Visualisierung von virtuellen Gedächtnissen in Augmented Reality

Visualization of virtual memories in Augmented Reality

Bachelorarbeit

Klaus Prüger

Prüfer der Bachelorarbeit: 1. Prof. Michael Beetz PhD
2. Prof. Dr.-Ing. Udo Frese

Supervisor Michaela Kümpel & Alina Hawkin

Eidesstattliche Erklärung

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

Bremen, den 17. März 2022

_____

Klaus Prüger

Abstract

Diese Bachelorarbeit befasst sich mit der Visualisierung von Trajektorien in der Welt, aus “imitation learning” Daten. Die Aufbereitung der Daten soll so zu einer neuen Perspektive und neuen Erkenntnissen führen. Die Handlungsabläufe werden bisher nicht im Kontext, also in der Umgebung der Handlung dargestellt. Nach der Design Science Research Methodik vorgehend, wurde eine Prototyp Anwendung für die Hololens geschrieben, um die Funktionalität zu demonstrieren. Anschließend wird der Nutzen evaluiert. In Ansätzen konnte gezeigt werden, dass sich die Perspektive eignet Handlungsstränge zu vergleichen.

Keywords: Trajektorien; KnowRob; Visualisierung; Hololens; Agenten; Bachelorarbeit

Inhaltsverzeichnis

1 Motivation

Ein wiederkehrendes Element im Alltag ist der Erwerb von Verbrauchsgegenständen in Supermärkten und Drogerien. Dieser wird wie viele weitere von der Technologisierung durchdrungen. Der stationäre Handel steht sich mit dem wachsenden Online-Handel einer Konkurrenz ausgesetzt. Das renommierte Bremer Institut "Institut of Artificial Intelligenc" kurz IAI, erforscht seit Jahren an "imitation learning", zu deutsch lernen durch Nachahmung, mit Ihrer Plattform namens "KnowRob" ermöglichen sie Agenten den Austausch von Erfahrungen im bewältigen von alltäglichen Aufgaben, sodass Sie ohne menschliche Einflussnahme Aufgaben erlernen und bewältigen können (Siehe Kapitel 2). Eine Anwendungsdomäne ist neben der Haushalts-/und Küchenhilfe der stationäre Handel. So tragen Sie indirekt zur Forschung der Steigerung der Attraktivität des stationären Handels bei.

Die Bewältigung einer alltäglichen Aufgabe kommt dabei einer Abfolge von Bewegungen gleich. "KnowRo" fasst Bewegungen als Bewegungskurven, Trajektorien auf. Trajektorien sind virtuell, binär oder textuell abgebildet, der Ansatz des "imitation learning" beruht primär auf dem Sehen bzw. der bildlichen Erfassung, jedoch wird die Schnittstelle zur echten Umgebung bisher außer acht gelassen. Mitarbeiter:innen wie Kunden in Drogeriemärkten könnten sich dies zur Nutze machen und sich zu Artikeln führen lassen. Des Weiteren könnten die ausgeführten Aktionen eines Agenten bei der Inventur oder befüllen der Regale anschließend übersichtlich nachvollzogen werden. Daher habe ich mir die Frage gestellt, hilft die Visualisierung der Trajektorien die Qualität der Aufgabenbewältigung der Agenten zu beurteilen? So ergibt sich meine These, die neue Perspektive hilft bei der Einschätzung und Verbesserung des autonomen Handeln. Die Herausforderung besteht in der Schaffung einer Schnittstelle für die verschiedenen Systeme. So wie in der gezielten Verarbeitung und Darstellung der Daten. Aus Gründen der zuvor erwähnten Anwendungsfällen, entschied ich mich für eine praktische Umsetzung, nach der Design Science Research Methodik. Sie ermöglicht theoretische Erkenntnisse in die praktische Lösungsumsetzung, durch die Evaluation mit einfließen zu lassen.Es mündete in einer Anwendung für die Hololens 1, mit der sich die Nutzer:innen Trajektorien darstellen lassen können (Siehe Kapitel 4). Die Entscheidung für Hololens als Endgerät lässt sich mit dessen AR Funktionalität begründen, ohne die Nutzer:innen auf eine Weise einzuschränken, wie es die Bedienung eines Smartphone würde. Die Daten werden per REST Schnittstelle von einem "KnowRob"-Server abgefragt. Die Besonderheit besteht in der Darstellung, unabhängig von der Erstellungs-Hoheit der Daten, es können alle sich auf dem "KnowRobServer befindlichen Trajektorien dargestellt werden. Zu Beginn, vor (und während) der Entwicklung habe ich mir einen Überblick über die aktuellen Ansätze der Forschung verschafft, um eine Idee zu bekommen, wie meine Umsetzung der Visualisierung der Trajektorien aussehen könnte (Siehe Kapitel 3). Die Umsetzungsschritte werden im

gleichnamigen Kapitel 4 “Umsetzung” beschrieben. Das darauf folgende Evaluationskapitel beurteilt die Problem Bewältigung der Umsetzung. Das verfolgte Ziel mit dieser Arbeit ist das laden und darstellen aller sich beliebig auf KnowRob befindenden Trajektorien.

2 NEEM - Episodische Erinnerung

Ein NEEM repräsentiert eine episodische Erinnerung eines Agenten.

2.1 NEEM Einführung

Narrative-enabled episodic memories kurz NEEMs sind von Agenten gesammelte Erfahrungen. Die Erfahrungen werden episodenweise gespeichert und aus diesen können Rückschlüsse für Aktionen gezogen werden, der Agent lernt. Ein Agent sei an dieser Stelle kurz erläutert, ein KI gestützter Roboter der wissensbasiert und autonom handelt.

NEEMs stellen den "geistigen" Teil des generalisierten Ansatzes dar, zu ermöglichen, dass Agenten selbständig diverse Aufgaben erfüllen können. Das Ziel ist ein fach übergreifendes Vokabular für beliebige Aktionen zu werden, diese zu beschreiben und in Beziehung zu setzen (Vgl. [1]). Sie sind den menschlichen Erinnerungen nachempfunden, sprich aus ihnen lassen sich (wie der Mensch es tut), vergangene Ereignisse aus den markanten Punkten rekonstruieren (Vgl. [1]). Erzeugt werden Sie auf zweierlei Wege. Erstens, der Agent versucht eine Aufgabe zu lösen bzw. ein Ziel zu erreichen und seine Versuche werden als NEEMs aufgezeichnet, die klsssische "Try and Error" Strategie. Zweitens, der Agent lernt durch Nachahmung des Menschen, da menschliche Bewegungen jedoch zu komplex sind und des Weiteren viele Details implizit dem Agenten verborgen bleiben ist eine einfache Aktivitätsaufzeichnung nicht zielführend. Für die Nachahmung zur Datengewinnung eignet sich hervorragend die virtuelle Realität kurz VR, in VR tritt der Benutzer mit Hilfe einer Brille in eine vom Computer simulierte Welt ein, da sich die Nachbildung von gewünschten naturgetreuen Umgebungen als unkompliziert herausstellte. Des Weiteren ermöglicht es eine gute Bewegungsaufzeichnung, wobei die Bewegungen weiterhin natürlich für den Menschen bleibt, es findet also keine Verzerrung statt. Einher geht mit der einfachen Erstellung von Szenarien auch ein unkomplizierter Wechsel der Szenarien.

Ein grober Ablauf der Datengewinnung sieht folgendermaßen aus: Erstens, die auszuführende Aktion wird formuliert. Zweitens, die Aktion wird von einem Menschen in VR ausgeführt. Drittens, die Aktion wird geloggt und in Folge auf den Agenten umgerechnet. Der Agent erhält eine grobe Ablaufbeschreibung, die er mit dem vermeintlich passenden VR Wissen füllt, "Try and Error" erfolgt (Vgl. [3]).

2.2 NEEM Aufbau

Ein NEEM besteht aus einem NEEM experience und einem NEEM narrative (NEEM = NEEM experience + NEEM narrative).
NEEM experience beinhaltet die Sensordaten, die der Agent beim Ausführen einer Aktion wahrnimmt, wie seine Position und Bilder durch Kameras. Diese Daten werden mit dem NEEM narrative verbunden, der diese in den Kontext setzt, sie werden einem Ziel und den aufgetretenen Effekten zugeordnet (Vgl. [1]).

Jedes NEEM hat genau ein NEEM-background ansässig, dieser listet alle physischen Objekte auf die bei den Aktionen beteiligt waren, wie der Ort des Geschehens, die Agenten und weitere Objekte. Ebenfalls ist die Beziehung der Objekte relevant für die Rekonstruktion der Aktionen und wird vermerkt (Vgl. [1]).

Gebündelt werden die NEEMs im NEEM-hub, von ihm aus können Agenten verbesserte Modelle laden, um Aufgaben besser bewältigen zu können.

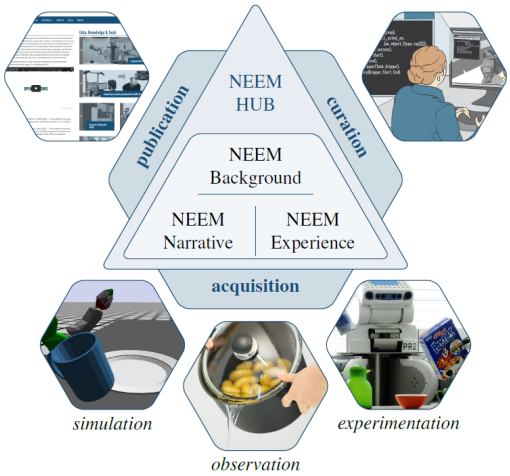

Abbildung 2.1: NEEM Aufbau Überblick [1]

2.2.1 Beispiel Logging

Zur Vertiefung folgt ein konkretes Beispiel.

Aktion: Ein Benutzer vermisst den Raum mit seiner Hololens.

Ein NEEM Logging Eintrag besteht jeweils aus einem Subjekt, Prädikat und Objekt. Ergänzt werden kann es um ein graph oder einer Start- und Endzeit (since und until), die angibt, wie lange die Teilaktion lief. Ein Subjekt ist entweder mit einem Objekt verknüpft

oder unterhält Daten in einem der folgenden Formate: string, boolean oder number. Ein Objekt zeigt auf eine Instanz. Das Prädikat loggt die ausgeführte Methode der Instanz oder eine seiner Eigenschaften.

Geloggt sieht eine Erinnerung folgendermaßen aus. Die Episode wird angelegt und dieser wiederum einer Aktion zugewiesen (Siehe Abbildung 2.2). Die Aktion bündelt alle Aktionen, die in dieser Episode ausgeführt werden. In Folge wird die zur Episode hinzugefügte Aktion angelegt (Siehe Abbildung 2.3). Neben dem Interesse welche Aktionen ausgeführt werden ist auch entscheidend, wer diese ausführt, daher wird der Akteur ebenfalls der Aktion zugewiesen (Siehe Abbildung 2.4) und angelegt (Siehe Abbildung 2.5). Von Bedeutung ist ebenfalls, die Zeit die eine Aktion in Anspruch nimmt, das Zeitintervall vermerkt die Start- und Endzeit (Siehe Abbildung 2.6). Abgeschlossen wird diese Episode mit einer einzigen Aktion, dem Vermessen des Raumes (Siehe Abbildung 2.7 und 2.8).

```
//Episode (1)
{
    "s": "http://www.ease-crc.org/ont/SOMA.owl#Episode_PVDQFA", //Subject
    "p": "http://www.w3.org/1999/02/22-rdf-syntax-ns#type", //Predicate
    "o": "http://www.ease-crc.org/ont/SOMA.owl#Episode", //Object
    "graph": "user"
}
{
    "s": "http://www.ease-crc.org/ont/SOMA.owl#Episode_PVDQFA",
    "p": "http://www.w3.org/1999/02/22-rdf-syntax-ns#type",
    "o": "http://www.w3.org/2002/07/owl#NamedIndividual",
    "graph": "user"
}
//Add new action to episode
{
    "s": "http://www.ease-crc.org/ont/SOMA.owl#Episode_PVDQFA",
    "p": "http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#includesAction",
    "o": "http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#Action_PVLMGF",
    "since": 0,
    "until": 0
},
```

Abbildung 2.2: Episode erzeugen und Aktion hinzufügen

```
//Create action
{
    "s": "http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#Action_PVLMGF",
    "p": "http://www.w3.org/1999/02/22-rdf-syntax-ns#type",
    "o": "http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#Action"
},
{
    "s": "http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#Action_PVLMGF",
    "p": "http://www.w3.org/1999/02/22-rdf-syntax-ns#type",
    "o": "http://www.w3.org/2002/07/owl#NamedIndividual"
},
```

Abbildung 2.3: Aktion anlegen

```
//Add user to current action
{
    "s": "http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#Action_PVLMGF",
    "p": "http://www.ease-crc.org/ont/SOMA.owl#isPerformedBy",
    "o": "http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#NaturalPerson_QRSLZF"
},
```

Abbildung 2.4: Akteur wird der Aktion zugewiesen

```
//Create user for current user
{
    "s": "http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#NaturalPerson_QRSLZF",
    "p": "http://www.w3.org/1999/02/22-rdf-syntax-ns#type",
    "o": "http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#NaturalPerson"
},
{
    "s": "http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#NaturalPerson_QRSLZF",
    "p": "http://www.w3.org/1999/02/22-rdf-syntax-ns#type",
    "o": "http://www.w3.org/2002/07/owl#NamedIndividual"
},
```

Abbildung 2.5: Benutzer anlegen

```
//Time intervals are assigned to actions
{
    "s": "http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#TimeInterval_RYAJLE",
    "p": "http://www.w3.org/1999/02/22-rdf-syntax-ns#type",
    "o": "http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#TimeInterval"
},
{
    "s": "http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#TimeInterval_RYAJLE",
    "p": "http://www.w3.org/1999/02/22-rdf-syntax-ns#type",
    "o": "http://www.w3.org/2002/07/owl#NamedIndividual"
},
{
    "s": "http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#TimeInterval_RYAJLE",
    "p": "http://www.ease-crc.org/ont/SOMA.owl#hasIntervalBegin",
    "o": {
        "$numberDecimal": "0"
    }
},
{
    "s": "http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#TimeInterval_RYAJLE",
    "p": "http://www.ease-crc.org/ont/SOMA.owl#hasIntervalEnd",
    "o": {
        "$numberDecimal": "0"
    }
},
{
    "s": "http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#Action_PVLMGF",
    "p": "http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#hasTimeInterval",
    "o": "http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#TimeInterval_RYAJLE"
},
```

Abbildung 2.6: Zeitintervall der Aktion zuweisen und Start- und Endzeit setzen

```
//Add areaServeying task to the current action
{
    "s": "http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#Action_PVLMGF",
    "p": "http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#executesTask",
    "o": "http://www.ease-crc.org/ont/SOMA.owl#AreaSurveying_BWLKCN",
    "since": 0,
    "until": 0
},
```

Abbildung 2.7: Vermessen den Aktionen zuweisen

```
//Define areaServeying task
{
    "s": "http://www.ease-crc.org/ont/SOMA.owl#AreaSurveying_BWLKCN",
    "p": "http://www.w3.org/1999/02/22-rdf-syntax-ns#type",
    "o": "http://www.w3.org/2002/07/owl#NamedIndividual"
},
{
    "s": "http://www.ease-crc.org/ont/SOMA.owl#AreaSurveying_BWLKCN",
    "p": "http://www.w3.org/1999/02/22-rdf-syntax-ns#type",
    "o": "http://www.ease-crc.org/ont/SOMA.owl#AreaSurveying"
},
```

Abbildung 2.8: Vermessungsaktion anlegen

3 Visualisierung der Trajektorien

Trajektorien sind Bewegungskurven.

