Karlsruher Institut für Technologie

Differenzbasierte Repräsentation räumlicher Relationen zur probabilistischen Szenenerkennung mittels hierarchischen Constellation Models

Bachelorarbeit
von

Joshua Enrico Link

An der Fakultät für Informatik
Institut für Anthropomatik und Robotik
Lehrstuhl Prof. Dr.-Ing. R. Dillmann

Erstgutachter:	Prof. Dr.-Ing. R. Dillmann
Zweitgutachter:	???
Betreuender Mitarbeiter:	Dipl.-Inform. Pascal Meißner

Bearbeitungszeit: 11. Juni 2017 – 10. September 2017

Hiermit erkläre ich an Eides statt, dass ich die von mir vorgelegte Arbeit selbstständig verfasst habe, dass ich die verwendeten Quellen, Internet-Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen – die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Karlsruhe, den **(Datum)** **ToDo**

Joshua Enrico Link

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einführung

In der Robotik ist die Servicerobotik wohl der Forschungsbereich, welcher den größten Alltagsbezug für den Menschen hat, da er sich mit der Entwicklung und Weiterentwicklung von autonomen Robotern beschäftigt, welche dem Menschen im Alltag assistieren. Man findet mittlerweile Roboter im Privaten, die das Putzen, Staubsaugen oder Rasenmähen übernehmen, in der Industrie, bei Montage und Fertigung, sowie auch in der Medizin, als Pflegehilfe, Botengänger oder Assistent.

Allerdings müssen die Roboter ihre Umwelt für komplexere Aufgaben so präzise wie möglich wahrnehmen und verstehen. Sie könne Aufgaben übernehmen bei denen sie gezielt Objekte umfahren, suchen und auch aufnehmen und benutzen. Dieser Funktionsumfang kann mit dem Prizip Programmieren durch Vormachen (PdV) ermöglicht werden, bei dem die Roboter Objekte und Tätigkeiten ihrer Umgebung kennen lernen, wieder erkennen und nachahmen können. So lässt sich die hohe Komplexität umgehen, die die manuelle Programmierung vieler Aufgaben mit sich bringen würde.

Um tatsächlich selbstständige Serviceroboter zu schaffen muss man aber noch zu einer Objekterkennung ein Kontextverständnis hinzufügen. Die Roboter müssen erkannte Objekte in einen Zusammenhang bringen, um die dadurch resultierenden Aufgaben zu verstehen. Zum Beispiel hat ein Teelöffel, welcher neben einer Tasse Tee liegt eine andere Aufgabe zu verrichten, als wenn er neben einem Becher Joghurt platziert ist. Nur am Kontext lässt sich dort entscheiden warum im einen Fall umgerührt und im anderen gelöffelt wird. Ebenso wäre ein Stück Butter verschieden zu verwenden, wenn es auf einem Frühstückstisch steht als wenn es mit anderen Zutaten neben einer Rührschüssel vorkommt.

Somit braucht man eine Szenenerkennung, welche zuverlässig die Objekte erkennen und ihren jeweiligen Kontext verstehen und einschätzen kann. Diese Erkennung ist nicht immer eindeutig, da der eben genannte Löffel ebenso zwischen einem Becher Joghurt und einer Tasse Tee liegen könnte, deshalb bietet es sich an mit Wahrscheinlichkeitsabschätzungen des vorliegenden Kontexts zu arbeiten.

2. Motivation und Problemstellung

Das Kapitel geht in Motivation auf die Relevanz des Themas und der vorliegenden Arbeit ein und in Fokus der Arbeit auf die Problemstellung die bearbeitet wird, sowie diverse Einschränkungen und Annahmen die für die Arbeit festgelegt sind.

2.1 Motivation

Wie schon in der Einführung erwähnt ist es elementar wichtig, dass man eine zuverlässige Szenenerkennung und ein gutes Kontextverständnis schafft um den Robotern die Möglichkeit zu bieten, sinnvoll mit ihrer Umwelt zu interagieren. Mit einer präzisen Wahrscheinlichkeitseinschätzung wie die momentane Umgebung beschaffen ist, lässt sich abschätzen welche Aufgaben es zu bewältigen und welche Probleme zu lösen gilt. Um dies zu gewährleisten muss man in das System möglichst viele Referenzdaten einspeisen, damit es jeden vorhanden Kontext erkennen kann. Szenen werden zu diesem Zweck aufgebaut un der Erkennung als neue Szene vorgestellt. Jede Szene die die Szenenerkennung auf diese Weise lernt, hilft das Chaos der sie umgebenen Objekte mehr und mehr zu interpretieren und einzuordnen.

Die Szenenerkennung nutzt also die Daten die sie zur Verfügung gestellt bekommt, bereits gelernte Szenen wiederzuerkennen. In dem bereits vorhandenen PSM(Probabilistic Scene Model)-System werden die Daten pro Szene zu einem Modell zusammengefasst, bei dem Auffällige Zusammenhänge berücksichtigt werden und scheinbar nicht miteinander in Verbindung stehende Objekte voneinander gelöst betrachtet werden. Zum Beispiel findet man eine Computermaus signifikant häufig vor dem Computerbildschirm und nie dahinter, allerdings kann dabei das räumliche Verhältnis der Maus zur Tastatur stark variieren. Die Maus wäre in diesem Beispiel manchmal direkt neben der Tastatur, manchmal dichter beim Bildschirm eben so oft weiter entfernt. In diesem Fall würde möglicherweise die Relation zwischen Maus und Tastatur wegfallen und nur jeweils das räumliche Verhältnis zum Bildschirm betrachtet werden. Dadurch gibt es Vorteile in der

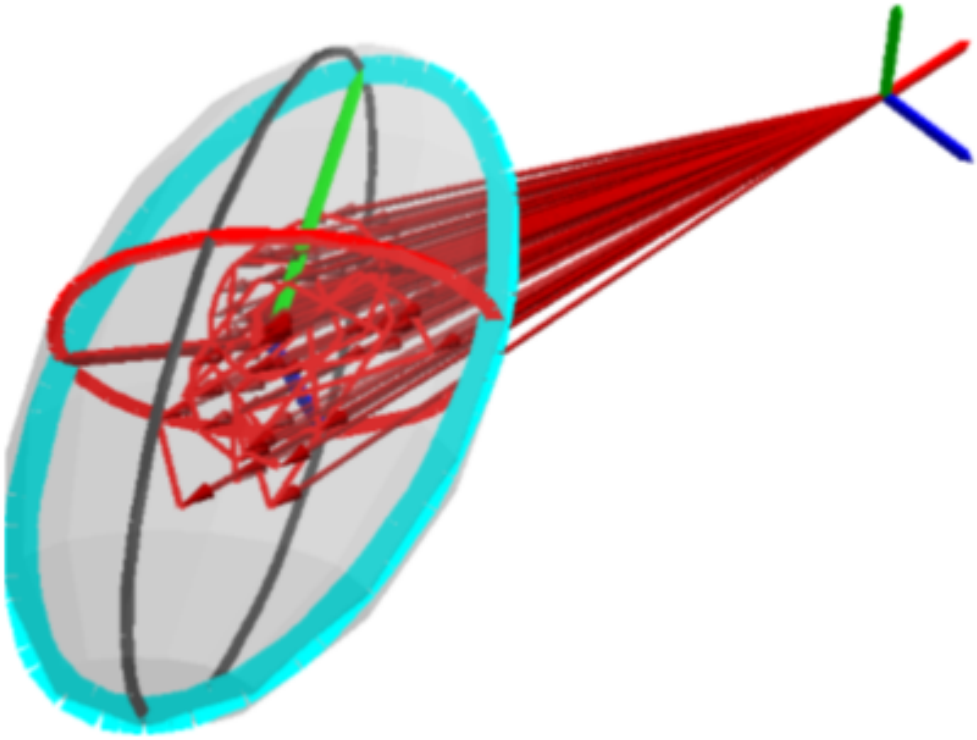

Abbildung 2.1: Beispiel: Relative Position eines Objekts zu einem anderen

Laufzeit und möglicherweise auch eine signifikantere Erkennung, allerdings findet natürlich auch ein Informationsverlust statt, der zu Fehlern führen kann. In dieser Arbeit wird ein Ansatz getestet, der dichter an den erhaltenen Daten arbeitet.

In Abbildung 2.1 sieht man wie das Modell eine Relation zwischen zwei Objekten aufgrund der erhaltenen Daten erstellt. Die roten Pfeile beschreiben hier die jeweiligen Relationen, die aus den Daten berechnet wurden und die Ellipsen zeigen, welchen Bereich das Modell als Basis für die Wahrscheinlichkeitsabschätzung benutzt. [Geh14]

2.2 Fokus der Arbeit

Ziel der vorliegenden Arbeit ist es das PSM-System zu überarbeiten und einen neuen Modus der Szenenerkennung im PSM-System zu entwickeln. Dieser Modus soll alternativ zu den bereits vorhandenen Modi auswählbar sein und das System erweitern. Außerdem soll sowohl die Positionierung als auch die Rotation der erkannten Objekten mit den bekannten Szenen verglichen werden um eine Wahrscheinlichkeitsabschätzung auszugeben, welche die Wahrscheinlichkeit aller möglichen Szenen ausgibt sowie auch die Wahrscheinlichkeit, dass es sich um keine der Szenen handelt.

Um eine interaktive Szenenerkennung zu schaffen ist es sinnvoll das System datengetrieben zu programmieren. Datengetriebene Entwicklung zeichnet sich dadurch aus, dass sich der Programmfluss ändert aufgrund der Daten die das System während der Laufzeit als Eingabe erhält. Das PSM-System wurde bereits datengetrieben programmiert und in der zu dieser Arbeit gehörigen Implementierung wurde auch am datengetriebenen Entwicklungsmodell festgehalten.

Da die Erkennung innerhalb des PSM-Systems nutzbar sein soll sind Eingabe- und Ausgabeschnittstellen sowie die Visualisierung vorgeschrieben und sollen für gute Vergleichbarkeit denen des vorhandenen Systems entsprechen.

Die Arbeit ist wie folgend strukturiert. In 3 Grundlagen wird auf die Grundkenntnisse eingegangen die man braucht um den Rest der Arbeit gut zu verstehen. Es werden in diesem Kapitel die für die Arbeit wichtigen Komponenten des bestehenden PSM-System erklärt. In 4 Konzept wird das theoretische Konzept thematisiert, welches zum Algorithmus geführt hat, dass den neuen Modus vom alten System unterscheidet. Außerdem wird der Algorithmus selbst erläutert. In 5 Implementierung wird die Software beschrieben und dokumentiert die im Zuge dieser Arbeit entstanden ist sowie die Änderungen die am bestehenden PSM System vorgenommen wurden. 6 Evaluation beschreibt die durchgeführten Experimente und interpretiert sie. In 7 Zusammenfassung und Ausblick wird die Arbeit noch einmal zusammen gefasst und ein Ausblick darauf gegeben in welche Richtung man das bestehende System weiter entwickeln und auf welche Weise man möglicherweise die Szenenerkennung noch weiter verbessern kann. Am Schluss stehen die Quellen welche zur Recherche für diese Arbeit genutzt wurden.

