ein Modell seiner Umgebung erstellen zu können. Anhand dieses Modells und der verfügbaren Kontrollprogramme kann ein Roboter seine Entscheidungen treffen. Ein fehlerhaftes Modell führt unausweichlich zu einem fehlerhaften Verhalten seitens des Roboters.

Für die Wahrnehmung der Umgebung ist in der Robotik die Perzeption zuständig. Das Ergebnis ihrer Analyse der Sensordaten ist ein wichtiger Bestandteil in einem Robotersystem. Ohne eine korrekte Interpretation der Sensordaten kann ein autonomer Haushaltsroboter seine Aufgaben nicht ausführen. Ein weiterer wichtiger Bestandteil autonomer Robo-

tersysteme ist die Wissensverarbeitung mit der dazugehörigen Wissensrepräsentation und dem Reasoning (englisch für *Schlussfolgern, logisches Denken*). Ein Roboter, der in einem menschlichen Haushalt zum Einsatz kommt, benötigt eine umfassende Wissensbasis, auf deren Grundlage er Objekte erkennen und Entscheidungen treffen kann. In klassischen Systemen die beiden Komponenten zur Perzeption und zur Wissensverarbeitung voneinander getrennt und werden von einer High-Level-Komponente aufgerufen, überwacht und gesteuert.

Die Aufgabe dieser Arbeit besteht darin, eine bereits existierende, aus verschiedenen Komponenten bestehende Perzeptionspipeline um eine Reasoning-Komponente zu erweitern. Sie soll die Ergebnisse der anderen Komponenten auf Konsistenz prüfen und gegebenenfalls korrigieren, um autonomen Haushaltsrobotern konsistentere Perzeptionsdaten anbieten zu können, auf deren Grundlage sie mit einer höheren Sicherheit agieren können. Zu diesem Zweck soll eine bereits existierende Physiksimulation mit bereitgestellten Reasoning-Methoden eingesetzt werden. Bei der Perzeptionspipeline handelt es sich um RoboSherlock (beschrieben in Abschnitt 3.3 auf Seite 23), einem Framework, das von der AG Künstliche Intelligenz (IAI) an der Universität Bremen entwickelt wird. Es erlaubt je nach Anwendungsfall die Kombination verschiedener, sich untereinander ergänzender Perzeptionsalgorithmen. Bei der Physiksimulation handelt es sich um eine Komponente des in der selben Arbeitsgruppe entwickelten CRAM-Frameworks. Das Paket `bullet_reasoning` (beschrieben in Abschnitt 3.2.2 auf Seite 20) bietet eine Schnittstelle zu dieser Simulation an.

1.1. Gliederung der Arbeit

Im folgenden Kapitel werden bestehende Perzeptions- und Reasoning-Systeme umrissen, um anschließend die Forschungsfrage zu stellen. In Kapitel 3 werden die für diese Arbeit grundlegenden Begriffe und Systeme erläutert. In Kapitel 4 wird die im Rahmen dieser Arbeit erstellte Software im Zusammenhang mit den bereits vorhandenen Systemen beschrieben. Kapitel 5 befasst sich mit dem Testen und der Evaluation der im Zuge dieser Arbeit entwickelten Software. Abschließend wird in Kapitel 6 die Forschungsfrage beantwortet und ein Ausblick auf mögliche Verbesserungen und Erweiterungen der entwickelten Software gegeben.

2. Forschungsfrage und Abgrenzung

Das Ziel dieser Arbeit ist, die klassische Perzeption, wie sie im Kontext von mobilen, im Alltag eingesetzten Robotersystemen verwendet wird, mit auf physikbasierter Simulation beruhenden Reasoning-Methoden zu ergänzen und zu verbessern. Im Mittelpunkt dieser Arbeit stehen autonome, mobile Haushaltsroboter, welche sich mit alltäglichen Aufgaben wie z. B. dem Decken eines Tisches beschäftigen. In einem solchen Umfeld eingesetzte Roboter besitzen meist eine mobile Basis, sodass sie sich frei in ihrer Umgebung bewegen können, und verschiedene Sensoren (z. B. Kameras, Laserscanner und Drehmomentsensoren), die ihnen bei der Orientierung, der Perzeption und der Durchführung ihrer Aufgaben helfen.

Damit ein solcher Roboter überhaupt agieren kann, ist es nötig, dass er seine Umgebung kennt und Änderungen in seiner Umwelt wahrnimmt. Weiterhin muss er in der Lage sein, Objekte, die er manipulieren soll, zu erkennen und zu klassifizieren. Mit der Objekterkennung beschäftigt sich in der Robotik die Perzeption. Viele Perzeptionssysteme erhalten als Eingabe ein zweidimensionales Farbbild und/oder ein dreidimensionales Tiefenbild von den Kameras eines Roboters. In diesen Bildern werden in der Perzeptionspipeline durch Anwendung verschiedener Algorithmen Objekte segmentiert und durch Klassifizierung erkannt. Die Ergebnisse werden an weiterverarbeitende Prozesse wie z. B. die Wissensrepräsentation oder das High-Level-Planning weitergegeben. Dort werden die Daten weiterverarbeitet und bilden eine wichtige Grundlage für den reibungslosen Betrieb eines autonomen Roboters.

2.1. Existierende Systeme

In diesem Abschnitt wird ein Überblick über bereits existierende Perzeptions- und Reasoning-Systeme gegeben. Weiterhin werden einige Systeme vorgestellt, die beide Komponenten miteinander verbinden.

2.1.1. Perzeption

In der Robotik gibt es insbesondere auf dem Gebiet der autonomen Roboter keinen omnipotenten Perzeptionsalgorithmus, der in jeder Situation perfekte Ergebnisse liefert. Daher gibt es viele verschiedene Versuche, auf die unterschiedlichen, im Alltag eines Haushaltsroboters auftretenden Situationen einzugehen.

In einigen Ansätzen, wie z. B. in [Kla+09], wird *CAD-Model-Matching* verwendet. Bei diesem Verfahren existiert eine Datenbank mit 3D-Modellen von bekannten Objekten. Aus einem zu analysierenden Tiefenbild werden potenziell zu Objekten gehörende Cluster extrahiert und mit den in der Datenbank vorhandenen 3D-Modellen verglichen, um Übereinstimmungen zu finden. Dies ermöglicht dem Roboter, ihm bereits bekannte Objekte zu identifizieren; unbekannte Objekte können nicht klassifiziert werden. Andere auf CAD-Model-Matching basierende Ansätze versuchen dem entgegenzuwirken, indem unbekannte Objekte mit 3D-Modellen aus einer im Internet verfügbaren Datenbank verglichen werden [KZM09].

Bei anderen Vorgehensweisen, wie in [CC12] beschrieben, wird ebenfalls vorausgesetzt, dass eine Datenbank existiert, in der die Attribute der bekannten Objekte abgespeichert sind. Diese Daten werden gewonnen, indem jedes Objekt mit einer Tiefenkamera aus verschiedenen Betrachtungswinkeln gescannt wird. Die dabei entstehenden Punktwolken werden dem untersuchten Objekt zugeordnet und in der Datenbank abgespeichert. Um die entsprechenden Objekte später bei einem Einsatz des Roboters wiederzuerkennen, werden die vom Roboter erfassten Punktwolken mit den abgespeicherten Punktwolken verglichen. Dabei werden Informationen zur geometrischen Form und zur Farbe eines Objekts sowie Beziehungen zwischen einzelnen Punktepaaren innerhalb der Cluster untersucht.

In [Mör+10] wird beschrieben, wie sowohl unbekannte Objekte erkannt als auch bekannte Objekte klassifiziert werden können. Im 2D-Bild einer Szene wird zunächst nach markanten Linien, Ecken und Kanten gesucht, aus denen im nächsten Schritt Konturen gebildet werden. Diese Konturen werden mit den zweidimensionalen Projektionen von einfachen, dreidimensionalen geometrischen Formen wie z. B. Quadern, Zylindern oder Kegeln verglichen. Unbekannte Objekte können so zumindest in ihrer Grundform erfasst werden, ohne dabei jedoch zu wissen, um welche Objektklasse es sich handelt. Bekannte Objekte können, wenn es sich um komplexe geometrische Formen handelt, anhand eines im Vorfeld bekannten 3D-Modells erkannt werden; Objekte mit primitiven geometrischen Formen können anhand von bestimmten Merkmalen in ihrer Geometrie (Features), ihrer Oberflächenbeschaffenheit

und ihrer Textur erkannt werden. Diese Merkmale werden a priori in einer Lernphase für jedes Objekt gesammelt und abgespeichert.

Ein in [CMS11] beschriebener Ansatz legt seinen Schwerpunkt auf die Erkennung von bekannten Objekten in komplexen Szenen. Zunächst wird in der Lernphase von jedem Objekt eine Reihe von Bildern aus verschiedenen Perspektiven aufgenommen. Aus diesen Bildern werden 3D-Modelle der Objekte angefertigt und zusammen mit in den Bildern erkannten objektspezifischen Features in einer Datenbank abgespeichert. Zur späteren Objekterkennung wird im benutzten Algorithmus Iterative Clustering-Estimation (ICE) angewandt. Das eigentliche Problem der Objekterkennung wird dabei in zwei Teile geteilt: *a*) zu bestimmten Regionen im Bild ein passendes Objekt zu finden, und *b*) für ein potenziell passendes Objekt die richtige Pose zu berechnen. ICE sucht im 2D-Bild zunächst nach Features, die in Zusammenhang miteinander stehen. Diese werden gruppiert und mit den Features der Objekte aus der Datenbank verglichen. Bei Übereinstimmung wird anhand des 3D-Modells die Pose des potenziell erkanntes Objekts berechnet und nach weiteren Features im Bereich des übereinstimmenden Objektes gesucht. Wird erkannt, dass möglicherweise mehrere solcher Gruppen zum selben Objekt gehören können, werden diese Gruppen von Features zusammengefasst. Dieser Vorgang wird iterativ wiederholt, bis sich keine weiteren Gruppen mehr bilden lassen. Jede Gruppe repräsentiert nun ein erkanntes Objekt.

Ein in [SBF13] beschriebenes Verfahren verwendet einen grundlegend anderen Ansatz. Hier werden Objektattribute wie Farbe, Form, Größe und materielle Beschaffenheit aus der natürlichen Sprache mit den dazugehörigen Objekten verknüpft und in einer Datenbank gespeichert. Wenn der Roboter den Auftrag „*Hol mir die rote Tasse*“ erhält, werden die von ihm wahrgenommenen Bildinformationen auf die ihm im Kommando gegebenen Objektmerkmale (Farbe: rot, Form: Zylinder) hin analysiert. Neben dem Erkennen von bekannten Objekten ist es somit auch möglich, bis dato unbekannte Objekte durch deren semantische Beschreibungen einzuordnen und zu identifizieren.

2.1.2. Reasoning

Da autonome Haushaltsroboter in Umgebungen eingesetzt werden, die durch eine hohe Dynamik geprägt sind und sie darin komplexe Manipulationsaufgaben durchführen sollen, ist die Wissensverarbeitung ein unerlässlicher Bestandteil eines solchen Robotersystems. Die Aufgaben der Wissensverarbeitung bestehen darin, neues Wissen zu erwerben, Wissen zu repräsentieren, Reasoning auf diesem Wissen zu betreiben und anhand der gewonnenen

Erkenntnisse neues Wissen zu generieren. Damit ist es möglich, generalisierte und flexible Steuerungsprogramme zu schreiben, die auf Veränderungen in der Umgebung reagieren können. [TJB10]

Eine große Herausforderung ist es, die Informationen aus der echten Welt in einer Struktur zu repräsentieren, die logisches Schlussfolgern zulässt. Dabei müssen Rohdaten von Kameras und anderen Sensoren, interpretierte Sensordaten wie Punktwolken und erkannte Objekte, bereits vom Roboter durchgeführte Aktionen und die dabei verwendeten Parameter und Pläne gespeichert werden. Dies gilt sowohl für Daten, die den aktuellen Zustand der Welt beschreiben als auch für Daten aus der Vergangenheit. All dieses Wissen aus den verschiedenen Quellen wird von der Wissensrepräsentation auf verschiedenen Abstraktionsebenen beschrieben, um den Daten eine semantische Bedeutung zu geben und Reasoning zu ermöglichen. Liegen die Daten in einer solchen Form vor, ist es dem Roboter möglich, fehlende Informationen durch Reasoning aus ihnen abzuleiten. Erhält ein Roboter z. B. von einer Person die Aufgabe „*Bring mir die Tasse*“, fehlen ihm wichtige Parameter. So weiß er nicht, welche Tasse er holen soll, oder wo sich die gewünschte Tasse befindet. Auf Basis des in der Vergangenheit gewonnenen Wissens lassen sich diese Parameter jedoch durch Reasoning inferieren. So könnte der Roboter durch in der Vergangenheit getätigte Beobachtungen herausfinden, welches die Lieblingstasse der Person ist. Da er nun weiß, um welche Tasse es sich handelt, könnte er in seiner Wissensbasis nach dem letzten ihm bekannten Ort der Tasse suchen.

Um erfolgreich in Umgebungen mit Menschen agieren und interagieren zu können, wird außer des Wissens über seine Umwelt eine große Menge an Wissen zur Durchführung von Aufgaben benötigt. Unter Laborbedingungen ist die Menge der Daten noch überschaubar und lässt sich manuell in die Wissensbasis eintragen. In einer echten Welt ist die Menge an Daten jedoch so groß, dass das Wissen auf eine andere Art und Weise beschafft werden muss. Reasoning kommt immer dann zum Einsatz, wenn dem Roboter nicht alle benötigten Informationen zum Durchführen einer Aufgabe vorliegen. Bei der Interaktion mit Menschen ist dies meistens der Fall. Die Befehle, die ein Haushaltsroboter erhält, enthalten nur selten alle von ihm benötigten Informationen. Erhält der Roboter z. B. den Befehl „*Back mir einen Pfannkuchen*“, fehlen ihm Angaben zu den Zutaten und der korrekten Zubereitung.

In [TNB10] wird beschrieben, wie fehlendes Wissen zur Bewältigung solcher – nach den Maßstäben eines Roboters – komplexen Aufgaben im Haushalt aus im Internet zugänglichen Anleitungen extrahiert wird. Da diese Anleitungen für Menschen geschrieben wurden, sind sie

in natürlicher Sprache verfasst und müssen von der Wissensverarbeitung analysiert werden, um die gewünschten Informationen zu erhalten. Aus dem gewonnenen Wissen kann anschließend ein Plan erstellt und in der Wissensbasis hinterlegt werden.

In [RBC] wird ein Framework vorgestellt, welches zur Beschaffung von neuem Wissen Videoaufnahmen von Menschen, die eine bestimmte Aktivität durchführen, analysiert. Dabei wird versucht, verschiedene Bewegungen und menschliches Verhalten im Video zu segmentieren und ihnen durch Reasoning eine semantische Bedeutung zu geben. Ziel ist es, menschliches Verhalten nicht nur zu kopieren, sondern durch die Semantik zu verstehen, **was** eine Person gerade tut. Somit kann das neu gewonnenen Wissen für Aufgaben in einem ähnlichen Kontext wiederverwendet werden.

Wie die von einem Objekt gegebenen Gebrauchseigenschaften (Affordanz) ermittelt werden können, wird in [Ten+] beschrieben. Um zu wissen, wie ein Roboter einen Gegenstand verwenden muss, um ihn korrekt einzusetzen, genügt es nicht, wenn der Gegenstand korrekt erkannt wird. Oftmals fehlt die Verknüpfung zwischen einem Objekt und seiner semantischen Bedeutung. Um diese Bedeutung herauszufinden, werden zu erkannten Objekten CAD-Modelle aus einer im Internet zugänglichen Datenbank geladen. In den Modellen wird nach primitiven geometrischen Formen gesucht, welche dem Roboter eine Interaktion mit dem Objekt erlauben. Wurden solche Formen gefunden, wird Reasoning betrieben, um aus ihnen die verschiedenen Interaktionsmöglichkeiten abzuleiten. So würde z. B. bei der Untersuchung des Modells eines Löffel eine annähernd zylindrische Formen gefunden werden, welche dem Stiel entspricht. Eine solche geometrische Form ermöglicht dem Roboter, das Objekt zu greifen. Diese Information wird in der Wissensbasis mit dem Objekt verknüpft und hilft dem Roboter bei der korrekten Handhabung.

2.1.3. Kombination von Perzeption und Reasoning

In den meisten klassischen Robotersystemen werden die Low-Level-Komponenten (z. B. Perzeption und Manipulation) von den High-Level-Komponenten (z. B. Planung und Reasoning) getrennt. In vielen Anwendungsfällen kann es jedoch nützlich sein, die verschiedenen Bestandteile miteinander zu kombinieren.

Das System K-CoPMan (*Knowledge-enabled Cognitive Perception for Manipulation*) [Pan+10] kombiniert Perzeption und Wissensverarbeitung miteinander, um komplexe Abfragen über den Zustand einer Welt zu ermöglichen, die zur Beantwortung sowohl Daten aus

der Wissensbasis, als auch von der Perzeption benötigt. Dem System liegen Informationen zur Umgebung aus einer statischen semantischen Karte vor. Diese Informationen werden durch Daten der Perzeption aktualisiert, um den dynamischen Teil der Welt zu repräsentieren. Dies wird ermöglicht, indem Perzeption und Wissensverarbeitung durch eine höhere Instanz kontrolliert werden. Erhält die Wissensverarbeitung eine Anfrage in Form eines Prädikates, das zur Beantwortung Perzeptionsdaten benötigt, wird es in einen Aufruf einer Perzeptionsroutine umgewandelt. Das Ergebnis dieses Aufrufs wird im Zuge der Auswertung des Prädikats von der Wissensverarbeitung interpretiert. Somit ist es möglich, Reasoning über den Zustand der Welt zu betreiben, ohne dass der Wissensverarbeitung die benötigten Daten zum Zeitpunkt der Abfrage zur Verfügung stehen.

In [Kun+] wird ein Framework vorgestellt, welches die Resultate eines klassischen Perzeptionssystems mit *Spatial Reasoning* (englisch für *räumliches Schlussfolgern*) kombiniert. Beim Spatial Reasoning werden im Vorfeld räumliche Zusammenhänge zwischen verschiedenen Objekten in räumlichen Modellen repräsentiert. Dabei wird die Tatsache ausgenutzt, dass Objekte, die in einem semantischen Kontext zueinander stehen, oft auch in einem bestimmten räumlichen Kontext zueinander stehen. So findet man z. B. häufig eine Computertastatur *vor* einem Computermonitor und nicht umgekehrt. Beim Klassifizieren der Objekte werden für jedes Objekt die von der Perzeption berechneten Wahrscheinlichkeiten, dass es sich um ein Objekt einer bestimmten Klasse handelt, mit den beim Spatial Reasoning errechneten Wahrscheinlichkeiten kombiniert.

Ein weiteres Beispiel für die Kombination von Perzeption und Reasoning wird in [BHT13] beschrieben. Das dort gestellte Ziel besteht darin, die menschliche Fähigkeit, komplexe Szenen (bestehend aus mehreren Objekten) aus dem Alltag anhand eines kurzen Eindrucks oder einer Momentaufnahme bewerten zu können, auf ein Perzeptionssystem zu übertragen. Dabei wird davon ausgegangen, dass das menschliche Gehirn ein mentales Modell der Umwelt mit grundlegenden physikalischen Regeln bildet, welches es zur Simulation der erfassten Szene verwendet. Ziel ist es, durch Kombination von Perzeption und kognitiven Fähigkeiten Vorhersagen über eine Szene treffen zu können. Dies ist z. B. für einen Roboter wichtig, der eine Manipulationsaufgabe durchführen soll. Wenn er ein Objekt aus der Szene aufnimmt (z. B. eine Untertasse), muss er im Vorfeld wissen, ob diese Manipulation negative Nebeneffekte erzeugt (eine Tasse, die auf der Untertasse stehen würde, könnte z. B. herunterfallen und zerbrechen). Soll ein Objekt in der Szene abgestellt werden, sollte der Roboter im Vorfeld wissen, ob sich ein bestimmter Ort zum Abstellen eignet. Die Untertasse würde z. B. mit hoher Wahrscheinlichkeit auf den Boden fallen und samt Tasse zerschellen, wenn der Roboter sie in einer instabilen Position an einer Tischkante abstellen

würde. Der Mensch besitzt die Fähigkeit, solche Entscheidungen intuitiv zu treffen, ohne aktiv darüber nachdenken zu müssen und ohne die konkreten physikalischen Gesetze kennen zu müssen. Diese kognitive Fähigkeit wird in [BHT13] durch die Verwendung einer Physik-Engine auf ein Reasoning-System übertragen. Das System erhält als Eingabe das Modell einer Szene (z. B. von der Perzeption). Mittels dieser Daten kann die Szene in die Physik-Engine übertragen und dort simuliert werden. Der Zustand der Szene wird nach der Simulation interpretiert, und erlaubt das Betreiben von Reasoning auf den durch die Simulation gewonnenen Informationen, um Aussagen über mögliche zukünftige Status der Szene treffen zu können.