3.1 Trajektorien

Als Trajektorien werden Bewegungskurven bezeichnet. In diesem Kontext sind es vom Menschen oder Agenten ausgeführte Aktionen, die zu einer Bewegung und damit Bewegungskurve führen. NEEMs ermöglicht eine Rekonstruktion dieser.

Bevor ich auf schon umgesetzte Visualisierungen von Trajektorien eingehe, folgen vorweg Gründe, die für eine Visualisierung sprechen.

3.1.1 Vorteile der Visualisierung

Einige Gründe sprechen für eine Visualisierung der Trajektorien. Die Visualisierung unterstützt den Ansatz des lernen durch Demonstration, so ermöglicht sie auf eine intuitive Weise für Menschen dem Agenten etwas vorzumachen, auch ohne Programmierkenntnisse (vgl. [4]). Dabei ist man nicht in einer virtuellen Welt, sondern in der natürlichen Umgebung, zusammen mit dem Agenten. So können sie direkt aufeinander Einfluss nehmen, denkbar wäre das bearbeiten der Trajektorien (vgl. [4]). Des Weiteren steigt die Nachvollziehbarkeit der Abläufe der Aktionen für die Menschen, da die Trajektorien im Raum erscheinen. VR wird somit von AR abgelöst und einher geht damit eine Zeitersparnis, da die Modellierung der Szenarien weg fällt.

In dem vorherigen Paragraph wurden viele Vorteile in Bezug auf die Demonstration genannt, jedoch darf die wegfallende Abstraktion bei der Beurteilung des Lernfortschrittes vom Agenten durch die Visualisierung nicht unerwähnt bleiben. So lässt sich eine Aktion vor und nach der Erweiterung des Wissensmodelles eines Agenten ausführen und beide Male als NEEM geloggt. Nun wären die Trajektorien darstell und vergleichbar.

Der Verlust des schnellen Szenariowechels soll an dieser Stelle nicht unerwähnt bleiben, jedoch schließt die verstärkte Nutzung von AR eine parallele Nutzung von VR nicht aus. Besonders im Hinblick auf die Zukunft, mit sich stetig verbessernder Technik und die damit einhergehende effizientere Modellierung.

3.1.2 Visualisierung der Trajektorien

In Folge werden mögliche Visualisierungen skizziert.

AR

Eine erste Idee der Visualisierung von Trajektorien in AR erhält man aus den Ergebnissen der Universität von Colorado, die sie im Paper "Augmented Reality Interface for Constrainted Learning from Demonstartion" festhalten. Sie benennen einige der zuvor genannten Vorteile (vgl. [4]). Eine Visualisierung der Trajektorien in AR erreichen Sie, indem Zeitpunkte auf den Kurven durch Kugeln dargestellt werden (vgl. [4]). Die Zeitpunkte können sogenannten "constraints" (Bedingungen) unterliegen, die erfüllt sein müssen für einen korrekten Ablauf (vgl. [4]). Eine Bedingung wäre die Begrenzung der Rotationsrichtung, sie ist jeweils farblich hervorgehoben und Verstöße gegen die Bedingung ebenfalls, hier zu sehen (erste Abbildung von Links) 3.1 (vgl. [4]).

Neben der Visualisierung sehen Sie auch eine Interaktion mit den Trajektorien vor, so können Zeitpunkte verschoben, um Bedingungen ergänzt oder gelöscht werden (vgl. [4]). Ein ausgewählter Zeitpunkt ist ebenfalls farblich abgegrenzt (Siehe Abbildung zweite von rechts 3.1) (vgl. [4]).

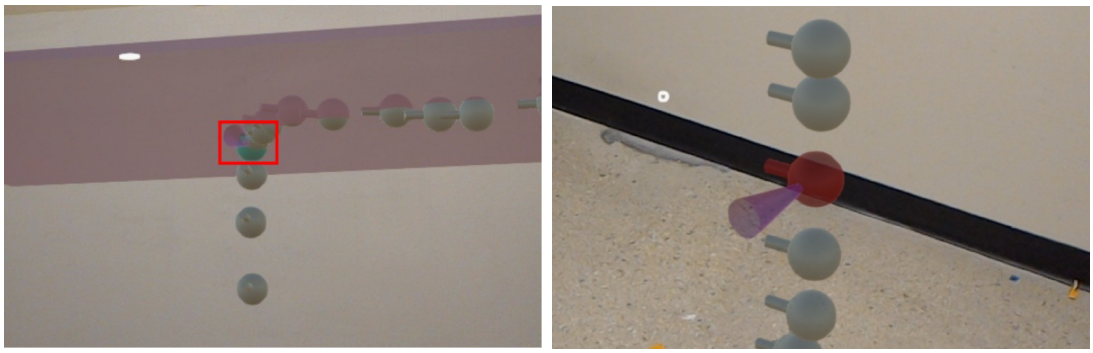

Abbildung 3.1: Ausgewählter Zeitpunkt: Dunkelgrüne Kugel im roten Kasten (Linke Grafik)
Rotationsbedingung: Rote Kugel (Rechte Grafik) [4]

VR

Neben den Überlegungen, wie die Visualisierung von Trajektorien in AR möglich wäre, bietet VR eine bereits etablierte Inspirationsquelle, aus dem potenzielle Abstraktionen möglich scheinen. Andrei Haidu und Michael Beetz präsentieren Ihre Ergebnisse im Paper "Automated acquisition of structured semantic models of manipulation activities from human VR demonstration". Sie verwenden NEEMs und rekonstruieren die Episoden in den virtuellen Welten. Aktionen werden farblich unterschieden, die Trajektorien unterhalten pro Zeitpunkt einen farblichen Kreis (Siehe Abbildung 3.2) (vgl. [2]).

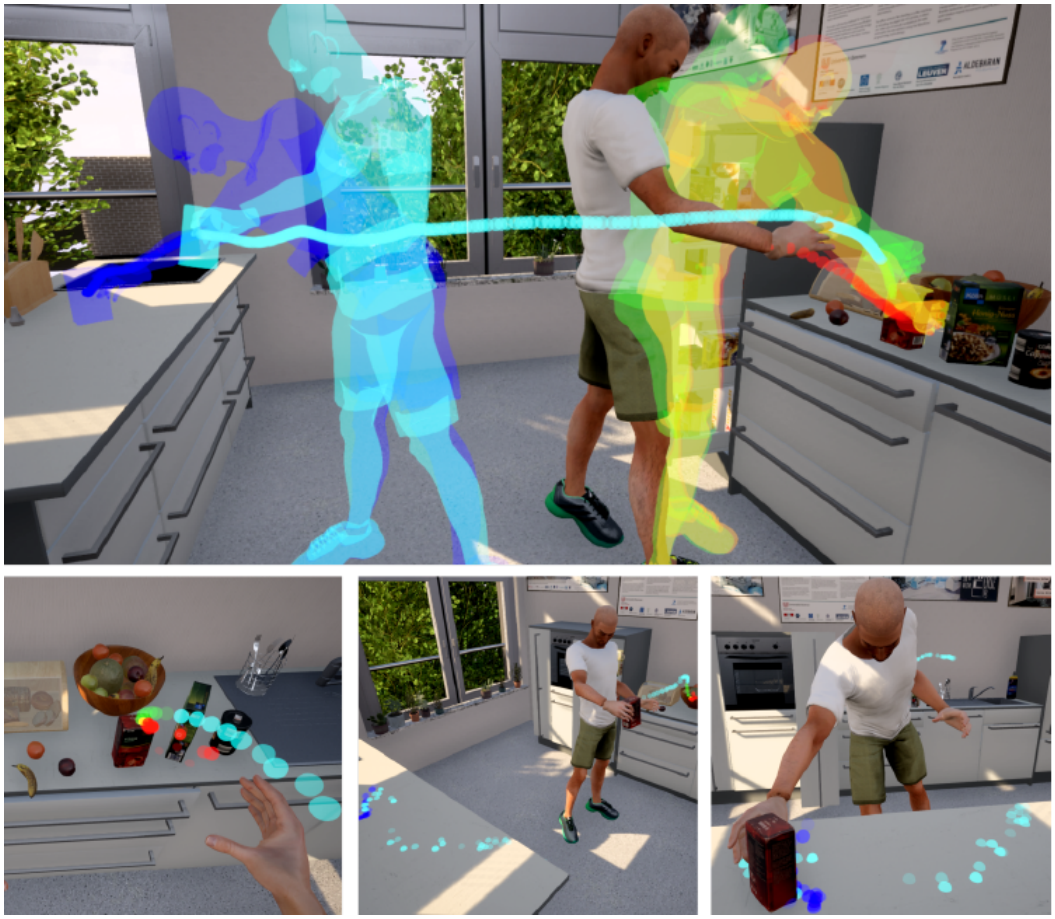

Abbildung 3.2: Objekt greifen und neu platzieren Aktionen visualisiert [2]

4 Umsetzung

Das kommende Kapitel stellt die Ergebnisse der Umsetzung der Anwendung dar, so wie dessen Erschaffungsprozess. Das Ziel war eine Anwendung für die Hololens, die in KnowRob gespeicherte Trajektorien separiert nach Aktionen darstellen kann.

4.1 Anwendung

Die Anwendung begrüßt den/die Nutzer:in mit einem vor sich schwebenden Menü, dieses unterhält sechs Knöpfe (Siehe Abbildung 4.9). Die GUI Elemente entstammen dem MRTK. Betätigung von Knopf eins löscht alle bis zu diesem Zeitpunkt erzeugten Trajektorien. Knopf zwei bis sechs unterhalten jeweils ein Szenario, diese zeigen die Trajektorien der Tätigkeiten eines gespeicherten Objektes (Siehe Abbildung 4.9). In Nachfolgenden wird genauer auf die fünf Szenarien eingegangen. Bevor eine Erzeugung von Trajektorien möglich ist, muss eine Positionskalibrierung stattfinden. Diese ist notwendig, damit die Trajektorien verlässlich an derselben Position erzeugt werden. Der/Die Nutzer:in muss dafür einmalig den QR-Code anschauen und bei Erkennung erscheint eine Pille an dessen Position, die über dem QR-Code schwebt (Siehe Abbildung 4.10). Damit wurde der World Anchor erzeugt. Der QR-Code ist zur Erhaltung des World Anchor nicht notwendig und kann aus dem Sichtfeld genommen werden. Eine Löschung des World Anchors wird durch das Ausführen der “air tap Geste” erreicht, bei dessen Ausführung darf kein virtuell erzeugtes Objekt anvisiert werden (Siehe Abbildung 4.11). Alle bis dahin erzeugten Trajektorien werden ebenfalls gelöscht. Eine erneute Erzeugung des World Anchors wird durch die erneute Anvisierung des QR-Codes erreicht. Die Anvisierung eines Knopfes, gefolgt von der “air tap” Geste betätigt den Knopf (Siehe Abbildung 4.12).

Es folgt die Vorstellung fünf exemplarischer Anwendungsfälle. Diese sollen einen breiten Eindruck über die Funktionsweise bieten. Das erste Szenario wird mit allen technischen Details dargestellt, alle weiteren Szenarien funktionieren analog, sie unterscheiden sich nur auf inhaltlicher Ebene in der Parametereingabe und in dessen Ergebnissen.

4.1.1 Szenario Eins

Parameterliste:

physicalObject:
"http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#PhysicalObject_PHONEA"

Query

```
is_action(A), has_time_interval(A, TimeInterval),
model_SOMA_EXT:has_interval_data(TimeInterval, Since, Until), (atom(Since) ->
atom_number(Since, S) ; S=Since ), (atom(Until) -> atom_number(Until, U) ; U=Until ),
Since1 is round(S), Until1 is round(U),
tf_plugin:tf_get_trajectory('http://www.ontologydesignpatterns.org/ont/dul
/DUL.owl#PhysicalObject_PHONEA', Since1, Until1, Trajectory).
```

Abbildung 4.1: Verwendete Abfrage im Szenario Eins

maxSolutionCount: 500

multiNum: 10

Ein Szenario basiert auf einem Agenten. Ein Agent ist ein etwas, was die Handlungen (Aktionen) vollziehen kann und dessen Bewegungen KnowRob konform aufzeichnet. In diesem Fall ist es ein Smartphone, das von einem Mensch durch die Regale bewegt wurde. Ebenso kann es ein Mensch sein, der Handlungen in VR vollzieht oder ein autonom handelnder Roboter. Der Agent wird in den Daten als "physikalisches/existierendes Objekt hinterlegt, er ist in der Parameterliste als "physicalObjectäufgeführt.

Dieses Szenario visualisiert alle ausgeführten Handlungen (Aktionen) des Agenten. Die ausgeführten Handlungen (Aktionen) werden bei dessen Darstellung farblich unterschieden (Siehe 4.2). Alle Punkte einer Bewegungskurve unterhalten dieselbe Farbe. Trajektorien überlagern sich teilweise oder ganz (Siehe 4.3).

Die maximal mögliche Anzahl an Aktionen die zurückgegeben werden könnte, beträgt 500. Sie wird durch den Parameter *maxSolutionCount* bestimmt. Wie viele Punkte einer Trajektorie visualisiert werden, hängt von der multiNum Variable ab. In diesem Fall wird jeder zehnte Punkt der Bewegungskurven erzeugt.

Abbildung 4.2: Szenario Eins

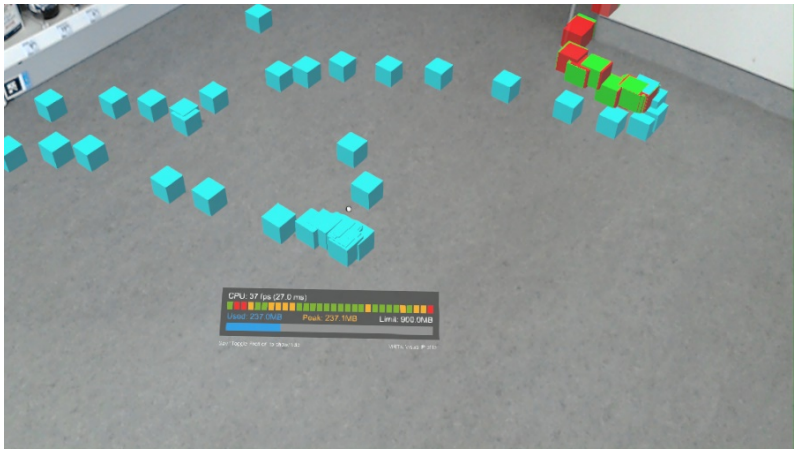

Abbildung 4.3: Überlagerung von Trajektorien

4.1.2 Szenario Zwei

Parameterliste:

physicalObject:
"http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#PhysicalObject_PHONEA"

Query

```
has_type(Tsk, soma:'AreaSurveying'), executes_task(A, Tsk), is_action(A),
has_time_interval(A, TimeInterval), model_SOMA_EXT:has_interval_data(TimeInterval,
Since, Until), (atom(Since) -> atom_number(Since, S) ; S=Since ), (atom(Until) ->
atom_number(Until, U) ; U=Until ), Since1 is round(S), Until1 is round(U),
tf_plugin:tf_get_trajectory('http://www.ontologydesignpatterns.org/ont/dul
/DUL.owl#PhysicalObject_PHONEA', Since1, Until1, Trajectory).
```

Abbildung 4.4: Verwendete Abfrage im Szenario Zwei

maxSolutionCount: 500

multiNum: 10

Tsk: 'AreaSurveying'

Szenario Zwei basiert auf denselben Daten, wie zuvor Szenario Eins. Es grenzt die Abfrage auf die Tätigkeit der Umgebungs-Begutachtung ein, genannt 'AreaSurveying'. Alle Trajektorien der Aktionen, dessen Tätigkeit als 'AreaSurveying' bezeichnet sind, werden dargestellt (Siehe Abbildung 4.5).