3. Grundlagen

In diesem Kapitel wird das bestehende Probabilistic Scene Model - System vorgestellt und erklärt. Dabei wird auf das Grundprinzip, die Struktur und die einzelnen Komponenten eingegangen.

3.1 PSM - Probabilistic Scene Model

Bei der Entwicklung des Systems war das Ziel, ein System zur Erkennung von Szenen zu entwickeln. Dabei sollten viele Informationen umfasst werden. Dies umfasste die beteiligten Objekte bzw. deren Erscheinung, welche von den zur Erkennung eingesetzten Werkzeugen abhängig ist. Die Auftrittshäufigkeit der Objekte sollte ebenfalls mit einfließen. Der Hauptinformationsträger sind die Relationen der Objekte untereinander, welche in Form von relativen Objektlagen berücksichtigt wurden. Es wurde berücksichtigt, dass Relationen nicht statisch, sondern dynamisch sind. Da eine Szene auch unabhängig von ihrem Ort der Demonstration erkannt werden soll, musste die Lagebeschreibung invariant gegenüber Rotation und Translation sein.

Weiterhin sollte Robustheit gegenüber verschiedenen Störfaktoren bestehen. Fehlende Objekte sollen eine Erkennung der Szene erlauben, wenn auch mit entsprechend reduzierter Konfidenz. Da jedes Objekt fehlen kann darf es kein zentrales Referenzobjekt geben, von dem die Relationen zu den anderen Objekten der Szene ausgehen. Überzählige Objekte sollen sich nicht negativ auf das Erkennungsergebnis auswirken. Generell soll sich die Funktionsweise des entwickelten Systems im Rahmen dessen bewegen, was plausibel erscheint.

Es wird im folgenden erst auf die Struktur des Systems, dann auf die Eingabeverarbeitung, das innere Datenmodell und letztendlich auf die eigentliche Erkennung eingegangen. [Geh14]

3.1.1 Aufbau

Das System ist in mehrere Komponenten aufgeteilt die verschiedenen Aufgaben dienen. Für diese Arbeit wichtig sind vorallem die Komponenten Learner und Inference. Der Learner ist für das Anlernen der Daten und das Einspeichern von Szenen zuständig. Die Inference übernimmt die Erkennung der Szene indem sie in Echtzeit Daten empfängt und die Wahrscheinlichkeiten der Szenen ausgibt. Es gibt außerdem eine Visualizer-Komponente, die dafür zuständig ist, dass verschiedene programminterne Prozesse über Rviz für den Nutzer sichtbar und verständlich gemacht werden.

3.1.2 Learner - Anlernen der Daten

Die Learnerkomponente berechet aus den Objekten, die sie bekommt, die Parameter für das Szenenmodell, dass für die Erkennung benötigt wird. Der Learner ist in mehrere Teilbereiche eingeteilt, Engine, OCM und Szenenmodell. Engine ist die Schnittstelle des Learners und kapselt die anderen beiden Architekturgliederungen voneinander ab.

Abbildung 3.1 beschreibt den Learner als Klassendiagramm. Die darin vorkommende *SceneLearningEnginge* ist die Schnittstelle des Moduls zum Rest des PSM- Systems. Sie liest die in der Launch-Datei gegebenen Parameter aus und überprüft, ob alle Parameter sich mit dem passenden Datentyp auslesen lassen. Außerdem ist die Klasse für Visualisierung des Modells zuständig, welches gerade gelernt wurde.

Die gegebenen Objekte könnene mehrere Szenen beschreiben, da man auch Objekte aus einer Datei auslesen kann und diese jeweils eine pattern variable haben, die beschreibt zu welcher Szene sie zugehörig sind. Für jede Szene die gerade gelernt wird, gibt es einen Lerner in der *SceneModelLearner*-Klasse, welche dafür verantwortlich ist, die einzelnen Lerner zu verwalten. Die Daten der verschiedenen Szenen werden auf die dazugehörigen Lerner aufgeteilt und es gibt eine seperaten Lerner für den Hintergrund, dessen Daten für den Fall wichtig sind, dass keine der Szenen in den gemessenen Objekten vertreten ist. Es ist sozusagen die Szene, die die Gegenwahrscheinlichkeit symbolisiert.

Alle Lerner erben vom *SceneLearner*, der eine abstrakte Basisklasse darstellt und eine Schnittstelle für das Lernen, Speichern und Visualisieren bietet. Der Lerner für den Hintergrund ist eine Instanz des *BackgroundSceneLearner*, welcher blos die Anzahl der Objekte abspeichert, um daraus einen validen Schwellenwert beziehungsweise eine sinnvolle Gegenwahrscheinlichkeit zu bestimmen. Der *ForegroundSceneLearner* ist eine abstrakte Klasse, sodass man das Modell der Berechnung leicht austauschen kann, allerdings konzentriert sich das PSM-System auf den Einsatz des OCM als Repräsentation der Szenenobjekte. Der *OCMForegroundSceneLearner* ist eine Unterklasse des *ForegroundSceneLearner* und kapselt die Lerner für den Vordergrund, welche Instanzen der Klasse *OCMSceneObjectLearner* sind. [Geh14]

3.1.3 Model - Szenenmodell

Das Szenenmodell wird als XML-Datei abgespeichert. Dadurch kann man manuell das erstellte Modell lesen, verstehen und verändern, falls dies zum testen nötig ist. Man

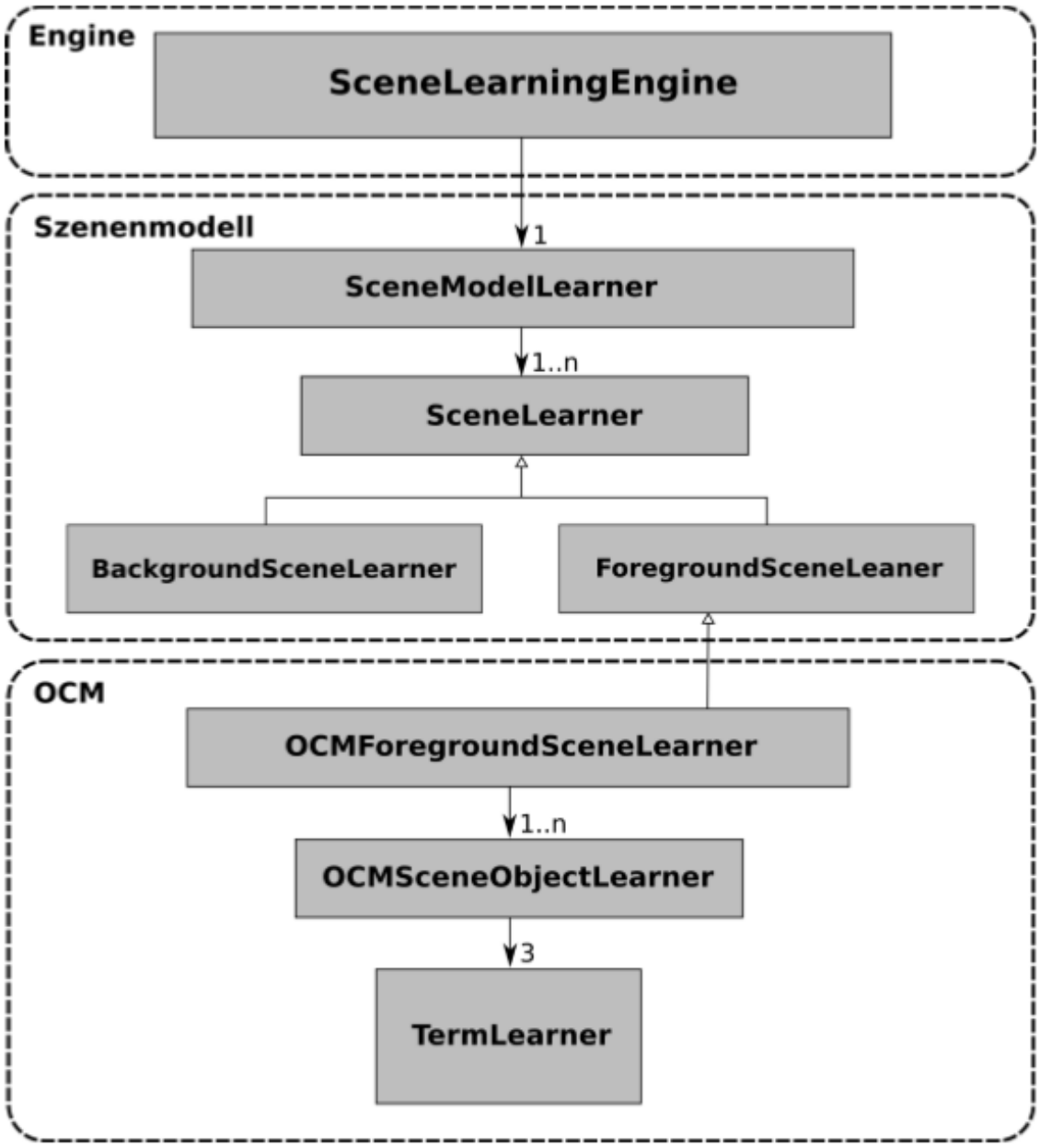

Abbildung 3.1: Klassendiagramm des Learner-Moduls

kann auch leicht mehrere erstellte Szenenmodelle kombinieren, da man alle Parameter verändern und leicht eine zusätzliche Szene aus einem anderen Modell hinzufügen oder ersetzen kann.

Abbildung 3.2 zeigt ein Beispielmodell für eine Frühstücksszene namens *breakfast* und die immer vorhandene Hintergrundszene *background*. Jeder Szene ist eine apriori-Wahrscheinlichkeit zugeordnet, welche beschreibt, wie wahrscheinlich es grundsätzlich ist, dass die Szene auftritt, allerdings ist diese im PSM-System generisch gleichverteilt und kommt nur der Vollständigkeit halber vor. Man kann diese Werte falls nötig manuell ändern, das erstellte Szenenmodell der Learner-Komponente wird aber stets gleichverteilte Werte beinhalten. Der Teil der Hintergrundszene im Modell beinhaltet Informationen über die Anzahl unterschiedlicher Objekte und die Größe des Bereichs auf dem Szenen erkannt werden.