2.2. Forschungsfrage

Auf modernen Robotersystemen werden pro Sekunde mehrere Bilder von den Perzeptionssystemen verarbeitet. In allen in Abschnitt 2.1.2 auf Seite 5 beschriebenen Verfahren zur Objekterkennung werden die verwendeten Algorithmen in jeder Iteration erneut auf die von den Sensoren empfangenen Daten angewandt, ohne dabei die Ergebnisse der vorherigen Durchläufe zu berücksichtigen; die gewonnenen Informationen werden in jeder Iteration nach einmaliger Analyse verworfen und nicht weiter verwendet. Würde man Daten aus der Vergangenheit jedoch in die Berechnungen zur Objekterkennung miteinbeziehen, könnten möglicherweise die Anzahl an Falscherkennungen verringert und Ergebnisse aus vorherigen Iterationen verbessert werden.

Im Rahmen dieser Arbeit soll ein Algorithmus entwickelt werden, der Daten aus der jeweils aktuellen Iteration und in vergangenen Iterationen gewonnene Daten verwendet und Reasoning auf ihnen betreibt. In den Prozess des Reasonings soll eine Physiksimulation eingebunden werden, die es ermöglicht, die vom Roboter wahrgenommene Welt und die erkannten Objekte und deren Verhalten unter dem Einfluss einer simulierten Physik zu untersuchen. Der zu entwickelnde Algorithmus soll in die Perzeptionspipeline des bestehenden Systems RoboSherlock integriert werden. Die von diesem System erzeugten Daten sollen mit Hilfe der Physiksimulation und durch Anwenden von Reasoning auf Konsistenz geprüft und gegebenenfalls korrigiert werden. Ziel ist es, durch konsistentere Daten die Genauigkeit, mit der autonome Haushaltsroboter agieren, zu erhöhen, damit diese ihre Aufgaben verlässlicher durchführen können. Im Folgenden beschriebene Ideen sollen dazu umgesetzt werden.

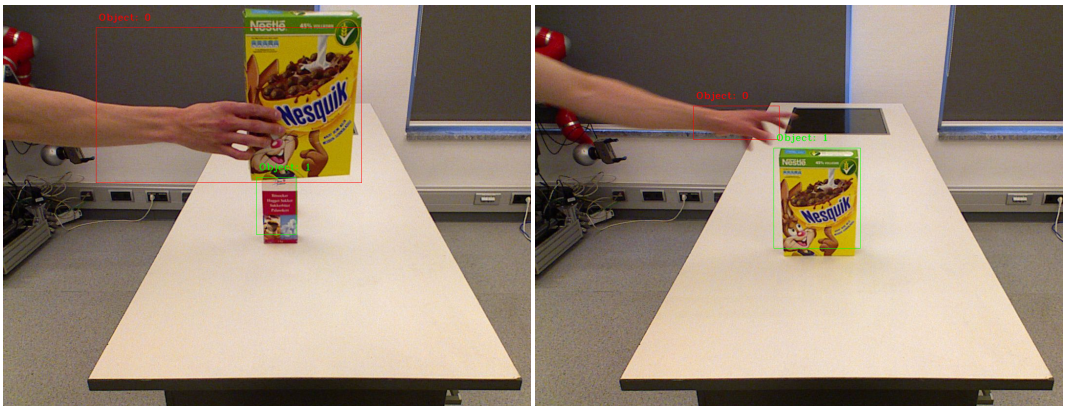

Abbildung 2.1.: *Links:* Ein Objekt und die Hand (inklusive Arm), in der es gehalten wird, werden als ein „schwebendes“ Objekt erkannt. *Rechts:* Eine Hand wird fälschlicherweise als Objekt erkannt.

1. Erkennen von Störungen im Sichtfeld des Roboters

   Oftmals kommt es vor, dass z. B. Menschen im Sichtfeld des Roboters agieren und einen Gegenstand durch die Luft bewegen. Dieser Gegenstand wird in einigen Fällen auch erkannt, im Belief State des Roboters wird aber fälschlicherweise angenommen, dass das Objekt sich in einem ruhenden Zustand befindet. In anderen Fällen werden menschliche Extremitäten als Objekte erkannt und in den Belief State übernommen. Beide Fälle sind in Abbildung 2.1 dargestellt.

2. Identifizieren von Objekten über mehrere Iterationen

   Ein Objekt, das über mehrere Iterationen im Sichtfeld des Roboter erscheint, soll nicht nur korrekt klassifiziert werden, sondern möglichst immer als dasselbe Objekt identifiziert werden. Dies soll auch der Fall sein, wenn es zwischenzeitlich verdeckt wurde, aus dem Sichtfeld des Roboters verschwunden ist oder in der Szene bewegt wurde.

3. Konsistenz bei der Objektklassifizierung

   Das Erkennen von bekannten Objekten beruht meist auf einer dem Roboter zugänglichen Datenbank, in der z. B. verschiedene Attribute zu jedem Objekt abgespeichert sind. Diese Attribute können z. B. Farbe, Größe oder Form sein. Die Ergebnisse der Perzeption hängen nicht nur von den verwendeten Algorithmen,

sondern auch stark von den durch die Umwelt gegebenen Bedingungen (Tageslicht, künstliche Beleuchtung, Schatten, Entfernung) und von den Ungenauigkeiten (Rauschen, Auflösung, Störsignale) der verwendeten Sensoren ab. Eine mangelnde Auflösung oder eine zu hohe Entfernung können dazu führen, dass die Maße eines Objektes nicht richtig erkannt werden; durch sich verändernde Lichtverhältnisse können erkannte Farbwerte stark von den Trainingsdaten aus der Datenbank abweichen und das Hintergrundrauschen in den Signalen kann die Rohdaten verfälschen. All diese Parameter unterliegen nichtdeterministischen Schwankungen. Dies kann dazu führen, dass ein und dasselbe Objekt in verschiedenen Iterationen jedes Mal als ein anderes Objekt erkannt wird, obwohl sich weder der Roboter noch das Objekt bewegt haben. Durch Reasoning über die in der Vergangenheit gesammelten Daten sollen konsistentere Ergebnisse erzeugt werden, um diesen Effekt zu minimieren

4. Korrigieren der Höhe und der Position von Objekten

   Bei der Objekterkennung werden zunächst Oberflächen erkannt, auf denen Objekte stehen könnten. Diese Flächen werden durch Segmentierung aus der Punktwolke der Szene entfernt. Dabei werden die Cluster von den auf den Oberflächen stehenden Objekten oftmals abgeschnitten (siege Abbildung 2.2). Im Belief State des Roboters sind diese Objekte also zum einen kleiner, als sie wirklich sind, zum anderen sind ihre *Centroids*[1] ein wenig versetzt. Wenn nun ein solches Objekt in die simulierte Welt an der erkannten Position und mit den erkannten Dimensionen eingefügt wird, so schwebt es zunächst über der Oberfläche, auf der es eigentlich stehen soll. Die Ausmaße der von den Objekten abgeschnittenen Stücke sollen mit Hilfe der Simulation erkannt und die Daten der betroffenen Objekte korrigiert werden.

5. Erkennen von verdeckten Objekten

   In alltäglichen Situationen, wie dem Decken eines Tisches, kommt es vor, dass sich im Verlaufe einer Aktion verschiedene Objekte gegenseitig verdecken (siehe Abbildung 2.3) und somit einstmals sichtbare Objekte aus dem Sichtfeld des Roboters verschwinden. Nicht sichtbare Objekte können von der Perzeption nicht wahrgenommen werden, was zur Folge haben kann, dass sie aus dem Belief State des Roboters entfernt werden und somit eine Diskrepanz zwischen dem Zustand

[1]↑Centroid, englisch für *geometrischer Schwerpunkt*

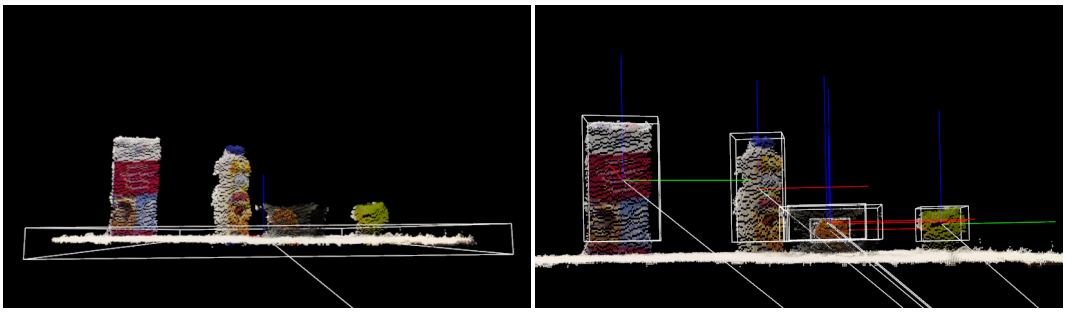

Abbildung 2.2.: *Links:* Die segmentierte Fläche nimmt auch Punkte ein, die eigentlich zu Objekten gehören. *Rechts:* Daher sind die erkannten, zu den Objekten gehörenden Cluster kleiner als sie in Wirklichkeit sind.

Abbildung 2.3.: Der Roboter bewegt sich an zwei Objekten vorbei, sodass das hintere vom vorderen verdeckt und von der Perzeption nicht mehr wahrgenommen wird.

der echten Welt und dem Belief State des Roboters entsteht. Um diesem Verhalten entgegenzuwirken, sollen in solchen Situationen die durch eine Simulation eröffneten Möglichkeiten dabei helfen, das Verdecken von Objekten zu erkennen.

Die Forschungsfrage dieser Arbeit lautet in Hinsicht auf die oben beschriebenen Gesichtspunkte:

> „*Können die Ergebnisse eines klassischen Perzeptionssystems durch das Anwenden von physikbasierter Simulation in Kombination mit Reasoning hinsichtlich ihrer Konsistenz verbessert werden?*“

2.3. Abgrenzung und Einschränkungen

Diese Arbeit befasst sich nicht mit der Entwicklung eines neuen Perzeptionsalgorithmus. Die bereits bestehenden Komponenten RoboSherlock und `bullet_reasoning` sollen miteinander kombiniert werden, um die von der Perzeption erhaltenen Daten auf Konsistenz zu überprüfen. Der Rahmen dieser Arbeit beschränkt sich dabei auf eine im Labor der IAI vorhandenen Küchenumgebung und eine Auswahl starrer Objekte aus dem Küchenalltag. Die Objekte werden während der in Kapitel 5 auf Seite 51 beschriebenen Experimente auf einem Tisch abgestellt und manipuliert.

Das Erkennen von deformierbaren oder unbekannten Objekten ist nicht vorgesehen. Weiterhin besteht die Einschränkung, dass Objekte in einer Szene einen Mindestabstand von 5 cm zueinander halten müssen. Objekte, die näher zueinander stehen, werden im ungünstigsten Fall von RoboSherlock als ein einzelnes Objekt wahrgenommen.

3. Grundlagen

Dieses Kapitel skizziert die verschiedenen Programmbibliotheken und Systeme, auf denen diese Arbeit aufbaut. Zunächst wird das Roboterbetriebssystem ROS beschrieben. Es bildet eine Kernkomponente, die die Kommunikation zwischen anderen Komponenten ermöglicht. Anschließend wird CRAM vorgestellt – eine Sammlung von Paketen, die das Design, die Implementierung und die Inbetriebnahme von Roboterplänen mit kognitiven Fähigkeiten unterstützt. Darauf folgt eine Beschreibung von RoboSherlock, einer Implementierung für die Perzeption von Robotern. An letzter Stelle werden die Datenbank *MongoDB* und die probabilistische Logik MLN umrissen.

3.1. Robot Operating System

Das Robot Operating System (ROS)[1] ist eine quelloffene Sammlung von Programmbibliotheken und Werkzeugen für die Entwicklung von Roboteranwendungen. Das Ziel von ROS ist die Bereitstellung einer hardware- und systemunabhängigen Entwicklungsumgebung. Es ist darauf ausgelegt, auf vielen verschiedenen Robotersystemen, die sich in ihrer Hardware grundlegend unterscheiden können, eingesetzt zu werden. [Qui+09]

ROS besitzt ein dezentralisiertes, modulares Design. Dies bietet den Vorteil, dass Komponenten, die für einen bestimmten Anwendungsfall nicht benötigt werden, nicht geladen werden müssen oder bestimmte Komponenten durch eigene Implementierungen ersetzt werden können. Die Kernkomponenten werden von der Firma *Willow Garage, Inc.*[2] entwickelt und gepflegt. Sie beinhalten das Erstellen und Verwalten von Paketen, eine Infrastruktur zur Kommunikation, Sprachanbindungen für verschiedene Programmiersprachen sowie Inspektions- und Analysewerkzeuge. Weitere Pakete (aktuell über 3000) werden von einer

[1] ↑ `http://www.ros.org`
[2] ↑ `http://www.willowgarage.com`

internationalen Gemeinschaft von Entwicklern erstellt, dokumentiert und in Stand gehalten.

ROS verwendet einen Standard, der als Message Passing Interface (MPI) bezeichnet wird. Dabei existieren neben einem Hauptknoten (*ROS Master*) beliebig viele Nebenknoten (*Nodes*). Diese Knoten, auf denen verschiedene Prozesse laufen, können auf unterschiedliche Computersysteme verteilt sein. Sie können über den Austausch von Nachrichten über sogenannte *Topics* und über *Services* miteinander kommunizieren und so z. B. Daten, Anfragen und dazugehörige Antworten untereinander versenden. Topics und Services besitzt eindeutige Namen, über die sie angesprochen werden können. Jede Node kann sich auf einem Topic als *Publisher* anmelden oder als *Subscriber* auf einem Topic horchen. Publishern ist es möglich, auf den Topics, auf denen sie angemeldet sind, Nachrichten in einem vom Topic festgelegten Nachrichtenformat zu veröffentlichen; Subscriber dieses Topics empfangen diese Nachrichten. Weiterhin können von jeder Node Services angeboten werden. Eine andere Node, die einem solchen Service eine Anfrage sendet, wartet so lange, bis diese von der Service-Node bearbeitet und eine Antwort zurückgeschickt wird. Somit dienen Topics dem asynchronen und Services dem synchronen Austausch von Nachrichten.

Nodes lassen sich über einen von ROS bereitgestellten *Parameter Server* konfigurieren. Jeder Parameter erhält – genau wie Topics und Services – einen eindeutigen Namen. Über einen Aufruf von außen lassen sich Daten in einem auf dem Parameter Server definierten Parameter speichern oder bereits gesetzte Parameter abrufen. Nodes nutzen diese Funktion, um Parameter und Konfigurationen während der Laufzeit zu speichern und abzufragen.

ROS ermöglicht es, alle Nodes über eine auf einem Topic bereitgestellte Uhr zu Synchronisieren. Somit spielt die Systemzeit, auf der die Nodes ausgeführt werden, für den Betrieb von ROS keine Rolle. Weiterhin ist es möglich, alle auf einer getroffenen Auswahl an Topics versendeten Daten (z. B. sämtliche Kameradaten und die Uhr) aufzuzeichnen. Diese Daten können später verwendet werden, um sie zu einem späteren Zeitpunkt (synchronisiert durch die Daten der aufgezeichneten Uhr) abzuspielen und an die entsprechenden Topics zu versenden. Somit ist es möglich, eine bestimmte Szene aufzunehmen, und dann die Aufnahme beliebig oft zum Testen der aktuellen Implementierung eines Robotersystems zu verwenden.

Durch diese Architektur stellt ROS eine sprachunabhängige Schnittstelle zur Interprozess-/Intermaschinenkommunikation und -konfiguration bereit, ohne dabei ein bestimmtes Pro-

tokoll vorzuschreiben. Es gibt Clientbibliotheken für die Programmiersprachen *C++*[3], *Python*[4], *Common Lisp*[5] und *Java*[6]. Weiterhin bietet ROS eine Integration in eine Vielzahl von anderen Programmbibliotheken wie z. B.

Gazebo[7]	Eine 3D-Robotersimulation mit dynamischer und kinematischer Physik.
OpenCv[8]	Eine Bibliothek für Computer Vision und Maschinelles Lernen.
Point Cloud Library[9]	Ein Framework, welches Algorithmen für die Manipulation und Verarbeitung von 2D- und 3D-Bildern bereitstellt.
MoveIt[10]	Eine Programmbibliothek für die Bewegungsplanung und Wegfindung von Robotern.

3.2. Cognitive Robot Abstract Machine

Cognitive Robot Abstract Machine (CRAM) [BMT10] ist ein auf ROS basierendes, quelloffenes Framework, welches das Design, die Implementierung und die Inbetriebnahme von kognitiven Robotersystemen ermöglicht. Es ist ausgerichtet auf autonome Haushaltsroboter, welche komplexe Manipulationsaufgaben aus dem menschlichen Lebensalltag durchführen sollen.

Es gibt bereits viele Middleware-Bibliotheken zur Erstellung von Steuerungsprogrammen für Roboter. Allerdings fehlen mächtige Tools, die eine effektive und effiziente Implementierung von High-Level-Fähigkeiten erlauben, wie z. B. Lernen, Wissensverarbeitung und Planung von Aktionen. In Alltagssituationen muss ein Haushaltsroboter fortwährend entscheiden, welche Aktionen er ausführen muss, und vor allen Dingen, **wie** er sie ausführen muss, um seine Aufgaben bewältigen zu können. Bereits für den Menschen simpelste Aktionen erfordern für einen Roboter einen hohen und komplexen Planungsaufwand. Bevor er

[3]↑ `http://isocpp.org`
[4]↑ `https://www.python.org`
[5]↑ `http://common-lisp.net`
[6]↑ `https://www.java.com`
[7]↑ `http://gazebosim.org`
[8]↑ `http://opencv.org`
[9]↑ `http://pointclouds.org`
[10]↑ `http://moveit.ros.org`

eine Aktion durchführen kann, müssen viele verschiedene Entscheidungen getroffen werden, wie z. B.:

- Wo muss der Roboter stehen, um sich in Reichweite eines zu greifenden Objekts zu befinden?
- Welche Hand muss er zum Greifen benutzen?
- Wie muss gegriffen werden?
- Wo muss gegriffen werden?
- Wie viel Kraft muss/darf beim Greifen angewandt werden?
- Wie muss das Objekt angehoben werden?
- Wie viel Kraft muss/darf beim Anheben des Objekts angewandt werden?
- Wie muss das Objekt bei weiteren Bewegungen gehalten werden?

Diese Entscheidungen hängen stark vom Kontext der Aufgabe ab. So gibt es z. B. bei einer leeren Tasse, die von einem Tisch in eine Spülmaschine befördert werden soll, nicht so viele Faktoren zu beachten wie bei einer Tasse, die mit heißem Tee gefüllt ist und einer Person übergeben werden soll. Im letzteren Fall darf die Tasse z. B. nicht so gehalten werden, dass die Flüssigkeit herausfließen kann. Außerdem sollte der Roboter die Tasse nicht am Henkel halten, um der entgegennehmenden Person zu ermöglichen, sie am Henkel zu erfassen und somit die Gefahr einer Verbrennung zu vermeiden.

CRAM bietet eine Softwareinfrastruktur für flexible und zuverlässige sensorgeführte Steuerungsprogramme (Pläne) für Roboter. Es erweitert die funktionale Programmiersprache *Common Lisp* und verwendet existierende Lisp-Compiler. Pläne spezifizieren das Verhalten eines Roboters in Reaktion auf Sensordaten, Änderungen im Belief State und erkannten Fehlschlägen im Ablauf eines Plans. Der Belief State wird ständig aktualisiert – z. B. nach durchgeführten Manipulationen oder mit Daten von passiven Perzeptionsmechanismen. Mit seiner eigenen Plansprache CRAM Plan Language (CPL) und seinen Reasoning-Mechanismen erlaubt CRAM das Implementieren von komplexen Steuerungsprogrammen, die flexibles, zuverlässiges und effizientes Verhalten produzieren. Diese Reasoning-Mechanismen erlauben es, Steuerungsentscheidungen erst während der Laufzeit auf der Basis von Wahrnehmungen der Umgebung und von bereits erlangtem Wissen zu treffen. Damit müssen Steuerungsentscheidungen nicht im Vorfeld fest programmiert werden.

3.2.1. KnowRob und Prolog

Services zur Wissensverarbeitung und für das Reasoning werden in CRAM durch *KnowRob*[11] [TB13] und eine eigene Prolog-Engine zur Verfügung gestellt. KnowRob ist ein speziell für autonome persönliche Roboter ausgelegtes System zur Wissensverarbeitung, basierend auf der logischen Programmiersprache *Prolog*[12]. Es kombiniert Wissensrepräsentation und Reasoning-Methoden mit Techniken zum Erlangen von Wissen und zum Einbringen des Wissens in ein System. Während der Ausführung eines Steuerungsprogramms kann neues Wissen automatisch durch Beobachtungen und Erfahrungen erlangt und für späteres Reasoning gespeichert werden. Wissen wird mittels Beschreibungslogik, basierend auf Prädikatenlogik erster Stufe, im OWL-Format repräsentiert.

Anfragen an die Wissensbasis werden in Form von Prologausdrücken gestellt. Prolog prüft die Erfüllbarkeit dieser aussagenlogischen Formeln anhand der vorliegenden Wissensbasis. Bei Erfüllbarkeit wird eine Variablenbelegung zurückgegeben, unter der die Aussage erfüllt wird. Zur Veranschaulichung dient Listing 3.1 auf der nächsten Seite. Dort wird in den Zeilen 1 bis 8 eine Wissensbasis in Form von Fakten definiert. In den Zeilen 10 bis 19 wird das Prädikat `in_front_of` definiert, welches – stark vereinfacht – eine Aussage darüber treffen kann, ob ein Roboter **vor** einem Tisch steht. In Listing 3.2 auf der nächsten Seite werden beispielhaft drei an diese Wissensbasis gestellte Abfragen und deren Rückgabewerte gezeigt. In den ersten beiden Abfragen wird `in_front_of` mit zwei gebundenen Variablen aufgerufen und der Ausdruck auf Erfüllbarkeit überprüft. Werden jedoch, wie in der letzten Abfrage, ungebundene Variablen übergeben, kann Prolog durch Inferenz alle Variablenbelegungen, die zur Erfüllbarkeit führen, bestimmen und ausgeben.