Abbildung 4.5: Szenario Zwei

4.1.3 Szenario Drei, Vier & Fünf

Parameterliste:

physicalObject:
"Product_8BiYIs" //Szenario Drei
"Product_PePr1F" //Szenario Vier
"Product_Qko7qe" //SZenario Fünf

Queries

```
has_type(Tsk, soma:'Grasping'), executes_task(A, Tsk), is_action(A),
has_time_interval(A, TimeInterval), model_SOMA_EXT:has_interval_data(TimeInterval,
Since, Until), (atom(Since) -> atom_number(Since, S) ; S=Since ), (atom(Until) ->
atom_number(Until, U) ; U=Until ), Since1 is round(S), Until1 is round(U),
tf_plugin:tf_get_trajectory('http://knowrob.org/kb/shop.owl#Product_8BiYIs', Since1,
Until1, Trajectory).
```

Abbildung 4.6: Verwendete Abfrage im Szenario Drei

```
has_type(Tsk, soma:'Grasping'), executes_task(A, Tsk), is_action(A),
has_time_interval(A, TimeInterval), model_SOMA_EXT:has_interval_data(TimeInterval,
Since, Until), (atom(Since) -> atom_number(Since, S) ; S=Since ), (atom(Until) ->
atom_number(Until, U) ; U=Until ), Since1 is round(S), Until1 is round(U),
tf_plugin:tf_get_trajectory('http://knowrob.org/kb/shop.owl#Product_PePr1F', Since1,
Until1, Trajectory).
```

Abbildung 4.7: Verwendete Abfrage im Szenario Fünf

```
has_type(Tsk, soma:'Grasping'), executes_task(A, Tsk), is_action(A),
has_time_interval(A, TimeInterval), model_SOMA_EXT:has_interval_data(TimeInterval,
Since, Until), (atom(Since) -> atom_number(Since, S) ; S=Since ), (atom(Until) ->
atom_number(Until, U) ; U=Until ), Since1 is round(S), Until1 is round(U),
tf_plugin:tf_get_trajectory('http://knowrob.org/kb/shop.owl#Product_Qko7qe', Since1,
Until1, Trajectory).
```

Abbildung 4.8: Verwendete Abfrage im Szenario Fünf

maxSolutionCount: 500

multiNum: 10

Tsk: 'Grasping'

Szenario Drei, Vier und Fünf basieren aus VR NEEMs. Weiteres folgt in Kapitel 5. Die Abfrage erbringt Informationen über die Position, an dem ein Produkt gegriffen wird. Bei den drei Szenarien handelt es sich jeweils um ein Produkt. Die ausgeführte Handlung ist greifen, 'Grasping'. Darstellen lässt sich ein Trajektorien-Punkt.

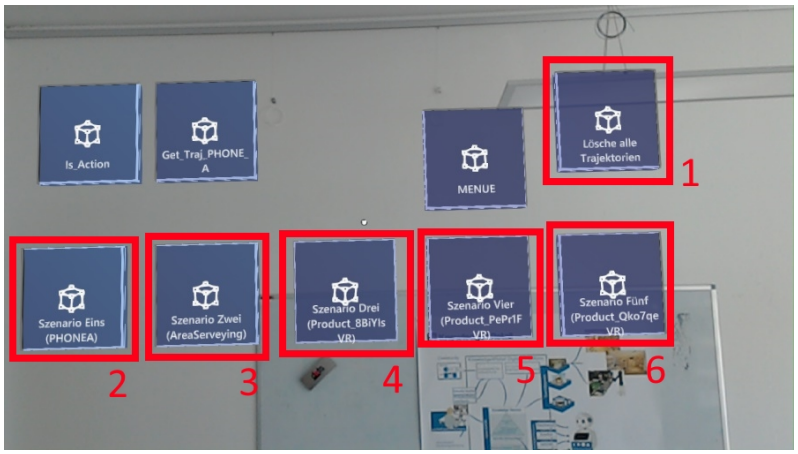

Abbildung 4.9: App Menü Funktionsübersicht

Abbildung 4.10: World Anchor setzen

Abbildung 4.11: World Anchor löschen

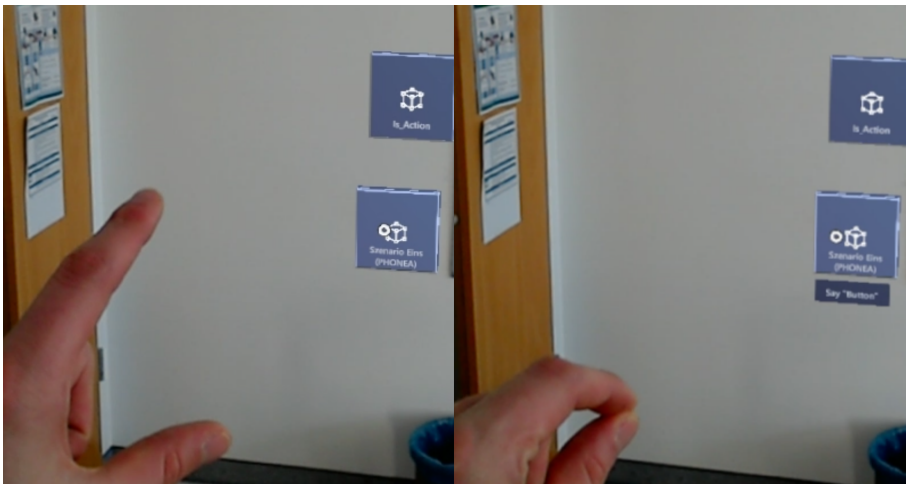

Abbildung 4.12: Knopf betätigen

4.2 Architektur

Die Architektur hat sich über die Entwicklungszeit verfeinert, sie wurde durch die Wahl der Technologien und des Refactorings geformt. In Folge schlüssel ich die Entwicklung der Software anhand der Architektur auf. Konzeptionell folgt das Projekt der Client-Server-Architektur. Der/Die Nutzer:in stellt Abfragen per Hololens Anwendung an den "KnowRo"-Server und dieser antwortet mit einem Ergebnis (Siehe Abbildung 4.13).

4.2.1 Client

Die verwendete Hardware nahm eine Vielzahl von architektonischen Entscheidungen der Anwendung vorweg. Wie bereits in der Motivation erwähnt, hielt ich die Hololens als geeignete Hardware für mein Vorhaben, da diese auf dem Kopf der Nutzer:innen getragen wird und die Hände für Interaktionen in der Welt frei bleiben. Denkbar sind Führungen zu Produkten und das Tragen eines Korbes, so wie das Halten der Hand eines Kindes.
Zur Entwicklung der Anwendung bot sich die Unity Engine an, sie verfügt über die Toolkit Erweiterung MRTK, die die Hololens unterstützt und ich konnte auf erste praktische Erfahrungen durch mein Bachelor-Projekt bauen. Nach dem Aufsetzen des Projektes sah ich mich der ersten Herausforderung gegenüber, die Datenverarbeitung. Nach dem vertraut machen mit den Daten, strebte ich eben dessen Verarbeitung an, da sie die Position und weitere Eigenschaften der zu erzeugenden Trajektorien unterhielten. Ich sah mich mit der Zusammenführung der tf und triples Daten konfrontiert, es benötigte ein Vergleich der Zeitfenster (timestamps) der beiden Dateien, um die aufgezeichneten Aktionen den Trajektorien Positionen zuordnen zu können. Dies erwies sich als sehr rechenintensiv, die die mobile Hardware, auch in Anbetracht des Akkubetriebs nicht leisten konnte. Nach Rücksprache erfuhr ich, dass "KnowRob" diesen Verarbeitungsschritt anbietet und folglich lagerte ich die Vorverarbeitung der Daten aus, es formte sich die Client-Server-Architektur (Weitere Details des Servers folgen in Abschnitt 4.2.3).
In Folge teilt sich die Funktionalität der Anwendung grob in zwei Bereiche, die Datenabfrage und die Erzeugung der Trajektorien. Das Ziel war es, den Aufruf der Abfrage so modular und generisch wie möglich zu halten, um eine nachvollziehbare Struktur zu schaffen, die um weitere Abfragen ergänzt werden kann. In Anlehnung an die Schichtenarchitektur erstreckt sich die Abfrage über drei vertikale Ebenen (Siehe Abbildung 4.14). Der Nutzer-Abfrage bzw. Parameter-Generierung. Die Businesslogik, sie bündelt das CommandHandler- und Data-Skript, also unterhält die Logik. Der CommandHandler dient als Abstraktion der Abfrage und koordiniert Ein- und Ausgaben. Das Data-Skript kapselt die Erzeugung der Trajektorien. Die unterste Ebene stellt die Schnittstelle zum Server dar.

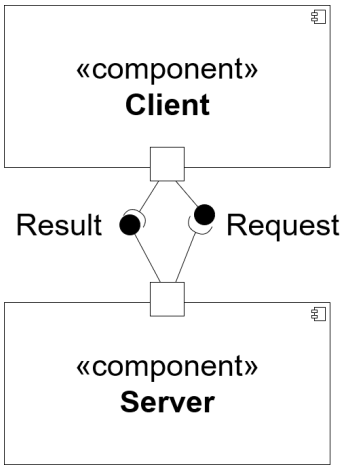

Abbildung 4.13: Konzeptionelle-Sicht Client-Server-Architektur

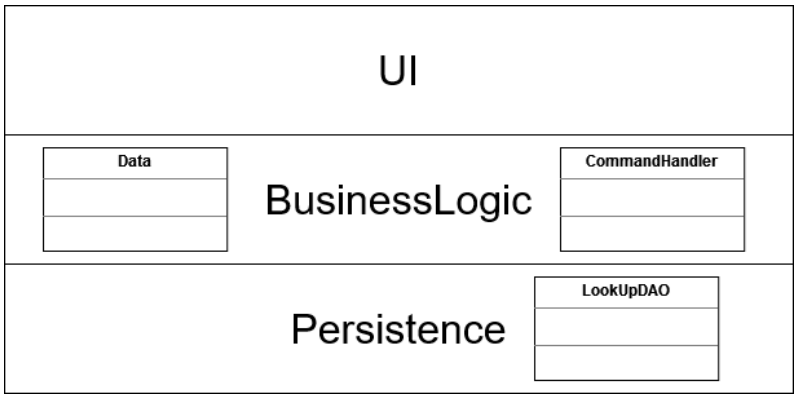

Abbildung 4.14: Schichtenarchitektur des Clients

Abfrage

Die Zusammenhänge sollen durch ein Sequenzdiagramm verdeutlicht werden (Siehe Abbildung 4.15, 4.16 und 4.17). Es ist auch online im Repository Ordner "Dokumentation" als ganzes abzurufen oder direkt unter [1]. Konkret wird erneut auf das Szenario Eins eingegangen, der gezeigte Ablauf ist dabei stellvertretend für alle Szenarien. Der/Die Nutzer:in interagiert mit der GUI, Er/Sie führt die "air tap" Geste aus, ein Avisieren des Knopfes vorausgesetzt. Der Knopfdruck lässt die Main Instanz die Parameter (Bezeichnung des Objektes), die er unterhält, an die CommandHandler Instanz weitergeben. Die Abfrage wird vom der CommandHandler Instanz an die Schnittstelle weitergegeben. Die LookupDAO Instanz ergänzt die Prolog Query um die Parameter und schickt die (REST) POST Abfrage an den Server (Details zur Schnittstelle in Sektion 4.2.4). Bei der Abfrage gilt

[1] `https://gitlab.informatik.uni-bremen.de/kprueger/bachelorarbeit/-/raw/master/Dokumentation/anwendungsfall_1_sequenzdiagramm_finished.drawio_mark.png?raw=true`

es drei Fälle zu unterscheiden, erstens den Erfolgsfall, die LookupDAO Instanz erhält eine Rückgabe. Diese übergibt die LookupDAO Instanz an den CommandHandler, dieser wiederum an die Data Instanz. Zweitens, ein Netzwerkfehler tritt auf, der Server ist nicht erreichbar, die Abfrage wird mit einer Exception abgebrochen. Drittens, ein interner Server Fehler tritt auf, der entweder in keiner Rückgabe mündet und eine Exception auslöst, oder eine fehlerhafte Rückgabe wird weitergegeben. Im Folgenden Abschnitt kommen wir auf Verarbeitung der Rückgabe zu sprechen.

Erzeugung der Trajektorien

Die Erzeugung der Trajektorien findet gekapselt im "Data-Skript" statt. Abhängig von den übergebenden Daten werden folgende Fälle unterschieden: Erstens die Daten weisen Trajektorien Koordinaten auf, zweitens die Daten weißen keine Trajektorien Koordinaten auf, drittens für das Objekt gibt es keine Rückgabe, viertens ein Eingabefehler der Abfrage liegt vor oder fünftens ein interner Serverfehler ist vorhanden.
Fall Eins stellt den Erfolgsfall dar, die Daten im JSON Format werden mit Hilfe der JSON.NET Bibliothek verarbeitet. Die JSON Rückgabe des Servers wies eine eigenwillige Struktur auf, JSON.NET bot sich als geeignetes Werkzeug für die Extraktion an, da es über die Angabe des Pfades ermöglicht, Daten aus individuelle strukturierten JSON Dateien zu entziehen (Siehe Abbildung 5.11). Die Positionsdaten (Koordinaten) der jeweilig aufgezeichneten Bewegungen werden rekursiv extrahiert. Anschließend wird für jedes Koordinaten Tripel ein Trajektorien-Objekt erzeugt (Siehe Datenmodell 4.37). Gleichermaßen erhalten die Trajektorien eine Einfärbung, je nach Zugehörigkeit zu dessen aufgezeichneter Aktion.
Fall Zwei stellt kein Misserfolg dar, jedoch bietet dieses Szenario keine verwertbaren Daten. Da keine Positionsdaten zu extrahieren sind, landet die Trajektorien Erzeugung unmittelbar in der Abbruchbedingung.
Fall Drei, der Server kann für das abgefragte Objekt in diesem Kontext keine Rückgabe generieren und antwortet mit "no solution found". Die Verarbeitung erzeugt eine Exception.
Fall Vier ist ein Fehlerfall, dieser kann unter den aktuellen Bedingungen jedoch nicht eintreten, da dem/der Nutzer:in nur begrenzt Zugriff auf die Eingabe gewährt wird, die eingerichteten Szenarien können betrachtet werden. Mit Erweiterungen wie Spracheingabe oder eines Textfeldes, worüber der/die Nutzer:in eigene Szenarien abfragen kann, sind fehlerhafte Nutzereingaben ein zu berücksichtigender Fall. Der Server kann die Query nicht auswerten und schickt die Fehlernachricht "bad query", die Verarbeitung mündet in einer Exception.
Fall Fünf, ein interne Serverfehler erzeugt eine fehlerhafte Rückgabe. Wie im Fall Drei mündet die Verarbeitung in einer Exception.
Kommen wir abschließend nochmal auf den Erfolgsfall zu sprechen, es wird eine Anfrage an den Server durch gereicht, dieser antwortet und die Trajektorien können erzeugt werden, sie erscheinen vor dem/der Nutzer:in. Wie erwünscht, sie können fortan betrachtet und nachvollzogen werden, jedoch ein Element fehlt, der Kontext. Die Trajektorien werden nach diesem Stand immer in Abhängigkeit des/der Nutzers:in erzeugt, die In-

formation des Ortes des Geschehens schlüsselt auf, warum vermutlich diese Bewegung vollzogen wurde und das war genau die Frage, die ich mir gestellt hatte. Darum gilt es eine verlässliche Positionsbestimmung zu ergänzen, damit die Trajektorien Ihre Aussagekraft steigern und sich immer an derselben Position befinden. Darum geht es im folgenden Abschnitt, Positionsbestimmung.