Die Szene namens *breakfast* beinhaltet zwei Szenenobjekte, *Cup* und *CoffeeBox*. Als Beispiel wurde eine einfache Szene mit geringer Objektanzahl gewählt um die Erklärung möglixhst simpel zu halten. Außerdem ist nur das Szenenobjekt *Coffebox* ausgeklappt in der Modellabbildung, damit die Abbildung übersichtlicher ist. Für jeden Term des OCM sind die entsprechenden Parameter im Objekt gespeichert. Jedes Objekt in der Szene hat eine apriori-Wahrscheinlichkeit, welche die Wichtigkeit des Objekts in der Szene verdeutlicht. Desweiteren enthält es Informationen über die Anzahl der Slots, welche zwar aus dem kompletten Modell hergeleitet werden kann, man spart aber Ladezeit dadurch. Die Terme die im OCM vorkommen sind *Scene Shape* und *Objekt Appearance*.

Der *Scene Shape*- Term wird im Modell mit *shape* bezeichnet und beinhaltet einen Baum, welcher die Relation zwischen den Szenenobjekten beschreibt. Der Baum besteht aus einem Wurzelknoten *root* und beliebig vielen Kindsknoten *child*, welche wiederum Kindsknoten beinhalten können. Der Wurzelknoten enthält das Volumen *volume* auf dem die Erkennung arbeiten soll und die Kindsknoten jeweils Daten die die relative Lage zum direkten Elternknoten bezeichnen. Diese Daten, die unter der Bezeichnung *pose* gespeichert sind, sind eingeklappt, da es unübersichtlich wäre, und enthalten Mittelwerte, Gewichte und Kovarianzmatrizen der einzelnen Gauss-Kernel.

Der *Objekt Appearance*-Term ist in zwei Abschnitte aufgeteilt. Erst wird jedem Objektnamen ein Index zugeordnet. Anschließend gibt es eine Wahrscheinlichkeitstabelle, die für jeden Index eine Auftrittswahrscheinlichkeit definiert. Der Index 0 wird bei der Zuordnung bewusst nicht belegt, da er ein Platzhalter für unbekannte Objekte ist. [Geh14]

3.1.4 Inference - Szenenerkennung

Die Szenenerkeennung oder auch Inferenz hat zur Aufgabe, aufgrund von dem gegebenen Szenenmodell und momentan erkannten Objekten, im folgenden auch Evidenz genannt, für jede Szene im Modell einzuschätzen, wie wahrscheinlich es ist, dass diese in der Evidenz vorkommt. Die verschiedenen Szenenwahrscheinlichkeiten werden dann miteinander verrechnet und es wird die relative Wahrscheinlichkeit für jede Szene ausgegeben, dass diese vorhanden ist. Diese Erkennung passiert in Echtzeit und ist als datengetriebenes Modul zu betiteln, da sich die Ausgaben und das Verhalten des Programms, maßgeblich

```
−<psm>
  −<scenes>
    −<scene type="background" name="background" priori="0.5">
       <description objects="2" volume="27"/>
     </scene>
    −<scene type="ocm" name="breakfast" priori="0.5">
      −<object name="CoffeeBox" type="ocm" priori="0.5">
         <slots number="2"/>
        −<shape>
          −<root volume="27">
            −<child name="Cup">
              +<pose></pose>
              </child>
            </root>
          </shape>
        −<appearance>
          −<mapping>
              <map id="1" name="CoffeeBox"/>
              <map id="2" name="Cup"/>
            </mapping>
          −<table>
              <entry values="0 1 0"/>
              <entry values="0 0 1"/>
            </table>
          </appearance>
        −<occlusion>
          −<table>
              <entry values="0 1"/>
              <entry values="0.5 0.5"/>
            </table>
          </occlusion>
        </object>
      +<object name="Cup" type="ocm" priori="0.5"></object>
      </scene>
  </scenes>
</psm>
```

Abbildung 3.2: Beispiel eines Szenenmodells - Frühstücksszene

dadurch verändert, welche Daten die Evidenz enthält, das heißt, welche Objekte erkannt werden.

In Abbildung 3.3 sieht man das Klassendiagramm der Szenenerkennung, welches vergleichbar zu dem des Learner-Moduls strukturiert ist. Die verschiedenen Teilbereiche sind wieder Engine, OCM und Szenenmodell. Engine ist die Schnittstelle und kapselt Szenenmodell und OCM voneinander ab. *SceneInferenceEngine* ist für das Auslesen der Parameter aus der Launch-Datei, für die Visualisierung des Erkenners und die Annahme und weitergabe der erkannten Evidenz zuständig. Diese wird an *SceneModelDescription* weiter gegeben, welche für jede Szene im Modell eine Instanz der Klasse *SceneDescription* erstellt. Ob es sich um eine Vorder- oder Hintergrundszene handelt, kann man dadurch erkennen, die daraufhin erstellte Instanz einer Unterklasse der Abstrakten Klasse *SceneContent* entweder vom Klassentyp *BackgroundSceneModel* oder *ForegroundSceneModel* ist.

In *ForeGroundSceneModel* sind Instanzen von *SceneObjektDescription* gekapselt, allerdings wird im PSM-System nur eine einzige Instanz benutzt, da dieses nur den Ansatz des OCM verfolgt und diese Kapselung dafür gedacht ist verschiedene Ansätze möglich zu machen. Die Unterklasse *OCMSceneObjektContent* ist der besagt Algorithmus des PSM-Systems und beinhaltet mehrere *TermEvaluator* die für die *Object Appearance*, *Object Existance* und *SceneShape* zuständig sind.

Die jeweils berechneten Wahrscheinlichkeiten werden eingespeichert und von der *SceneInferenceEngine* über mehrere Klassen hinweg abgefragt und über das Visualisierungsmodul ausgegeben. [Geh14]

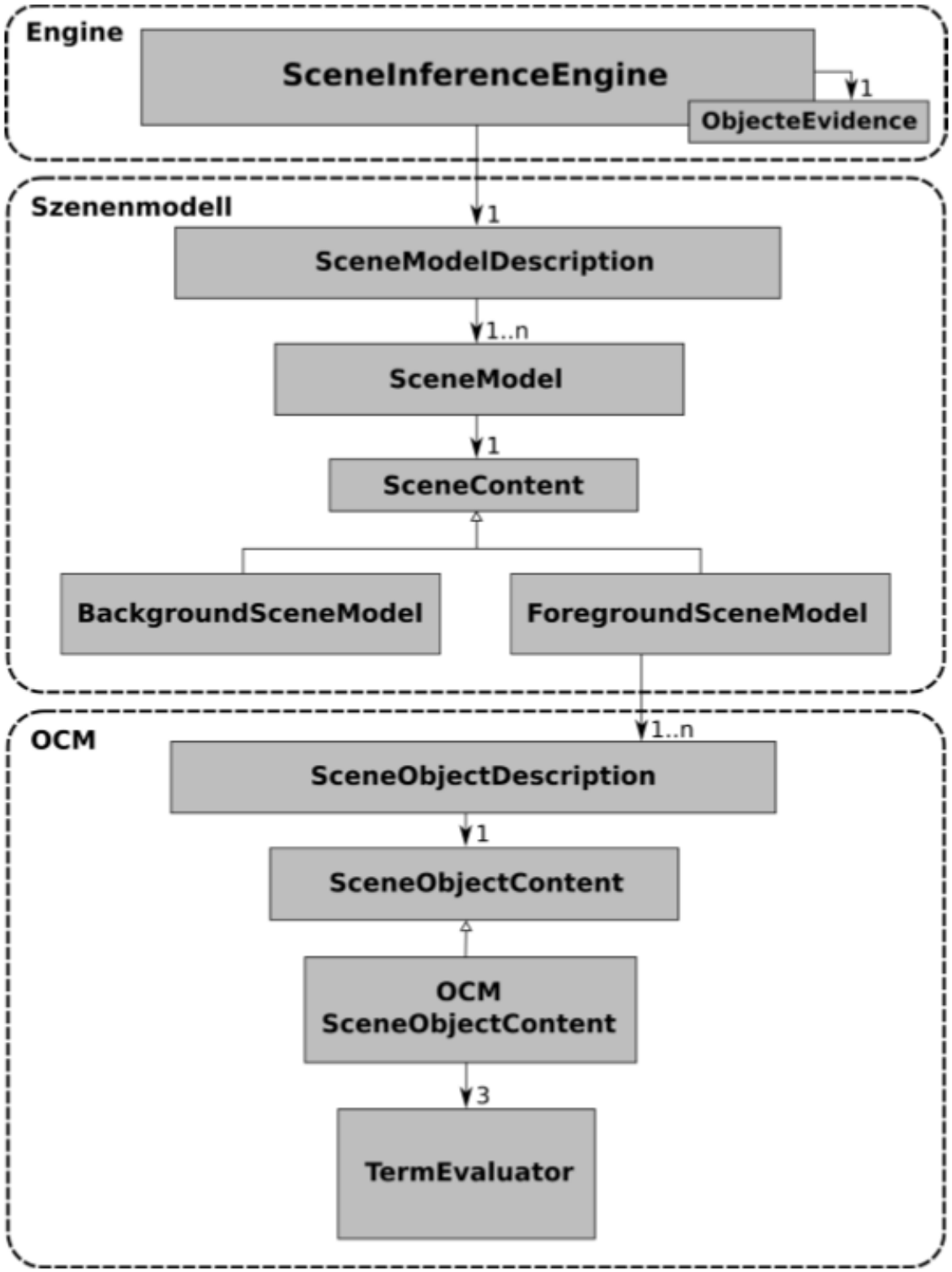

Abbildung 3.3: Klassendiagramm des Inference-Moduls

4. Konzept

Das folgende Kapitel thematisiert das Konzept, dass im Zuge der vorliegenden Arbeit entwickelt wurde, um die vorgestellte Problemstellung zu lösen. Zwischen vielen verschiedenen möglichen Ansätzen einen neuen Modus für das PSM System zu entwickeln, entschied ich mich für einen differenzbasierten Vergleich der Positions- und Rotationsrelationen. Dieser wird im ersten Abschnitt erläutert und anschließend wird der Algorithmus sprachlich, grafisch und in Pseudocode erklärt. Zum Schluss wird die Wahrscheinlichkeitsabschätzung für den Algorithmus erklärt und begründet.

4.1 Ansatz

Im vorhandenen PSM-System werden im Learner die erhaltenen Positions- und Rotationsdaten zu einem Modell zusammengefasst, dass die vorkommen aller Objekte in Relation zueinander zusammenfasst. Dieser Vorgang führt dazu, dass teilweise Zusammenhänge in den Daten betont werden, aber auch zu einem Informationsverlust da Ausreißer und mutmaßliche Fehlmessungen dadurch verloren gehen. Deshalb kann es sinnvoll sein direkt auf den gemessenen Daten zu arbeiten, um ein Ergebnis zu erhalten welches alle Daten berücksichtigt.