Während ein Roboter versucht eine Aufgabe zu lösen, kann es jedoch vorkommen, dass Wissen benötigt wird, welches nicht in der Wissensbasis vorhanden ist. Um das benötigte Wissen zu erlangen, ist der Zugriff auf externe Quellen wie z. B. das Perzeptionssystem oder eine auf ein bestimmtes Problem spezialisierte Reasoning-Prozedur erforderlich. Dies wird durch sogenannte *Computable Predicates* (berechenbare Prädikate) ermöglicht. Wird der Wissensverarbeitung eine Anfrage gestellt, die zur Beantwortung ein Computable Predicate auswerten muss, geschieht dies durch einen Aufruf der entsprechenden externen Quelle. Die dabei als Rückgabe erhaltenen Daten werden bei der Beantwortung der Anfrage verwendet.

[11] ↑ `http://knowrob.org`
[12] ↑ `http://www.swi-prolog.org`

Listing 3.1: Beispiel für eine stark vereinfachte Wissensbasis in Prolog.

```
tisch ( tisch1 ).
tisch ( tisch2 ).
roboter ( roboter1 ).
roboter ( roboter2 ).
position ( tisch1 , 10.5, 4, 1.3).
position ( tisch2 , 5, 1, 1.3).
position ( roboter1 , 20, 2, 1.2).
position ( roboter2 , 7, 3, 1.2).

in_front_of ( Roboter , Tisch ) :-
    tisch ( Tisch ),                  % 'Tisch' muss ein Tisch ein
    roboter ( Roboter ),              % 'Roboter' muss ein Roboter sein.
    Tisch \= Roboter ,                % Bei 'Tisch' und 'Roboter' darf es sich nicht
                                      % um ein und dasselbe Objekt handeln.
    position ( Tisch , XT, _, _),     % Die X-Komponente der Position des Tisches
                                      % wird der ungebundenen Variable 'XT' zugewiesen.
    position ( Roboter , XR, _, _), % Die X-Komponente der Position des Roboters
                                      % wird der ungebundenen Variable 'XR' zugewiesen.
    XR < XT.                          % Die X-Komponente der Position des Roboters muss
                                      % kleiner sein als die X-Komponente der Position
                                      des Tisches.
```

Listing 3.2: Beispiel für an Prolog gestellte Abfragen, basierend auf der Wissensbasis aus Listing 3.1.

```
?- in_front_of ( roboter2 , tisch1 ).
true

?- in_front_of ( roboter2 , tisch2 ).
false

?- in_front_of ( Roboter , Tisch ).
Roboter = roboter2 ,
Tisch = tisch1
```

3.2.2. Auf Simulation basierendes Reasoning

Das Paket `bullet_reasoning`[13], welches in `cram_physics`[14] enthalten ist, bietet eine Schnittstelle zwischen einer mittels der *Bullet-Physik*[15] simulierten Welt (im Folgenden *Bullet-Welt* genannt) und CRAM. Eine initialisierte Welt besteht zunächst nur aus einem Boden (siehe Abbildung 3.1). Es lassen sich allerdings nach Belieben Umgebungen, Roboter und Gegenstände einfügen. Modelle der im Labor der AG Künstliche Intelligenz (IAI) vorhandenen Küchenumgebung sowie des verwendeten Roboters liegen im Unified Robot Description Format (URDF) vor und lassen sich über den ROS Parameter Server in die *Bullet-Welt* laden. Mit URDF ist es möglich, einen Roboter bis ins Detail genau zu modellieren. Jedes beweg-

[13]↑`https://github.com/cram-code/cram_physics/tree/master/bullet_reasoning`
[14]↑`https://github.com/cram-code/cram_physics`
[15]↑`http://bulletphysics.org`

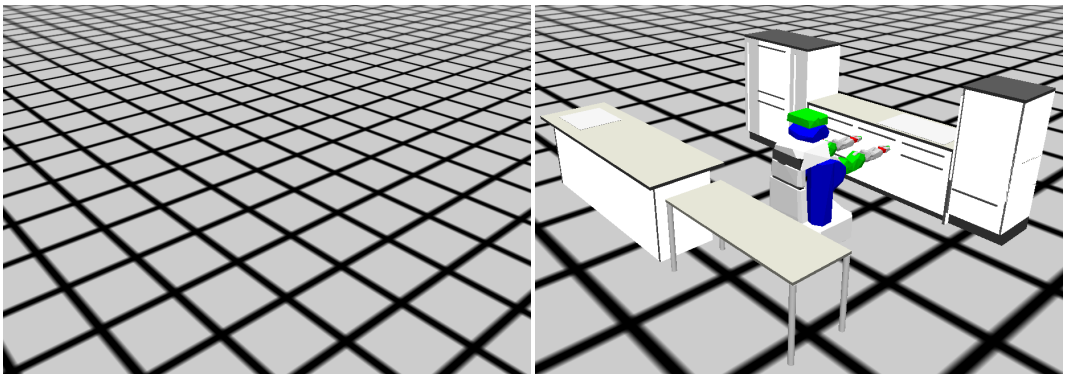

Abbildung 3.1.: *Links:* Eine nur aus einem Boden bestehende in der Bullet-Physik-Engine simulierte Welt. *Rechts:* Eine Welt, in der die Küchenumgebung und das Robotermodell geladen wurden.

liche Glied, jede Verbindung und jedes Gelenk kann darin beschrieben werden, sodass der Roboter detailgetreu in die simulierte Welt eingefügt werden kann.

Die Interaktion mit der Bullet-Welt erfolgt über von `bullet_reasoning` zur Verfügung gestellte Prologprädikate. So lassen sich z. B. die Position des Roboters und die Stellungen seiner Gelenke an die Gegebenheiten in der echten Welt anpassen. Weiterhin können Objekte (z. B. Haushaltsgegenstände) in die Welt eingefügt werden. Objekte, welche aus primitiven geometrischen Formen wie z. B. Kugeln, Quadern, Kegeln oder Zylindern bestehen, können je nach Form unter Angabe von Radius, Höhe, Tiefe und Breite erzeugt werden. Objekte mit komplexeren Formen können ebenfalls erstellt werden, vorausgesetzt sie liegen im STL-Format vor (siehe Abbildung 3.2).

Für die Simulation der Welt wird die Physik-Engine Bullet-Physik verwendet. Diese ermöglicht das Simulieren von Gravitation, Masse, Kraft, Reibung und Bewegung. Auf ein Objekt können beliebig viele Kräfte einwirken. Steht z. B. ein Objekt auf einem Tisch und wird gleichzeitig vom Roboter verschoben, so wirken Gravitation und eine vom Roboter ausgehende Beschleunigung auf das Objekt. Die Simulation wird über die Schnittstelle von `bullet_reasoning` ausgelöst (siehe Prädikat `simulate` in Tabelle 3.1). Sie erfolgt in kleinen Zeitschritten von z. B. 10 ms. Dabei werden alle auf ein Objekt einwirkenden Kräfte addiert und eine daraus resultierende Bewegung berechnet. Anschließend werden die Geschwindigkeitsvektoren und die Positionen der Objekte aktualisiert.

`bullet_reasoning` stellt eine Vielzahl von prologähnlichen Prädikaten, deren Auswertung

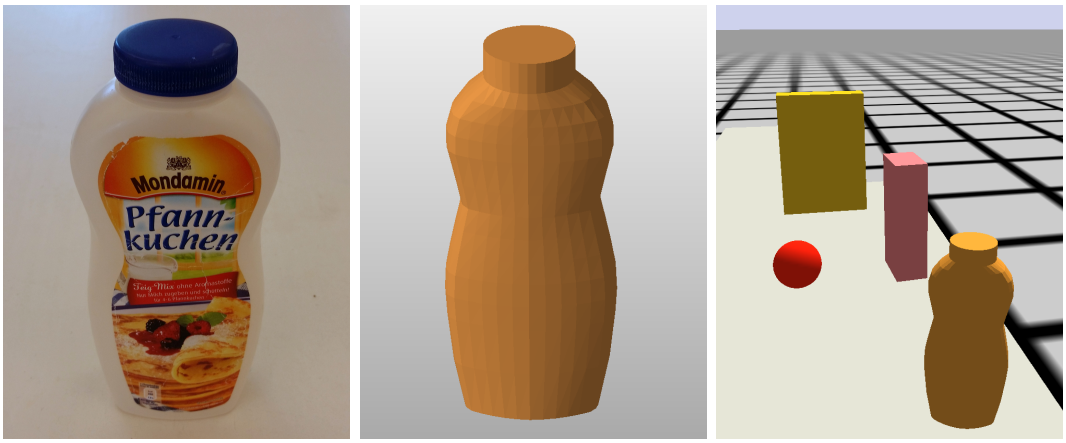

Abbildung 3.2.: *Links:* Objekt mit komplexer geometrischen Form in der echten Welt. *Mitte:* Dasselbe Objekt als 3D-Modell. *Rechts:* Objekte primitiver und komplexer geometrischer Formen in der simulierten Welt.

mittels eines Prolog-Interpreters geschieht, zur Verfügung. Über die Prädikate lassen sich Aussagen über eine simulierte Welt und die darin enthaltenen Objekte treffen. Tabelle 3.1 beschreibt eine Auswahl an Prädikaten, die im Rahmen dieser Arbeit von Relevanz sind. Sämtliche Prädikate können, wie im vorausgehenden Abschnitt beschrieben, mit ungebundenen Variablen aufgerufen werden, um erfüllende Variablenbelegungen als Antwort zu erhalten. So kann beispielsweise das Prädikat `object()` mit einer gebundenen Variable `world` und einer ungebundenen Variable `?objects` aufgerufen werden, um in `?objects` alle Objekte zu erhalten, die sich in der Welt `world` befinden.

Tabelle 3.1.: Auszug aus den in `bullet_reasoning` verfügbaren Prädikaten.

Prädikat	Beschreibung
`household-object-type(world, object, type)`	Das Objekt `object` vom Typ `type` existiert in der Welt `world`.
`object(world, object)`	Das Objekt `object` existiert in der Welt `world`.
`object-pose(world, object, pose)`	Das Objekt `object` mit der Pose `pose` existiert in `world`.
`visible-from(world, camera-pose, object)`	Das Objekt `object` in der Welt `world` ist aus der Perspektive der Pose `camera-pose` sichtbar.

Tabelle 3.1.: (*Fortsetzung*)

Prädikat	Beschreibung
`visible(world, robot, object)`	Das Objekt `object` in der Welt `world` ist aus der Perspektive von Roboter `robot` sichtbar.
`stable(world, object)`	Das Objekt `object` in der Welt `world` ist stabil (bewegt sich nicht).
`stable-household(world)`	Alle Haushaltsgegenstände in `world` sind stabil (bewegen sich nicht).
`contact(world, objectA, objectB)`	Die Objekte `objectA` und `objectB` aus der Welt `world` berühren sich.
`occluding-objects(world, camera-pose, occluded-object, object)`	Die Objekte `object` und `occluded-object` befinden sich in der Welt `world`. `occluded-object` wird aus der Perspektive der Pose `camera-pose` von `object` verdeckt.
`retract(world, object)`	Das Objekt `object` wird aus der Welt `world` entfernt.
`assert(...)`	Wird verwendet, um Zustände in der simulierten Welt zu verändern. Es können z. B. neue Objekte eingefügt, oder bereits existierende Objekte bewegt werden.
`simulate(world, duration)`	Die Welt `world` wird für die Dauer `duration` simuliert.
`copied-world(world, ?copy)`	Die Welt `world` wird kopiert. Die Kopie wird der ungebundenen Variable `copy` zugewiesen.

3.3. RoboSherlock

RoboSherlock[16] ist ein quelloffenes, in *C++* geschriebenes Framework, welches ermöglicht, viele verschiedene Perzeptionssysteme in Kombination mit wissensbasierten Reasoning-

[16]↑`http://www.pr2-looking-at-things.com/rs_home.html`

Techniken zu kombinieren. Nach dem heutigen Stand der Technik sind viele Perzeptionsalgorithmen darauf ausgerichtet, ein spezielles Problem möglichst fehlerfrei zu lösen. Eine omnipotente Lösung für alle denkbaren Anwendungsgebiete wurde noch nicht entwickelt. Doch gerade im Bereich der in einem menschlichen Haushalt anfallenden alltäglichen Manipulationsaufgaben müssen autonome Roboter in der Lage sein, viele verschiedene Objekte und deren Charakteristika zu erkennen. [Bee+14]

Ein im Rahmen dieser Arbeit betrachtetes Robotersystem erhält in jeder Iteration Rohdaten von z. B. RGB-D-Kameras, monokularen Kameras oder Laserscannern. Diese großen Mengen an Daten liegen in keiner formalisierten Struktur vor – sie sind unstrukturiert. Die Bedeutung von strukturierten Daten ergibt sich durch deren Struktur, bzw. durch ihr Format. Ein Beispiel für strukturierte Daten sind strukturierte Datenbanken. In ihnen können Daten nur in einem vorher bestimmten Format abgelegt werden. Beim Auslesen der Daten ist dieses Format bekannt, wodurch sich die Bedeutung ableitet.

Beispiele für unstrukturierte Daten sind in natürlicher Sprache verfasste Texte, Fotografien sowie Audio- und Videoaufnahmen. All diese Daten liegen zwar in einer wohldefinierten Syntax vor, eine Bedeutung lässt sich daraus aber nicht ableiten. Als Beispiel sei eine 2D-Rastergrafik genannt. Eine solche Grafik besitzt eine Breite, eine Höhe und für jedes Pixel einen Farbwert. Dass auf einer solchen Grafik aber z. B. ein bestimmter Gegenstand dargestellt wird, lässt sich anhand dieses Wissens nicht ableiten. [GS09]

Um die großen Mengen an unstrukturierten Daten zu verarbeiten, verwendet RoboSherlock die Unstructured Information Management Architechture (UIMA)[17]. Dabei handelt es sich um ein Framework zur Entwicklung von Anwendungen, die dem Extrahieren von Wissen aus vorliegenden Daten dienen. Die vorhandenen Daten werden dabei sequenziell von einer beliebigen Anzahl Analysis Engines (AEs) analysiert. Eine AE ist ein Programm, das Artefakte (z. B. Dokumente) analysiert und dabei Informationen extrahiert. Im Kontext von UIMA werden diese Artefakte Subject of Analysis (SofA) genannt. Es gibt Primitive AEs, welche aus einem einzigen *Annotator* bestehen, und Aggregate AEs, welche aus beliebig vielen Annotatoren bestehen. [oV09; FL04]

Annotatoren enthalten die eigentliche Logik, die bei der Analyse der SofAs verwendet wird. Ein Annotator beschränkt sich dabei auf eine bestimmte Aufgabe. Unabhängig vom Kontext, in dem ein Annotator eingesetzt wird, soll dieser sich nur mit seiner Berechnungslogik bei der Analyse der Daten beschäftigen. Eine Kommunikation mit anderen Annotatoren

[17]↑`http://uima.apache.org`

ist nicht vorgesehen. Ein Annotator erzeugt als Ergebnis eine Merkmalstruktur (*Feature Structure*). Dabei handelt es sich um eine Reihe aus Schlüssel-Wert-Paaren in denen Werte entweder atomar sind und einen einzelnen Wert darstellen oder aus einer weiteren Merkmalstruktur bestehen. Diese erzeugten Metadaten, welche nun in einer strukturierten Form vorliegen, werden vom Annotator mit dem Teil der SofA, der von ihm analysiert wurde, verknüpft. [oV09]

Die eingangs erhaltenen unstrukturierten Daten werden zusammen mit den von den Annotatoren erstellten Merkmalstrukturen in einer Common Analysis Structure (CAS) repräsentiert, einer zentralen Datenstruktur innerhalb eines UIMA-Workflows, über die alle Komponenten miteinander kommunizieren können. Eine CAS in RoboSherlock besteht aus folgenden Elementen:

Bilder — Die zu interpretierenden Sensordaten (Rohdaten).

SofAs — Teile der Rohdaten, von denen angenommen wird, dass sie eine Struktur besitzen und ihnen eine semantische Bedeutung zugewiesen werden kann.

Annotationen — Informationen, die von verschiedenen Annotatoren aus den Eigenschaften der SofAs geschlossen wurden.

Typsystem — Ein einheitliches Typsystem, welches sicherstellt, dass die verschiedenen Komponenten von RoboSherlock kompatibel miteinander arbeiten können.

In RoboSherlock gibt es zwei Arten von primitiven AEs. Die sogenannten *Hypotheses Generators* suchen in jeder Iteration in den unstrukturierten Daten einer CAS nach Bereichen, die zu Objekten oder Objektgruppen gehören könnten und generieren aus diesen Bereichen SofAs. Die zweite Art von AEs besteht lediglich aus Annotatoren und verknüpft die SofAs mit den durch die Analyse gewonnen Informationen. Diese Informationen können z. B. Größe, Farbe, Pose, Typ oder Form eines Objekts sein. Wurden Daten einem SofA hinzugefügt (annotiert), sind sie für von weiteren Annotatoren durchgeführte Analysen verfügbar. Dabei kann es durchaus sein, dass verschiedene Annotatoren widersprüchliche und ineinander inkonsistente Annotationen erstellen. Dies ist ein durchaus gewolltes Verhalten, da jeder Annotator für den Bereich, in dem er eingesetzt wird, spezialisiert ist. Die Annotationen bieten somit die Grundlage für eine weiterführende Verarbeitung. So kann z. B. eine Gewichtung der verschiedenen Ergebnisse beim späteren Reasoning dazu führen, dass bestimmten Annotationen mehr Aussagekraft als anderen zugesprochen wird.

3.4. MongoDB

MongoDB[18] ist eine quelloffene und – im Gegensatz zu relationalen Datenbankmanagementsystemen (RDBMS) – schemafreie Datenbank. Sie ist ausgelegt auf große Datenmengen, dokumentenorientiert, einfach skalierbar und performant. [Boe10] Datenbanken in MongoDB sind in *Collections* aufgeteilt. Diese bestehen wiederum aus Dokumenten. In Dokumenten werden Daten als Schlüssel-Wert-Paare gespeichert. Werte können aus Daten (z. B. Zahlen, Zeichenketten oder Binärdaten), einem weiteren Dokument oder Arrays von Daten und Dokumenten bestehen. Da diese Dokumente keinen vorgeschriebenen Aufbau besitzen, sind sie schemalos.

Komplexe Daten (z. B. Arrays oder Objekte) lassen sich problemlos in einem Feld speichern (siehe Abbildung 3.3). Dies wird durch das Datenformat Binary JSON (BSON) ermöglicht. Durch das Speichern von Listen in einzelnen Feldern können One-to-Many- und Many-to-Many-Relationen, ohne die Verwendung von Verweistabellen, aufgebaut werden. Eine höhere Performanz als RDBMSs erreicht MongoDB durch eine Abweichung vom Paradigma der Normalisierung der Daten. Somit entstehen zwar potenziell Redundanzen, bei Abfragen müssen aber weniger zeitintensive *Joins* vorgenommen werden. [BRA12; Boe10]

Da es durch MongoDB ermöglicht wird, große Mengen von Daten in kurzer Zeit abzuspeichern und auszulesen, wird im Rahmen von RoboSherlock eine MongoDB-Datenbank verwendet, in der in jeder Iteration die Rohdaten der Sensoren und die von den Annotatoren erstellten Metadaten gespeichert werden.