4.2.2 Positionsbestimmung

Die Position der Trajektorien soll fortan nicht mehr von dem/der sich variable im Raum befindenden Nutzer:in ausgehen, sondern von einem festen Ausgangspunkt. Der feste Ausgangspunkt bildet den Koordinatenursprung für die Trajektorien. Zur Gewährleistung der Darstellung jeglicher aufgezeichneter Trajektorien, kommt ein fester Raum oder feste Objekte in diesem als Ausgangspunkt nicht infrage. Daher entschied ich mich für ein leicht zu handhabendes und deutlich erkennbares Bild, einen QR-Code (Siehe Abbildung 4.10). Befestigt auf einem Ständer, ist eine sichere Nutzer:in Interaktion gewährleistet und damit kann komfortabel der Ursprung justiert und weiterhin aufzeichnungsunabhängig Trajektorien positionstreu dargestellt werden.
Die Bilderkennung ermöglicht die Vuforia Engine, die zu erkennenden Bilder, in meinem Fall ein Abbild des QR-Codes, wird in die Vuforia Cloud geladen, indessen eine (Schwarz/Weiß) Bildverarbeitung stattfindet. Das Bild wird nach Helligkeit segmentiert, dann wird gruppiert und anschließend werden Regionen gebildet, wobei Objekte getrennt werden. Merkmale bilden scharfe Kanten, sie werden mit Kreuzen angedeutet (Siehe Abbildung 4.18). So könnte grob aus meinem Wissensstand die Bildverarbeitung abgelaufen sein. Das verarbeitete Bild wird heruntergeladen und in die Anwendung eingepflegt, die Bilderkennung passiert auf lokaler Ebene. Somit ist ein flexible aber zugleich fester Ursprung vorhanden, jedoch birgt es zwei Herausforderungen, die es zu bewältigen gilt, der Ursprung muss im Blickfeld des/der Nutzers:in liegen und die dauerhafte Bilderkennung nimmt begrenze Ressourcen der tragbaren Hololens ein.
Eine einmalige Bilderkennung die, einen dauerhaften Ursprung erzeugt, zugleich leicht zu entfernen ist, schien mir die passende Lösung. Eine einmalige Bilderkennung erzeugt ein dauerhaftes Objekt an dessen Stelle, verknüpft am Raum. Ermöglicht wird es von dem passiven Raumscan der Hololens gepaart mit der World Anchor Bibliothek. World Anchor verbinden erzeugte Unity-Objekte mit einem World Anchor, wodurch dies an dessen Position im Raum gehalten wird. World Anchor können, gelöscht, gespeichert und geladen werden.

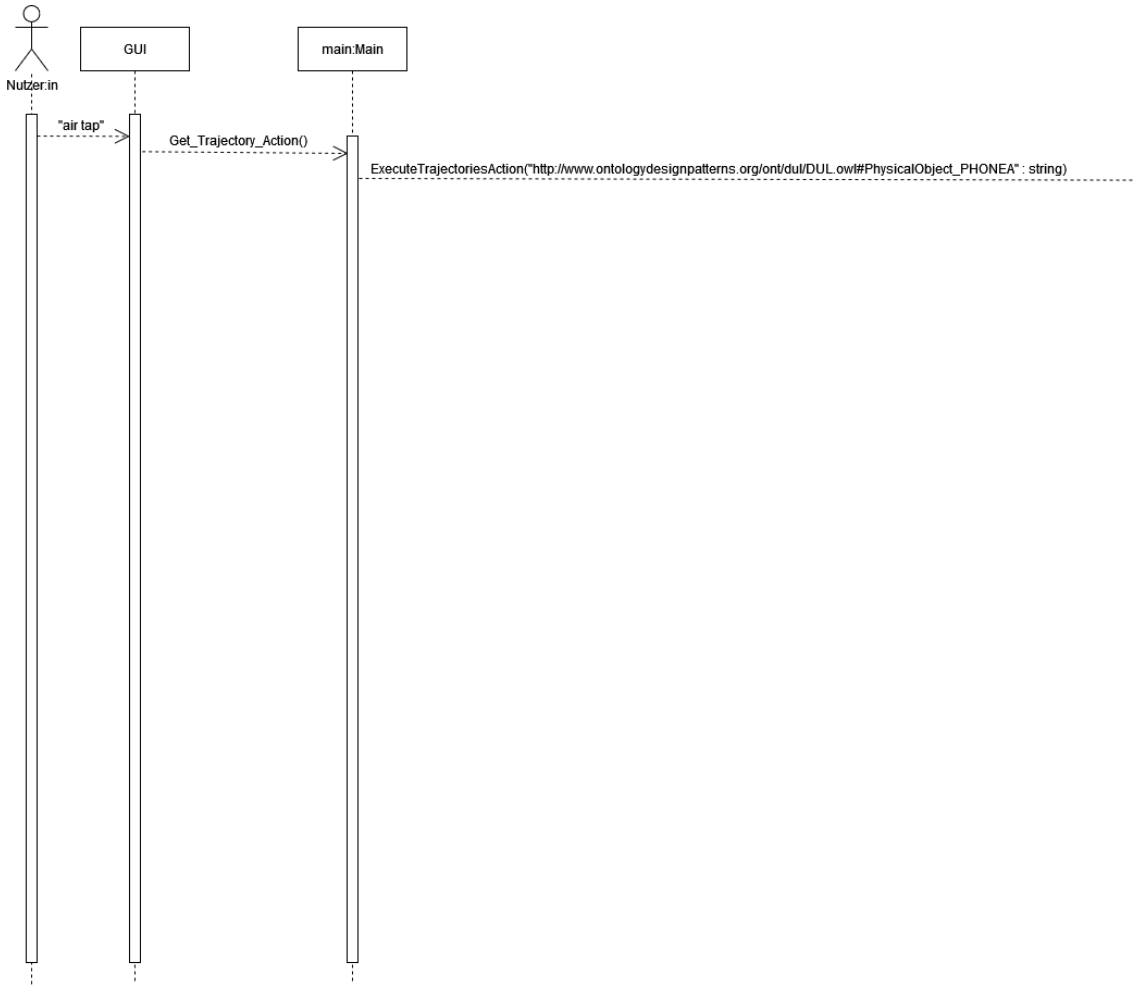

Abbildung 4.15: Anwendungsfall Eins, Teil 1

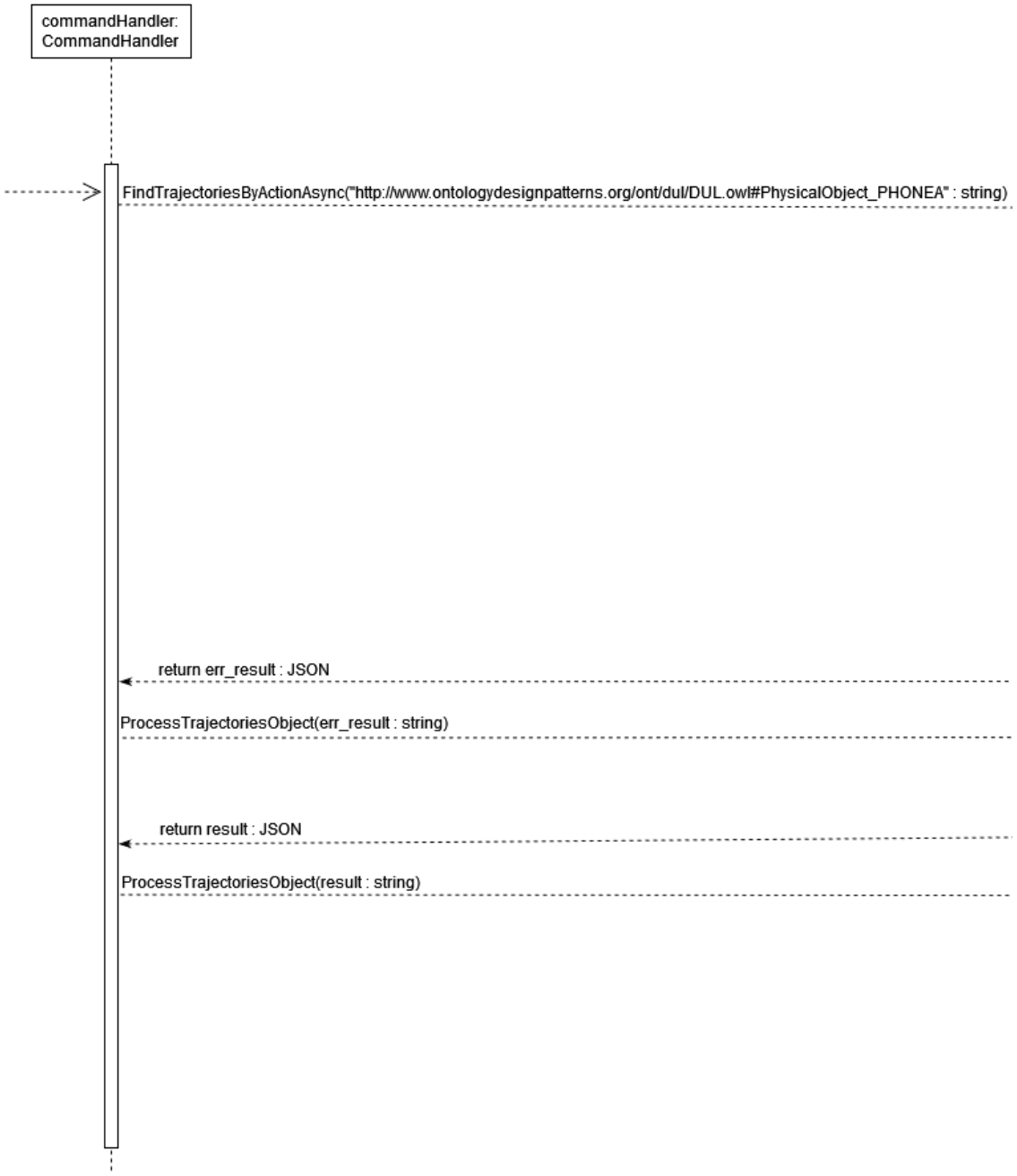

Abbildung 4.16: Anwendungsfall Eins, Teil 2

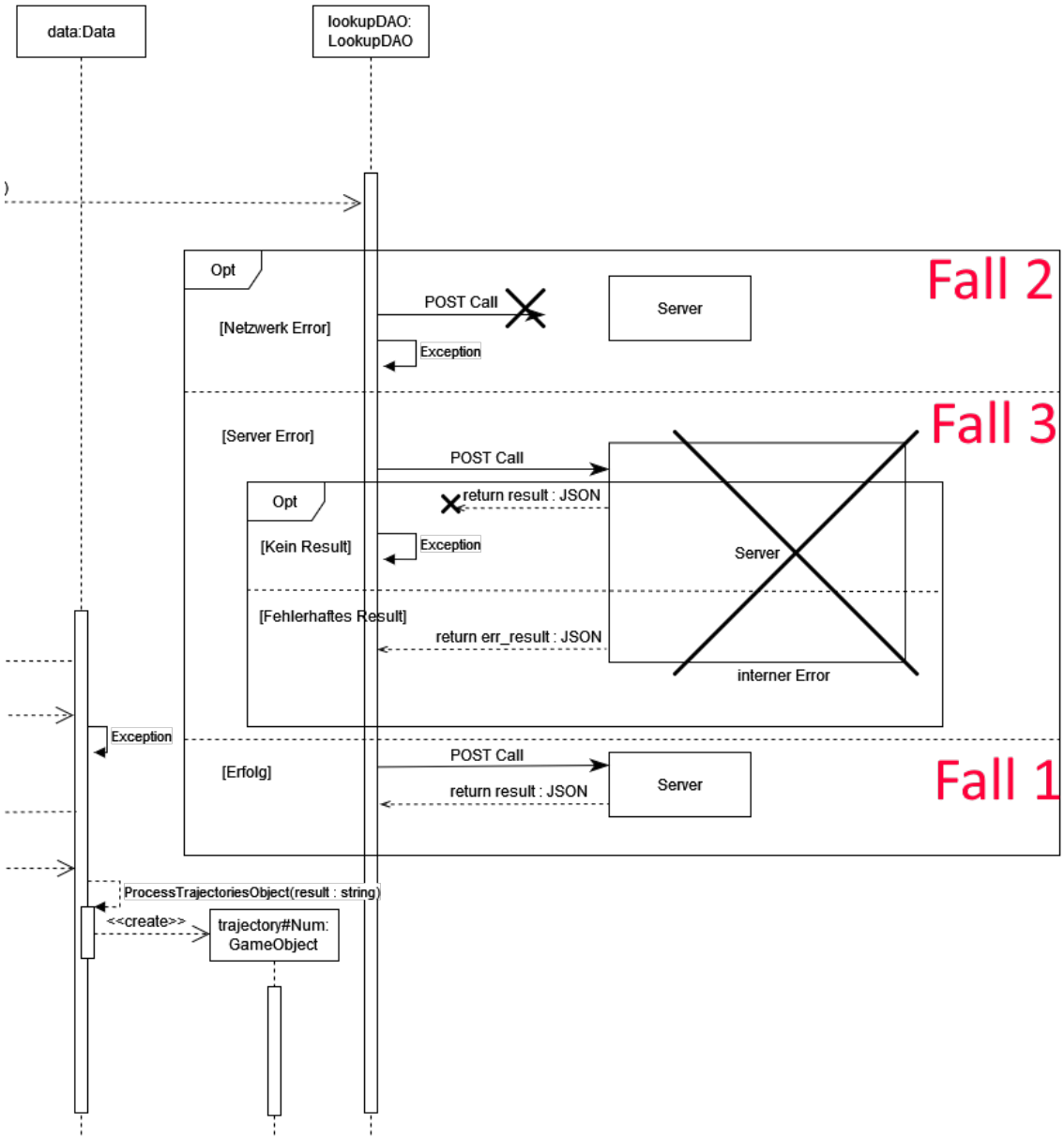

Abbildung 4.17: Anwendungsfall Eins, Teil 3

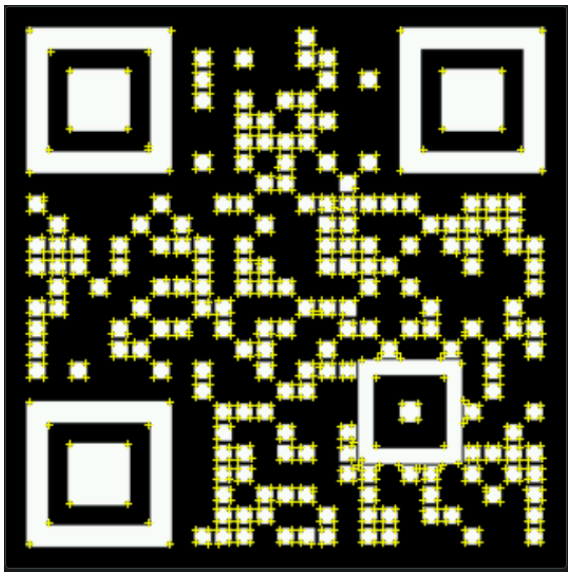

Abbildung 4.18: Bilderkennung

Anwendung auf Hololens deployen

Einmalige Elemente

- Unity Version 18.4.x installieren
- + Vuforia Augmented Reality Support
- + UWP Build Support (.NET)
- + Microsoft Visual Studio Community 2017
- Das Windows 10 SDK installieren (.NET framework development kit und IP over USB auswählen),
  `https://developer.microsoft.com/de-de/windows/downloads/windows-sdk/`
- Projekt vom Repository klonen;
  git clone https://gitlab.informatik.uni-bremen.de/kprueger/bachelorarbeit.git
- Projekt öffnen
- File -> Build Settings -> Universal Windows Platform auswählen und "switch platform" wählen, siehe Punkt Eins 4.19
- Die Scene "MainScene" öffnen