Wir betrachten folgendes Szenario. Ein Roboter soll seine Aufgaben aufgrund von einer Szenenerkennung einschätzen und durchführen. In der Szene "Kaffee" gibt es eine volle Kaffeetasse und einen Teelöffel und seine Aufgabe ist es mit dem Löffel den Kaffee umzurühren. In seinen Referenzdaten zu der Szene war der Löffel meist direkt neben der Tasse und nur in einem Fall ein Stück weiter entfernt. Allerdings gilt jede einzelne aufgezeichnete Refernzszene auf äquivalente Weise als Beispiel für die Szene "Kaffee". Das Parametermodell würde diese Ausreißerdaten allerdings glätten und kaum berücksichtigen, sodass der Roboter die Szene selbst mit genau dem Aufbau aus den Refernzdaten möglicherweise nicht erkennen würde. Wenn man allerdings die Erkennung direkt auf den Referenzdaten basiert, erkennt die Szenenerkennung den Ausreißer auch, da sie ja eine Instanz der Szene mit diesem vergleicht, welche diesem entspricht.

Abbildung 4.1: Beispiel: Kaffeetasse - Ausreißer in den Daten

Abbildung 4.1 verdeutlicht das genannte Szenario. Die roten Punkte stehen für die gemessenen Positionen und die grünen Pfeile stehen für die räumliche Relation zwischen den Objekten. Links sieht man, dass die meisten Messungen einen kleinen Abstand zwischen Löffel und Tasse haben, rechts ist der Ausreißer dargestellt, der möglicherweise vom alten System nicht als die gelernte Szene erkannt wird.

Im differenzbasierten Modus sollen also gemessene Objekte direkt mit den Referenzdaten verglichen werden, die das System bereits gelernt hat. Der Algorithmus betrachtet alle Objekte vollvermascht, sodass er die Szenenreferenz findet, die die maximale Ähnlichkeit zu den gemessenen Objekten hat. Darauf basierend wird die Wahrscheinlichkeit abgeschätzt, dass die gemessenen Objekte die Referenzszene enthalten oder repräsentieren. Dabei stören zusätzliche Objekte die Erkennung nicht und eine Unvollständigkeit der Szene führt zu einer kleineren Wahrscheinlichkeit aber nicht zu direkter Ablehnung, da die Szene noch durch weitere Objekterkennungen vervollständigt werden könnte.

4.2 Erkennungsalgorithmus

Der Algorithmus wurde mit den in Ansatz genannten Annahmen und Einschränkungen entwickelt und hat zur Aufgabe zu jeder Szene die auf Vorkommen geprüft wird die Instanz der Szene in den Daten zu finden, die am dichtesten an den gemessenen Daten liegt und so die höchste Wahrscheinlichkeit aufzeigt, dass die gemessenen Daten die Szene enthalten. Nachdem die Wahrscheinlichkeit einer Szene bestimmt ist wird diese mit den anderen Szenen genau wie im bestehenden PSM-System verrechnet, sodass am Ende die Relative Wahrscheinlichkeit für alle zutestenden Szenen angegeben wird.

4.2.1 Algorithmus: Beschreibung

Der Algorithmus läuft wie folgt ab. Für jedes Objekt, der zu testenden Szene, wird überprüft ob es sich um ein Objekt handelt, welches gerade von der Objekterkennung erkannt

wird. Wenn dies nicht der Fall ist, wird das Objekt übersprungen. Wenn dies allerdings der Fall ist, wird das Objekt zeitweise zu unserem Referenzobjekt und der Algorithmus führt für jede Instanz der zu testenden Szene in den Daten folgendes aus:

Es wird über alle anderen Objekte der Szeneninstanz iteriert und für jedes Objekt abgefragt, ob es ein gemessenes Objekt gibt, welches die Repräsentation für das Datenobjekt sein kann. Falls das Objekt nicht in den momentan wahrgenommenen Objekten vorkommt, ist die Szene allein aus Sicht dieses Objekts betrachtet unwahrscheinlich. Deshalb wird eine Objektwahrscheinlichkeit von 0 eingespeichert, welche aussagt, dass dieses Objekt allein die Szene als nicht auffindbar beschreibt. Wenn allerdings eine Instanz der Objekts in der Iteration gefunden wird, berechnet der Algorithmus die Positions- und Rotationsrelation zwischen den beiden Objekten aus den Daten. Genauso wird die Position und Rotationsrelation der gemessenen Objekte zueinander berechnet, welche mutmaßlich den Objekten aus der Datenbank entsprechen sollten. Nach diesen Berechnungen wird die Differenz verglichen die zwischen den beiden Paaren jeweils besteht. Dabei werden Positionsrelationen und Orientierung beziehungsweise Rotation jeweils getrennt betrachtet. Aus dem Grad der Ähnlichkeit dieser Differenzen lässt sich nun die Wahrscheinlichkeit ableiten, ob das Objekt so vorkommt wie die Szene aufgezeichnet wurde. Somit hat man eine Objektwahrscheinlichkeit.

Nun werden alle Objektwahrscheinlichkeiten zu einer Szenenwahrscheinlichkeit zusammengefasst. Diese wird innerhalb der Iteration über die Szeneninstanzen maximiert. Außerdem wird dieser Maximalwert wiederum innerhalb der äußersten Schleife maximiert, welche über alle Objekte iteriert, die sowohl in der zu testenden Szene sind als auch von der Objekterkennung erkannt wurden. Es wird also der absolute maximale Wert der Szenenwahrscheinlichkeit bestimmt, den man mit einer der Szeneninstanzen mit der beschriebenen Schleife mit jedwedem Referenzobjekt erzeugen kann. Dieser Maximalwert ist die Wahrscheinlichkeit, ob die zu testende Szene in den Gemessenen Objekten und potentiell weiteren unbekannten Objekten enthalten ist, welche von der Objekterkennung noch erkannt werden könnten.

Abbildung 4.2 beschreibt den Algorithmus als vereinfachtes Flussdiagramm. Einfache Linien verdeutlichen den Programmfluss in den verschiedenen Fällen und die Flussrichtung ist stehts nach unten. Die Schleifen sind zusammengefasst damit das Diagramm übersichtlich bleibt.

4.2.2 Algorithmus: Pseudocode

Um das Algorithmuskonzept weiter zu verdeutlichen beschreibt Abbildung 4.3 den Algorithmus nochmals mit Pseudocode. Damit der Code verständlich wird hier eine kurze Erklärung zu der Abbildung. Die Einrückungen wurden statt den geschweiften Klammern verwendet, die in den meisten höheren Programmiersprachen vorkommen. Die Parameter sind wie folgt definiert. Szenenmodell beschreibt das Szenenmodell, dass die vorkommenden Objekte in der zu testenden Szene beinhaltet. Der Parameter Gemessen beschreibt die Objekte die momentan erkannt werden und real oder in einer Simulation vorhanden sind. Der Parameter Daten steht für die Instanzen die zur zu testenden Szene als Referenzen gespeichert sind.

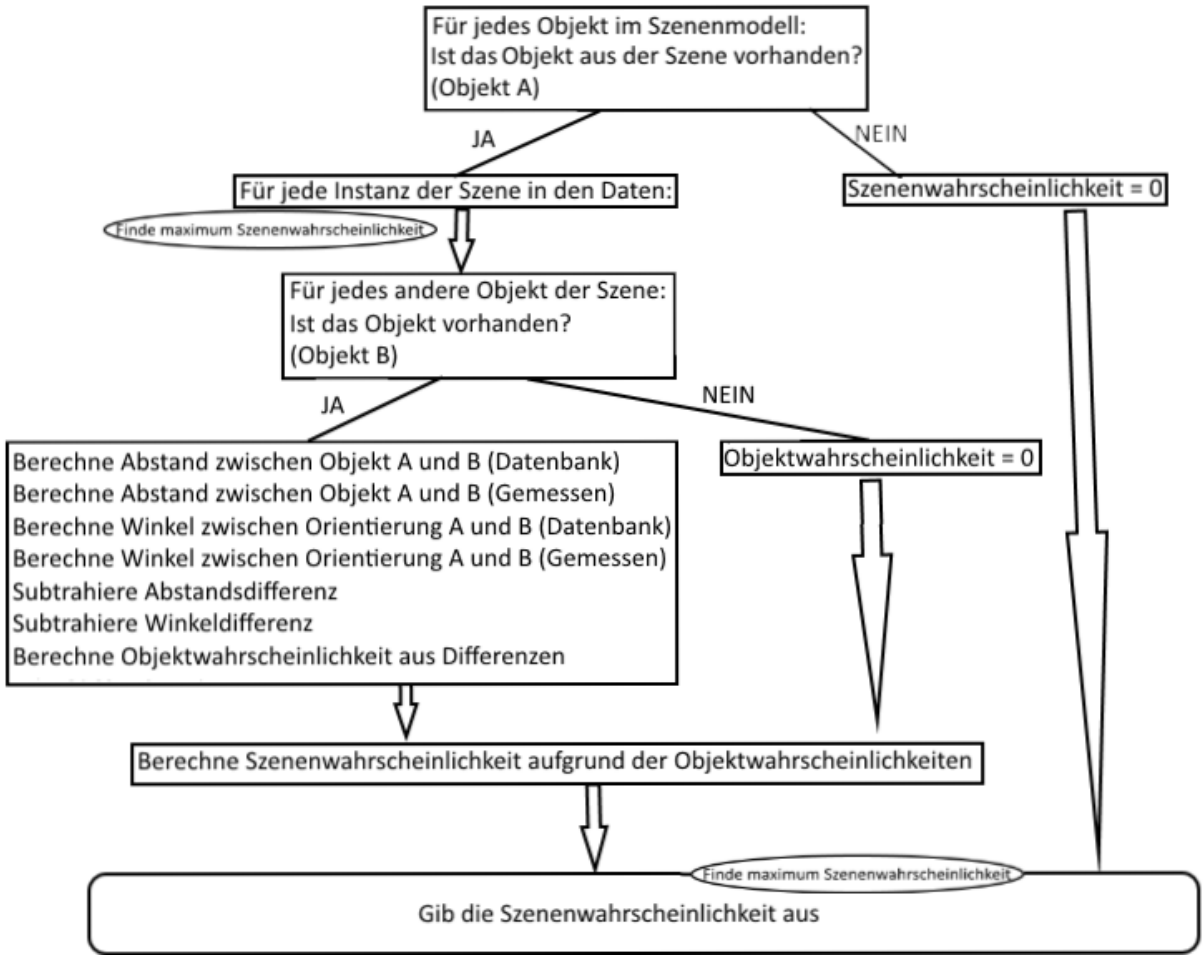

Abbildung 4.2: Algorithmus als vereinfachtes Flussdiagramm

```
BerechneSzenenWahrscheinlichkeit ( Objekt[] Szenenmodell, Objekt[] Gemessen, Szeneninstanz[] Daten )
    SzenenWahrscheinlichkeit = 0;

    foreach - Objekt A in Szenenmodell

        if  - Gemessen->EnthaeltObjektMitName ( A->Name )
            ReferenzObjekt = Gemessen->FindeObjektMitName ( A->Name );

            foreach - Szeneninstanz S in Daten
                SzenenWahrscheinlichkeitsSumme = 1;

                foreach - Objekt B in S
                    Objektwahrscheinlichkeit = 0;

                    if - Gemessen->EnthaeltObjektMitName ( B->Name )
                        ZweitesObjekt = Gemessen->FindeObjektMitNamen ( B->Name );
                        Objektwahrscheinlichkeit = BerechneObjektWahrscheinlichkeit( ReferenzObjekt, ZweitesObjekt, A, B);

                    SzenenWahrscheinlichkeitsSumme += Objektwahrscheinlichkeit;

                NeueSzenenWahrscheinlichkeit = SzenenWahrscheinlichkeitsSumme / Szenenmodell->Laenge;

                if - NeueSzenenWahrscheinlichkeit > SzenenWahrscheinlichkeit
                    SzenenWahrscheinlichkeit = NeueSzenenWahrscheinlichkeit;
```