3.5. Markov Logic Network

Ein Markov Logic Network (MLN) ist im Gegensatz zur Pädikatenlogik erster Stufe (FO) eine probabilistische Logik. Ausdrücke in der FO können entweder erfüllbar oder nicht erfüllbar sein. Es gibt keine Möglichkeit der Auswertung, die ein anderes Ergebnis zulässt. MLN kombiniert *Markov-Netzwerke* (ein statistisches Modell) mit der FO. Um eine gegebene Aussage auswerten zu können, muss als Grundlage eine Wissensbasis vorliegen. In ihr steht eine Reihe von Ausdrücken aus der FO (bestehend aus atomaren Ausdrücken, welche mit Junktoren verbunden werden). Diese Sammlung von Ausdrücken bildet ein Markov-Netzwerk. Soll

[18] `https://www.mongodb.org`

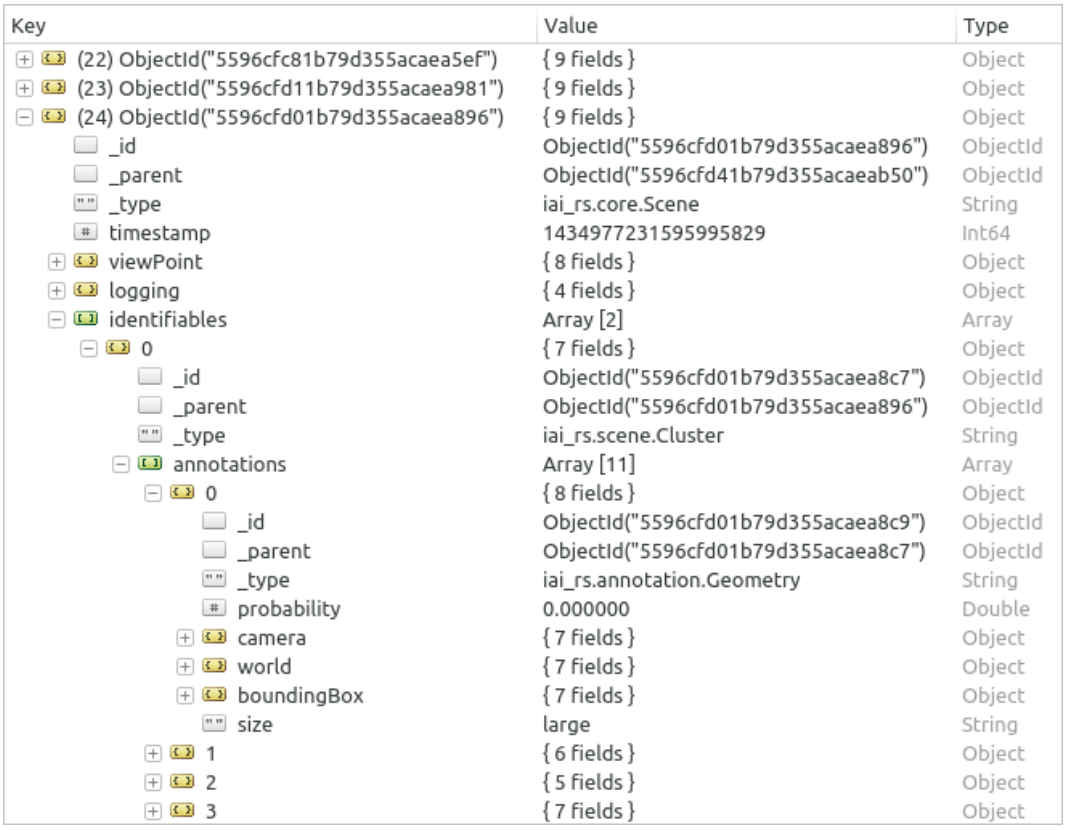

Abbildung 3.3.: Beispielhafte Darstellung eines Ausschnitts einer MongoDB-Datenbank mit dem Programm *Robomongo* (`http://robomongo.org`). Zu sehen sind drei Objekte vom Datentyp `iai_rs.core.Scene`. Das dritte Objekt hält in einigen Schlüssel-Wert-Paaren primitive Daten (z. B. `String` oder `Int64`), in anderen hält es Arrays oder Dokumente.

in der FO ein Ausdruck anhand einer solchen Wissensbasis auf Erfüllbarkeit geprüft werden, genügt es, wenn nur ein Ausdruck aus der Wissensbasis nicht erfüllbar ist, damit auch der zu validierende Ausdruck als unerfüllbar gewertet wird. MLN reagiert in einem solchen Fall auf die Weise, dass der zu überprüfende Ausdruck nicht sofort als unmöglich erfüllbar, sondern als weniger wahrscheinlich erfüllbar eingestuft wird. [RD06]

MLN wird im Rahmen dieser Arbeit vom in Abschnitt 4.1.2.3 auf Seite 39 beschriebenen `MLNAtomsGenerator` zur Objektklassifizierung, basierend auf einer Wissensbasis verschiedener Objektattribute, verwendet.

Im Folgenden wird die Verwendung von MLN anhand eines Beispiels[19] veranschaulicht. In diesem Beispiel soll stark vereinfacht der Zusammenhang zwischen der Tatsache, dass jemand raucht und der Erkrankung an Krebs modelliert werden. Es existieren folgende Prädikate:

```
1 Raucht(p)                  // Eine Person ist Raucher.
2 Krebs(p)                   // Eine Person hat Krebs.
3 Freunde(p,p)               // Zwei Personen sind miteinander befreundet.
```

Die unten stehenden Ausdrücke aus der FO postulieren, dass aus der Gegebenheit, dass eine Person Raucher ist, folgt, dass diese Person an Krebs erkrankt. Weiterhin wird das Prädikat `Freunde` als symmetrisch definiert. Der letzte Ausdruck modelliert den Zusammenhang zwischen einer Freundschaft und dem Rauchverhalten beider beteiligten Personen.

```
1 Raucht(p) => Krebs(p)                                  // Rauchen führt zu Krebs.
2 Freunde(p1,p2) <=> Freunde(p2,p2)                      // Ist p1 mit p2 befreundet, ist auch
3                                                        // p2 mit p1 befreundet (Symmetrie).
4 Freunde(p1,p2) => (Raucht(p1) <=> Raucht(p2))          // Sind zwei Personen befreundet, so
5                                                        // sind entweder beide oder keiner
6                                                        // von ihnen Raucher.
```

Diese Ausdrücke müssen für die weitere Verwendung gewichtet werden, um anschließend in die Wissensdatenbank geschrieben zu werden:

```
1 //  Gewichtung      Ausdruck
2       1.126769       Raucht(x) => Krebs(x)
3       1.577776       Freunde(x, y) => (Raucht(x) <=> Raucht(y))
```

In echten Anwendungsfällen ist es kiene Seltenheit, dass hunderte Ausdrücke gewichtet werden müssen. Zu diesem Zweck existieren Programme, die diesen Vorgang, unter Angabe des

[19]↑Dieses Beispiel wurde von `https://github.com/opcode81/ProbCog/wiki/MLN-Tutorial` (besucht am 04.07.2015) übernommen.

zu verwendenden Algorithmus, automatisiert durchführen. Im Rahmen dieser Arbeit wurde das *MLN-Learning-Tool*[20] aus *ProbCog*[21] verwendet.

Um eine Anfrage an die erstellte Wissensbasis zu stellen, werden zunächst Eingabedaten benötigt. Diese können z. B. folgendermaßen aussehen:

```
1 Krebs(Ann)              // Ann hat Krebs.
2 !Krebs(Bob)             // Bob hat keinen Krebs.
3 !Freunde(Ann,Bob)       // Ann und Bob sind nicht befreundet.
```

Unter Verwendung dieser Eingabe und der Wissensdatenbank können nun Abfragen durch Inferieren beantwortet werden. Zu jedem Resultat wird angegeben, mit welcher Wahrscheinlichkeit es zutrifft.

```
Raucht                                  // Abfrage
0.436830    Raucht(Ann)                 // Ergebnis

Raucht(Ann) ∨ Raucht(Bob)               // Abfrage
0.152667    Raucht(Ann) ∧ Raucht(Bob)   // Ergebnis

Raucht(Ann) ∧ Raucht(Bob)               // Abfrage
0.528921    Raucht(Ann) ∨ Raucht(Bob)   // Ergebnis

Raucht(Bob)                             // Abfrage
0.244758    Raucht(Bob)                 // Ergebnis
```

Die Ergebnisse deuten darauf hin, dass Ann mit einer höheren Wahrscheinlichkeit als Bob raucht.

[20] ↑ https://github.com/opcode81/ProbCog/wiki/MLN-Learning-Tool
[21] ↑ https://github.com/opcode81/ProbCog

4. Umsetzung

In diesem Kapitel wird beschrieben, wie die in Abschnitt 2.2 auf Seite 9 formulierten Ziele in der Implementierung umgesetzt wurden. Um RoboSherlock um eine Reasoning-Komponente zu erweitern, wurde der Annotator `objectReasoningAnnotator` (OR) geschrieben. Für die Kommunikation mit `bullet_reasoning` wurde ein ROS-Service erstellt. Im nun folgenden Abschnitt 4.1 werden zunächst die im Rahmen dieser Arbeit verwendeten Analysis Engines und Annotatoren beschrieben, um anschließend in Abschnitt 4.2 auf Seite 41 auf die Implementierung einzugehen.

4.1. Verwendete Analyse Engines und Annotatoren

In diesem Abschnitt werden die in der Implementierung verwendeten Analysis Engines (AEs) und die darin genutzten Annotatoren beschrieben. Sie bilden die Perzeptionspipeline, die für das Erkennen und Klassifizieren der Objekte in einer Szene zuständig ist. Die Annotatoren werden in der Reihenfolge ausgeführt, in der sie an dieser Stelle beschrieben werden. Viele Annotatoren verwenden bei ihrer Analyse Daten, die zuvor von den jeweils vor ihnen ausgeführten Annotatoren zur Common Analysis Structure (CAS) hinzugefügt wurden. Zur Veranschaulichung beziehen sich einige Beschreibungen auf die Beispielaufnahme einer Szene. Ein Ausschnitt des nicht verarbeiteten Rohbildes dieser Szene ist in Abbildung 4.1 zu sehen.

4.1.1. Lowlevel-Analysis-Engine

Die Lowlevel-Analysis-Engine extrahiert alle für die Objekterkennung und -klassifizierung benötigten Daten aus der Szene. Die von jedem Annotator extrahierten und weiterverarbeiteten Daten werden zur CAS hinzugefügt und nach dem Durchlaufen einer jeden Iteration in eine MongoDB-Datenbank geschrieben. Diese Daten werden später von der in Abschnitt

Abbildung 4.1.: Zu sehen ist ein Ausschnitt des Rohbildes einer RGB-Kamera. Sechs Objekte stehen auf einer Arbeitsfläche der Küche.

4.1.2 auf Seite 39 beschriebenen Analysis Engine (AE) benötigt. Es folgt eine Beschreibung aller von der Lowlevel-Analysis-Engine verwendeten Annotatoren.

4.1.1.1. CollectionReader

Der `CollectionReader` liest bei seiner Initialisierung einmalig eine semantische Karte der Umgebung, in welcher der Roboter eingesetzt wird, ein. Der Pfad zu dieser Karte muss dem `CollectionReader` über eine Konfigurationsdatei mitgeteilt werden.

Außerdem blockiert dieser Annotator in jeder Iteration so lange den weiteren Verlauf der AE, bis er von allen verwendeten Kameras ein Bild empfängt. Die erhaltenen Bilder werden dann in der CAS als Rohdaten gespeichert. Der `CollectionReader` kann wahlweise Daten von echten Kameras empfangen oder die Kameradaten aus einer MongoDB-Datenbank extrahieren. Das gewünschte Verfahren kann über die Konfigurationsdatei des Annotators festgelegt werden. In dieser AE werden die Daten von echten Kameras empfangen.

4.1.1.2. ImagePreprocessor

Dieser Annotator stellt sicher, dass das RGB-Bild in den benötigten Auflösungen 1280 px × 960 px und 640 px × 480 px in der CAS vorhanden ist. Liegt das Bild nur in einer der beiden

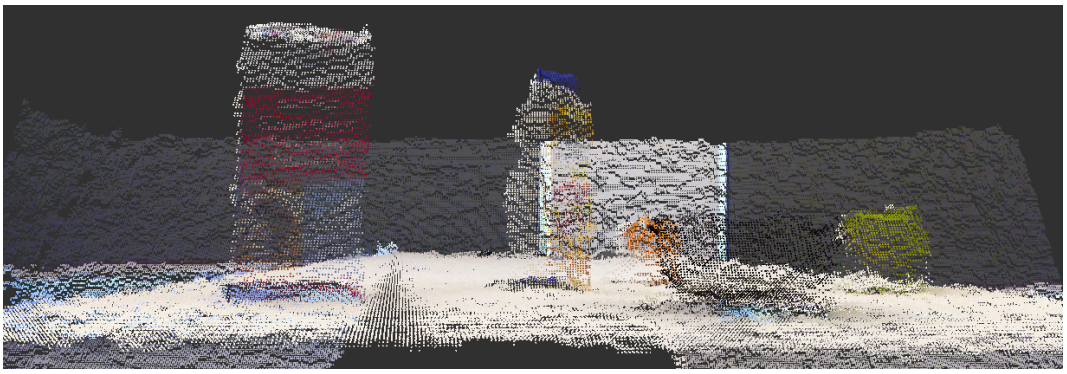

Abbildung 4.2.: Ausschnitt der aus dem RGB-Bild und den Tiefeninformationen erstellten Punktwolke der aufgenommenen Szene.

Auflösungen vor, so wird es in die jeweils andere Auflösung skaliert. Ähnliches gilt für das Tiefenbild. Es wird überprüft, ob es in den Auflösungen 1280 px × 960 px und 640 px × 480 px jeweils in vorzeichenloser 16-Bit-Integer-Darstellung und in 32-Bit-Gleitkommazahl-Darstellung vorliegt. Wenn mindestens ein Format vorhanden ist, kann es in die anderen Formate konvertiert werden.

Weiterhin erstellt der `ImagePreprocessor` aus den 2D- und 3D-Daten eine Punktwolke, welche die aufgenommene Szene repräsentiert (siehe Abbildung 4.2). Alle skalierten oder konvertierten Bilder sowie die erstellte Punktwolke werden der CAS hinzugefügt.

4.1.1.3. RegionFilter

Der `RegionFilter` filtert die Bereiche aus der Punktwolke der Szene heraus, die für die Objekterkennung relevant sind. Dazu liest er aus der vom `CollectionReader` geladenen semantischen Karten die Spezifikationen aller Objekte vom Typ `CounterTop` aus. Dieser Typ beschreibt Objekte, die in der Welt als Tischplatte oder als Arbeitsfläche dienen. Anhand der Spezifikationen kann nun für jeden einzelnen Punkt in der Punktwolke bestimmt werden, ob dieser sich im relevanten Bereich befindet. Als relevanter Bereich gelten alle Orte, die sich in oder über einem Objekt vom Typ `CounterTop` befinden. Alle nicht relevanten Punkte werden aus der Punktwolke entfernt, sodass lediglich für die Objekterkennung bedeutungsvolle Punkte zurückbleiben (siehe Abbildung 4.3). Da somit die Anzahl der vorhandenen Punkte in der Punktwolke um ein erhebliches Maß reduziert werden kann, wird die Rechenzeit für

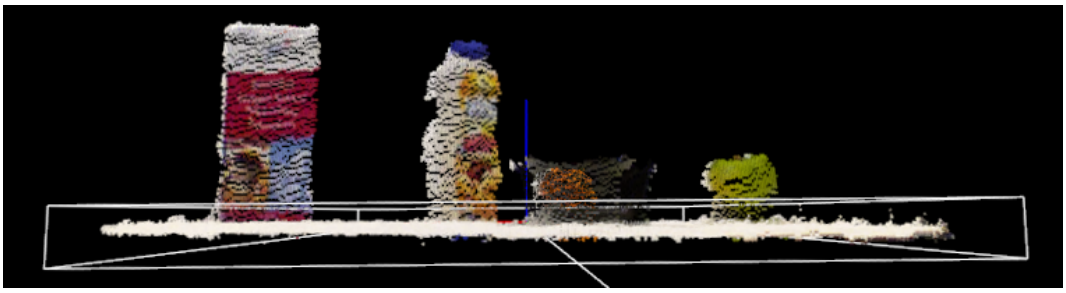

Abbildung 4.3.: Die Punktwolke, in der ursprünglich die gesamte Szene erfasst war, enthält jetzt nur noch Punkte aus relevanten Regionen.

noch folgende Prozesse potenziell um ein Vielfaches verringert. Die neu entstandene Punktwolke wird der CAS hinzugefügt. Dieser Annotator ist für den in Abschnitt 2.2 auf Seite 9 (Punkt 4) beschriebenen Effekt verantwortlich, dass die auf dem Tisch stehenden Cluster beschnitten werden.

4.1.1.4. NormalEstimator

Dieser Annotator berechnet für alle in der Punktwolke erkannten Oberflächen ihren Normalenvektor (siehe Abbildung 4.4). Anhand des Normalenvektors einer Oberfläche kann bestimmt werden, ob diese der Kamera zu- oder abgewandt ist. Die errechneten Vektoren werden der CAS hinzugefügt.

4.1.1.5. PlaneAnnotator

Der `PlaneAnnotator` erkennt Flächen, auf denen potenziell Objekte stehen können, und segmentiert die dazugehörigen Punkte der Punktwolke (siehe Abbildung 4.5). Damit können Punkte, die zu diesen Flächen gehören, bei der Erkennung von Objekten ignoriert werden.

4.1.1.6. PointCloudClusterExtractor

Der `PointCloudClusterExtractor` extrahiert aus der Szene alle nicht zu einer Ebene gehörenden Cluster und fügt sie der CAS hinzu. Ein Cluster ist eine Gruppe von Punkten, die

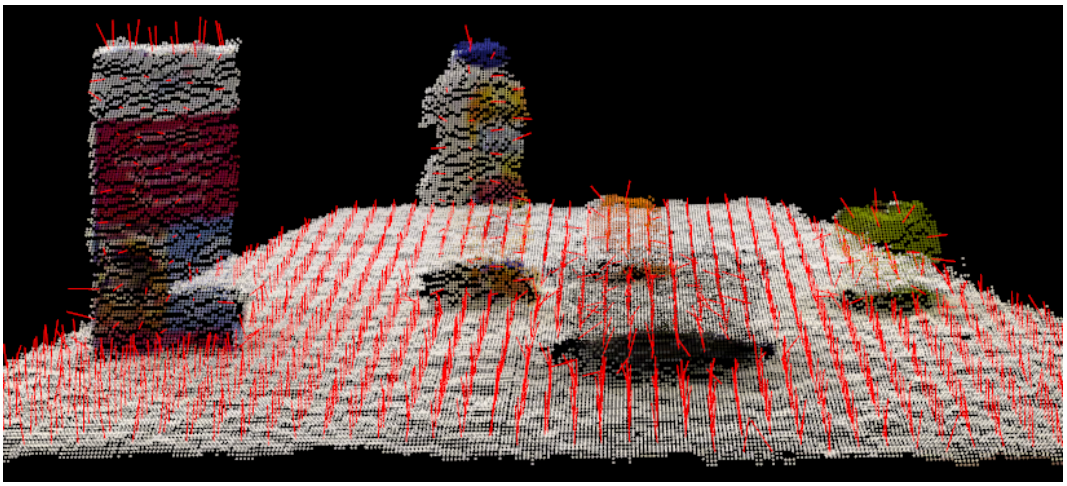

Abbildung 4.4.: Die roten Pfeile stellen die Normalenvektoren der Oberflächen in der Punktwolke dar.

in einem räumlichen Zusammenhang zueinander stehen. In Abbildung 4.6 sind beispielhaft die Cluster der Objekte aus Abbildung 4.1 zu sehen.

4.1.1.7. Cluster3DGeometryAnnotator

Dieser Annotator berechnet für jedes gefundene Cluster seine Bounding Box und deren Pose im Koordinatensystem der Welt und im Koordinatensystem der Kamera. Die Bounding Box enthält Angaben zu Höhe, Tiefe, Breite und Volumen. Die berechneten Daten werden der CAS hinzugefügt. In Abbildung 4.7 werden die Bounding Boxes visualisiert.

4.1.1.8. ClusterTFLocationAnnotator

Der `PointCloudClusterExtractor` verwendet die vom `CollectionReader` in der CAS gespeicherte semantische Karte, sowie die vom `Cluster3DGeometryAnnotator` berechneten Posen, um für jedes in der Szene gefundene Cluster zu bestimmen, ob es sich über einem Tisch bzw. einer Arbeitsfläche oder innerhalb einer Schublade befindet. Das Ergebnis wird mit Hilfe eines semantischen Symbols (`on top of` oder `inside of`) zur CAS hinzugefügt.

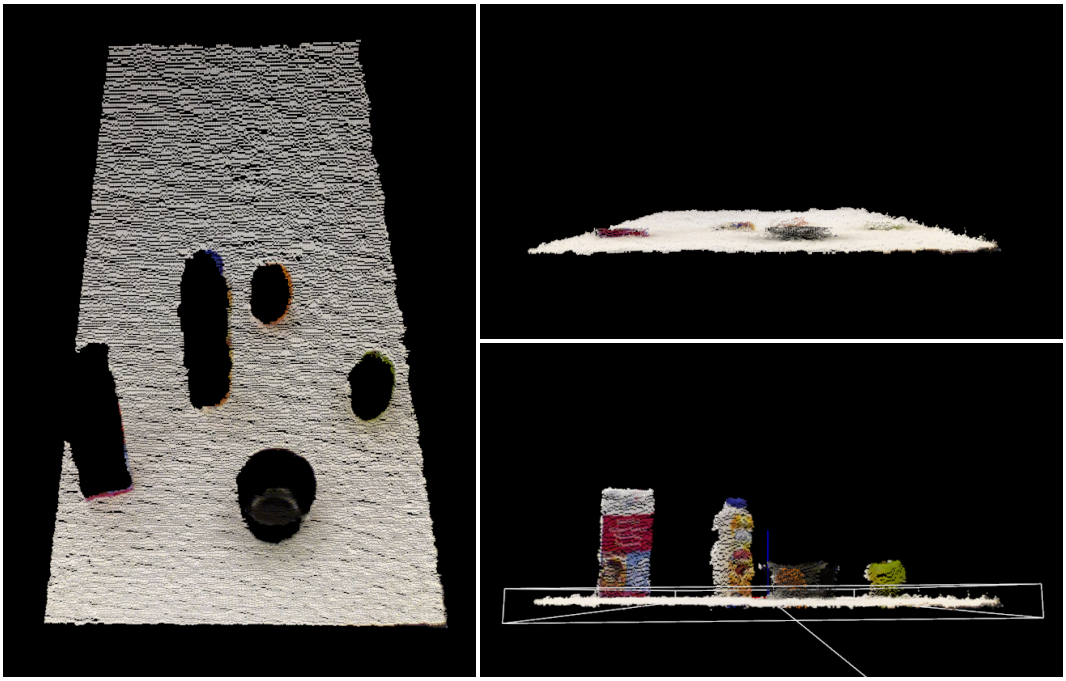

Abbildung 4.5.: *Links:* Die segmentierte Ebene. Die schwarzen Stellen innerhalb der Ebene entstehen durch die Schatten, die die Objekte werfen. Innerhalb der Schatten kann von der Tiefenkamera nichts wahrgenommen werden. *Oben:* Die segmentierte Ebene aus einer anderen Perspektive. Die zu den Objekten gehörenden Punkte sind nicht in der Ebene enthalten. *Unten:* Zum Vergleich ist hier die Punktwolke zu sehen, wie sie vor der Segmentierung aussah.