- Das MRTK foundation Paket installieren, `https://github.com/microsoft/MixedRealityToolkit-Unity/releases/tag/v2.5.4`
- "Player Settings ..." auswählen, siehe Punkt Zwei 4.19
- Einstellungen vergleichen, siehe Abbildungen 4.20 und 4.22
- Unter "Publishing Settings", InternetClient, InternetClientServer, Webcam, Microfon und SpatialPerception anwählen, siehe 4.21
- Die IP Adresse des Servers im Skript des "LookupDAO"-Objektes eintragen 4.25

Projekt bauen

1. File $->$ "Build Settings..." $->$ Build
2. "App" Ordner anlegen und/oder auswählen
3. "BA_AR.sln" doppelklicken
4. Die Hololens 1 per USB an den Rechner anschließen oder mit WLAN verbinden
5. Release, x86 und "Remotecomputer" auswählen und bauen, siehe Abbildung 4.23
6. (Einmalig) unter Projekt oder Debuggen -> "BA_AR Eigenschaften" auswählen oder in das aufkommende Popup-Fenster die IP Adresse der Hololens eintragen
7. Authentifizierungsmodus: "Universell (unverschlüsseltes Protokoll)" setzen, siehe 4.24

Projekt bauen (Error)

Was build Errors beheben kann

- Projekt neu laden, Assets $->$ Reimport All
- TextMeshPro neu installieren, Windows $->$ TextMeshPro $->$ Import TMP Essentail Resources
- App, Library löschen, sowie alle MRTK und .csproj Dateien (außer Assembly-CSharp.csproj) löschen
- PC neustarten

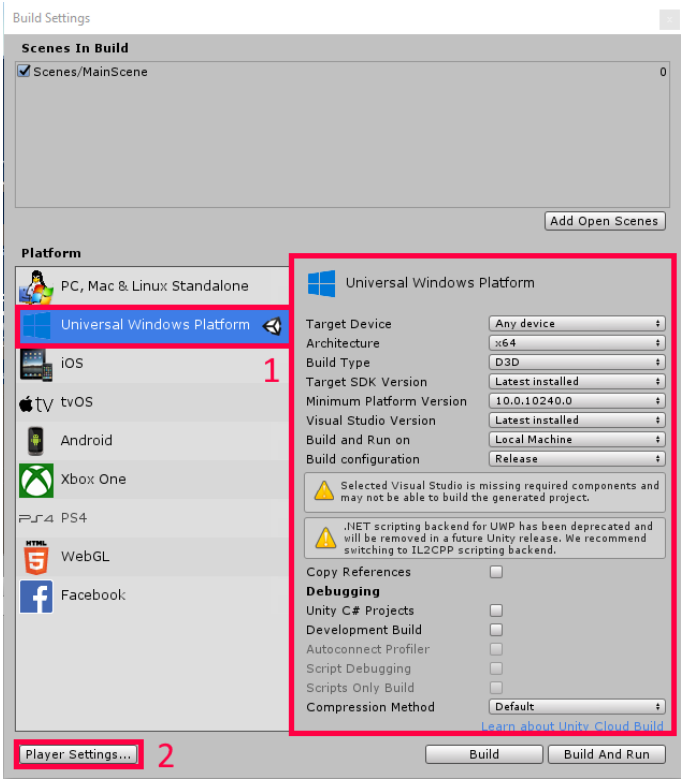

Abbildung 4.19: Build Settings

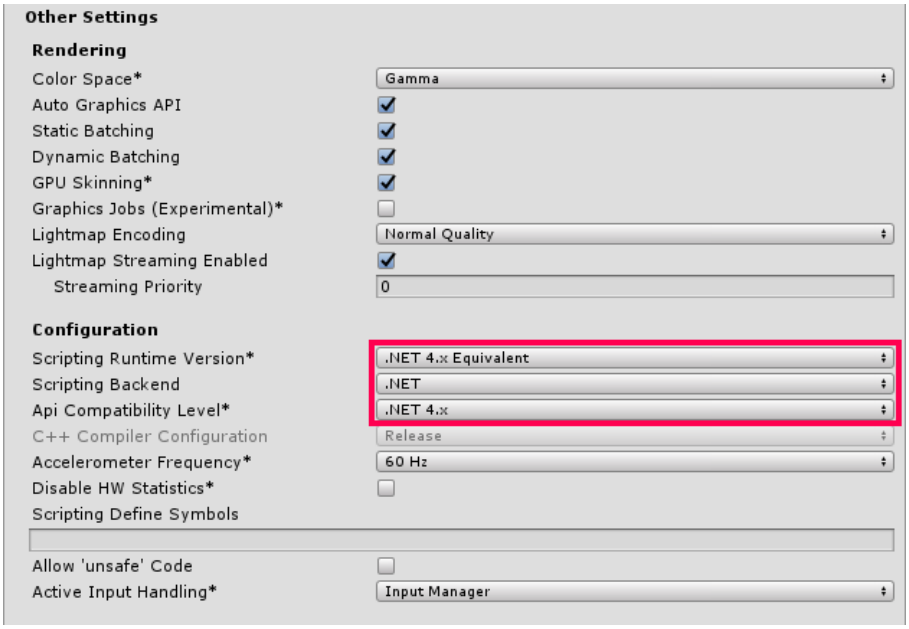

Abbildung 4.20: Other Settings

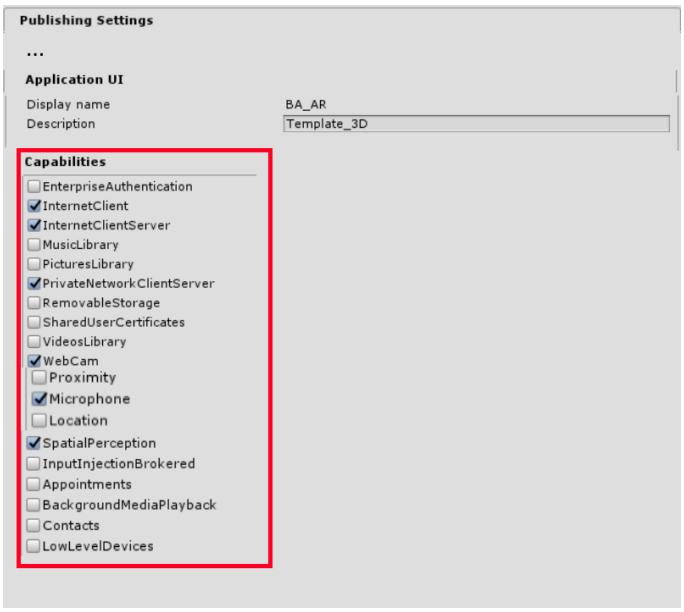

Abbildung 4.21: Publishing Settings

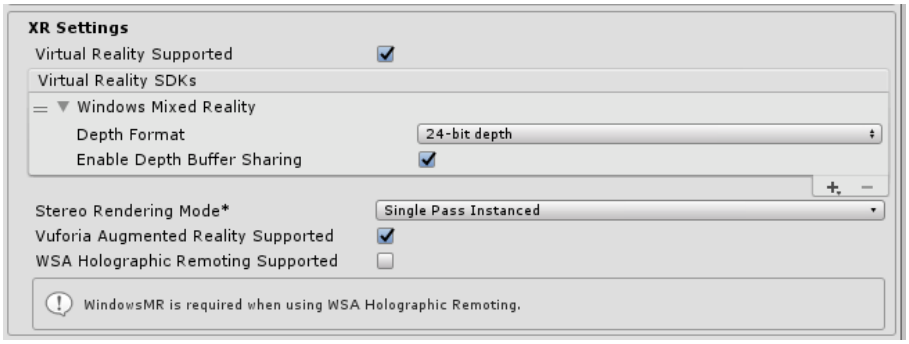

Abbildung 4.22: XR Settings

Abbildung 4.23: Visual Studio 2017 build panel

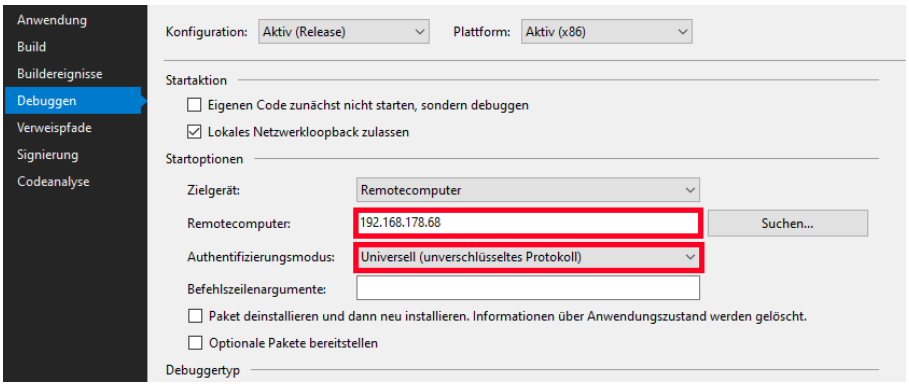

Abbildung 4.24: Hololens IP eingeben

```
public class LookUpDAO : MonoBehaviour
{
    /// <summary>
    /// REST Server URI
    /// </summary>
    private readonly string uri = "http://192.168.178.29:62226/knowrob/api/v1.0/query";

    ...
```

Abbildung 4.25: Server IP setzen

Diese Sektion schließt die Umsetzung der Client Anwendung ab. Es folgt eine Aufschlüsselung des Servers und der Schnittstelle im Gesamtsystem.

4.2.3 Server

Vorab sei erwähnt, dass ich den Server nicht selber geschrieben habe, sondern mich freundlicherweise auf die Arbeit des IAI Institutes stützen darf.

Neben der Hardware formte der Server die Entwicklung in vielen Bereichen bedeutend: Der Schnittstelle, das Rückgabeformat und der Abfrage. Die REST Schnittstelle und das JSON Austauschformat (middleware) werden weiter in Sektion 4.2.4 thematisiert. Die vom Client gerichtete Anfrage folgt der Konvention der logischen Programmiersprache Prolog.Es können Abfragen an das System gestellt werden, mit sogenannten Queries. Um also die Trajektorien eines Szenarios zu erhalten, muss diese in eine Prolog Query übersetzt werden. Meine Prolog Query sieht folgendermaßen aus:

```
has_type(Tsk, soma:'AreaSurveying'), executes_task(A, Tsk), is_action(A),
has_time_interval(A, TimeInterval), model_SOMA_EXT:has_interval_data(TimeInterval,
Since, Until), (atom(Since) -> atom_number(Since, S) ; S=Since ), (atom(Until) ->
atom_number(Until, U) ; U=Until ), Since1 is round(S), Until1 is round(U),
tf_plugin:tf_get_trajectory('http://www.ontologydesignpatterns.org/ont/dul
/DUL.owl#PhysicalObject_PHONEA', Since1, Until1, Trajectory).
```

Abbildung 4.26: Trajektorien Abfrage als Prolog Query

Sie fragt alle Trajektorien des Szenarios ab, die über den gesamten Zeitraum aufgezeichnet wurden, die Bewegungen werden jeweils nach den ausgeführten Tätigkeiten (Aktionen) unterschieden. Des weiteren werden alle Aktionen heraus gefiltert die nicht der 'AreaSurveying' entsprechen.

```
tf_plugin:tf_get_trajectory('http://www.ontologydesignpatterns.org/ont/dul/DUL.owl#PhysicalObject_PHONEA', Since1, Until1, Trajectory)
```

Abbildung 4.27: Trajektorien Abfrage

fragt alle Trajektorien eines Szenarios
($'http://knowrob.org/kb/dm-market.owl\#DMShelfW100\_BHVSONZL'$) über die den Zeitraum *Since* und *Until* ab. Ein Szenario ist als "physical Object" gespeichert. Dieses übergibt der/die Nutzer:in bei der Abfrage.

```
has_time_interval(A, TimeInterval)
```

Abbildung 4.28: Zeiträume der Aktionen

fragt ab ob die Aktion für dieses Szenario zu einem Zeitpunkt ausgeführt wurde.

```
model_SOMA_EXT:ha_interval_data(TimeInterval, Since, Until)
```

Abbildung 4.29: Timeinterval

unterhält den Zeitraum.

```
(atom(Since) -> atom_number(Since, S) ; S=Since ),
(atom(Until) -> atom_number(Until, U) ; U=Until )
```

Abbildung 4.30: Typ der Zeiträume von atom zu atom num ändern

wandelt den Typ der Zeiträume von atom zu atom_number.

```
Since1 is round(S), Until1 is round(U)
```

Abbildung 4.31: Runden der Zeiträume

rundet die Zeiträume.

```
is_action(A)
```

Abbildung 4.32: Aktionsauflistung

geht alle bekannten Aktionen durch, die ausführt werden können.

Die Elemente schränken die Abfrage weiter und können weggelassen werden, wie es in Szenario Eins der Fall war.

```
executes_task(A, Tsk)
```

Abbildung 4.33: Task Abfrage

prüft ob die Aktionen dem Task angehören und bei Verneinung werden sie aus der Liste der Aktionen genommen.

```
has_type(Tsk, soma:'AreaSurveying')
```

Abbildung 4.34: Typ des Tasks setzen

setzt den Typ des Tasks.

Der Server besteht grob aus den Komponenten KnowRob, ROS und einer Mongo Datenbank (Siehe Abbildung 4.35). ROS stellt die Schnittstelle zu den Robotern bzw. Agenten dar und ist eng in KnowRob integriert. Die Mongo Datenbank unterhält das gesammelte Wissen, also in meinem Fall die Aufzeichnungen der Bewegungen.

Bei meiner verwendeten Version handelt es sich um eine Multi-Docker Container Anwendung. Das KnowRob-init image stützt das KnowRob image initial beim Start des Servers. Beide Container KnowRob-init und KnowRob stehen in Abhängigkeit zum Mongo Container, ohne diesen ist ein Start fehlerhaft.