Abbildung 4.3: Algorithmus als PseudoCode

Die Variable Name, die jedes Objekt gesetzt hat ist eine Identifizierung um Objekte ihren mutmaßlichen Vorkommen in der Messung zuzuordnen. Die Funktion Enthaelt-ObjektMitName(string Name) gibt true aus, falls die aufrufende Liste von Objekten ein Objekt mit dem gegebenen Namen enthält. Ansonsten wird false ausgegeben.

Die Funktion FindeObjektMitName(string Name) gibt das Objekt aus der Liste zurück, welches den gegebenen Namen trägt. Falls kein Objekt mit dem gegebenen Namen existiert wird NULL zurückgegeben. BerechneObjektWahrscheinlichkeit(Objekt C, Objekt D, Objekt A, Objekt B) nimmt vier Objekte und berechnet die Positions- und Orientierungsunterschiede zwischen den Parametern C und D sowie zwischen A und B. Anschließend werden die Differenzen abgeglichen und basierend auf den Unterschieden eine Wahrscheinlichkeitsabschätzung zwischen 0 und 1 abgegeben, wobei 1 für ”mit der Szene übereinstimmend” und 0 für ”weit entfernt” steht. Auch die Komplette Funktion speichert am Ende eine Wahrscheinlichkeit zwischen 0 und 1. Sie wird nicht ausgegeben, da auf die eingespeicherte Wahrscheinlichkeit über eine andere Schnittstelle zugegriffen wird und der Algorithmus nur den Zweck erfüllt die Wahrscheinlichkeit zu berechnen.

4.3 Wahrscheinlichkeitsabschätzung

Zuerst zeigen wir, dass die Abschätzung beim einzelnen Objekt präzise ist, anfolgend wird die Berechnung der gesammten Szenenwahrscheinlichkeit erklärt. Die Wahrschein-

lichkeiten sind von gewissen Schwellenwerten abhängig, da der Maßstab und die Szene die überprüft wird unterschiedliche Anforderungen haben kann.

Bei einer Beispielszene, die den gedeckten Frühstückstisch repräsentiert, hat man sicher noch eine gewisse Tolleranz, wenn es um die räumliche Positionierung der Objekte zueinander geht, allerdings gibt es kaum Kompromisse bei der Rotation. Wenn die Teller umgekehrt wären, würde es nicht mehr dem gewohnten gedeckten Tisch ensprechen. Wenn hingegen ein Ball in einer Szene vorkommt, wird dieser mehr Tolleranz gegenüber Rotation aber möglicherweise einen kleineren Schwellenwert bei der Position haben. Man sieht also, dass die Schwellenwerte nötog sind, um in verschiedenen Szenenkontexten sinnvolle Ergebnisse zu bekommen.

4.3.1 Objektwahrscheinlichkeit

Es gibt eine Instanz der zu testenden Szene. Außerdem gibt es ein Referenzobjekt mit Positions- und Rotationsdaten. Es gibt ein Testobjekt mit Positions- und Rotationsdaten. Es gibt ein gemessenes Referenzobjekt, welches dem gewählten Referenzobjekt in der aktuellen Objekterkennung entspricht. Nun wird das Objekt in den erkannten Objekten gesucht, welches die selbe identifikation, wie unser Testobjekt hat. Dieses nennen wir gemessenes Testobjekt. Falls wir kein Objekt finden, welches die gewollten Eigenschaften hat ist die Objektwahrscheinlichkeit gleich null, da die Szene aus der Perspektive des Testobjekts nicht auffindbar ist.

Die Objektwahrscheinlichkeit beschreibt die Wahrscheinlichkeit für das Testobjekt, dass die Szene aus der es entspringt in den aktuell erkannten Objekten vorkommt. Diese wird bestimmt indem das Verhätnis zwischen Testobjekt und Referenzobjekt mit dem zwischen gemessenem Testobjekt und gemessenem Referenzobjekt abgeglichen wird. Dieser Vergleich vergleicht einerseits die Positionsrelationen sowie auch den Winkelunterschied zwischen den Orientierungen der Objekte. Die beiden Einzelvergleiche arbeiten mit Schwellenparametern, die bestimmen wie groß der Unterschied sein muss, damit die Wahrscheinlichkeit null entspricht. Beim Winkel kann dieser Wert zwischen 0 und 180 liegen, beim Positionsvergleich ist dieser Parameter eine beliebige Zahl größer oder gleich null. Ausformuliert bedeuten diese Parameter: Wie weit kann man die Szene verändern, bevor die Erkennung sie nicht mehr erkennen soll?

In Abbildung 4.4 kommen Testobjekt T, Referenzobjekt R, gemessenes Testobjekt gT, gemessenes Referenzobjekt gR und die Parameter, Positionsparameter pP und Rotationsparameter rP, vor. Alle Objekte haben geben mit **.p** ihre Position als Vektor und mit **.r** ihre Rotation als Matrix in einem Weltkoordinatensystem zurück. Die Funktion RotationDiff nimmt zwei Rotationsmatrizen und berechnet den Rotationsunterschied zwischen diesen. Position(T, R, gT, gR) berechnet den Differenzvektor zwischen den Relationen von R zu T und von gR zu gT und bestimmt anschließend dessen länge. Die Funktion Rotation(T, R, gT, gR) berechnet die Rotationsmatrix, um die man Rotieren müsste um aus der Orientierungsrelation von R und T zu der vergleichbaren von gR und gT zu rotieren und gibt den Winkel dieser Rotation aus. Die Funktionen PositionParametisiert(T, R, gT, gR, pP) und RotationParametisiert(T, R, gT, gR, rP) benutzen

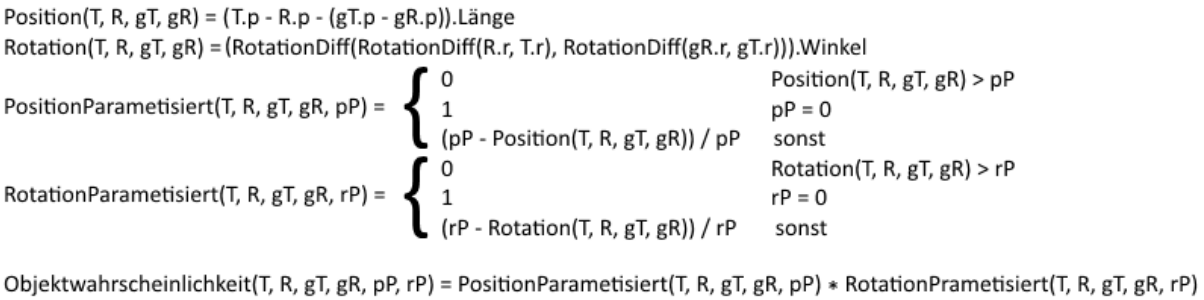

Abbildung 4.4: Formeln zur Objektwahrscheinlichkeit

die Schwellenparameter pP und rP, um aus der Distanz oder dem Winkel die Wahrscheinlichkeitsabschätzungen zu gewinnen. Obejektwahrscheinlichkeit(T, R, gT, gR, pP, rP) kombiniert die Wahrscheinlichkeit, die rein durch die Positionsrelationen begründet ist, sowie die Wahrscheinlichkeit, die nur aus der Rotation basiert, zu einer gesammten Objektwahrscheinlichkeit.

Im gesamten handelt es sich um eine Wahrscheinlichkeitsabschätzung die sich aus den Komponenten der Positions- und der Rotationswahrscheinlichkeit zusammensetzt. Die beiden einzelnen Komponenten beschreiben jeweils wie wahrscheinlich ist, dass aufgrund der gegebenen Objekte die zu testende Szene in der Objekterkennung repräsentiert wird, wenn man nur die jeweilige Komponente betrachtet.

4.3.2 Szenenwahrscheinlichkeit

Die Szenenwahrscheinlichkeit setzt sich aus den Objektwahrscheinlichkeiten der Szene zusammen. Mit jedem möglichen Referenzobjekt wird in jeder Instanz der Szene in den Daten die Objektwahrscheinlichkeit für jedes andere Objekt berechnet und anschließend in eine mögliche Szenenwahrscheinlichkeit zusammengefasst. Diese möglichen Szenenwahrscheinlichkeiten bestimmen, wie wahrscheinlich es ist, dass die Szene in den gemessenen Objekten repräsentiert ist, wenn man nur genau die gewählte Instanz der Daten und das gewählte Referenzobjekt betrachtet. Aus den möglichen Szenenwahrscheinlichkeiten wird die maximale Szenenwahrscheinlichkeit als das entgültige Ergebnis ausgewählt. Man findet also die Wahrscheinlichkeit dafür, dass die Szene vorhanden ist, ausgehend von den Referenzdaten, die am dichtesten an den gemessenen Daten sind.

Die Objektwahrscheinlichkeiten werden jeweils mit einem A-priori Wert multipliziert und ihre Wichtigkeit in der Szene zu gewichten. Objekte müssen nicht in jeder Datenaufzeichnung der Szene vorhanden sein und der A-priori Wert zeigt auf wie signifikant und wichtig das Auftreten eines Objekts in einer Szene ist. Zum Beispiel ist eine Gabel auf fast jedem gedeckten Mittagstisch, tiefe oder flache Teller werden sich jedoch teils in den Daten abwechseln und nicht beide in jeder Aufzeichnung in der Szene vorhanden sein. Somit hätte die Gabel einen höheren A-priori Wert.

Die gewichteten Objektwahrscheinlichkeiten werden nun aufaddiert. Jede Objektwahrscheinlichkeit hat einen Wert, der größer oder gleich 0 und kleiner oder gleich 1 beträgt.