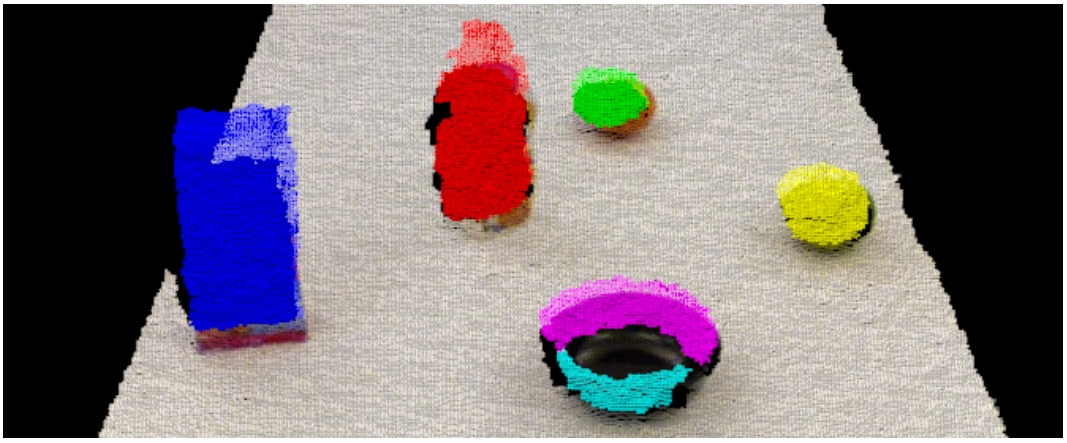

Abbildung 4.6.: Die erkannten Cluster sind farbig markiert. Um die Zugehörigkeiten der einzelnen Punkte zu ihren Clustern zu betonen, wurden sie in verschiedenen Farben dargestellt. Jede Farbe repräsentiert ein unterschiedliches Cluster.

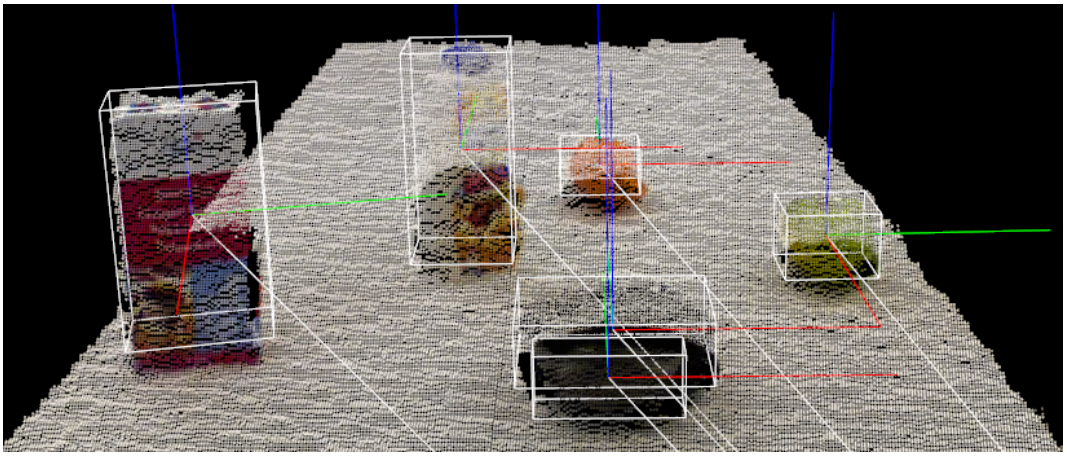

Abbildung 4.7.: Die weißen Kästen stellen die Bounding Boxes der gefundenen Cluster dar. Ihr Ursprung liegt in ihrem jeweiligen Zentrum. Die farbigen Linien zeigen jeweils die drei Achsen im Koordinatensystem der jeweiligen Cluster. Die weißen Linien, die aus den Ursprüngen der Cluster entspringen, führen zum Ursprung des Koordinatensystems der Welt und stellen somit die Translation der Bounding Boxes dar.

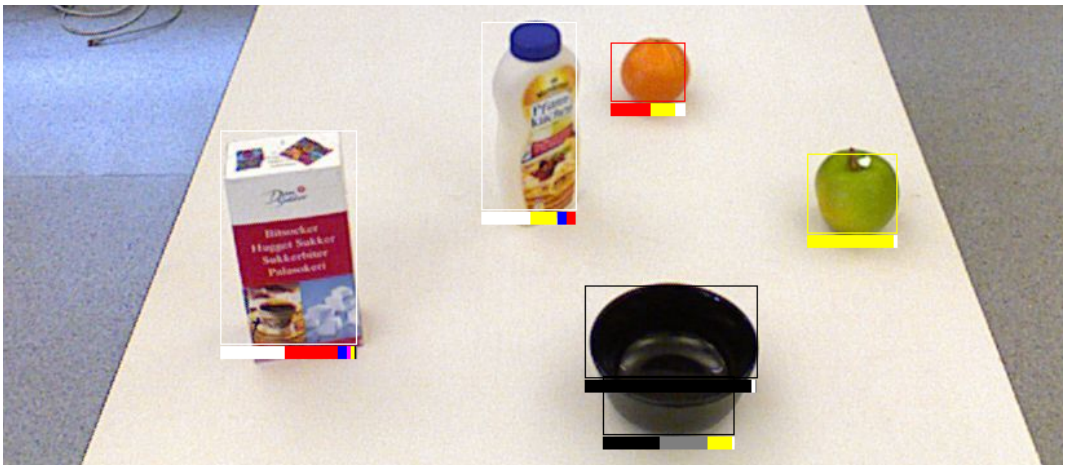

Abbildung 4.8.: Zu jedem Cluster wird die Häufigkeitsverteilung der Farben dargestellt.

4.1.1.9. ClusterColorHistogramCalculator

Dieser Annotator berechnet für jedes Cluster die Farbverteilung und erstellt daraus ein Farbhistogramm. Die Häufigkeitsverteilung wird für die Farben Rot, Gelb, Grün, Cyan, Blau, Magenta, Schwarz und Grau berechnet. Das erstellte Histogramm wird anschließend der CAS hinzugefügt. Abbildung 4.8 zeigt am Beispiel die zu den Clustern gehörigen Häufigkeitsverteilungen.

4.1.1.10. PrimitiveShapeAnnotator

Der `PrimitiveShapeAnnotator` bestimmt für jedes Cluster dessen primitive geometrische Form und fügt das Ergebnis zur CAS hinzu. Es wird zwischen den Formen `round` und `box` unterschieden.

4.1.1.11. SacModelAnnotator

Der `SacModelAnnotator` erkennt, ob ein Cluster eine zylindrische Form hat. Ist dies der Fall, wird diese Information in Form einer Annotation zur CAS hinzugefügt.

4.1.2. Reasoning-Analysis-Engine

Die in Abschnitt 4.1.1 auf Seite 31 beschriebenen Annotatoren bilden eine Grundlage für die Reasoning-Analysis-Engine. Die von der Lowlevel-Analysis-Engine weiterführende Analyse der CAS erfordert, dass zu untersuchenden Daten in einer MongoDB-Datenbank vorliegen. Im Folgenden werden die von dieser AE verwendeten Annotatoren beschrieben.

4.1.2.1. CollectionReader

Die Funktionsweise dieses Annotators wurde bereits in Abschnitt 4.1.1.1 auf Seite 32 erläutert. In dieser AE werden die Daten vom `CollectionReaders` allerdings nicht von echten Kameras empfangen, sondern aus der MongoDB-Datenbank gelesen, welche von der Lowlevel-Analysis-Engine zum Speichern der Daten verwendet wurde.

4.1.2.2. ObjectIdentityResolution

Dieser Annotator erstellt auf Grundlage der vom `PointCloudClusterExtractor` extrahierten Cluster Objektinstanzen mit eindeutigen IDs (siehe Abbildung 4.9). Dazu werden werden die Cluster in jeder Iteration analysiert und es wird versucht, sie anhand ihrer Eigenschaften Objektinstanzen zuzuordnen, die bereits in früheren Iterationen erstellt wurden. Um dies zu erreichen, werden die Cluster der aktuellen Iteration mit den Clustern der Objekte der vorherigen Iterationen verglichen und auf Ähnlichkeiten in ihren Charakteristika überprüft. Wurde eine Übereinstimmung gefunden, werden die Metadaten des entsprechenden Objekts mit den Daten des kongruenten, in dieser Iteration erkannten Clusters aktualisiert. Dies soll eine Identifizierung der erkannten Objekte über die Zeit ermöglichen.

4.1.2.3. MLNAtomsGenerator

Dieser Annotator verwendet ein Markov Logic Network (MLN) (beschrieben in Abschnitt 3.5 auf Seite 26) und nutzt damit die Vorteile der probabilistischen Logik, um Objekte anhand ihrer Attribute klassifizieren zu können (siehe [NBB14]). Um eine Klassifizierung zu ermöglichen, wird eine Wissensbasis benötigt. Diese wird im Rahmen der Arbeit erstellt, indem jedes der zu erlernenden Objekte aus verschiedenen Perspektiven von den Kameras des Roboters erfasst wird. Anschließend werden Farbe, geometrische Form und Größe der Objekte bestimmt und in einer Trainingsdatenbank erfasst. Tabelle 4.1 zeigt die möglichen Werte

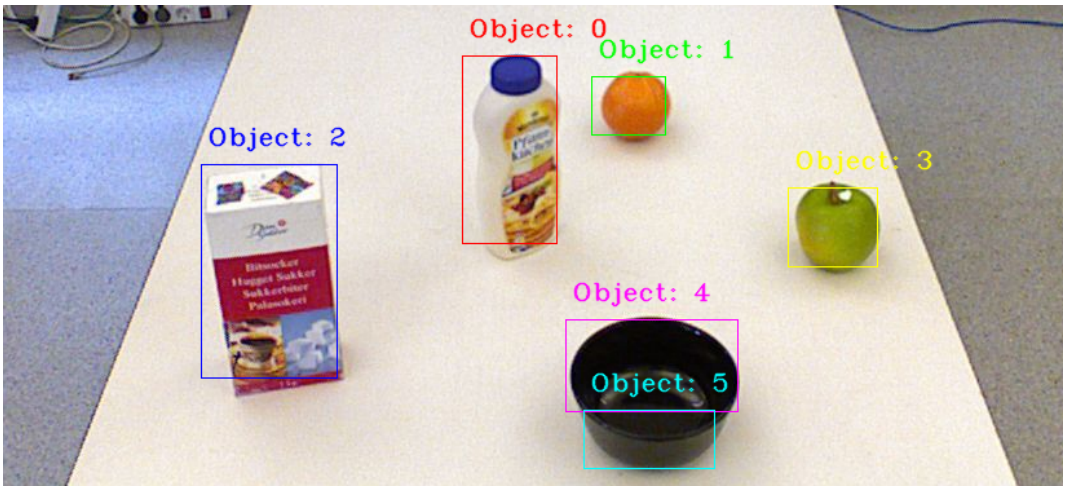

Abbildung 4.9.: Die vom `ObjectIdentityResolution`-Annotator erstellten Objekte. Jedes besitzt eine eindeutige ID.

Tabelle 4.1.: Vom `MLNAtomsGenerator` zur Klassifizierung verwendete Attribute und ihre möglichen Wertzuweisungen.

Attribut	Wertebereich
`size`	{`small`, `medium`, `large`, `extraLarge`}
`shape`	{`round`, `box`, `cylinder`, `flat`}
`color`	{`red`, `yellow`, `green`, `cyan`, `blue`, `magenta`, `white`, `black`, `grey`}

für diese drei Objektattribute. Aus der erstellten Trainingsdatenbank wird mit dem *MLN Learning Tool* eine Wissensbasis erstellt. Die Generierung einer Wissensbasis muss nur einmalig erfolgen. Sie bleibt nach jedem Betrieb des Roboters erhalten, sodass sie immer wieder verwendet werden kann.

Anhand der bereits in der CAS annotierten Daten und der erstellten Wissensbasis kann der `MLNAtomsGenerator` nun berechnen, mit welcher Wahrscheinlichkeit es sich bei einem detektierten Cluster um ein Objekt einer bestimmten Klasse handelt. Die Klasse, die den höchsten Wahrscheinlichkeitswert besitzt, wird dem Cluster in der CAS als Objekttyp in Form einer Annotation hinzugefügt. Abbildung 4.10 zeigt die den Cluster zugeordneten Attribute und Objektklassen.

Da bei diesem Verfahren jedes erkannte Cluster mit einer Klasse assoziiert wird, in der

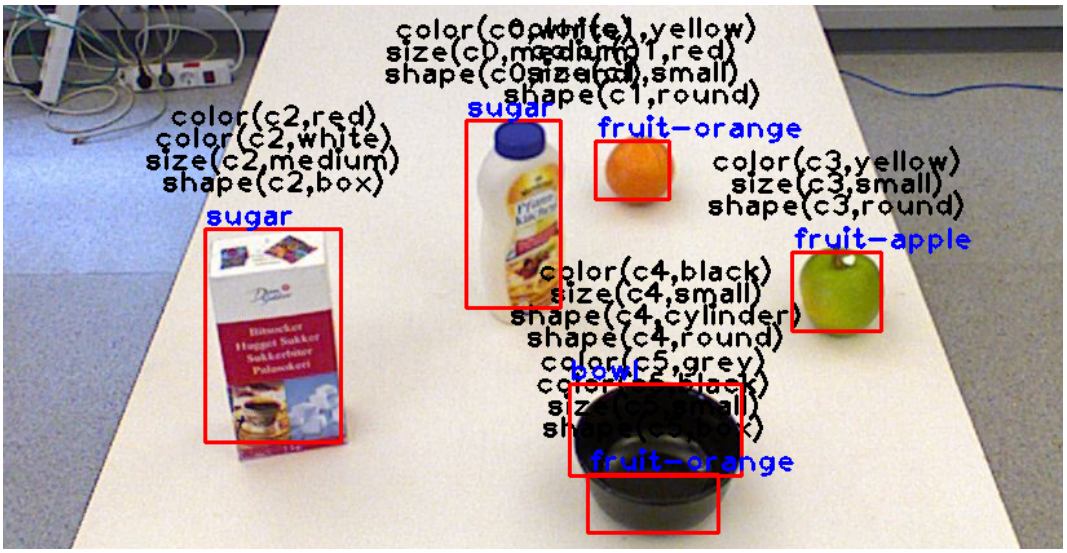

Abbildung 4.10.: Die Objektattribute von jedem Cluster werden in schwarzer Schrift, die erkannte Objektklasse in blauer Schrift dargestellt.

Wissensbasis allerdings nur die gelernten Objektklassen vorhanden sind, können unbekannte Objekte nicht als solche erfasst werden. Sie werden der Objektklasse zugeordnet, der sie am ähnlichsten sind.

4.2. Implementierung

In diesem Abschnitt wird die im Rahmen dieser Arbeit durchgeführte Implementierung beschrieben. Diese teilt sich in zwei große Komponenten auf – das `bullet_reasoning_interface` (`btri`) und den `ObjectReasoningAnnotator`, einen Annotator für RoboSherlock. Der gesamte Quelltext ist auf dem dieser Arbeit beigefügten Datenträger einzusehen.

4.2.1. Bullet Reasoning Interface

Das `bullet_reasoning_interface` (`btri`)[1] wurde als ROS-Paket in `cram_physics` integriert und daher in der Sprache *Common Lisp* geschrieben. Es bildet eine Schnittstelle zu der in Abschnitt 3.2.2 auf Seite 20 beschriebenen simulierten Bullet-Welt. Diese Schnittstelle wurde in Form eines ROS-Service umgesetzt. Das bedeutet, dass dem `btri` von einer beliebigen ROS-Node eine Anfrage gesendet werden kann. Das `btri` bearbeitet diese Anfrage und schickt anschließend eine Antwort an die aufrufende Node zurück. Über das `btri` ist es möglich, Objekte in eine Bullet-Welt zu laden, vorhandene Objekte zu bewegen, Objekte zu entfernen und die Welt zu simulieren. Weiterhin können auf Wunsch diverse Berechnungen durchgeführt werden. Berechnet werden kann, ob Objekte stabil stehen, ob sie ein Küchenmöbel berühren, ob sie mit anderen Objekten kollidieren, ob sie aus der Sicht des Roboters sichtbar sind und ob sie von anderen Objekten verdeckt werden. Tabelle 4.2 zeigt alle Parameter, die beim Aufrufen des Services gesetzt werden können. Die relativ hohe Anzahl an Parametern erklärt sich durch die Tatsache, dass die Berechnungen durch die Bullet-Physik-Engine relativ viel Zeit in Anspruch nehmen (siehe Tabelle 5.1). Da der Service in verzögerungsarmen Echtzeit-Robotersystemen eingesetzt werden soll, ist es somit dem Aufrufer überlassen zu entscheiden, welche Berechnungen durchgeführt werden sollen.

Tabelle 4.2.: Die Parameter des vom `btri` angebotenen Services.

Parameter	Typ
`operationsWithCopy`	`bool`
Gibt an, ob die an den Objekten durchgeführten Operationen in einer Kopie der simulierten Welt durchgeführt werden.	
`removeAllObjects`	`bool`
Gibt an, ob vor den an den Objekten durchzuführenden Operationen alle bisher in die Welt geladenen Objekte aus dieser entfernt werden sollen.	
`operations`	`ObjectOperation[]`
Ein Array von durchzuführenden Objektoperationen.	
`updateCameraPose`	`bool`
Gibt an, ob die Pose der Kamera aktualisiert werden soll.	

[1]↑Im der Arbeit angehängten Datenträger zu finden unter `cram_physics/bullet_reasoning_interface/`

Tabelle 4.2.: (*Fortsetzung*)

Parameter	Typ
`cameraPose`	`geometry_msgs/PoseStamped cameraPose`
Die zu aktualisierende Kamerapose. Alle Berechnungen, die die Sichtbarkeit oder das Verdecken von Objekten betreffen, werden aus der Sicht dieser Pose getätigt.	
`simulate`	`bool`
Gibt an, ob eine Simulation durchgeführt werden soll.	
`simulateWithCopy`	`bool`
Für den Fall, dass eine Simulation durchgeführt werden soll, gibt dieser Wert an, ob dies mit einer Kopie der Welt geschehen soll.	
`duration`	`duration`
Gibt die gewünschte Dauer der Simulation an.	
`computeStability`	`bool`
Gibt an, ob für jedes Objekt (gegebenenfalls erst nach dem Durchführen einer Simulation) berechnet werden soll, ob es stabil ist.	
`computeContactWithKitchenBeforeSimulation`	`bool`
Gibt an, ob für jedes Objekt berechnet werden soll, ob es vor Durchführung der Simulation Teile der Küchenmöbel berührt hat.	
`computeContactWithKitchenAfterSimulation`	`bool`
Gibt an, ob für jedes Objekt berechnet werden soll, ob es nach Durchführung der Simulation Teile der Küchenmöbel berührt.	
`computeCollisionsBeforeSimulation`	`bool`
Gibt an, ob für jedes Objekt berechnet werden soll, welche der anderen Objekte in der Welt es vor Durchführung der Simulation berührt hat.	
`computeCollisionsAfterSimulation`	`bool`
Gibt an, ob für jedes Objekt berechnet werden soll, welche der anderen Objekte in der Welt es nach Durchführung der Simulation berührt hat.	
`computeVisibilityBeforeSimulation`	`bool`
Gibt an, ob für jedes Objekt berechnet werden soll, ob es vor Durchführung der Simulation aus Sicht der Kamerapose sichtbar war.	
`computeVisibilityAfterSimulation`	`bool`
Gibt an, ob für jedes Objekt berechnet werden soll, ob es nach Durchführung der Simulation aus Sicht der Kamerapose sichtbar ist.	

Tabelle 4.2.: (*Fortsetzung*)

Parameter	Typ
`computeOcclusionsBeforeSimulation`	`bool`
Gibt an, ob für jedes Objekt berechnet werden soll, von welchen der anderen Objekten in der Welt es vor Durchführung der Simulation aus Sicht der Kamerapose verdeckt wurde.	
`computeOcclusionsAfterSimulation`	`bool`
Gibt an, ob für jedes Objekt berechnet werden soll, von welchen der anderen Objekten in der Welt es nach Durchführung der Simulation aus Sicht der Kamerapose verdeckt wird.	