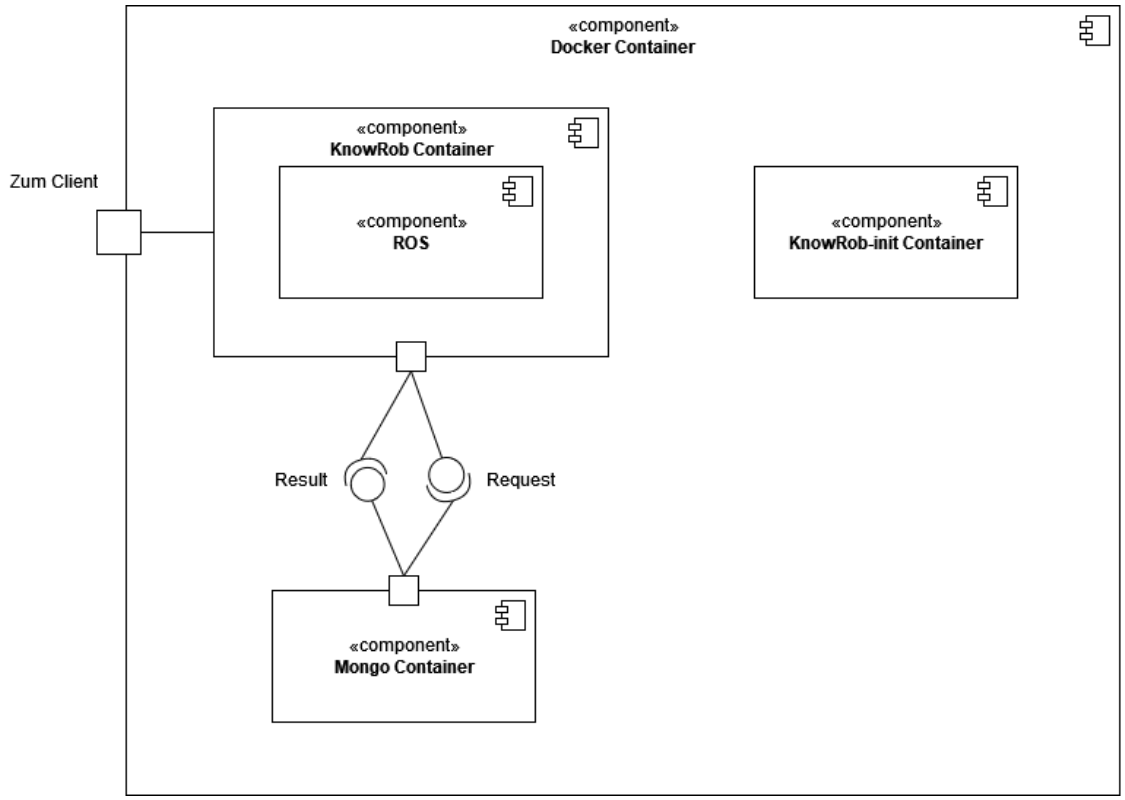

Abbildung 4.35: Konzeptionelle-Sicht des Servers

Server aufsetzen

Voraussetzung: Docker, Docker-Compose und Git installiert

1. (Einmalig) Das Repository des Servers klonen, git clone *git@github.com:K4R-IAI/knowrob_k4r.git*
2. (Einmalig) In den Branch "v1.0" wechseln, *git checkout v1.0*
3. Im Terminal in den "Docker" Ordner des Servers navigieren
4. Server starten, Linux: *sudo docker-compose up*; Windows: *docker-compose up*

Der Server kann im Intranet über dessen IP gefolgt vom Port 62226 erreicht werden.

Server beenden

1. Im Terminal des im Vordergrund laufenden Prozesses des Servers, *Strg+c drücken*

Nach Aufschlüsselung der Entstehung der Client Anwendung, so wie Aufbau und Rolle des Servers, ist ein Element ausstehend, die REST Schnittstelle. Und wieso ich mich für sie entschieden habe wird jetzt thematisiert?

4.2.4 Schnittstelle

Die Wahl fiel auf eine REST Schnittstelle, für Sie sprach in meinem Fall nicht nur die breite Unterstützung und fortlaufende Entwicklung dieser Schnittstelle, sondern, dass neben XML auch JSON als Datenübertragungsformat möglich ist. KnowRob gibt Daten gezielt im JSON Format aus und so nahm es mir die Entscheidung ab, ebenfalls meine Abfragen und gleichermaßen Rückgaben im JSON Format zu verarbeiten. Komfortabler weise brachte der KnowRob-Server eine REST Schnittstelle mit.

Die REST Schnittstelle bietet eine POST Abfrage an. Sie benötigt eine zuvor thematisierte Query und die Angabe, wie viele Resultate maximal zurückgegeben werden sollen. Dabei kann es vonnöten sein, die Obergrenze des *maxSolutionCount* anzuheben, dieser ist standardmäßig auf 100 gesetzt.

maxSolutionCount Obergrenze anpassen

1. Per Terminal auf den KnowRob Container verbinden, Linux: *sudo docker exec -u 0 -it knowrob_container bash*; Windows: *docker exec -u 0 -it knowrob_container bash*

2. (Einmalig, wenn noch nicht getan) *sudo apt update*

3. Den Nano Editor installieren, *sudo apt install nano*

4. Das “rosprolog_rest.py” Skript editieren, *sudo nano src/rosprolog/scripts/rosprolog_rest.py*

5. Die Obergrenze von “maxSolutionCount” anpassen, siehe 4.36

```
...

# Query interface
query = api.model('Query', {
    'query': fields.String(required=True, description='The query string'),
    'maxSolutionCount': fields.Integer(required=True, default=999, description='The
maximal number of solutions'),
    'response': fields.List(fields.Raw, readonly=True, description='The response
list')
})

...
```

Abbildung 4.36: maxSolutionCount Obergrenze anpassen

Alle Komponenten der Architektur fanden in vergangenen Sektionen Erläuterung, abschließend möchte ich auf die verwendeten Daten eingehen, auf die ich meine Entwicklung stützt und getestet habe.

4.3 Daten

Die zu handhabenden Daten teilen sich in zwei Kategorien, die zu erzeugenden Daten und die Daten auf dem Server, die abgefragt werden. Letzteres sind die Daten, auf dessen Basis ich die Anwendung entwickelt und getestet habe. Sie entspringen nicht meiner Erzeugung, sondern wurden mir freundlicherweise vom IAI Institut zur Verfügung gestellt. Ihre Erzeugung erfolgte per Smartphone, es wurde durch die Regale des Drogeriemarktes geführt. Die entsprechenden triples und tf JSON Dateien lud ich in die Mongo Datenbank über KnowRob.

Daten in den Server laden
Bevor die gewünschten Daten in die Mongo Datenbank geladen werden können, müssen diese einmalig auf den Server übertragen werden. Die Übertragung habe ich auf zwei wesentliche Methoden vollzogen. Die Ergänzung eines Volumes, auf den die Daten kopiert werden oder das Hochladen der Daten auf einen Cloud-Dienst/Git, die die Daten auf den Server übertragen. Die Methoden können je nach Betriebssystem abweichen.

Windows-Server
Voraussetzung der Server ist gestartet und die Daten liegen in einer Cloud. Man hat Zugriff auf den Server.

1. Auf den KnowRob Container per Terminal verbinden, *docker exec -u 0 -it knowrob_container bash*

2. (Einmalig) *sudo apt update* ausführen

3. (Einmalig) das Git klonen oder Zugriff auf die Cloud herstellen und Daten herunterladen, *git clone "Git mit hinterlegten Daten"*

4. *source devel/setup.bash* ausführen

5. Prolog Konsole starten, *rosrun rosprolog rosprolog_commandline.py*

6. triples Datei laden, *lang_triple:triple_import_json('Absoluter Pfad/"Dateiname".json').* und warten bis *true.* auftaucht

7. Prolog Konsole verlassen, *halt.*

8. tf Datei laden, *mongoimport –collection=tf –file="Absoluter Pfad/Dateiname.json" –jsonArray –uri=mongodb://mongo_container:27017/roslog*

9. Die Verbindung vom KnowRob Container trennen, *exit*

Linux-Server
Die Daten liegen auf dem Server. Man hat Zugriff auf den Server.

1. (Einmalig) die Docker-Compose Datei um ein Volume ergänzen, 4.38

2. tf und triples Dateien in das Volume kopieren, *sudocp“Dateipfad/tf − /triples − Dateiname″.json/var/lib/docker/volumes/docker_lft/_data*

3. Dateirechte setzen, *sudo chmod 777 /var/lib/docker/volumes/data_lft/_data/“tf-/triples-Dateiname”.json*

4. Den Server starten, zuvor in den “Docker” Ordner des Servers gehen, *sudo docker-compose up*

5. Auf den KnowRob Container per Terminal verbinden, *sudo docker exec -u 0 -it knowrob_container bash*

6. Dateien befinden sich im Ordner “/home/data”, weiter bei Punkt 3 vom Windows-Server

Die erzeugten Daten bzw. Modelle belaufen sich auf ein Trajektorien-Modell (Siehe Datenmodell 4.37). Neben einer Bezeichnung zur Unterscheidung, enthält es dessen Positionsdaten im Raum. Die Farbe macht die Trajektorien unterscheidbar und zugleich einer Bewegung bzw. Aktion zugehörig. Die Positionsdaten des Elternobjekts ermöglichen die Erzeugung der Trajektorien im Verhältnis zu diesem. Bei dem Elternobjekt handelt es sich um den World Anchor. Der World Anchor setzt voraus, dass alle Objekte die ihn als Koordinatenursprung verwenden möchten, ein untergeordnetes Objekt von ihm werden, daher wird eine Referenz zum World Anchor übergeben. Darüber hinaus wird die Skalierung des Objektes als anzeige Größe benötigt.

Diese Sektion beschließt das Kapitel der Umsetzung, es folgt das Evaluationskapitel, der dritte Schritt der Design Science Research Methodik.

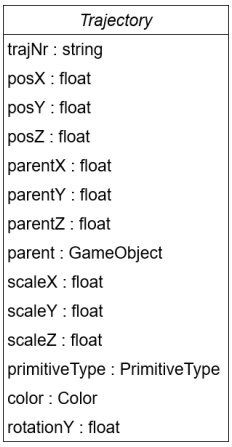

Abbildung 4.37: Datenmodell Trajektorie

```
version: '3'
services:
  mongo:
    image: mongo:4.4.3-bionic
    container_name: ${MONGODB}

  knowrob-init:
    image: iaik4r/knowrob-init:v0.3
    container_name: ${KNOWROB_INIT}
    depends_on:
      - 'mongo'
    environment:
      - KNOWROB_MONGODB_PKG_URI=${KNOWROB_MONGODB_PKG_URI}

  knowrob:
    image: iaik4r/knowrob:v1.0.5
    container_name: ${KNOWROB}
    depends_on:
      - 'mongo'
    environment:
      - KNOWROB_VERSION=${KNOWROB_VERSION}
      - KNOWROB_PORT=${KNOWROB_PORT}
      - KNOWROB_MONGODB_URI=${KNOWROB_MONGODB_URI}
      - SANDBOX=${SANDBOX}
    ports:
      - "${KNOWROB_PORT}:${KNOWROB_PORT}"
    entrypoint: /entrypoint.bash
    command: roslaunch knowrob_k4r knowrob_k4r.launch
    volumes:
      - lft:/home/data

volumes:
  # volume to transfer scanned data
  lft:
```

Abbildung 4.38: Docker Compose Datei um ein Volume ergänzen

5 Evaluation

Im Folgenden wird geprüft, wie gut die Umsetzung geeignet ist, das Ausgangsproblem zu lösen. Darüber hinaus wird der weitere Nutzen der Anwendung gemessen.

5.1 Evaluation

KnowRobs Wissenserfassung beruht auf “imitation learning”, bisher wird die Schnittstelle zur echten Umgebung außer Acht gelassen. Inwieweit mein praktischer Ansatz dem gerecht wird, soll die Evaluation aufschlüsseln. Sie gliedert sich in drei Sektionen, Kriterien 5.1.1, Forschungsfrage 5.1.2 und VR NEEMs - VR Episodische Erinnerung 5.1.3.

5.1.1 Kriterien

Während und nach der Entwicklung habe ich meine Umsetzung aus verschiedenen Perspektiven betrachtet, die in Fragen gemündet sind. Ihre Beantwortung soll einen Überblick verschaffen, was meine Umsetzung leistet und wo Hürden bestehen. Sie stellen den Bewertungshorizont dar, die Kriterien.

Mit den Entwicklungsdaten habe ich versucht folgende Fragen zu beantworten:
Lässt sich aus der Rückgabe der Abfrage verwertbare Informationen extrahieren? Bzw. Abfragen erzeugen die verwertbare Informationen zurückgeben? Ist es möglich, mit meinem Ansatz die Trajektorien darzustellen? Lässt die Perspektive (Aufbereitung der Daten) einen Erkenntnisgewinn über die Daten zu? Sind weitere Abfragen möglich? Sind die Darstellungen verlässlich zu wiederholen?

Im zweiten Teil geh ich auf die Erkenntnisse des Verhaltens meiner Anwendung ein, die Testdaten ergaben. Wie verhält sich meine Anwendung mit Fremddaten? Ist es möglich, Ergebnisse per Abfrage zu beziehen? Lassen sich die Ergebnisse darstellen? Verhält sich die Darstellungen gleich zu meinen Entwicklungsdaten?

5.1.2 Forschungsfrage

Szenario Eins und Zwei, die auf den Entwicklungsdaten basieren wurden im Institut eigenen Drogeriemarkt aufgenommen, es wird in Folge auch als Lab bezeichnet (Abbildungen 5.1).

Abbildung 5.1: Drogerie-Regale

Das Ziel war aufgezeichnete Trajektorien in dessen Umgebung wieder darzustellen, um im Kontext der Umgebung Informationen zu erhalten. Die Szenarien stellen Beispiele dar, mit denen ich Antworten auf die Fragen suche. Doch zuvor möchte ich kurz erläutern, was mit "verwertbaren Informationen" gemeint ist, sie stellt die Rückgabe der Abfrage dar, indem sich die Trajektorien-Punkte zur Darstellung extrahieren lassen (Siehe Abbildung 5.11).

Die Abfrage für Szenario Eins habe ich ausgeführt und die Verarbeitung der Rückgabe, sowie die Visualisierung der Trajektorien funktionierten (Siehe Abbildung 5.2). Die Visualisierung zeigt Überlagerungen von Trajektorien auf. Wiederkehrende Bewegungen lassen sich in ihre unterschiedlichen- und gemeinsamen Bewegungsabschnitte unterteilen (Siehe Abbildung 5.3). Sind die Bewegungen des Agenten über die Zeit akkurater bzw. benötigt er weniger Versuche? Eine von vielen Fragen die sich durch die Sicht ergründen lassen. Eine weitere Eingrenzung der Handlung, wie im Fall Szenario Zwei auf die Umgebungs-Begutachtung ermöglicht eine detailliertere Sicht, welche Handlung wie oft und wo angewandt wurde (Siehe Abbildung 4.5). Auch hier lässt sich über die Zeit vergleichen, ob sich die Strategie eines Agenten bezüglich seiner Aufgabenbewältigung geändert hat.