Szene S
Instanz I
Referenzobjekt R
TestObjekt T
O(R, T, pP, rP) = Objektwahrscheinlichkeit(T, R, FindeGemessenes(T), FindeGemessenes(R), pP, pR)
Szenenwahrscheinlichkeit(S, pP, rP) = MaxForeach(S,MaxForeach(I, 1 * apriori(R) + SumForeach(I, O(R, T, pP, rP) * apriori(T))))

Abbildung 4.5: Formeln zur Szenenwahrscheinlichkeit

Außerdem wird noch 1 aufaddiert, da es ja ein Referenzobjekt gibt, bei dem man davon ausgeht, dass es genau an der richtigen Position ist und die gemessene Orientierung hat. Anschließend teilt man die Summe durch die Anzahl aller Objekte, um den durchschnittlichen Wert der Objektwahrscheinlichkeit zu erhalten. Dieser entspricht dem gewichteten Erwartungswert der Objektwahrscheinlichkeit von einem Objekt, dass man zufällig aus der zu testenden Instanz der Szene zieht. Dieser gewichtete Erwartungswert entspricht der Wahrscheinlichkeit, dass die Szene auf Basis der gemessenen Objekte vorhanden sein könnte. Parametisiert muss dieser Wert der Szenenwahrscheinlichkeit nicht mehr werden, da dies schon auf der Ebene der Objektwahrscheinlichkeit passiert.

In Abbildung 4.5 zeigt auf wie sich die Szenenwahrscheinlichkeit ergibt. In der Grafik die Formeln beinhaltet steht S für die zu testende Szene, I für eine Instanz aus den Daten, die die Szene beschreibt, R für eine Referenzobjekt welches variabel wählbar ist und T für das Testobjekt für das gerade die Objektwahrscheinlichkeit berechnet wird. Die Funktion O(R, T, pP, rP) berechnet die Objektwahrscheinlichkeit für das gegebene Testobjekt T in Hinblick auf das Referenzobjekt R und mit den Parametern pP und rP, die die Distanztoleranz für die Position und die Winkeltoleranz für die Orientierung angeben. FindeGemessenes(A) bestimmt jeweils das Objekt in den gemessenen Daten, welches dem gegeben Objekt A entspricht, das heißt die selbe Identifikation wie A hat. Szenenwahrscheinlichkeit(S, pP, rP) nimmt eine zu testende Szene S an und die beiden Schwellenparameter.

Die Funktion MaxForeach iteriert über alle gegebenen Instanzen von S und findet den Maximalwert, welcher beim zweiten Parameter errechnet werden kann. Dieser Parameter, der eine Funktion enthält, die einen Zahlenwert zurückliefert, verwendet I als die aktuelle Instanz, die gerade in der Iteration geprüft wird. MaxForeach kann aber auch eine Instanz als ersten Parameter enthalten und verhät sich dann anders. In diesem Fall wird über alle Objekte in der Szeneninstanz iteriert und jeweils das aktuelle Objekt als Referenzobjekt R ausgewählt. Der zweite Parameter muss eine Zahl zurückliefern und R kann frei in dem Ausdruck verwendet werden. Die Funktion gibt den Maximalwert aus, den sie über alle Schleifendurchläufe findet.

apriori(A) gibt den gespeicherten A-priori Wert des Objekts A zurück, sodass man die Wahrscheinlichkeiten gewichten kann. Die SumForeach iteriert über alle Objekte der gegebenen Instanz I bis auf das momentane Referezobjekt R. Am Ende jedes Schleifenschritts wird der im zweiten Parameter definierte Zahlenwert aufaddiert. Die Variable

T entspricht hier dem Testobjekt, welches das Iterationsobjekt der Schleife ist, also das Objekt welches aktuell in der Iteration geprüft wird.

Es wird also die Instanz in den Daten gesucht deren Objekte die höchsten Erwartungswert haben, wenn es darum geht aufgrund der Objekte und der gemessenen Daten einzuschätzen, ob die zu testende Szene in der Messung repräsentiert wird. Es ist sinnig den Maximalwert zu suchen, da jede Messung gleichermaßen vollwertiges Beispiel für die Szene gilt und damit die maximale Wahrscheinlichkeit bestimmt wird, dass die Szene, die es zu testen gilt, vorhanden ist. Man könnte natürlicch aucch wiederum den Erwartungswert bestimmen, wenn man eine zufällige Instanz aus allen Szenen zieht, um aus allen Szeneninstanzen ein Modell zu generieren, welches die Szene gut repräsentiert, da aber gegeben ist, dass jede einzelne Messung schon alleinstehend die Szene vollends repräsentiert, würde dieses Vorgehen die Erkennung nur unpräziser machen.

5. Implementierung

Im Kapitel Implementierung wird alles beschrieben und erklärt, was am bestehenden PSM-Projekt verändert und hinzugefügt wurde. Außerdem werden ausgewählte hinzugefügte und veränderte Klassen sowie launch-Dateien dokumentiert, sodass das Kapitel das Verständnis und die Nutzung der Neuheiten im System vereinfacht. Nachdem der Umbauprozess beschrieben wird, bei dem eine Klasse komplett aus dem PSM-Projekt ausgetauscht wurde, widmet sich das Kapitel der Umsetzung des Algorithmuskonzepts und der Einbettung in das vorhandene System.

5.1 Umbau PSM

Da das Paket *pbd_msgs* nicht konstenlos zur Verfügung gestellt wird, mussten alle Vorkommen der Klassen aus diesem Paket zu alternativen Ersatzklassen geändert werden. Teilweise konnte man dies durch simple Ersetzung erreichen, allerdings gab es nicht für jede Klasse eine Ersatzklasse mit dem selben Funktionsumfang. In den Fällen, in denen Funktionen fehlten oder geringfügig anders funktionierten, konnte man den Umbau durch kleine Anpassungen erreichen oder musste eigene Funktionen schreiben, welche die nötigen Operationen verichten konnten. Alle auf diese Weise programmierten Funktionen wurden hinreichend auf Gleichheit mit ihren Ursprungsfunktionen in ihrer Funktionsweise getestet, indem die Ergebnisse bei gleichen Eingangsparametern abgeglichen wurden. Außerdem wurde eine Datenbankschnittstelle für das PSM-System hinzugefügt, da es Daten zum anlernen, wegen einem parallel laufenden anderen Projekt, schon in Datenbanken gibt und diese vom System noch nicht verwertet werden konnten.

5.1.1 Klassenaustausch

Der größte Arbeitsaufwand ergab sich dadurch, dasss das die repräsentation der Szenenobjekte im System ausgerauscht werden musste. Im Zuge dieser Arbeit wurde im ganzen PSM-System *pbd_msgs::PbdObject* durch *ISM::Object*, die Objektrepräsentation aus dem

include/inference/model/foreground/ocm/shape/HierarchicalShapeModel.h
include/inference/model/foreground/ocm/shape/HierarchicalShapeModelNode.h
include/learner/SceneLearner.h
include/learner/SceneLearningEngine.h
include/learner/SceneModelLearner.h
include/learner/background/BackgroundSceneLearner.h
include/learner/foreground/ForegroundSceneLearner.h
include/learner/foreground/ocm/OcmForegroundSceneLearner.h
include/learner/foreground/ocm/SceneObjectLearner.h
include/learner/foreground/ocm/ocm/OcmSceneObjectLearner.h
include/learner/foreground/ocm/ocm/OcmTree.h
launch/learner.launch
src/inference/model/foreground/ocm/shape/HierarchicalShapeModel.cpp
src/inference/model/foreground/ocm/shape/HierarchicalShapeModelNode.cpp
src/learner.cpp
src/learner/SceneLearner.cpp
src/learner/SceneLearningEngine.cpp
src/learner/SceneModelLearner.cpp
src/learner/background/BackgroundSceneLearner.cpp
src/learner/foreground/ForegroundSceneLearner.cpp
src/learner/foreground/ocm/OcmForegroundSceneLearner.cpp
src/learner/foreground/ocm/ocm/OcmSceneObjectLearner.cpp
src/learner/foreground/ocm/ocm/OcmTree.cpp
include/visualization/psm/ProbabilisticPrimarySceneObjectVisualization.h
include/visualization/psm/ProbabilisticSecondarySceneObjectVisualization.h
include/visualization/psm/helper/CoordinateFrameVisualizer.h
include/visualization/psm/helper/GaussianKernelVisualizer.h
include/visualization/psm/helper/KinematicChainVisualizer.h
include/visualization/psm/helper/SampleVisualizer.h
src/visualization/psm/ProbabilisticPrimarySceneObjectVisualization.cpp
src/visualization/psm/ProbabilisticSecondarySceneObjectVisualization.cpp
src/visualization/psm/helper/CoordinateFrameVisualizer.cpp
src/visualization/psm/helper/GaussianKernelVisualizer.cpp
src/visualization/psm/helper/KinematicChainVisualizer.cpp
src/visualization/psm/helper/SampleVisualizer.cpp
include/trainer/PSMTrainer.h
include/trainer/TreeNode.h
include/trainer/generator/heuristic/DirectionRelationHeuristic.h
include/trainer/generator/heuristic/HeuristicalTreeGenerator.h
include/trainer/source/ObjectSetList.h
include/trainer/source/PbdSceneGraphSource.h
src/trainer/PSMTrainer.cpp
src/trainer/TreeNode.cpp
src/trainer/generator/heuristic/DirectionRelationHeuristic.cpp
src/trainer/generator/heuristic/HeuristicalTreeGenerator.cpp
src/trainer/source/ObjectSetList.cpp
src/trainer/source/PbdSceneGraphSource.cpp