Über den Parameter `operations` kann angegeben werden, welche Aktionen innerhalb der simulierten Welt ausgeführt werden sollen. Es können Objekte erstellt, bewegt, oder entfernt werden. Für jede Operation muss eine Objekt-Id angegeben werden, welche bestimmt, auf welches Objekt eine Operation angewendet werden soll. Soll ein neues Objekt in der simulierten Welt erstellt werden, muss ein Objekttyp angegeben werden. Dieser Typ entspricht der vom `MLNAtomsGenerator` erkannten Objektklasse eines Objekts. Für jeden Objekttyp sind im Paket `cram_projection_demos` in einer internen Datenbank Informationen hinterlegt. Für Objektklassen mit komplexen geometrischen Formen liegen 3D-Modelle vor; für Klassen mit primitiven Formen wie Boxen und Kugeln sind die Abmessungen hinterlegt. Weiterhin kann beim Erstellen eines Objekts eine Farbe angegeben werden, in der das Objekt erscheinen soll. Soll ein Objekt erstellt oder ein bereits in der Welt existierendes Objekt bewegt werden, müssen eine Pose und die Bounding Box des Objekts angegeben werden. Der Parameter `operations` erwartet als Eingabe ein Array vom *Message*-Typ `ObjectOperation`, welcher in diesem Paket definiert ist (siehe Listing 4.1 auf der nächsten Seite).

Das `btri` sendet nach dem Bearbeiten einer Anfrage eine Antwort, in deren Feld `worlds` ein Array vom Typ `WorldIntel` an den Aufrufer zurückgegeben wird. Dieser Typ ist eine Message, die ebenfalls in diesem Paket definiert ist (siehe Listing 4.2 auf der nächsten Seite). Abhängig davon, wie viele Kopien der simulierten Welt beim Aufruf des `btri` erstellt wurden, enthält das Feld `worlds` Informationen von bis zu drei Welten (der „normalen“ simulierten Welt, der Kopie, in der die Operationen stattgefunden haben, und der Kopie, in der simuliert wurde). Diese Informationen beinhalten wiederum Daten über jedes in der jeweiligen Welt vorhandene Objekt. Die Daten sind vom Typ `ObjectIntel` und treffen Aussage darüber, ob ein Objekt stabil ist, ob es sichtbar ist, ob es Kontakt mit einem Küchenmöbel hat, welche anderen Objekte es berührt, von welchen anderen Objekten es verdeckt wird, welche

Listing 4.1: Definition der Message `bullet_reasoning_interface/ObjectOperation`

```
# Valid operations
uint32 SPAWN_OBJECT = 0
uint32 MOVE_OBJECT = 1
uint32 REMOVE_OBJECT = 2
uint32 QUERY_STATUS = 3

# Object's id
uint32 id

# Operation to execute
uint32 operation

# Object's type. Only necessary when spawning an object.
uint32 type

# Object's stamped pose. Only necessary when spawning or moving an object.
geometry_msgs/PoseStamped poseStamped

# Object's color. Only necessary when spawning an object.
std_msgs/ColorRGBA color

# Object's bounding box in world's fixed frame. Only necessary when spawning or moving an object.
geometry_msgs/Vector3 boundingBox
```

Pose und welche Maße es besitzt, und darüber, ob die Operation, welche auf das Objekt angewandt werden sollte, erfolgreich durchgeführt werden konnte. Eine genaue Definition des Typs `ObjectIntel` findet sich im Anhang im Listing A.1 auf Seite 67.

Listing 4.2: Definition der Message `bullet_reasoning_interface/WorldIntel`

```
# Available world types
string CURRENT_WORLD = "current world"
string OPERATED_WORLD = "operated world"
string SIMULATED_WORLD = "simulated world"

# Type of this world
string worldType

# Indicates if this world is stable
bool isStable

# Information for every object in this world
ObjectIntel[] objects
```

4.2.2. ObjectReasoningAnnotator

Der `objectReasoningAnnotator` (OR)[2] ist ein für RoboSherlock geschriebener Annotator und wurde daher in der Programmiersprache *C++* verfasst. In ihm werden die in

[2]↑Im der Arbeit angehängten Datenträger zu finden unter `iai_robosherlock/object_reasoning/`

Abschnitt 2.2 auf Seite 9 beschriebenen Ideen umgesetzt. Er verbessert die in Abschnitt 4.1 auf Seite 31 beschriebene Perzeptionspipeline dahingehend, dass zuverlässigere und konsistentere Ergebnisse produziert werden.

Die durch den OR betriebene Analyse ist in zwei Phasen aufgeteilt. In der ersten Phase wird Reasoning auf den Daten der aktuellen CAS und der CASs vergangener Iterationen betrieben. Anschließend wird das `btri` zum Aktualisieren der Bullet-Welt aufgerufen, um daraufhin in der zweiten Phase anhand der vom `btri` zurückgegebenen Ergebnisse weitere Erkenntnisse gewinnen zu können. Zu Beginn einer jeden Iteration werden alle vom `PointCloudClusterExtractor` annotierten Cluster und alle von der `ObjectIdentity Resolution` annotierten Objekte aus der CAS extrahiert. Bevor Reasoning auf den Objektdaten betrieben wird, wird der erste Punkt der im Rahmen dieser Arbeit umzusetzenden Ziele abgearbeitet:

1. Erkennen von Störungen im Sichtfeld des Roboters

   Da die von menschlichen Extremitäten erzeugten Cluster im Vergleich zu den in einer Küche vorkommenden Gegenständen eine erheblich größere Bounding Box besitzen (siehe Abbildung 2.1), wurde das Erkennen von Störungen im Bild von den Dimensionen der Bounding Boxes der Cluster abhängig gemacht. Überschreitet die Höhe, die Breite, die Tiefe oder das Volumen einer Bounding Box einen bestimmten Wert, wird davon ausgegangen, dass eine Störung in der aktuellen Iteration vorliegt. In diesem Fall wird keine weitere Analyse auf den Daten der aktuellen Iteration betrieben, sondern auf das Eintreffen neuer Daten gewartet.

Konnte keine Störung festgestellt werden, beginnt die erste Phase des Reasonings.

2. Identifizieren von Objekten über mehrere Iterationen

   Im `ObjectIdentityResolution`-Annotator wird bereits versucht, Objekte über mehrere Iterationen hinweg zu identifizieren. Dabei werden jedoch häufig falsche Ergebnisse produziert, die vom OR auf Konsistenz geprüft und gegebenenfalls korrigiert werden.

   Um Objekte über mehrere Iterationen hinweg identifizieren zu können, werden die in der aktuellen Iteration erkannten Objekte mit Objekten aus vorherigen

Iterationen verglichen. Um festzustellen, ob es sich bei zwei miteinander zu vergleichenden Objekten um dasselbe Objekt handelt, werden die Position, die Dimensionen, der Typ und die vom `ObjectIdentityResolution`-Annotator zugewiesene ID des jeweiligen Objekts sowie deren Änderung über die Zeit analysiert. Stand beispielsweise ein Objekt A in der vorherigen Iteration an einer Position x und steht in der aktuellen Iteration ein Objekt B an einer Position y, die minimal von x abweicht, und stimmen Dimensionen und Typen der Objekte überein, ist mit einer hohen Wahrscheinlichkeit davon auszugehen, dass es sich bei den Objekten A und B um dasselbe Objekt handelt. Ähnliches gilt für Objekte, die ihre Positionen getauscht haben oder in der Szene bewegt wurden.

Schwieriger ist das Identifizieren von Objekten, wenn die von den anderen Annotatoren stammenden Daten fehlerhaft sind. Um in solchen Fällen ein gewisses Maß an Robustheit zu erlangen, werden die verschiedenen Objektattribute bei der Analyse unterschiedlich gewichtet. Das beruht auf der Tatsache, dass z. B. die Klassifizierung von Objekten öfter fehlschlägt als die Bestimmung der Objektdimensionen. Weiterhin werden die Objektattribute nicht nur mit den aktuellsten verfügbaren Daten verglichen, sondern es werden Daten aus weiter in der Vergangenheit liegenden Iterationen mit einbezogen.

Objekte, die in der aktuellen Iteration das erste Mal vom Roboter gesehen wurden, werden als neue Objekte in den Bestand aufgenommen. Wurde ein Objekt A aus der aktuellen Iteration als ein Objekt B aus einer vorherigen Iteration identifiziert, werden die Daten von B mit den Daten von A aktualisiert.

3. Konsistenz bei der Objektklassifizierung

   Da die durch den `MLNAtomsGenerator` durchgeführte Klassifizierung von Objekten in einem hohen Maße anfällig gegen Änderungen in der Belichtung der Szene ist, kommt es häufig zu falschen Klassifizierungen. Um diesem Effekt entgegenzuwirken, werden im OR bei der Bestimmung der Klasse eines Objekts ebenfalls die Ergebnisse aus Iterationen aus der Vergangenheit betrachtet. Wurde ein Objekt beispielsweise in der aktuellen Iteration vom `MLNAtomsGenerator` als `sugar` eingeordnet, in den letzten 20 Iterationen aber als `mondamin`, ist davon auszugehen, dass die Klassifizierung der aktuellen Iteration fehlerhaft war.

An diesem Punkt ist die erste Phase des Reasonings beendet. Objekte wurden identifiziert und aktualisiert oder als neue Objekte in den Bestand aufgenommen; die Konsistenzprüfung

der Klassifizierung wurde abgeschlossen. Im nächsten Schritt werden die neuen Daten an die Bullet-Welt weitergegeben, um deren Zustand an den Zustand der echten Welt anzupassen. Dies geschieht über einen Aufruf des `btri`. Dabei wird für jedes Objekt eine entsprechende Operation übergeben, welche angibt, ob ein Objekt neu erstellt, die Position aktualisiert oder ob es aus der Bullet-Welt entfernt werden soll. Weiterhin wird das `btri` angewiesen, die Welt zu simulieren und Objektkollisionen sowie Berührungen mit den Küchenmöbeln und Verdeckungen von Objekten zu berechnen. Auf Basis der vom `btri` zurückgegebenen Ergebnisse beginnt nun die zweite Phase des Reasonings, in der die beiden verbleibenden in Abschnitt 2.2 auf Seite 9 beschriebenen Ziele umgesetzt werden. Wurden jedoch in der ersten Reasoning-Phase keine Daten erfasst, die eine Änderung in der Bullet-Welt erfordern, wird für die aktuelle Iteration auf einen Aufruf des `btri` verzichtet, um wertvolle Zeit einsparen zu können (vergleiche Tabelle 5.1). Das Durchführen der zweiten Phase ist damit hinfällig.

4. Korrigieren der Höhe und der Position von Objekten

   Nach der Durchführung der Simulation in der Bullet-Welt werden die vom `btri` zurückgegebenen Posen der Objekte mit den Posen der von der Perzeption erkannten Objekte verglichen. Für jedes Objekt wird berechnet, wie weit es sich in der Bullet-Welt unter Einfluss der Gravitation in der z-Komponente seiner Pose im globalen Koordinatensystem – also Richtung Boden – bewegt hat. Anhand dieser Bewegung kann erkannt werden, ob ein Objekt wahrscheinlich auf einer Oberfläche steht, oder ob es in der Luft „schwebt“. Geht man z. B. davon aus, dass die Perzeption von den zu den Objekten gehörenden Clustern maximal 5 cm abschneidet, ein Objekt sich während der Simulation aber 7 cm nach unten bewegt hat, kann daraus abgeleitet werden, dass das Objekt in der echten Welt keinen Kontakt zum Tisch hat, sondern sich in der Luft befindet – z. B. weil es von einer Person in der Hand gehalten wird (siehe Abbildung 2.1). Hat sich ein Objekt aber weniger als diese 5 cm bewegt, kann davon ausgegangen werden, dass es nicht weiter fallen konnte und nun festen Kontakt zu einem Küchenmöbel hat. Die Distanz, über die sich dieses Objekt bewegt hat, entspricht dann dem Stück, das die Perzeption vom zum Objekt gehörenden Cluster abgeschnitten hat.

   Aus den Bewegungen aller Objekte in der Bullet-Welt, die nicht als „schwebend“ eingeordnet wurden, kann nun die Höhenkorrektur berechnet werden, um der vom `RegionFilter` verursachten Beschneidung der Objektcluster entgegenzuwirken.

5. Erkennen von verdeckten Objekten

   Existierte ein Objekt in der vorherigen Iteration noch in der Bullet-Welt, ist in der aktuellen Iteration in der Bullet-Welt aus Sicht des Roboters aber nicht mehr zu sehen, kann es dafür zwei Gründe geben: Entweder wurde das Objekt in der echten Welt tatsächlich entfernt oder es wird von mindestens einem anderen Objekt verdeckt. Um überprüfen zu können, welcher Fall zutrifft, wird dem `btri` beim Aufrufen des Services die Anweisung gegeben, für jedes Objekt zu berechnen, ob und von welchen anderen Objekten es aus Sicht des Roboters verdeckt wird. Wird es in der Bullet-Welt von keinem anderen Objekt verdeckt, ist davon auszugehen, dass es in der echten Welt aus der Szene entfernt wurde. Wird es allerdings in der Bullet-Welt von einem anderen Objekt verdeckt, kann angenommen werden, dass in der echten Welt ein anderes Objekt vor das betroffene Objekt gestellt wurde und es deshalb für den Roboter nicht mehr sichtbar ist. Trifft letzterer Fall zu, nimmt der OR an, dass sich das Objekt noch an seiner alten Position befindet.

Dem `ObjectReasoningAnnotator` können folgende für das Reasoning relevante Parameter in einer Konfigurationsdatei[3] übergeben werden:

`max_object_size`
`max_object_volume`

`max_object_size` gibt die maximale Länge an, die ein erkanntes Objekt pro Dimension haben darf; `max_object_volume` gibt das maximale Volumen an, das ein erkanntes Objekt besitzen darf. Wird ein Objekt erkannt, das einen dieser Werte überschreitet, wird die aktuelle Iteration, wie in Punkt 1 beschrieben, vom Reasoning ausgenommen.

`height_correction_limit`
`height_correction_control_threshold`
`height_correction_tolerance_limit`
`height_correction_tolerance_factor`

Das Resultat der Berechnung, ob ein Objekt auf einer Oberfläche steht, oder ob es „schwebt“, hängt von der in Punkt 4 bereits ermittelten Höhenkorrektur δ, der Distanz σ und den vier oben genannten Parametern ab. σ ist Länge der Strecke, die sich ein Objekt während der Simulation in der z-Komponente des globalen Koordinatensystems bewegt hat. Wurde noch keine Höhenkorrektur berechnet ($\delta \leq 0$), wird für ein Objekt, für das

[3] Die Konfigurationsdatei befindet sich im Wurzelverzeichnes des Pakets `object_reasoning` unter `descriptors/annotators/ObjectReasoningAnnotator.xml`

$\sigma \leq$ `height_correction_limit` gilt, angenommen, dass es nicht schwebt. Wurde schon eine Höhenkorrektur $\delta > 0$ bestimmt, muss $\sigma \leq \delta \cdot$ `height_correction_tolerance_factor` und $\sigma \leq$ `height_correction_tolerance_limit` gelten, damit ein Objekt als nicht schwebend eingeordnet wird.

`object_not_seen_tolerance`

Bei diesem Parameter handelt es sich um eine Zeitangabe. Wenn ein Objekt länger als `object_not_seen_tolerance` nicht vom Roboter gesehen wurde und es von keinem anderen Objekt verdeckt wird, wird es aus dem Belief State des Roboters entfernt. Dieser Parameter existiert, da es vorkommen kann, dass obwohl sich ein Objekt permanent für den Roboter sichtbar in der Szene befindet, dieses Objekt in einer Iteration n erkannt, in Iteration $n+1$ nicht erkannt und in Iteration $n+2$ wieder erkannt wird. Um das Objekt in der Iteration $n+1$ nicht sofort aus dem Belief State zu entfernen, nur um es eine Iteration später wieder einzufügen, wurde `object_not_seen_tolerance` als Schwellwert eingeführt. So werden Objekte erst entfernt, wenn sie über eine längere Zeit (z. B. 1,5 s) nicht gesehen wurden.

`overlapping_tolerance`

Wenn die kürzeste Distanz zwischen zwei Bounding Boxes der Cluster zweier Objekte kleiner-gleich `overlappingTolerance` ist, wird davon ausgegangen, dass es sich bei beiden Objekten um ein einziges Objekt handelt. In diesem Fall werden beide Objekte zu einem zusammengefasst.

`occlusion_threshold`

Wenn ein Objekt A einen Anteil von `occlusion_threshold` der Fläche von einem Objekt B im RGB-Bild verdeckt, wird davon ausgegangen, dass Objekt B von Objekt A verdeckt wird.

`visible_threshold`

Wenn im RGB-Bild weniger als ein Anteil von `visible_threshold` der Fläche eines Objekts sichtbar ist, wird davon ausgegangen, dass es von anderen Objekten verdeckt wird.

5. Experimente

Abbildung 5.1.: Der Roboter *PR2* von *Willow Garage, Inc.*[1]

In diesem Kapitel werden die im Rahmen dieser Arbeit geschriebenen Algorithmen evaluiert. Dazu wurden Tests in verschiedenen Szenarien in der im Labor der AG Künstliche Intelligenz (IAI) vorhandenen Küchenumgebung durchgeführt. Bei dem verwendeten Roboter handelt es sich um den Personal Robot 2 (PR2) der Firma *Willow Garage, Inc.* (siehe Abbildung 5.1), welcher einen Laserscanner, eine RGB-Kamera und eine Tiefenkamera besitzt.

[1] ↑ Foto von *Willow Garage, Inc.*, `https://www.flickr.com/photos/willowgarage/5362755871` (besucht am 13.07.2015)

Abbildung 5.2.: Die im Rahmen der Experimente verwendeten Objekte. Von links nach rechts: `nesquik`, `sugar`, `mondamin`, `bowl`, `fruit-apple` und `fruit-orange`.

Für die Szenarien stand eine Auswahl von sechs Objekten (siehe Abbildung 5.2) zur Verfügung. Von ihnen wurde im Vorfeld der Experimente eine Wissensbasis erstellt, um eine Klassifizierung der Objekte in einer Szene durch den `MLNAtomsGenerator` zu ermöglichen. Die verschiedenen Szenarien werden im folgenden Abschnitt beschrieben.

5.1. Szenarien

Um die vom `objectReasoningAnnotator` (OR) produzierten Ergebnisse in Bezug auf die in Abschnitt 2.2 auf Seite 9 formulierten Punkte zu testen, wurden drei verschiedene Szenarien konstruiert. In allen drei Szenarien werden Objekte auf einem Tisch abgestellt und durch menschlichen Einfluss manipuliert. In den Szenarien 2 und 3 bewegt sich der Roboter dabei um den Tisch herum. Jedes Szenario besitzt mehrere Schwerpunkte, die beim Testen in den Fokus gerückt werden. Abbildung 5.3 zeigt zu jedem Szenario eine Repräsentation aus der Bullet-Welt.

5.1.1. Szenario 1

In diesem Szenario (siehe Abbildung 5.3, *links*) steht der Roboter während des gesamten Ablaufs still. Zu Beginn ist der Tisch leer. Nacheinander werden nun die Objekte `fruit-apple`, `sugar` und `nesquik` darauf abgestellt, sodass sie sich gegenseitig verdecken. Dabei werden absichtlich viele Störungen durch menschliche Arme und Hände und durch in den Händen gehaltene Objekte erzeugt. Der Roboter soll das Verdecken der Objekte und die

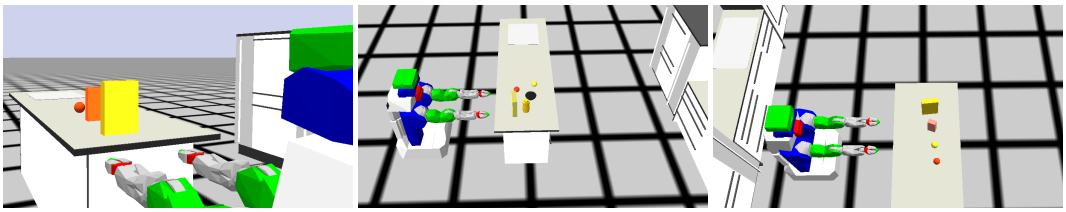

Abbildung 5.3.: *Links:* Szenario 1 – `nesquik` verdeckt `sugar`; ganz hinten befindet sich `fruit-apple`. *Mitte:* Szenario 2 – Es befinden sich auf dem Tisch `fruit-apple` (gelbe Kugel), `fruit-orange` (orange Kugel), `bowl`, `mondamin` und `nesquik`. *Rechts:* Szenario 3 – Sobald der Roboter die Stirnseite des Tisches erreicht hat, verdeckt `nesquik` `sugar` und `sugar` verdeckt `fruit-apple`, gefolgt von `fruit-orange`.

Störungen in dem ihm sichtbaren Bereich erkennen. Die Dauer eines Durchlaufs beträgt circa 2:30 Minuten.

5.1.2. Szenario 2

Das zweite Szenario beginnt damit, dass die Objekte `fruit-orange`, `fruit-apple`, `bowl`, `mondamin` und `nesquik` bereits auf dem Tisch stehen (siehe Abbildung 5.3, *Mitte*). `mondamin` wird aus Sicht des Roboters von `nesquik` verdeckt. Der Roboter beginnt, sich aus seiner Perspektive kontinuierlich nach rechts um den Tisch herum zu bewegen, bis er sich auf der gegenüberliegenden Seite spiegelverkehrt zur Ausgangsposition befindet. Während er sich bewegt, werden ohne Eingriff von außen – durch die Konstellation der Objekte bedingt – verschiedene Objekte von anderen Objekten verdeckt. `bowl` wird vom Tisch entfernt, sobald es von `nesquik` verdeckt wird. Sobald der Roboter die Stelle, an der `bowl` stand, komplett einsehen kann, wird `sugar` an eben diese Stelle gestellt. Durch die Bewegung des Roboters entstehen weitere Verdeckungen von Objekten. Sobald `fruit-apple` von `sugar` verdeckt wird, werden die Positionen von `fruit-orange` und `fruit-apple` vertauscht.