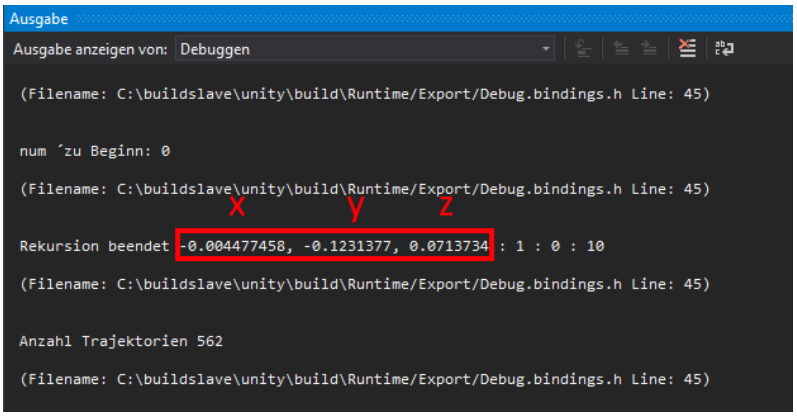

Abbildung 5.2: Ausschnitt der Konsolen Rückgabe von Szenario Eins

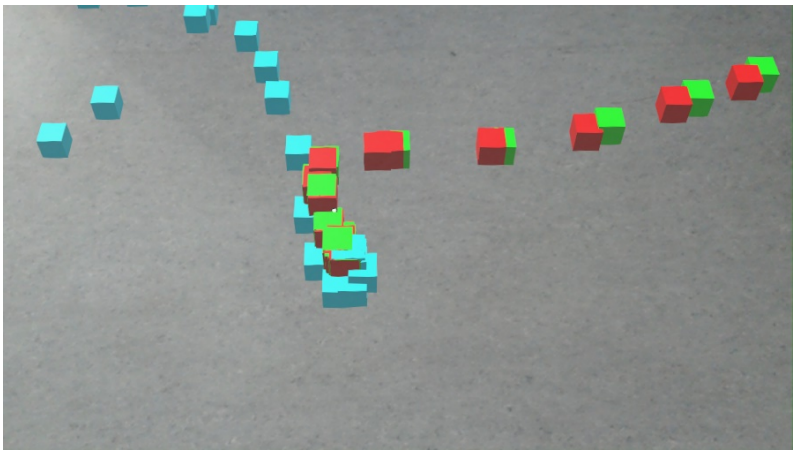

Abbildung 5.3: Überlagerung von Trajektorien

Die farbliche Unterscheidungskraft kommt mit einem Kompromiss. Jede Trajektorie erhält eine Farbe aus einem Array, wodurch Doppelungen nicht auszuschließen sind. Eine zuvor automatisch generierte Farb-Erzeugung pro Trajektorie erwies sich als zu unzuverlässig, sie erzeugte Renderfehler und alle Trajektorien wurden in Weiß erzeugt (Siehe Abbildung 5.4). Da ich keinen Einfluss auf die Reihenfolge der Trajektorien Koordinaten in der Rückgabe habe, ist eine Analyse vonnöten, die Aufschluss über Überlagerungen gäbe und die Farbe der Trajektorien-Punkte unterscheiden lässt. Jedoch würde der Vergleich jeder Trajektorien Koordinaten mit denen der anderen Trajektorien Koordinaten, eine exponentielle Aufgabe darstellen, die nicht mit dieser Hardware zu vereinbaren wäre. Darüber hinaus lag eine solche Optimierung nicht in meinem Fokus. Eine Analyse der Rückgabe ausgeschlossen, experimentierte ich mit weiteren Faktoren die die Unterscheidbarkeit der Trajektorien fördern sollte. Sie sollten die Farbbegrenzung kompensieren. Ich probierte eine deutlichere Abgrenzung durch Formen zu erreichen, ich entschied sie seien ungeeignet, da sie die Vergleichbarkeit und Perspektive verzerren durch die unterschiedlichen Größen die Ihre Formen mit brachten (Siehe Abbildung 5.5). Stattdessen die Trajektorien-Punkte

um die eigene Y-Achse entgegen der Anderen Trajektorien zu drehen, erwies sich als geeigneter, beschränkte die Punkte jedoch auf die Würfelform, da Kanten benötigt werden. Die zusätzlich gewonnene Unterscheidung kompensierte nicht die aufkommenden Irritationen in kurzen, beengten Überlagerungen, durch die vermehrten Kanten (Siehe Abbildung 5.6).

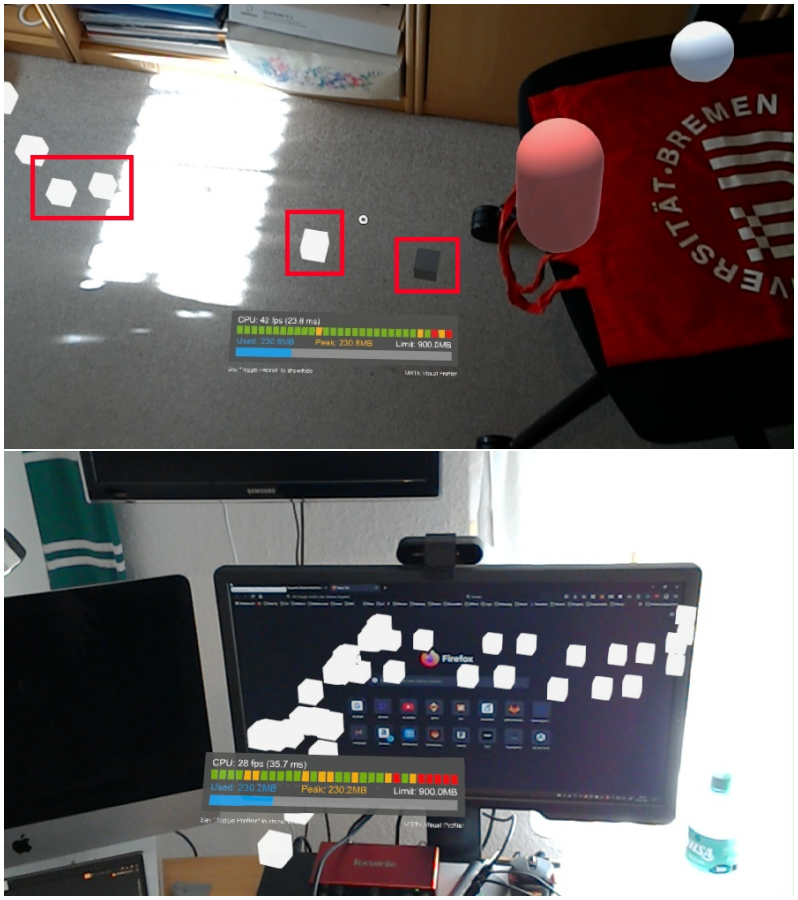

Abbildung 5.4: Renderfehler Trajektorien

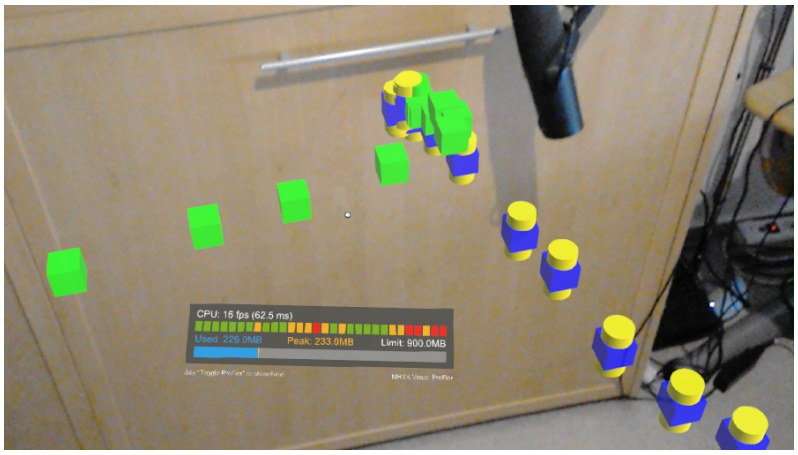

Abbildung 5.5: Trajektorien unterscheiden sich durch Formen

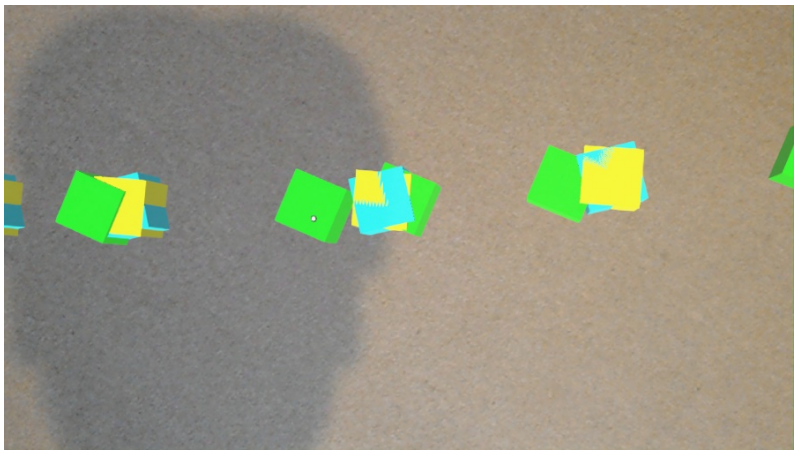

Abbildung 5.6: Trajektorien-Punkte sind um die Y-Achse gedreht

Der Vergleich über die Zeit einer Handlungsausführung setzt eine örtlich verlässliche Darstellung voraus. Der World Anchor kann manuell an dieselbe Stelle des Startes der Handlungsausführung platziert werden und dient als Ursprung der Trajektorien. Im Laufe der Entwicklung habe ich unzählige Male die Platzierung vorgenommen und eine Ausführung in Videoform festgehalten (Siehe Abbildung 5.7). Eine Zentimeter genaue Annäherung an den Ursprung ist möglich (Siehe Abbildung 5.8).
Im Zuge der Tests kam ein weiterer Kompromiss auf, die Darstellung der Trajektorien hängt neben dem Ursprung von der Achsenausrichtung ab und diese variiert. Das Weltkoordinatensystem wird mit dem Start der Anwendung festgelegt, die zu diesem Zeitpunkt befindliche Blickrichtung markiert die Z-Achse. Die Trajektorien sind um die Y-Achse rotiert, dies ist auf dessen Implementierung im Raum zurückzuführen. Die Trajektorien werden ausgehend vom Punkt des World Anchors im Raum erzeugt. Sie direkt zu Kindern des World Anchor zu machen und mit der Methode *localPosition* in dessen Koordinatensystem zu setzen schien mir eine Lösung, sie stellt sich aber als noch unpraktikabler heraus,

da sie die Skalierung des Objektes ansetzt. Die Skalierung war zusammengestaucht (Siehe Abbildung 5.9).

Abbildung 5.7: World Anchor plaziert

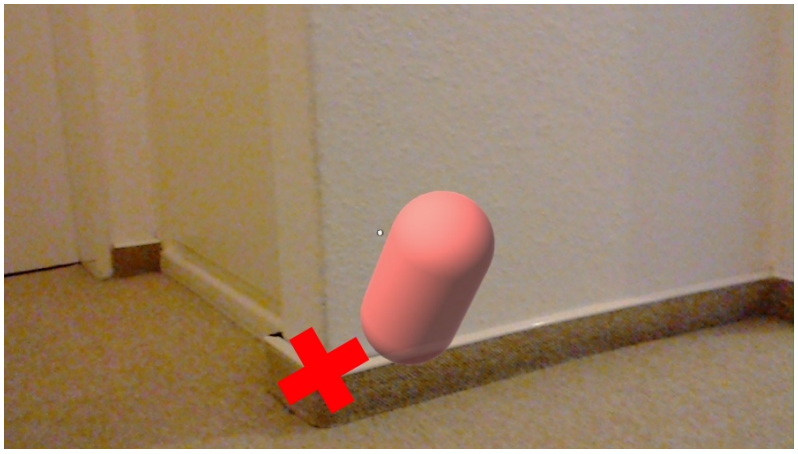

Abbildung 5.8: World Anchor am NEEM aufzeichne Koordinatenursprung setzen

Abbildung 5.9: Szenario Eins mit localPosition Methode platziert

5.1.3 VR NEEMs - VR Episodische Erinnerung

Die Fremddaten wurden mir freudlicherweise von Nazmi zur Verfügung gestellt. Er hat diese Daten im Rahmen seiner Bachelorarbeit "Generierung von Episodic Memories in einer virtuellen Umgebung" erzeugt. Sie wurden in der Nachbildung des Labs in VR aufgenommen.
Diese Daten bilden die Grundlage der Szenarien Drei, Vier und Fünf. Sie dienen stellvertretend zur Demonstration bzw. zum Test, ob Abfragen von Fremddaten möglich ist. Als Fremddaten sind in diesem Kontext Daten die durch eine andere Domäne bzw. durch eine andere Quelle erzeugt wurden zu verstehen. Szenario Drei, Vier und Fünf ähneln einander stark, sie zeigen das Greifen eines Produkts. Es handelt sich jeweils um eine Trajektorie, die einen Trajektorien-Punkt aufweist. Hier zeigen sich die Grenzen der Anwendung, die Daten entscheiden über dessen Anwendbarkeit und den Nutzen. Inhaltlich sind sie nicht sehr tiefgründig aus dem Blickwinkel der Darstellung von Trajektorien, sie bieten keine längeren Trajektorien. Es war mir nicht möglich eine aussagekräftigere Abfrage zu verwenden bzw. Objekte und Handlungen zu finden.
Wie zuvor Szenario Eins und Zwei, habe ich diese ebenfalls an dem dafür vorgesehenen Ort getestet, dem Lab. Die Abfragen waren erfolgreich (Siehe Abbildung 5.10). Die Darstellung des Trajektorien-Punktes vermutlich auch, dies kann ich jedoch nicht eindeutig nachweisen, da ich ihn nicht gesehen habe. Es zeigt eine weitere Herausforderung die die Diversität der Daten mit sich bringt, die Rückgabe der Abfrage zeigte vom Aufbau keine Unterschiede zu den von Szenario Eins und Zwei, aber auf der inhaltlichen Ebene. Die Koordinaten weisen eine andere Skalierung auf, die Koordinaten aus Szenario Drei, Vier und Fünf sind um ein vielfaches größere Zahlen (Siehe Abbildung 5.11). Daher meine Schlussfolgerung, die Trajektorien-Punkte sind vermutlich außerhalb des Gebäudes erzeugt worden oder liegen außerhalb des Render Bereiches der Hololens.
Da ich keine weiteren Fremddaten zum Testen habe, entschied ich die VR NEEMs Koordinaten zu manipulieren, ich verringerte ihren Wert, damit er im Darstellungsbereich liegt (Siehe Abbildung 5.12). Unter diesen Umständen wollte ich untersuchen, ob neben der

Abfrage und Verarbeitung von Fremddaten, ich auch dessen Darstellung verifizieren kann. Die Ausführung von Szenario Drei mit den modifizierten Daten ließ dessen Trajektorien-Punkt sichtbar darstellen. Das Erzeugen von Trajektorien von Fremddaten ist möglich (Siehe Abbildung 5.13).