Abbildung 5.1: Liste der Dateien die beim Umbau verändert wurden

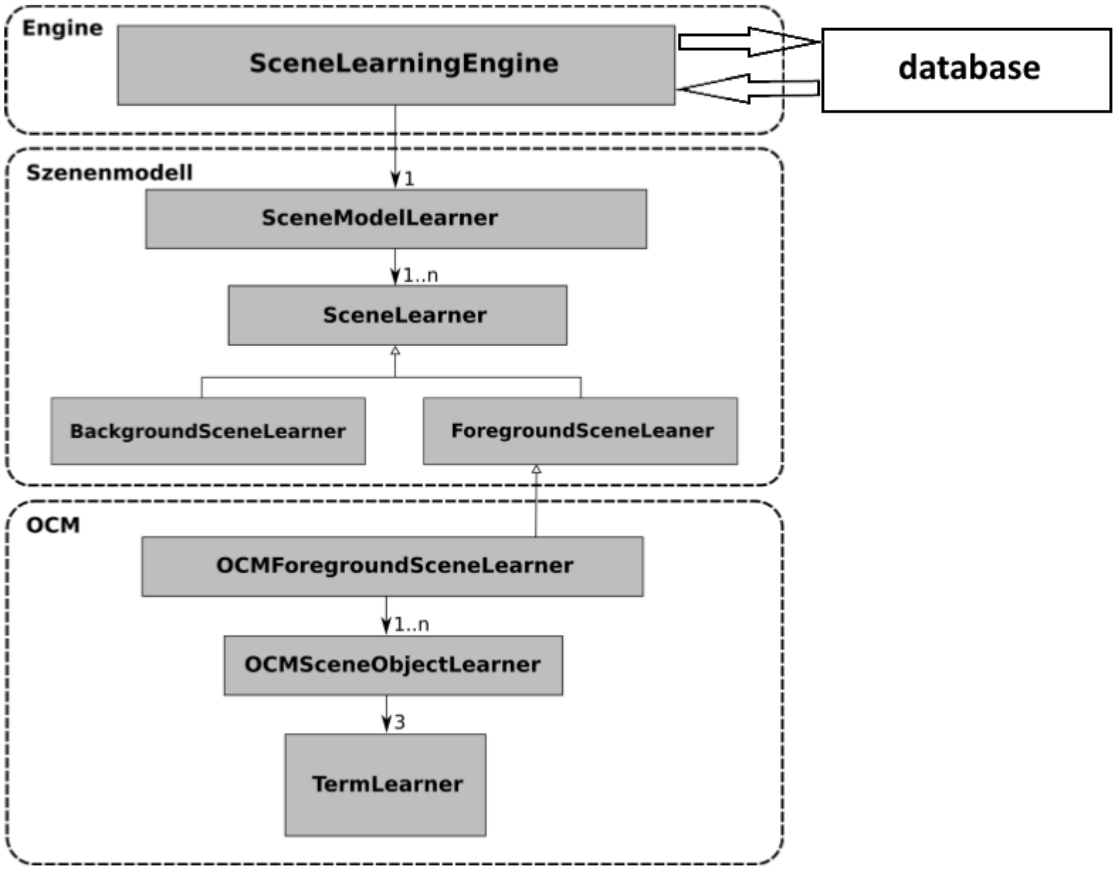

Abbildung 5.2: Datenbankanbindung des Learner-Moduls

ISM-Projekt ersetzt. Später wurden dann die Objekte aus dem ISM-System teils durch *AsrObject* repräsentiert und damit wurde dies auch im PSM-System eingeführt.

In Abbildung 5.1 sind Dateien aufgelistet, die im Zuge des Umbaus geändert werden mussten. Die Liste ist in drei Teile aufgeteilt, die die Änderungen verschiedenen Ordnern zuordnen. Der erste Teil ist im *asr_psm*-Ordner, welcher den Kern des PSM-Projekts enthält. Der zweite Teil ist in *asr_psm_visualizations* enthalten, des Komponente die die Visualisierung des PSM-Systems übernimmt. Der dritte und letzte Part ist in *asr_relation_graph_generator* enthalten.

5.1.2 Datenbankeinbindung

Im vergleichbaren ISM-Projekt, welches eine ähnliche Zielsetzung wie das PSM-Projekt hat, allerdings einen nicht stochastischen Ansatz verfolgt, kann man die Szenen, die das System lernen soll aus einer Datenbank auslesen. Aus Gründen der Vergleichbarkeit wie auch dem Komfort ist es sinnig, diese Datenbankschnittstelle für das PSM System nachzurüsten. Um dies zu erreichen wurde ein Datenbankpfad in die Launch-Datei des

Learners *learner.launch* hinzugefügt und der veraltete rosbagfiles Parameter damit ersetzt. Die Datenbank wird innerhalb der *SceneLearningEngine* Klasse ausgelesen und die erhaltenen Daten, welche jeweils Vorkommen und von verschiedenen Objekten in diversen Szenen repräsentieren, werden anschließend konvertiert, sodass sie für das System sinnvolle AsrObject-Instanzen werden.

Für den Zugriff auf die Datenbank nutzt die Engine-Klasse die im ISM-System gestellte Hilfsklasse *TableHelper*, die auf die Erkennungssysteme zugeschnittene Funktionen beinhaltet, die man für den Datenbankzugriff nutzen kann.

In Abbildung 5.2 sieht man deutlich, wo der Datenbankzugriff im Learner Klassendiagramm geschiet, nämlich bei der *SceneLearningEngine*-Klasse im gekapselten Bereich der Engine. Die Pfeile simbolisieren den Austausch zwischen Programmcode und Datenbank. Der Programmcode schickt anfragen, die Datenbank liefert die gewollten Daten, falls die Anfragen korrekt sind. [Geh14]

5.2 Differenzbasierter Erkennungsalgorithmus

Einerseits beschreibt dieses Kapitel, wie der im Konzept vorgestellte Algorithmus programmiertechnisch umgesetzt, andererseits wie dieser in das bestehende System eingebunden wurde.

5.2.1 Algorithmus

Der Algorithmus wurde umgesetzt, wie er in Kapitel 4.2 beschrieben wurde. Zu diesem Zweck wurden die Klassen *DifferenceForegroundInferenceAlgorithm* und *DifferenceBackgroundInferenceAlgorithm* hinzugefügt, welche einen Modus der Szenenerkennung mit dem beschriebenen Algorithmus implementieren, sowie die dazugehörige Hintergrundwahrscheinlichkeit bestimmen. Im Fall des Differenzbasierten Erkennungsalgorithmus wird die Hintergrundszenenwahrscheinlichkeit nicht aufgrund der Anzahl der Objekte oder dem Volumen der Szene berechnet, sondern man stellt manuell einen Schwellenwert ein, da die eben genannten Parameter keinen direkten Einfluss auf den Erkennungsalgorithmus haben. Man muss also einschätzen, wahrscheinlich eine Szene sein muss um wahrscheinlicher als jede beliebige Szene zu sein und den Parameter entsprechend setzen, da die Wahrscheinlichkeiten miteinander in Relation gesetzt werden und man die Szenenwahrscheinlichkeit aller Szenen auch in Bezug auf die Szenenwahrscheinlichkeit der Hintergrundszene berechnet.

In Abbildung 5.3 sieht man eine vereinfachte Version des Headers der Klasse *DifferenceBackgroundInferenceAlgorithm*. Darin sind alle Klassenvariablen und Methoden der Klasse aufgelistet, bis auf den Konstruktor. Dieser initialisiert verschiedene Variablen,wie zum Beispiel die Schwellenwerte der Positionierung und Rotation, welche den maximalen Bereich angeben indem Objekte überhaupt noch erkannt werden sollen. Außerdem wird im Konstruktor auf die Datenbank zugegriffen und die Szeneninstanzen, die die zu testende Szene repräsentieren, werden in der Variable *mPattern* abgespeichert.

Die Methode *doInference* ist die Hauptschnittstelle zum restlichen System und muss von

```
class DifferenceForegroundInferenceAlgorithm : public ForegroundInferenceAlgorithm {
  public:

    void doInference(std::vector<ISM::Object> pEvidenceList, std::ofstream& pRuntimeLogger);

    double getProbability();

    double differenceBetween(ISM::Object pRoot, ISM::Object pTar, ISM::Object pDiffRoot, ISM::Object pDiffTar);

  private:

    double mMaximumDistance;

    double  mMaximumRotation;

    double mProbability;

    ISM::RecordedPatternPtr mPattern;

    std::string mDataBaseName;

    std::string patternName;

    boost::shared_ptr<ISM::TableHelper> tableHelper;

    ISM::Object findObjectOfType(std::vector<ISM::Object> pList, std::string pTypeAndObservedId);

    void normalizeVector3d(Eigen::Vector3d input);

};
```

Abbildung 5.3: Header-Datei der Klasse DifferenceForegroundInferenceAlgorithm

DifferenceForegroundInferenceAlgorithm implementiert werden, da die Klasse von der abstrakten Klasse *ForegroundInferenceAlgorithm* erbt, die wiederum von der abstrakten Klasse *InferenceAlgorithm* erbt, welche dies vorschreibt. Die Methode bekommt die Objekte der Objekterkennung gegeben, sowie auch das im Learner erstellte Szenenmodell und berechnet damit die Szenenwahrscheinlichkeit. Das Szenenmodell wird nur genutzt um Objekte zu identifizieren, da sich die Erkennung nur auf die aus der Datenbank ausgelesenen Daten stützt. In der *doInference*-Methode ist der eigentliche Algorithmus mit mehreren *for*-Schleifen und unter Nutzung der anderen Methoden und Variablen der Klasse implementiert.

Die *getProbability*-Methode gibt die aktuelle Wahrscheinlichkeit aus, die in der Klasse eingespeichert ist. Auf diese wird zugegriffen, wenn die Ausgabe des PSM-Systems die Wahrscheinlichkeit der verschiedenen Szenen abfragt um diese in Relation zu setzen und auszugeben. Auch diese Methode muss in der Klasse Implemetiert sein, da sie von abstrakten Klassen geerbt wird.

Die Methode *differenceBetween* implementiert berechnung der Objektwahrscheinlichkeit, die in 4.3.1 beschrieben wird. Es werden vier Objekte übergeben und erst sowohl die Positions- als auch die Rotationsrelation zwschen dem ersten und zweiten Paar berechnet und anschließend die Relationen verglichen. Bei diesem vergleich bekommt man eine gewisse Differenz, welche mit Nutzung der Schwellenwerte zu der Objektwahrscheinlichkeit, also einem *double*-Wert, umgeformt werden. Die Methode ist als public aufgeführt, damit man sich auch mit anderen Klassen an dieser bedienen kann, falls man in Zukunft eine differenzbasierte Objektwahrscheinlichkeit benötigt.

Die Variablen *mMaximumDistance* und *mMaximumRotation* sind die Schwellenwerte für Distanz und Rotation, welche zur Wahrscheinlichkeitsbestimmung notwendig sind. Für *mMaximumDistance* ist jeder Wert größer oder gleich 0 zulässig, für *mMaximumRotation* nur Werte von 0 bis 180. Wenn einer der Werte auf 0 gesetzt ist ist die Erkennung nicht auf einen Tolleranzwert von 0 gesetzt, sondern die zum Parameter gehörige Sparte ist bei der Erkennung deaktiviert. Dies hat den Sinn diese Deaktivierung ohne weitere Variablen zu implementieren. Außerdem schafft man in der Realität sowieso nicht, dass exakt die gleiche Position oder Rotation erkannt wird und hätte das Problem, dass bei Schwellenwert 0, der jeweilige Teil der Objektwahrscheinlichkeit nur 0 oder 1 sein könnte. Das ginge gegen jedweden stochastischen Ansatz.

In *mProbility* wird die Wahrscheinlichkeit eingespeichert, sobald sie berechnet ist. Diese wird dann von *getProbability* ausgegeben. Desweiteren wird die eingespeicherte Wahrscheinlichkeit als Vergleichswert genutzt, um die maximale Wahrscheinlichkeit zu finden.