Der Roboter soll, ausgelöst durch seine langsame Bewegung, das langsame Verdecken der verschiedenen Objekte erkennen. Weiterhin soll er erkennen, dass verschiedene Objekte, die sich in einigen Attributen sehr ähnlich sind, ihre Positionen getauscht haben. Die Dauer eines Durchlaufs beträgt circa 4:00 Minuten.

5.1.3. Szenario 3

Im letzten Szenario befinden sich zu Beginn die Objekte `fruit-orange`, `fruit-apple`, `sugar` und `nesquik` in einer Reihe aufgestellt auf dem Tisch (siehe Abbildung 5.3). Der Roboter betrachtet den Tisch von der Längsseite und bewegt sich während des Experiments aus seiner Perspektive kontinuierlich nach links um den Tisch herum, bis er die Kopfseite des Tisches erreicht hat. Währenddessen verdecken sich die Objekte bedingt durch ihre Aufstellung gegenseitig, bis nur noch `nesquik` zu sehen ist. Von diesem Moment an wird abwechselnd *1.*) ein hinter `nesquik` stehendes Objekt vom Tisch entfernt und *2.*) der Roboter ein wenig zur Seite bewegt, damit einige der verdeckten Objekte wieder sichtbar werden. Anschließend bewegt er sich zur Stirnseite des Tisches zurück, sodass wieder nur `nesquik` sichtbar ist. Dieser Vorgang wird so lange wiederholt, bis nur noch `nesquik` auf dem Tisch zurückbleibt.

Der Roboter soll auch hier das langsame Verdecken der Objekte registrieren. Die Schwierigkeit besteht darin, zu erkennen, dass ein Objekt während der Bewegung des Roboters Stück für Stück verdeckt wird. Ab einem bestimmten Punkt wird es von der Perzeption nicht mehr erfasst, obwohl es noch zu einem Teil sichtbar ist (siehe Abbildung 2.3). Wird ein Objekt aus der Szene entfernt, während es von einem anderen Objekt verdeckt wird, muss dies vom Roboter erkannt werden, sobald er die letzte Bekannte Position des als verdeckt geglaubten Objekts einsehen kann. Die Dauer eines Durchlaufs beträgt circa 3:00 Minuten.

5.2. Ergebnisse

Tabelle 5.1.: Durchschnittliche Dauer eines Aufrufs des `btri`

	Szenario 1	Szenario 2	Szenario 3
Durchschnittliche Dauer in ms	601	931	807

In diesem Abschnitt werden die in den Experimenten gesammelten Daten präsentiert. Bei der Durchführung der Experimente wurden in jedem Szenario für jede Iteration und für jedes Objekt die Daten der Ground Truth, des Belief States des Roboters ohne den Einsatz des `objectReasoningAnnotators` (OR) und des Belief States des Roboters nach dem Einsatz

Tabelle 5.2.: Ergebnisse zur Erkennung von Störungen im Sichtfeld des Roboters

Szenario	GT	OR	Falsch negativ	Falsch positiv
Szenario 1	43	43	0	0
Szenario 2	22	25	0	3
Szenario 3	0	3	0	3

des OR, erfasst. Tabelle 5.1 zeigt für jedes Szenario die durchschnittliche Dauer, die für einen Aufruf des `btri` benötigt wurde. Da die Berechnungen zu Sichtbarkeiten und Kollisionen zeitaufwendig sind, dauert ein Aufruf länger, je mehr Objekte in der Welt sind. In den nun folgenden Tabellen sind vom OR stammende Ergebnisse farbig hinterlegt. Außerdem werden folgende Abkürzungen verwendet:

GT	Ground Truth	`n`	`nesquik`
RS	RoboSherlock	`b`	`bowl`
OR	`objectReasoningAnnotator`	`m`	`mondamin`
`a`	`fruit-apple`	`o`	`fruit-orange`
`s`	`sugar`		

5.2.1. Erkennen von Störungen im Sichtfeld des Roboters

Tabelle 5.2 zeigt, wie oft in jedem Szenario Störungen durch Hände und/oder Arme sowie „schwebende“ Objekte, die gerade bewegt werden, erkannt wurden. In der Spalte GT steht die Anzahl der tatsächlichen Störungen, in der Spalte *OR* steht die Anzahl der vom OR erkannten Störungen. Die Spalte *Falsch negativ* bezieht sich auf Störungen, die nicht erkannt wurden; die Spalte *Falsch positiv* bezieht sich auf falsch erkannte Störungen. Die Tabelle zeigt nur Daten vom OR, da RoboSherlock ohne den OR nicht in der Lage ist, Störungen im Bild zu erkennen.

5.2.2. Identifizieren von Objekten über mehrere Iterationen

Ziel des `ObjectIdentityResolution`-Annotators von RoboSherlock ist, in einer Szene erscheinende Objekte über mehrere Iterationen hinweg identifizieren zu können. Dazu wird jedes in einer Iteration erkannte Cluster einem Objekt zugeordnet. Erkennt die `ObjectIdentityResolution` in späteren Iterationen weitere Cluster, von denen sie annimmt, dass sie zu Objekten gehören, die sie bereits identifiziert hat, ordnet sie die Cluster den entsprechenden Objekten zu. Jedes Objekt erhält dabei eine eindeutige ID, sodass es über mehrere Iterationen hinweg identifizierbar ist. Die Tabellen 5.3 und 5.4 zeigen, in wie vielen Fällen diese Identifizierung erfolgreich war. Eine Identifizierung wird als erfolgreich gewertet, wenn ein zu einem Objekt gehörendes Cluster dem Objekt zugeordnet wird, dem es bei seiner ersten Erkennung auch zugeordnet wurde.

Tabelle 5.3 zeigt die Erkennungsrate pro Objekt für alle Zeitpunkte, an denen das jeweilige Objekt nicht verdeckt war. In den Spalten GT steht für jeweils jedes Szenario und jedes Objekt, wie oft es theoretisch hätte klassifiziert und identifiziert werden können (also aus der Perspektive des Roboter sichtbar war). Die Spalten RS % geben prozentual an, in wie vielen der Fälle ein Objekt von RoboSherlock ohne den OR identifiziert werden konnte. Analog dazu gibt die Spalte OR % die Erkennungsrate von RoboSherlock mit dem OR an. In der letzten Zeile stehen die pro Szenario für alle Objekte zusammengefassten Werte. Für Tabelle 5.4 gelten die gleichen Bedingungen, allerdings wurden hier pro Objekt nur Zeitpunkte einbezogen, zu denen es weder verdeckt war, noch eine Störung im Sichtfeld des Roboters existierte. Daher sind in den Spalten GT überwiegend weniger Zeitpunkte pro Objekt angegeben, an denen es hätte klassifiziert und identifiziert werden können.

5.2.3. Konsistenz bei der Objektklassifizierung

Tabelle 5.5 zeigt, in wie vielen Fällen ein Objekt korrekt klassifiziert wurde. Die Werte in der Spalte GT geben für jedes Szenario an, wie oft ein Objekt theoretisch hätte klassifiziert werden können (es also sichtbar für den Roboter war). Die mittlere Spalte (RS %) zu jedem Szenario gibt an, wie oft RoboSherlock ohne den OR ein Objekt korrekt klassifiziert hat, die rechte Spalte (OR %) gibt an, wie oft der OR ein Objekt richtig klassifiziert hat. Bei letzteren zwei Spalten handelt es sich um Prozentwerte, die sich auf die Angaben aus Spalte GT beziehen. In der letzten Zeile stehen die pro Szenario für alle Objekte zusammengefassten Werte.

Tabelle 5.3.: Ergebnisse zur Identifizierung von Objekten über mehrere Iterationen. Pro Objekt und Szenario wurden nur Zeitpunkte in die Statistik aufgenommen, an denen das jeweilige Objekt nicht verdeckt war.

	Szenario 1			Szenario 2			Szenario 3		
Objekt	GT	RS %	OR %	GT	RS %	OR %	GT	RS %	OR %
a	12	9	100	94	41	99	60	39	100
s	40	28	38	30	59	100	23	100	100
n	23	45	59	107	44	100	94	100	100
b	–	–	–	15	79	93	–	–	–
m	–	–	–	74	44	99	–	–	–
o	–	–	–	88	55	98	41	58	100
Σ	78	31	54	408	48	99	218	75	100

Tabelle 5.4.: Ergebnisse zur Identifizierung von Objekten über mehrere Iterationen. Pro Objekt und Szenario wurden nur Zeitpunkte in die Statistik aufgenommen, an denen das jeweilige Objekt nicht verdeckt war und im jeweiligen Szenario keine Störung im Sichtfeld des Roboters existierte.

	Szenario 1			Szenario 2			Szenario 3		
Objekt	GT	RS %	OR %	GT	RS %	OR %	GT	RS %	OR %
a	10	0	100	75	26	99	60	39	100
s	12	100	100	27	65	100	23	100	100
n	10	100	100	85	55	100	94	100	100
b	–	–	–	15	79	93	–	–	–
m	–	–	–	69	47	99	–	–	–
o	–	–	–	67	47	100	41	58	100
Σ	32	69	100	338	47	99	218	75	100

Tabelle 5.5.: Ergebnisse zur Konsistenz bei der Objektklassifizierung

	Szenario 1			Szenario 2			Szenario 3		
Objekt	GT	RS %	OR %	GT	RS %	OR %	GT	RS %	OR %
a	12	100	100	94	100	99	60	100	100
s	40	40	40	30	100	40	23	96	100
n	23	0	0	107	77	100	94	73	98
b	–	–	–	15	80	100	–	–	–
m	–	–	–	74	42	76	–	–	–
o	–	–	–	88	96	100	41	95	100
Σ	75	37	37	408	82	90	218	87	99

5.2.4. Korrigieren der Höhe und der Position von Objekten

In Tabelle 5.6 werden die Ergebnisse für die Höhenkorrektur der Objekte dargestellt. Für jedes Szenario sind für jedes Objekt die Abweichungen der z-Komponente der Objektposition und der Objekthöhe im Vergleich zur Ground Truth angegeben. Für jedes Szenario stehen in den linken beiden Spalten (Δz) jeweils die Abweichungen für die Objektposition für RoboSherlock ohne den OR (Spalte *RS*) und für RoboSherlock mit dem OR (Spalte *OR*). In den beiden rechten Spalten (Δh) stehen zu jeder Szene die Abweichungen für die Objekthöhe für Durchläufe von RoboSherlock ohne den OR (Spalte *RS*) und für RoboSherlock mit dem OR (Spalte *OR*). Die letzte Zeile enthält die pro Szenario für alle Objekte zusammengefassten Mittelwerte.

5.2.5. Erkennen von verdeckten Objekte

In Tabelle 5.7 ist abzulesen, wie oft das Verdecken eines Objekts im Szenario erkannt wurde. Auf einen Vergleich zu einer Durchführung von RoboSherlock ohne den OR wurde verzichtet, da RoboSherlock bisher nicht in der Lage war, verdeckte Objekte zu erkennen. In der linken Spalte (GT) zu jedem Szenario steht für jedes Objekt, in wie vielen Iterationen es tatsächlich verdeckt war. In der mittleren Spalte (OR) steht zu jedem Szenario, wie oft trotz der Verdeckung vom OR angenommen wurde, dass sich das Objekt in der Szene befindet. Die rechte Spalte (%) gibt für jedes Szenario den Prozentwert der erkannten Verdeckungen an. In der letzten Zeile stehen pro Szenario die für alle Objekte zusammengefassten Mittelwerte. Objekte, die bereits zu Beginn eines Szenarios verdeckt waren, wurden in dieser Statistik

Tabelle 5.6.: Abweichungen der z-Komponente der Positionen der Objekte und Abweichungen der Objekthöhen, verglichen mit der Ground Truth. Δz gibt die Abweichung zur z-Komponente der Objektposition im Vergleich zur Ground Truth in cm an; Δh gibt die Abweichung der Objekthöhe im Vergleich zur Ground Truth in Prozent an.

	Szenario 1				Szenario 2				Szenario 3			
	Δz in cm		Δh in %		Δz in cm		Δh in %		Δz in cm		Δh in %	
Obj.	RS	OR	RS	OR	RS	OR	RS	OR	RS	OR	RS	OR
a	2,15	0,31	26	27	2,00	0,94	24	16	2,77	0,24	24	40
s	2,41	0,21	7	16	2,20	0,20	7	14	2,80	0,43	7	17
n	2,69	0,24	1	17	2,23	0,64	2	9	2,68	0,30	2	17
b	–	–	–	–	1,56	0,32	22	14	–	–	–	–
m	–	–	–	–	2,00	0,49	6	12	–	–	–	–
o	–	–	–	–	1,74	0,31	31	18	2,93	0,42	30	33
∅	2,42	0,25	11	20	1,95	0,48	15	14	2,80	0,35	16	27

nicht berücksichtigt, da nicht davon auszugehen ist, dass diese Objekte vom Roboter erkannt werden.

Tabelle 5.7.: Ergebnisse zur Erkennung von verdeckten Objekten

	Szenario 1			Szenario 2			Szenario 3		
Objekt	GT	OR	%	GT	OR	%	GT	OR	%
a	60	59	98	9	0	0	25	25	100
s	19	17	89	0	0	–	7	7	100
n	0	0	–	0	0	–	0	0	–
b	–	–	–	22	22	100	0	0	–
m	–	–	–	0	0	–	0	0	–
o	–	–	–	17	17	100	12	12	100
Σ	79	76	96	48	39	81	44	44	100

5.3. Interpretation der Ergebnisse

Es folgt eine Interpretation der während der Experimente gesammelten Daten.

1. Störungen erkennen

   Störungen wurden in allen drei Szenarien korrekt vom OR erkannt. In den Szenarien 2 und 3 wurden jedoch jeweils drei Iterationen falsch als Störung klassifiziert. In allen sechs Fällen lag dies an im Vorfeld falsch erkannten Clustern von Objekten. Deren Bounding Boxes wiesen zu große Maße auf, sodass die entsprechenden Cluster vom OR als Störungen eingestuft wurden. Ein Vergleich mit RoboSherlock ohne die Verwendung des OR wird an dieser Stelle nicht gezogen, da RoboSherlock ohne den OR nicht in der Lage ist, Störungen im Sichtbereich des Roboters zu erkennen.

2. Objekte über mehrere Iterationen verfolgen

   In allen drei Szenarien kann der OR deutlich bessere Ergebnisse bei der Identifizierung von Objekten über mehrere Iterationen hinweg vorweisen als RoboSherlock ohne den OR. Besonders groß ist die Diskrepanz zwischen beiden Methoden in der zweiten Szene. Dies lässt sich dadurch erklären, dass RoboSherlock ohne den OR nicht in der Lage ist zu erkennen, dass ein Objekt verdeckt wird. Daher kann es passieren, dass ein Objekt als ein anderes Objekt identifiziert wird, welches sich zwar noch in der Szene befindet, aber von einem anderen Objekt verdeckt wird.

   Auffällig ist, dass sich die Werte für den OR im ersten Szenario in Tabelle 5.4 im Vergleich zu Tabelle 5.3 stark verbessert haben. Ursache dafür sind die vielen im ersten Szenario absichtlich durch „schwebende“ Objekte und menschliche Gliedmaßen erzeugten Störungen (siehe Abschnitt 5.1.1 auf Seite 52). Wurde ein Objekt (z. B. `sugar`) von einer Person über den Tisch ins Sichtfeld des Roboters gehalten, konnte der OR dies erfolgreich als Störung einordnen (sieh Tabelle 5.2). Erkennt der OR, dass eine Störung vorliegt, analysiert er die Szene nicht weiter, da die störenden Cluster falsche Ergebnisse hervorrufen würden. Daher werden Objekte, die eine Störung verursachen, vom OR nicht erfasst und somit auch nicht identifiziert. Ein nicht identifiziertes Objekt wird in die Statistik als fehlgeschlagene Identifizierung aufgenommen.

3. Konsistenz bei der Objektklassifizierung

   Bei der Klassifizierung von Objekten gab es im ersten Szenario zwischen RoboSherlock ohne den OR und RoboSherlock mit dem OR keine Unterschiede. Beide Methoden haben dort relativ schlecht abgeschnitten, da es viele Störungen im Bild gab, die das Klassifizieren erschwerten. Im dritten Szenario weist der OR mit 99 % korrekten Klassifizierungen bessere Ergebnisse als das klassische RoboSherlock mit 87 % auf. Im zweiten Szenario, in dem es viele Objekte, viele Verdeckungen und einige Manipulationen gab, konnte der OR insgesamt besser abschneiden, war aber dem klassischen RoboSherlock bei den Objekten `fruit-apple` und `sugar` unterlegen. Das sehr schlechte Abschneiden des OR beim Objekt `sugar` kam zustande, da `sugar` in dem Moment, als es auf den Tisch gestellt wurde, vom OR als das Objekt `bowl` identifiziert wurde, welches vorher bereits auf dem Tisch stand, aber wieder aus der Szene entfernt wurde.

4. Höhe und Position von Objekten korrigieren

   Besonders auffällig ist hier, dass sich durch den OR die Annäherung an die korrekte Objektposition in allen Szenarien signifikant um ein Vielfaches verbessert hat. Gleichzeitig hat sich aber die Annäherung an die korrekte Objekthöhe in den Szenarien 1 und 3 erheblich verschlechtert. Dies deutet auf einen Fehler in der Implementierung hin, da beide Größen direkt abhängig voneinander sind.

5. Verdeckte Objekte erkennen

   Im dritten Szenario konnte der OR alle Verdeckungen von Objekten erfolgreich erkennen, im ersten Szenario 96 %. Im zweiten Szenario wurden hingegen 81 % der Verdeckungen erkannt. Grund für diesen im Vergleich niedrigen Wert ist, dass in dem Moment, als das Objekt `fruit-orange` von `sugar` verdeckt wurde, die Positionen von `fruit-apple` und `fruit-orange` vertauscht wurden. Somit wurde das Objekt `fruit-apple` in den Schatten von `sugar` gelegt. Das Erkennen einer solchen Verdeckung ist allerdings nicht mit herkömmlichen Perzeptions- und Reasoningmethoden zu bewerkstelligen.

   Auch zu diesem Punkt fällt ein Vergleich mit dem klassischen RoboSherlock weg, da das Erkennen von verdeckten Objekten dort nicht vorgesehen ist.

6. Auswertung und Ausblick

In diesem Kapitel wird nach einer Zusammenfassung die eingangs gestellte Forschungsfrage beantwortet, um dann abschließend mit einem Ausblick auf mögliche Verbesserungen der entwickelten Software abzuschließen.

6.1. Zusammenfassung

In der Einleitung wurde eine Übersicht über die von einem autonomen Haushaltsroboter zu bewältigenden Aufgaben und die dabei vorhandenen Schwierigkeiten sowie die entstehenden Probleme gegeben. Anschließend wurden verschiedene Perzeptions- und Reasoning-Systeme vorgestellt, die Ansätze zur Bewältigung dieser Probleme darstellen. Weiterhin wurden Frameworks präsentiert, die beide Komponenten miteinander verbinden, um bessere Ergebnisse zu erzielen. Es folgte die Formulierung einiger Ideen zur Verwendung von physikbasierter Simulation und Reasoning-Mechanismen, um die von einem bestehenden Perzeptionssystem gelieferten Resultate auf Konsistenz zu prüfen und gegebenenfalls zu verbessern. Im daran anschließenden Teil wurden die verschiedenen Programmbibliotheken und Systeme, auf denen diese Arbeit aufbaut, umrissen. Darauf folgte die Beschreibung der Implementierung und der im Rahmen der Experimente zum Testen der Implementierung verwendeten Szenarien. Im letzten Kapitel wurden die Ergebnisse der Experimente präsentiert und bewertet. Dabei wurde das klassische RoboSherlock mit RoboSherlock unter Verwendung des `objectReasoningAnnotators` verglichen.

6.2. Beantwortung der Forschungsfrage

Die eingangs formulierte Forschungsfrage lautet:

> *„Können die Ergebnisse eines klassischen Perzeptionssystems durch das Anwenden von physikbasierter Simulation in Kombination mit Reasoning hinsichtlich ihrer Konsistenz verbessert werden?“*

Die Auswertung der aus den Experimenten gewonnenen Ergebnisse und der dabei gezogene Vergleich zwischen dem klassischen RoboSherlock und RoboSherlock unter Verwendung des `objectReasoningAnnotators` (OR) haben gezeigt, dass die Konsistenz der Resultate der Objektklassifizierung und der Identifizierung von Objekten durch den OR signifikant gesteigert werden konnte. Durch das erfolgreiche Erkennen von verdeckten Objekten und Störungen im Sichtfeld des Roboters bietet der OR dem Perzeptionssystem RoboSherlock einen erheblichen Mehrwert. Die präziseren Daten ermöglichen einem Roboter, seine Aufgaben mit einem höheren Grad an Zuverlässigkeit zu bewältigen und tragen einen wesentlichen Teil dazu bei, autonome Robotersysteme in dynamischen Umgebungen stabiler, verlässlicher und genauer agieren zu lassen.

Somit wurde gezeigt, dass das Perzeptionssystem RoboSherlock durch das Hinzufügen einer Reasoning-Komponente, die Konsistenzprüfungen und Fehlerkorrekturen auf Basis von physikbasierter Simulation und logischer Schlussfolgerung durchführt, in mehreren Aspekten verbessert werden konnte.