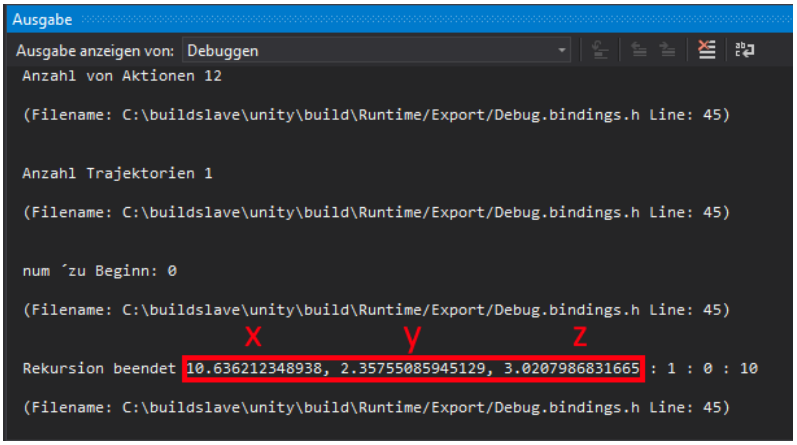

Abbildung 5.10: Ausschnitt der Konsolen Rückgabe von Szenario Drei

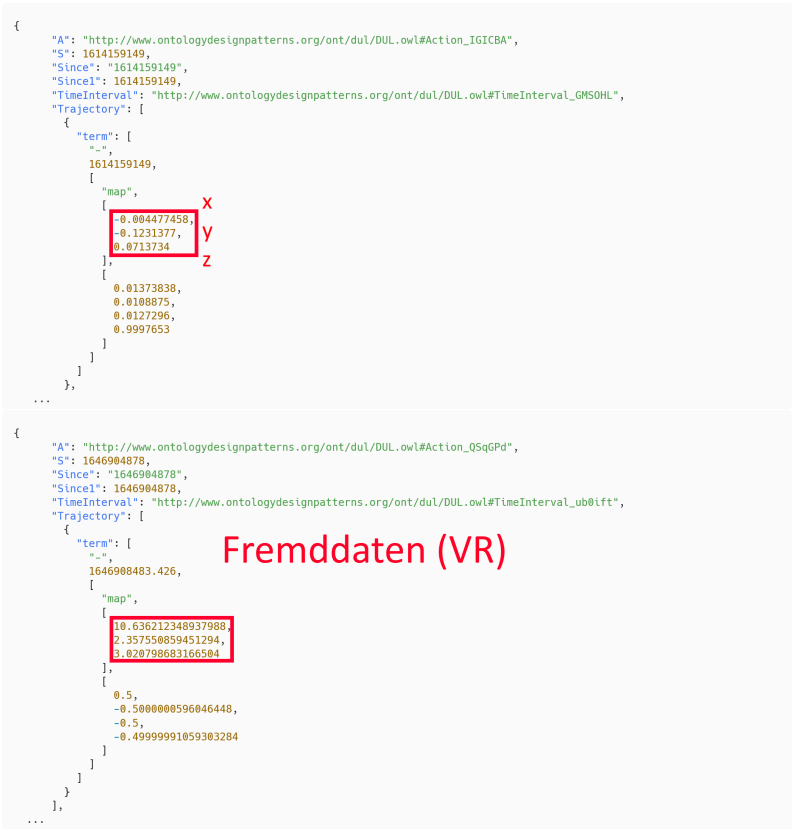

Abbildung 5.11: Trajektorien Koordinaten Ausschnitte

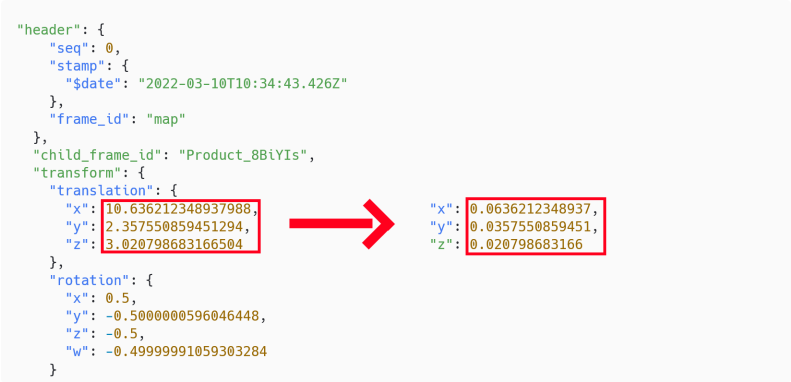

Abbildung 5.12: Manipulation der VR Daten

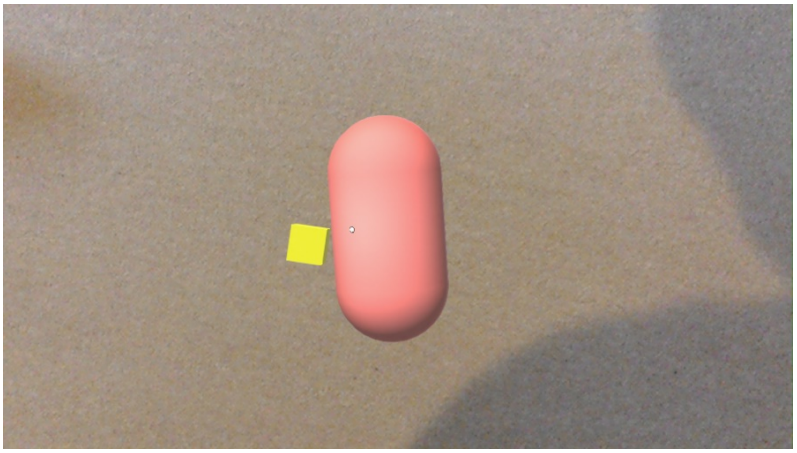

Abbildung 5.13: Szenario Drei mit modifizierten Daten

5.2 Fazit

Die Evaluation zeigt, dass der Ansatz Abfragen, dessen Verarbeitung und die Visualisierung ermöglicht. In Ansätzen wurde der Nutzen durch die Aufbereitung der Daten exemplarisch dargestellt. Zugleich zeigt es deutlich die Kompromisse auf, die den Nutzen mindern, wie die Achsenausrichtung oder die begrenzte Farbauswahl bei Trajektorien. KnowRobs größte Stärke, das Zulassen von Wissen aus diversen Quellen, stellt hier die größte Herausforderung dar. Die Datenvielfalt zeigt am deutlichsten die Grenzen der Umsetzung auf. Es ist eine Skalierungsangleichung vonnöten, entweder von Seiten der Anwendung oder der Daten bzw. bei dessen Erzeugung. Eine vorherige Konvertierung, vor der Darstellung stellt auch eine Möglichkeit dar.

6 Fazit

6.1 Fazit

Diese Arbeit verfolgte das Ziel, eine praktische Umsetzung zu entwickeln die es ermöglicht erlernte Aufgaben Abläufe der Agenten in der Welt, durch die Hololens zu visualisieren. Das Lernen der Aufgabenbewältigung beruht auf "imitation learning", was wiederum durch das Sehen bzw. Bildverarbeitung und weiteren Sensoren geschieht. Bisher wurde die Schnittstelle zur Welt außer vor gelassen. In diesem Zuge stellte ich mir die Frage "Hilft die Visualisierung der Trajektorien die Qualität der Aufgabenbewältigung der Agenten zu beurteilen?". Das Vorgehen der Design Science Research Methodik, lies mich Erkenntnisse einordnen und gab grob den Aufbau der Arbeit vor. Zu Beginn der Umsetzung sah ich mich damit konfrontiert, dass die Daten lagen im JSON Format vor und waren nach der KnowRob Konvention in "triples" und "tf" Dateien geteilt. Die Darstellung der Trajektorien, bedurfte eine Analyse der Dateien, um über gemeinsame Zeitpunkte die Daten zusammen zuführen. Die Hardwarelimitierung unterbund das Vorhaben und es musste ein Alternative gefunden werden. Ich brachte in Erfahrung, dass KnowRob diese Funktionalität bietet und es formte sich die Client-Server-Architektur. Die Datenverarbeitung ausgelagert, stellt die Anwendung REST Abfragen an den Server und hat die Rückgabe zu verarbeiten. Da der Server intern mit Prolog arbeitet, mussten die Abfragen in Queries formuliert und in REST gekapselt werden. Die Verarbeitung der Rückgabe zur Extraktion der Koordinaten der Trajektorien-Punkte erfolgt mit der Unterstützung der JSON.NET Bibliothek. Aus den Koordinaten wurden die Trajektorien erzeugt und visualisiert. Zu diesem Zeitpunkt wurden die Trajektorien Kontext los vom Ursprung des/der Benutzers:in erzeugt, es bedarf eine Zuordnung zu dessen Aufzeichnungsursprung. Die Zuordnung ermöglicht die Bilderkennung der Vuforia Bibliothek, ein hinterlegter QR-Code kann an den Ursprung gelegt, erkannt werden und erzeugt einen World Anchor. Der erzeugte World Anchor stellt ein verankertes Objekt im Raum dar, von dem aus die Trajektorien vortan erzeugt werden.
Die anschließende Evaluierung bestätigte die Funktionalität. Sowohl aus Daten die während der Entwicklung zum Testen gebrauch fand, als auch Fremddaten ließen sich abfragen und Trajektorien darstellen. Es konnten Überlagerungen der Trajektorien gezeigt werden, sie sollen zeigen, dass sich Daten über die Zeit visualisieren lassen, um mögliche Veränderungen in den Tätigkeitsabläufen fest zu können. Letzte Darstellung setzte eine Manipulation der Koordinaten voraus, damit sie in der Welt darstellbar waren. Eine fehlende Angleichung der Skalierung, um Quellen unabhängige Daten kontextbezogen darzustellen ist ein von weiteren Herausforderungen, die die Evaluation aufzeigt. Der Anwendungsstart setzt die Ausrichtung der Achsen des Welt-Koordinatensystem und kann

so die Darstellung der Trajektorien verzerren. Der dritte Punkt ist die farbliche Unterscheidbarkeit, sie ist durch Renderfehler begrenzt und eine Analyse die Trajektorien auf Überlagerungen prüft, um sie verlässlich farblich unterscheidbar zu machen, lässt die Hardwarelimitierung nicht zu. Einen weiteren Unterscheidungsfaktor einzubinden erwies sich als unpraktikabel. Unterscheidung durch Formen verzerrte das Bild und eine Rotation der Trajektorien-Punkte um die eigene Y-Achse erwies sich als zu unübersichtlich und störend.
Hinsichtlich dieses Stands der Umsetzung muss ich die Forschungsfrage verneinen, die Anwendung ist zu unzuverlässig und weißt zu viele Kompromisse auf. Die Daten boten aus der Sicht der Trajektorien Visualisierung eine zu geringe inhaltliche Substanz. Es war in Ansätzen gelungen den Nutzen der Anwendung im Bezug der Analysemöglichkeit der Trajektorien zu zeigen.
Im Ausblick werden die Punkte erneut aufgegriffen und skizziert wie die Forschung weiter gehen könnte, um sich erneut der Forschungsfrage befassen zu können.

6.2 Eigene Meinung/Reflektion

Zum Abschluss, bevor ich einen kurzen Ausblick über die Weiterführung dieser Forschung gebe, möchte ich über diese Arbeit und den Prozess reflektieren. Was habe ich persönlich gelernt? Was würde ich anders machen, in künftigen Arbeiten?
Inhaltlich bin ich durch diese Arbeit auf vielen Ebenen gewachsen. Sie forderte mich in verschiedenste technische Bereichen. Darunter fallen, Docker, Prolog, REST, JSON.NET, Vuforia, Unity und das MRTK. Aufseiten der Softskills förderte es meine Planungs- und Kommunikationsfähigkeiten. Ich war gezwungen eigenständig ein Projekt zu planen, durchzuführen und meine Fortschritte selber zu bewerten. Zugleich war die Kommunikation zu meinen Betreuerinnen und Experte:innen entscheidend, Rücksprachen halfen beim Vorgehen und förderten die Auseinandersetzung mit Problemen, da man sie verständlich vermitteln musste, um Hilfestellung erhalten zu können. Ein letzter Punkt betrifft die Planung und Forschrittsbewertung, so hätte ich neben meiner Kernaufgabe (Entwicklung der Anwendung) auch Engagement in angrenzende Aufgaben invertieren muss. So hätte ich mich selber frühzeitig um die Erzeugung eigener Fremddaten kümmern müssen oder vorab Zeit dafür einplanen müssen. Dies hätte aus meiner Sicht einen erheblicheren Erkenntnisgewinn in der Evaluation eingebracht und die inhaltliche Aussagekraft gesteigert. Dazu gehört auch die vermehrte Berücksichtigung von Literatur, um nicht nur die Vorteile zu sehen und Beispiele wie eine Umsetzung aussehen könnte, sondern dass auch eine konstruktive Auseinandersetzung mit vermeintlich aufkommenden Herausforderungen vorangetrieben wird. Sodass Herausforderungen vorab vermieden oder bei der Evaluation erwartet werden und die Bewertung auf einer aussagekräftigeren Ebene geschieht.

Abschließend kann ich sagen, dass diese Arbeit mir einen weitreichenden Einblick in die Forschung gewährt hat.

7 Ausblick

Wie kann auf diese Arbeit aufgebaut werden? Die Erkenntnisse genommen und etwas Neues erschaffen werden? Wenn man der Design Science Research Methodik dieser Arbeit treu bleibt, dann bestimmt sie bereits das weitere Vorgehen. Sie sieht vor die gewonnenen Erkenntnisse praktisch einzuarbeiten und erneut zu evaluieren. Die Arbeit erbrachte die Kenntnis über eine Angleichung der Daten bezüglich der Skalierung zur Darstellung. Eine Ergänzung der möglichen Farben zur Unterscheidung oder einer Analyse zur Erkennung von Überlagerungen, um sie farblich unterscheiden zu lassen. Die letzte Erkenntnis betrifft die Achsenausrichtung, die Trajektorien benötigt eine unabhängigere Darstellung von dem Koordinatensystem der Anwendung.
Neben den direkt aus der Arbeit resultierenden Erkenntnissen, kamen im Prozess weitere Ideen zu Erweiterungen, die die Aussagekraft der Trajektorien Visualisierung steigern könnte und aus meiner Sicht einer Untersuchung bedarf. Darunter fällt die Steigerung der Performance der Rückgabe Verarbeitung. Eine Auslagerung dieser auf einen Server wäre denkbar. Zugleich würde es die Hardwarelimitierung lösen und eine "Überlagerungsanalyse" ermöglichen, die die Farbenlimitierung umgeht. Ein weiteres Feld wäre die Nutzer:in Interaktion. Mit den erzeugten Trajektorien interagieren, sie benennen oder feste Farben zuweisen, erhöht die Wiedererkennung. Aus den neu geladenen Daten Objektlisten mit Aktionen generieren lassen, die sich darstellen lassen. Bis hin zur gemeinsamen Betrachtung durch Erstellung einer Session, um vor Ort über die visualisierten Ergebnisse diskutieren zu können. Nicht nur zwischen Menschen, auch live Visualisierungen über ausgeführte Tätigkeiten und gedanklich als nächstes auszuführende Handlungen von Agenten während der Bewältigung einer Aufgabe sind denkbar. Und fördern das Verständnis über die Denkweise des gegenwärtigen Agenten. Wo hängt er besonders häufig? Und wie geht er an die Aufgabe heran, sind nur einige Fragen.

Diese Anregungen beschließen meine Bachelorabeit, ich hoffe, Sie haben ebenfalls eine Erkenntnis aus dieser Arbeit ziehen können.

Abbildungsverzeichnis

Glossar

Agent Ein Agent ist ein KI gestützter und autonom handelnder Roboter.

AR Erweiterte Realität kurz AR ergänzt im Sichtfeld die Welt um sogenannte Hologramme, diese werden auf ein Display vor den Augen projiziert.

Hololens Die Hololens ist eine Brille für den Menschen der die AR Funktionalität zugänglich macht.

IAI Institut of Artificial Intelligence; IAI Web Präsenz.

MRTK Das MixedRealityToolkit kurz MRTK ist eine Erweiterung der Unity Engine und unterstützt die Entwicklung Mixed Reality Erfahrungen.

NEEM Narrative-enabled episodic memories kurz NEEM repräsentiert abschnittsweise Erinnerungen eines Agenten.

ROS Roboter Operation System kurz ROS ist eine übergreifendes Open Source Betriebsystem zur Ansteuerung von Robotern.

VR Virtuelle Realität kurz VR beschreibt das eintauchen und interagieren mit Hilfe einer Brille und Kontrollern in einer computergenerierten Welt.

World Anchor Raumanker, erzeugt einen festen wiederkehrenden Punkt relativ im Raum. Es erzeugt ein eigenes Koordinatensystem..

Literaturverzeichnis

[1] Michael Beetz, Daniel Beßler, Sebastian Koralewski, Mihai Pomarlan, Abhijit Vyas, Alina Hawkin, Kaviya Dhanabalachandran, and Sascha Jongebloed. *NEEM Handbook*. University Bremen, Am Fallturm 1, 28359 Bremen. Online; besucht am 12.01.2022; `https://ease-crc.github.io/soma/owl/current/NEEM-Handbook.pdf`.

[2] Andrei Haidu and Michael Beetz. Automated acquisition of structured, semantic models of manipulation activities from human vr demonstration. Technical report, Institute for Artificial Intelligence, University of Bremen, Germany, 2020.

[3] Gayane Kazhoyan, Alina Hawkin, Sebastian Koralewski, Andrei Haidu, and Michael Beetz. Learning motion parameterizations of mobile pick and place actions from observing humans in virtual environments. Technical report, Institute for Artificial Intelligence, University of Bremen, Germany, 2020.

[4] Matthew B. Luebbers, Connor Brooks, Minjae John Kim, Daniel Szafir, and Bradley Hayes. Augmented reality interface for constrained learning from demonstration. Technical report, University of Colorado Boulder Boulder, CO, USA.