Die Variable *mPattern* speichert die Objekte die aus der Datenbank ausgelesen wurden, um sie im Laufe der Inferenz zu nutzen. Auf *mPattern* wird nur im Konstruktor schreibend zugegriffen und sonst nur lesend, damit die erhaltenen Daten niemals verfälscht werden und das System robust gegenüber Wiederholungen ist. Die Objekte sind in sogenannten Pattern gespeichert, die eine Szene repräsentieren, welche wiederum Sets enthalten. Diese Sets werden dann im Algorithmus jeweils setweise betrachtet und mit den erkannten Objekten verglichen.

Die Variable *mDataBaseName* enthält den Pfad zur Datenbank, der im Konstrukt zur

Abfrage genutzt wird. Das System wurde nur mit absoluten Pfaden genutzt sollte, aber auch mit realtiven Pfaden arbeiten können.

patternName ist der Szenenname der gerade zu testenden Szene. Dieser ist wichtig, um die richtigen Objekte aus der Datenbank zu lesen, damit die Szenenwahrscheinlichkeit der korrekten Szene bestimmt wird.

Die Variable *tableHelper* zeigt auf eine Instanz der Hilfsklasse *Tablehelper*, welche viele Funktionen beinhaltet die bei dem Zugriff auf Datenbanken nützlich sind. Im Konstruktor wird die Funktion *getRecordedPattern* genutzt, welche einen *string* annimmt, der die Bezeichnung für eine Szene ist und die Objekte dieser Szene zurückliefert. Die Objekte sind unteranderem als Sets gespeichert, sodass man setweise über die Objekte iterieren kann.

Die Methode *findObjectOfType* bekommt eine Liste von Objekten und eine eindeutige Identifikation eines Objekts übergeben und gibt das Objekt aus der Liste mit der gegebene Identifikation zurück. Falls kein Objekt der Liste die gegebene Identifikation hat, wird ein NULL Objekt zurückgegeben.

Die Methode *normalizeVector3d* normiert einen gegebene 3D Vektor, das heißt, dieser wird auf Länge 1 gebracht. Die Funktion wurde für verschiedene Berechnungen benutzt, die während des Testens und des Forschens an dem Algorithmus und dem Modus an sich, wird allerdings momentan nicht mehr genutzt und ist veraltet. Der Vollständigkeit halber und falls noch weiter daran geforscht wird und die Funktion eventuell nochmal gebraucht wird, wurde sie in der Klasse behalten.

In Abbildung 5.4 wird der Programmcode innerhalb der *DoInference*-Methode vereinfacht dargestellt. In rot gefärbt sind dabei Kommentare, die den Programmcode erklären. Welche Objekte im Algorithmus wie benannt sind, wird im Kapitel 4.2 erklärt. Es fällt auf, dass der Algorithmus geringfügig vom Konzept abweicht, da die beiden äußeren Schleifen vertauscht sind. Allerdings macht dies für das Ergebnis keinen Unterschied und hat den Sinn, dass man niemals von einem Referenzobjekt ausgeht, welches nicht gemessen wurde, da dies ein unnötiger Schleifendurchlauf wäre.

5.2.2 Einbindung PSM

Da davon abgesehen wird die Parametisierung des Szenenmodells zu nutzen, braucht man den Learner und das Szenenmodell nur noch zu dem Zweck, dass überliefert wird, welche Szenen potentiell erkannt werden sollen. Die einzigen Informationen die der Algorithmus nutzt sind der Name der Szene und Anzahl der enthaltenen Objekte, sowie die Indentifikationen dieser. Das Learner-Modul musste für den neuen Algorithmus also nicht weiter überarbeitet werden sondern wird genauso eingesetzt, wie es auch im restlichen PSM-System eingesetzt wird. Es wird nur eben nicht auf alle möglichen Daten zugegriffen die zur Verfügung gestellt werden. In der Launch-Datei der Erkennung *inference.launch* wurde ein neuer Parameter hinzugefügt um den Pfad der Datenbank anzugeben. Die Datenbank enthält die Referenzinstanzen von jeder zu testenden Szene. Im Projekt gab es schon verschiedene Modi mit denen man die Erkennung laufen lassen konnte, die unter dem Parameter *inference_algorithm* in der Launch-Datei eingestellt

```
mProbability = 0.0; Ausgabewahrscheinlichkeit
double maxProbability = 0.0;    Maximale Wahrscheinlichkeit, die bis jetzt berechnet wurde
for(auto recordedObject : pEvidenceList){  Äußere Schleife über alle Objekte die gefunden wurden
    std::string currentType = recordedObject.type + recordedObject.observedId;    Setzt gemessenes Referenzobjekt für diesen Schleifendurchlauf
    for(ISM::ObjectSetPtr objectSet : mPattern->objectSets){    Schleife über Szeneninstanzen aus der Datenbank
        ISM::Object testReference;  Das Referenzobjekt aus der Datenbank
        for(auto object : objectSet->objects) {         Schleife über alle Objekte in der Szeneninstanz des Schleifendurchlaufs
            std::string objectType = object->type + object->observedId;   Bestimme die eindeutige Identifikation des Objekts
            if(objectType.compare(currentType) == 0)    Überprüfe, ob das Objekt der ID des gemessenen Referenzobjekts entspricht
                testReference = *object;        Falls ja, wird das Referenzobjekt gesetzt
        }
        if(testReference.type.compare("")){   Überprüfe, ob ein Referenzobjekt gesetzt ist - Falls ja, fahre fort
            double testProbability = 0.0;           Szenenwahrscheinlichkeit, die nur lokal vorkommt
            for(auto object : objectSet->objects) {  Schleife über alle Objekte in der Szeneninstanz des Schleifendurchlaufs
                std::string objectType = object->type + object->observedId;     Bestimme die eindeutige ID des Testobjekts

                double innerTestProbability = 0.0;      Setze die Objektwahrscheinlichkeit = 0
                if(objectType.compare(currentType) != 0){       Überprüfe ob es sich nicht um das Referenzobjekt handelt - Falls ja, fahre fort
                ISM::Object innerRecordedObject = findObjectOfType(pEvidenceList, objectType);    Finde das gemessene Testobjekt in der Messung
                if(innerRecordedObject.type.compare("") != 0)Überprüfe ob ein Objekt gefunden wurde - Falls ja, fahre fort
                    innerTestProbability = differenceBetween(recordedObject, innerRecordedObject, testReference, *object);
                }
                testProbability += innerTestProbability;        Addiere die Objektwahrscheinlichkeit auf
            }
            int objectCount = objectSet->objects.size();    Bestimme die Anzahl der Objekte der Szeneninstanz
            if((testProbability + 1.0) / (objectCount * 1.0) > maxProbability) Überprüfe, ob eine größere Szenenwahrscheinlichkeit gefunden wurde
                maxProbability = (testProbability + 1.0) / (objectCount * 1.0);   Falls ja, speichere diese
        }
    }
}
if(maxProbability > mProbability)   Überprüfe, ob eine größere Ausgabewahrscheinlichkeit gefunden wurde
    mProbability = maxProbability;      Falls ja, speichere diese
}
```

Abbildung 5.4: Programmcode des Algorithmus

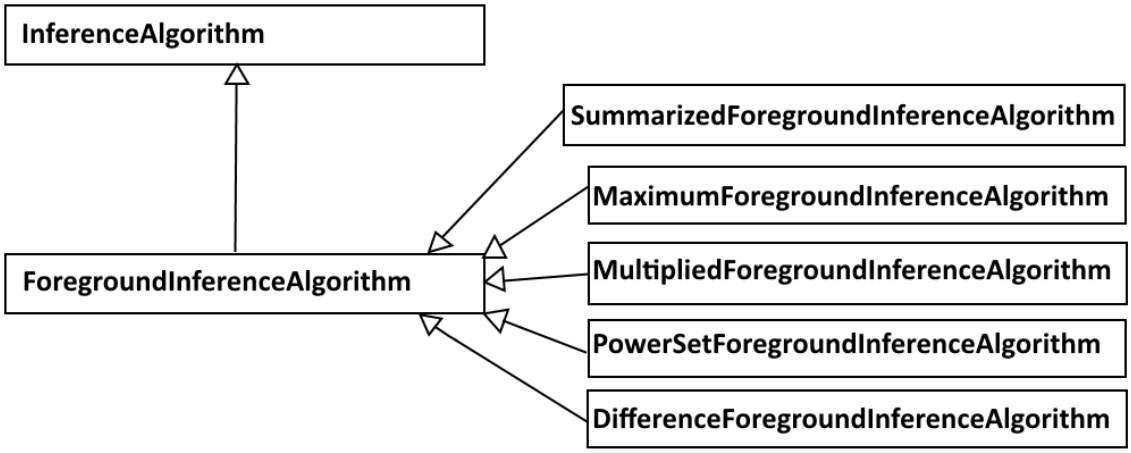

Abbildung 5.5: Klassendiagramm - InferenceAlgorithm-Klasse und Erben

werden konnten. Zu dem neuen Modus, für den man den Parameter auf *difference* setzen muss, gibt es noch die Varianten *powerset*, *summarized*, *multiplied* und *maximum*. Von diesen wurde zuletzt nur noch der *maximum*-Modus genutzt.

Abbildung 5.5 zeigt teilweise die Vererbung der Klasse *InferenceAlgorithm*. Es wurde der Teilbaum der Erben für die Klasse *BackgroundInferenceAlgorithm* weggelassen, da dieser symmetrisch zu der anderen Hälfte der Vererbung ist. Zu Jeder Vordergrundklasse gibt es eine Hintergrundklasse die statt mit *Foreground* mit *Background* gekennzeichnet ist. Um den Algorithmus richtig einzubinden wurde demnach sowohl die Klasse *DifferenceForegroundInferenceAlgorithm* als auch die Klasse *DifferenceBackgroundAlgorithm* hinzugefügt.

6. Evaluation

6.1 Experiment 1: Frühstück

6.2 Experiment 2: Office

Viele Bilder, beschreiben Daten
Text interpretiert
Fazit am Ende

[DSS93]

7. Zusammenfassung und Ausblick

Zwei Sätze zu jedem größeren Kapitel

[DSS93]

Literaturverzeichnis

[DSS93] Randall Davis, Howard Shrobe und Peter Szolovits: *What is a Knowledge Representation?* AI Magazine, 14(1):17–33, 1993. `http://www.aaai.org/ojs/index.php/aimagazine/article/view/1029`.

[Geh14] Joachim Gehrung: *Probabilistische Szenenerkennung durch hierarchische Constellation Models über räumliche Relationen aus demonstrierten Objekttrajektorien.* Seiten 5–63, 2014. `http://www.aaai.org/ojs/index.php/aimagazine/article/view/1029`.