6.3. Ausblick

Diese Arbeit hat sich in einem sehr eingeschränkten Rahmen unter Laborbedingungen mit dem Kombinieren von Perzeption und Reasoning beschäftigt. Daher wäre es interessant, in einer weiterführenden Arbeit zu untersuchen, wie sich der `objectReasoningAnnotator` bei einer Skalierung der Szenarien verhält. Durch Skalierung kann eine höhere Komplexität der Szenen erreicht werden, um der realen Umgebung, in der ein Haushaltsroboter eingesetzt wird, näher zu kommen. Dafür könnten Objekte mit komplexeren geometrischen Formen verwendet, die Anzahl der verschiedenen Objekttypen erhöht, eine höhere Anzahl an Objekten in der Szene verwendet und die Anzahl an menschlichen, die Szene verändernden, Einflüssen erhöht werden.

Weiterhin kann untersucht werden, in welchen Situationen der `objectReasoningAnnotator` schlechtere Ergebnisse als das klassische RoboSherlock erzeugt. Durch gezieltes Testen könnten die Probleme eingegrenzt und deren Ursachen gefunden werden. Die dabei gewonnenen Informationen könnten anschließend dazu genutzt werden, die im Rahmen dieser Arbeit entstandenen Reasoning-Methoden zu verbessern.

Eine andere Möglichkeit der Weiterführung dieser Arbeit besteht darin, den Reasoning betreibenden Annotator nicht, wie in dieser Arbeit geschehen, am Ende der Perzeptionspipeline anzuordnen, sondern ein iteratives Verfahren zu entwickeln. In einem solchen Verfahren könnte das vom Reasoning inferierte Wissen von den Perzeptionsalgorithmen verwendet werden, um ihre Ergebnisse aus der vorherigen Iteration zu verbessern. Diese verbesserten Ergebnissen würden nun wieder an die Reasoning-Komponente weitergegeben werden. Das könnte so lange wiederholt werden, bis ein vorher festgelegtes Maß an Kontinuität in den Ergebnissen erreicht wurde.

A. Quellcode

Listing A.1: Definition der Message `bullet_reasoning_interface/ObjectIntel`

```
# Object types
uint32 BOWL = 0
uint32 NESQUIK = 1
uint32 MONDAMIN = 2
uint32 FRUIT_APPLE = 3
uint32 FRUIT_ORANGE = 4
uint32 SUGAR = 5


# Object's id
uint32 id

# Indicates if the object is stable.
bool isStable

# Indicates if the object is visible.
bool isVisibleBeforeSimulation
bool isVisibleAfterSimulation

# Indicates if the object had contact with kitchen before simulation.
bool contactWithKitchenBeforeSimulation

# Indicates if the object had contact with kitchen after simulation.
bool contactWithKitchenAfterSimulation

# Has collision with this objects.
uint32[] collisionWithBeforeSimulation
uint32[] collisionWithAfterSimulation

uint32[] occludedByBeforeSimulation
uint32[] occludedByAfterSimulation

# Object's stamped pose.
geometry_msgs/PoseStamped poseStamped

# Object's bounding box in world's fixed frame.
geometry_msgs/Vector3 boundingBox

# Which operation had to be executed?
uint32 operation

# Did the operation succeed?
bool operationSucceeded
```

B. Inhalt des Datenträgers

`Thesis.pdf`	Dieses Dokument.
`iai_robosherlock`	Eine Kopie des RoboSherlock-Projekts. Der im Rahmen dieser Arbeit entwickelte `objectReasoningAnnotator` befindet sich im Unterverzeichnis `object_reasoning`.
`cram_physics`	Eine Kopie des CRAM-Pakets `cram_physics`. Das im Rahmen dieser Arbeit entwickelte `bullet_reasoning_interface` befindet sich im Unterverzeichnis `bullet_reasoning_interface`.
`cram_projections_demo`	Eine Kopie des CRAM-Pakets `cram_projections_demo`.
`Experimente`	Ordner, der zu jedem Szenario eine Videoaufnahme eines Testlaufs enthält.

C. Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig angefertigt, nicht anderweitig zu Prüfungszwecken vorgelegt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe. Sämtliche wissentlich verwendete Textausschnitte, Zitate oder Inhalte anderer Verfasser wurden ausdrücklich als solche gekennzeichnet.

Bremen, den 16.07.2015

Andreas Romero Früh

Literaturverzeichnis

[Bee+14] Michael Beetz, Ferenc Balint-Benczedi, Nico Blodow et al. *RoboSherlock: Unstructured Information Processing for Robot Perception*. Technical Report 75. Am Fallturm 1, 28359 Bremen, Deutschland: Universität Bremen, 2014.

[BHT13] Peter W. Battaglia1, Jessica B. Hamrick und Joshua B. Tenenbaum. „Simulation as an engine of physical scene understanding“. In: *Proceedings of the National Academy of Sciences of the United States* 110.45 (5. Nov. 2013). Hrsg. von National Academy of Sciences, S. 18327–18332. ISSN: 0027-8424. DOI: 10.1073/pnas.1306572110. URL: http://www.pnas.org/content/110/45/18327 (besucht am 12. 07. 2015).

[BMT10] Michael Beetz, Lorenz Mösenlechner und Moritz Tenorth. „CRAM – A Cognitive Robot Abstract Machine for Everyday Manipulation in Human Environments“. In: *The IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems*. (IROS 2010). (Taipei International Convention Center, Taipei, Taiwan, 18.–22. Okt. 2010). Hrsg. von Institute of Electrical und Inc. Electronics Engineers. Piscataway, NJ, USA, 2010, S. 1012–1017. ISBN: 978-1-4244-6675-7. DOI: 10.1109/IROS.2010.5650146.

[Boe10] Marc Boeker. *MongoDB. Sag Ja zu NoSQL*. Frankfurt am Main: Software & Support Media GmbH, 2010. ISBN: 978-3-86802-244-5.

[BRA12] Alexandru Boicea, Florin Radulescu und Laura Ioana Agapin. „MongoDB vs Oracle – Database Comparison“. In: *Third International Conference on Emerging Intelligent Data and Web Technologies*. EIDWT 2012. (Bucharest, Rumänien, 19.–21. Sep. 2012). Hrsg. von The Institute of Electrical und Inc. Electronics Engineers. Piscataway, NJ, USA, 2012, S. 330–335. ISBN: 978-0-7695-4734-3. DOI: 10.1109/EIDWT.2012.32.

[CC12] Changhyun Choi und Henrik I. Christensen. „3D Pose Estimation of Daily Objects Using an RGB-D Camera“. In: *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. (IROS 2012). (Hotel Tivoli Marina Vilamoura, Vilamoura, Algarve, Portugal, 7.–12. Okt. 2012). Hrsg. von Institute of Electrical und Inc. Electronics Engineers. Piscataway, NJ, USA, 2012, S. 3342–3349. ISBN: 978-1-4673-1737-5. DOI: 10.1109/IROS.2012.6386067.

[CMS11] Alvaro Collet, Manuel Martinez und Siddhartha S. Srinivasa. „The MOPED framework: Object Recognition and Pose Estimation for Manipulation“. In: *The International Journal of Robotics Research* 30.10 (Sep. 2011), S. 1284–1306. ISSN: 0278-3645, 1741-3176. DOI: 10.1177/0278364911401765. URL: http://ijr.sagepub.com/content/30/10/1284.short (besucht am 01. 07. 2015).

[FL04] David Ferrucci und Adam Lally. „UIMA: an architectural approach to unstructured information processing in the corporate research environment“. In: *Natural Language Engineering* 10 (3–4 Sep. 2004), S. 327–384. ISSN: 1351-3249, 1469-8110. DOI: 10.1017/S1351324904003523. URL: http://journals.cambridge.org/abstract_S1351324904003523 (besucht am 06. 07. 2015).

[GS09] Thilo Götz und Oliver Suhre. „Design and implementation of the UIMA Common Analysis System“. In: *IBM Systems Journal* 43.3 (2009), S. 476–489. ISSN: 0018-8670. DOI: 10.1147/sj.433.0476. URL: http://domino.research.ibm.com/tchjr/journalindex.nsf/a3807c5b4823c53f85256561006324be/221395678866f87d85256eed00780cd1!OpenDocument (besucht am 06. 07. 2015).

[Hum96] Robert Hummel. „Uncertainty reasoning in object recognition by image processing“. In: *Lecture Notes in Computer Science*. Bd. 1093: *Reasoning with Uncertainty in Robotics*. International Workshop, RUR '95 Amsterdam, The Netherlands December 4 – 6, 1995 Proceedings. Berlin, Deutschland, Nov. 1996, S. 131–145. ISBN: 978-3-540-61376-3, 978-3-540-68506-7. DOI: 10.1007/BFb0013956. URL: http://link.springer.com/book/10.1007/BFb0013951 (besucht am 12. 07. 2015).

[Kla+09] Ulrich Klank, Dejan Pangercic, Radu Bogdan Rusu et al. „Real-time CAD Model Matching for Mobile Manipulation and Grasping“. In: *2009 9th IEEE-RAS International Conference on Humanoid Robots*. (Humanoids 2009). (Pierre et Marie Curie University, Paris, France, 7.–10. Dez. 2009). Piscataway, NJ, USA, 2009, S. 290–296. ISBN: 978-1-4244-4597-4. DOI: 10.1109/ICHR.2009.5379561.

[Kun+] Lars Kunze, Chris Burbridge, Marina Alberti et al. „Combining Top-down Spatial Reasoning and Bottom-up Object Class Recognition for Scene Understanding“. In: S. 2910–2915. DOI: 10.1109/IROS.2014.6942963.

[KZM09] Ulrich Klank, Muhammad Zeeshan Zia und Beetz Michael. „3D Model Selection from an Internet Database for Robotic Vision“. In: *Open-Source Software workshop of the International Conference on Robotics and Automation*. ICRA 2009 Workshop. (Kobe International Conference Center, Kobe, Japan, 12.–17. Mai 2009). Hrsg. von Institute of Electrical und Inc. Electronics Engineers. 2009, S. 2406–2411. DOI: 10.1109/ROBOT.2009.5152488.

[Mör+10] Thomas Mörwald, Johann Prankl, Andreas Richtsfeld et al. „BLORT - The Blocks World Robotic Vision Toolbox“. In: *Best Practice in 3D Perception and Modeling for Mobile Manipulation*. ICRA 2010 Workshop. (Dena'ina Center, Anchorage, Alaska, USA, 3.–7. Mai 2010). Hrsg. von Institute of Electrical und Inc. Electronics Engineers. 2010.

[NBB14] Daniel Nyga, Ferenc Balint-Benczedi und Michael Beetz. „PR2 Looking at Things: Ensemble Learning for Unstructured Information Processing with Markov Logic Networks“. In: *2014 IEEE International Conference on Robotics and Automation*. (ICRA 2014). (Hong Kong Convention and Exhibition Centre, Hong Kong, China, 31. Mai–7. Juni 2014). Hrsg. von Institute of Electrical und Inc. Electronics Engineers. Piscataway, NJ, USA, 2014, S. 3916–3923. ISBN: 978-1-4799-3686-1. DOI: 10.1109/ICRA.2014.6907427.

[oV09] o. V. *UIMA Tutorial and Developers' Guides. Written and maintained by the Apache UIMA Development Community*. Hrsg. von International Business Machines Corporation & The Apache Software Foundation. 2009.

[Pan+10] Dejan Pangercic, Moritz Tenorth, Dominik Jain et al. „Combining Perception and Knowledge Processing for Everyday Manipulation“. In: *The IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems*. (IROS 2010). (Taipei International Convention Center, Taipei, Taiwan, 18.–22. Okt. 2010). Hrsg. von Institute of Electrical und Inc. Electronics Engineers. Piscataway, NJ, USA, 2010, S. 1065–1071. ISBN: 978-1-4244-6675-7. DOI: 10.1109/IROS.2010.5651006.

[Qui+09] Morgan Quigley, Brian Gerkey, Ken Conley et al. „ROS: an open-source Robot Operating System“. In: *Open-Source Software workshop of the International Conference on Robotics and Automation*. ICRA 2009 Workshop. (Kobe Interna-

tional Conference Center, Kobe, Japan, 12.–17. Mai 2009). Hrsg. von Institute of Electrical und Inc. Electronics Engineers. 2009.

[RBC] Karinne Ramirez-Amaro, Michael Beetz und Gordon Cheng. „Automatic Segmentation and Recognition of Human Activities from Observation based on Semantic Reasoning“. In: S. 5043–5048. DOI: 10.1109/IROS.2014.6943279.

[RD06] Matthew Richardson und Pedro Domingos. „Markov logic networks“. In: *Machine Learning* 62 (1–2 Feb. 2006), S. 107–136. ISSN: 0885-6125, 1573-0565. DOI: 10.1007/s10994-006-5833-1. URL: http://link.springer.com/article/10.1007/s10994-006-5833-1 (besucht am 05. 07. 2015).

[SBF13] Yuyin Sun, Liefeng Bo und Dieter Fox. „Attribute based object identification“. In: *2013 IEEE International Conference on Robotics and Automation*. (ICRA 2013). (Kongresszentrum Karlsruhe, Karlsruhe, Deutschland, 6.–10. Mai 2013). Hrsg. von Institute of Electrical und Inc. Electronics Engineers. Piscataway, NJ, USA, 2013, S. 2096–2103. ISBN: 978-1-4673-5640-4. DOI: 10.1109/ICRA.2013.6630858.

[TB13] Moritz Tenorth und Michael Beetz. „KnowRob: A knowledge processing infrastructure for cognition-enabled robots“. In: *The International Journal of Robotics Research* 32.5 (Apr. 2013), S. 566–590. ISSN: 0278-3645, 1741-3176. DOI: 10.1177/0278364913481635. URL: http://ijr.sagepub.com/content/32/5/566.short (besucht am 12. 07. 2015).

[Ten+] Moritz Tenorth, Stefan Profanter, Ferenc Balint-Benczedi et al. „Decomposing CAD Models of Objects of Daily Use and Reasoning about their Functional Parts“. In: S. 5943–5949. DOI: 10.1109/IROS.2013.6697218.

[TJB10] Moritz Tenorth, Dominik Jain und Michael Beetz. „Knowledge Processing for Cognitive Robots“. In: *KI - Künstliche Intelligenz* 24 (3 Sep. 2010), S. 233–240. ISSN: 0933-1875, 1610-1987. DOI: 10.1007/s13218-010-0044-0. URL: http://link.springer.com/article/10.1007/s13218-010-0044-0 (besucht am 12. 07. 2015).

[TNB10] Moritz Tenorth, Daniel Nyga und Michael Beetz. „Understanding and Executing Instructions for Everyday Manipulation Tasks from the World Wide Web“. In: *2010 IEEE International Conference on Robotics and Automation*. (ICRA 2010). (Egan Center & Dena'ina Center, Anchorage, Alaska, USA, 3.–8. Mai 2010). Hrsg. von Institute of Electrical und Inc. Electronics Engineers. Piscataway,

NJ, USA, 2010, S. 1050–4729. ISBN: 978-1-4244-5038-1, 978-1-4244-5040-4. DOI: 10.1109/ROBOT.2010.5509955.

Abkürzungen

Es folgt eine Auflistung der in diesem Dokument verwendeten Abkürzungen. Die Zahlen hinter einer ausgeschriebenen Form beziehen sich auf Seiten im Dokument, auf denen die entsprechende Abkürzung verwendet wird.

btri `bullet_reasoning_interface` 41, 42, 44, 46, 48, 49, 54, 55

AE Analysis Engine 24, 25, 31, 32, 39

BSON Binary JSON 26

CAD Computer-aided design 4, 7

CAS Common Analysis Structure 25, 31–35, 38–40, 46

CPL CRAM Plan Language 18

CRAM Cognitive Robot Abstract Machine 2, 15, 17–20

FO Pädikatenlogik erster Stufe 26, 28

IAI AG Künstliche Intelligenz 2, 13, 20, 51

ICE Iterative Clustering-Estimation 5

JSON JavaScript Object Notation 26, 79

MLN Markov Logic Network 15, 26, 28, 39

MPI Message Passing Interface 16

OR `objectReasoningAnnotator` 31, 45–47, 49, 52, 54–56, 58, 61–65

OWL Web Ontology Language 19
Siehe auch Glossar auf Seite 84.

PR2 Personal Robot 2 51

RDBMS Relationales Datenbankmanagementsystem 26

RGB Rot-Grün-Blau 32, 33, 50, 51, 83
Siehe auch Glossar auf Seite 83.

ROS Robot Operating System 15–17, 20, 31, 42

SofA Subject of Analysis 24, 25

STL Stereolithografie 21
Siehe auch Glossar auf Seite 83.

UIMA Unstructured Information Management Architechture 24, 25

URDF Unified Robot Description Format 20
Siehe auch Glossar auf Seite 83.

Glossar

Es folgt eine Auflistung der in diesem Dokument verwendeten Fachbegriffe. Die Zahlen hinter einer Beschreibung beziehen sich auf Seiten im Dokument, auf denen der erklärte Begriff verwendet wird.

B

Belief State

Der *Belief State* ist der vom Roboter angenommene Zustand der Welt, in der er sich befindet. Vgl. *Ground Truth* im Glossar auf der nächsten Seite. . . 10–12, 18, 50, 54

Bounding Box

Eine *Bounding Box* ist ein Quader, der ein anderes (oft komplexeres) Objekt vollständig umschließt. Eine minimale Bounding Box ist der kleinstmögliche umschließende Quader. Bounding Boxes werden häufig zur Berechnung von Kollisionen verwendet, da der Rechenaufwand um ein Vielfaches kleiner ist als bei der Verwendung von komplexen geometrischen Formenen. 35, 37, 44, 46, 50, 61

C

Cluster

Ein *Cluster* (englisch für *Haufen, Gruppe*) ist eine Reihe von Punkten aus einer Punktwolke, die in einem räumlichen Kontext zueinander stehen (z. B. indem sie zu einem Objekt gehören). 4, 11, 12, 34, 35, 37–41, 46, 48, 50, 56, 61

F

Feature

Ein *Feature* (englisch für *Merkmal, Besonderheit, Eigenschaft*) ist in der digitalen Bildverarbeitung ein bestimmtes, aus einem Bild extrahiertes Konstrukt. Es wird durch einen Ortsvektor und seine Attribute beschreibende Werte repräsentiert. Features können z. B. Kanten, Endpunkte von Linien, Ecken, Kreise und Ellipsen oder Verbindungspunkte mehrerer Flächen sein. [Hum96] 4, 5

G

Ground Truth

Die *Ground Truth* bezeichnet den tatsächlichen Zustand der Welt, in der sich der Roboter befindet. Vgl. *Belief State* im Glossar auf der vorherigen Seite. . . . 54, 55, 58, 59

P

Perzeptionspipeline

Bei einer *Perzeptionspipeline* handelt es sich um eine Reihe von Perzeptionsalgorithmen, die hintereinander angewandt werden. Häufig basieren dabei die Berechnungen eines Algorithmus auf den Ergebnissen eines oder mehrerer vorausgehender Algorithmen. 2, 3, 9, 31, 46, 65

Pose

Eine *Pose* beschreibt die Lage eines Objektes im dreidimensionalen Raum. Sie besteht aus einem Vektor, der die Position angibt, und einem Quaternion, welches die Rotation angibt. 5, 22, 23, 25, 35, 42–45, 48

Punktwolke

Eine *Punktwolke* ist analog zu einer 2D-Rastergrafik die Beschreibung eines dreidimensionalen Bildes. Die Koordinaten von Punkten in Punktwolken werden in einem dreidimensionalen Koordinatensystem angegeben. Dies ermöglicht die Erfassung und Darstellung von räumlichen Ausschnitten. 4, 6, 11, 33–36, 81

R

RGB-D-Kamera

Eine *RGB-D-Kamera* ist eine Kamera, die den RGB-Farbraum verwendet und zusätzlich Tiefeninformationen wahrnimmt. Dies ermöglicht die dreidimensionale Darstellung des aufgenommenen Bildes. 24

Rot-Grün-Blau (RGB)

Ein *Rot-Grün-Blau-Farbraum* ist ein additiver Farbraum, in dem Farben durch das Mischen der drei Grundfarben Rot, Grün und Blau beschrieben werden. 1, 32, 33, 50, 51, 83

S

Semantische Karte

Eine *semantische Karte* beschreibt aus beliebig vielen Objekten bestehende Umgebungen. Jedes Objekt besitzt einen eindeutigen Namen, einen Typ (welcher eine semantische Bedeutung besitzt), Abmessungen (Höhe, Tiefe, Breite) und eine Transformation vom globalen Ursprung der Welt zum Ursprung des Objekts. 8, 32, 33, 35

Stereolithografie (STL)

STL (Stereolithografie) ist ein Dateiformat zur Darstellung von dreidimensionalen Objekten. 21

U

Unified Robot Description Format (URDF)

Das *Unified Robot Description Format*[1] (englisch für *Vereinheitlichtes Format zur Beschreibung von Robotern*) ist ein XML-Format zur Beschreibung von Robotermodellen. 20

[1] ↑ `http://wiki.ros.org/urdf`

W

Web Ontology Language (OWL)

Web Ontology Language ist eine Spezifikation des *World Wide Web Consortiums (W3C)*, um Ontologien anhand einer formalen Beschreibungssprache erstellen, publizieren und verteilen zu können. 19