HAW HAMBURG

BACHELOR THESIS
Friedrich Dudda

Visualisierung von Systeminformationen autonomer Systeme durch Augmented Reality mithilfe von ARCore und ROS

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Engineering and Computer Science
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Friedrich Dudda

Visualisierung von Systeminformationen autonomer Systeme durch Augmented Reality mithilfe von ARCore und ROS

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Informatik Technischer Systeme*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Stephan Pareigis
Zweitgutachter: Prof. Dr. Philipp Jenke

Eingereicht am: 11. Juli 2022

Friedrich Dudda

Thema der Arbeit

Visualisierung von Systeminformationen autonomer Systeme durch Augmented Reality mithilfe von ARCore und ROS

Stichworte

AR, ARCore, ROS, Android, Unity

Kurzzusammenfassung

Bei der Arbeit mit Robotik-System ist es sowohl in der Entwicklung, als auch in der Anwendung wichtig, die Sensordaten des Systems verständlich darzustellen. Eins der meist verwendesten Frameworks für die Entwicklung von Robotik-Systemen ist das Robot-Operating-System (ROS). In dieser Arbeit wird eine Android Anwendung vorgestellt welche sich mit einem bestehenden ROS-Netzwerken verbinden kann, um die verschiedene ROS-Daten des Systems in Augmented Reality darzustellen. Dabei müssen keine Änderungne an dem bestehenden ROS-System vergenommen werden. Um die visualisierung in Augmented Reality umzusetzen wurde Googles ARCore verwendet. Die Daten werden sowohl in der Umgebung, als auch am Roboter selbst dargestellt und sind fest in der realen Welt verankert. Dadurch können die Daten aus verschiedenen Perspektiven betrachtet werden. Durch die Anwendung kann der tf-Frame des Roboters visualisiert werden. Dabei ist es möglich mit dem virtuellen tf-Frame zu interagieren, um so die einzelnen Bauteile des Roboters analysieren zu können. Zusätzlich können Point-Cloud-Daten des Roboters in Augmented Reality visualisiert werden. Die Daten werden dabei direkt in die Umgebung, in der sie aufgenommen wurden projiziert. Dadurch kann ein gztes räumliches Verständnis für die Daten hergestellt werden. Über die GUI der Anwendung kann dabei festgelegt werden welche Daten Visualisiert werden sollen, zusätzlich können verschiedene Einstellungen an der Visualisierung vorgenommen werden. Die Funktionsweise der Anwendung wird mit einer Reihe von Versuchen geprüft, als Test Geräte wurden Dafür Samsung Galxy Tab S7+ und ein Google Pixel 6 verwendet. Als Autonomes System wurde der Husky aus dem TIQ Projekt genutzt.

Friedrich Dudda

Title of Thesis

Visualization of system information of autonomous systems through augmented reality using ARCore and ROS

Keywords

AR, ARCore, ROS, Android, Unity

Abstract When working with a robotic system, it is important, both in development and in application, to present the system's sensor data in an understandable way. One of the most commonly used frameworks for developing robotic systems is the Robot Operating System (ROS). In this work, an Android application is presented which can connect to an existing ROS network to display the various ROS data of the system in augmented reality. No changes need to be made to the existing ROS system. Google's ARCore was used to implement the visualization in augmented reality. The data is displayed both in the environment and on the robot itself and is firmly anchored in the real world. This allows the data to be viewed from different perspectives. The tf frame of the robot can be visualized, through the application. It is possible to interact with the virtual tf frame in order to be able to analyze the individual components of the robot. In addition, point cloud data from the robot can be visualized in augmented reality. The data is projected directly into the environment in which it was recorded. This allows a complete spatial understanding of the data to be established. The GUI of the application can be used to specify which data is to be visualized, and various settings can be made to the visualization. The functionality of the application is checked with a series of tests, the devices used are the Samsung Galaxy Tab S7+ and the Google Pixel 6. The Husky from the TIQ project was used as the autonomous system.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation

Das Robot-Operating-System (ROS) [20] ist eines der meist verwendeten Framworks für die Entwicklung und Forschung in der Robotik. ROS ist vielfältig einsetzbar und ermöglicht es, Kommunikationskanäle zwischen den Komponenten eines Systems strukturiert und einfach aufzubauen. [20] Bei der Arbeit mit Robotern ist es sowohl in der Entwicklung als auch in der Anwendung wichtig, die Sensordaten des Roboters geeignet zu visualisieren, um den Status des Roboters verstehen zu können und gegebenes Fehlverhalten erklären zu können. Ein beliebtes Software-Tool um diese Visualisierung in ROS umzusetzen ist RViz. RViz bietet die Möglichkeit, bei der Entwicklung von Robotik Anwendung mit ROS, Sensordaten des Roboters auf einem Computer-Bildschirm darzustellen.

Allerdings bringt diese Form der Visualisierung auch einige Nachteile mit sich. In der Robotik sind die zu visualisierenden Daten eines Roboters oft an eine Position in seiner Umgebung gebunden, wie beispielsweise erkannte Markierungen oder Hindernisse. Eine verständliche und intuitive Darstellung dieser Daten auf einem 2D Computer-Bildschirm ist schwer umzusetzen, da dafür ein räumliches Verständnis für die Daten hergestellt werden muss. So ist es über einen Computer-Bildschirm beispielsweise schwierig zu validieren, ob die 3D-Position eines erkannten Features korrekt ist. Eine Darstellung der Daten an der Position, an der sie erzeugt wurden, kann wertvolle Kontextinformationen liefern und dabei helfen, die Daten besser einzuordnen.

Ein weiteres Problem stellt die Realtime Auswertung von Daten dar. Ein einzelner Entwickler kann nicht gleichzeitig die Daten auswerten und die Ausführung des Roboters beaufsichtigen, weil der Fokus bei der Analyse der Daten auf dem Bildschirm und nicht auf dem Roboter liegt. Dadurch wird ein sicheres und effizientes Arbeiten beeinträchtigt.

AR bietet die Möglichkeit, die Systemdaten von Robotern in die reale Welt, den Arbeitsbereich des Roboters zu übertragen. Dadurch kann die Interaktion zwischen Menschen und Robotern intuitiver und effizienter gestaltet werden.

Durch Googles ARCore und Apples ARKit besteht mittlerweile die Möglichkeit ein einfaches Smartphone oder Tablet aus dem täglichen Gebrauch als Hardware zur Ausführung einer AR-Anwendung zu verwenden. Dadurch steht AR einer breiteren Masse an Nutzern zur Verfügung und gewinnt somit immer mehr an Bedeutung. [17] Durch AR ergeben sich neue und interessante Möglichkeiten, Daten intuitiver und zugänglicher zur Verfügung zu stellen und die Daten Auswertung interaktiver zu gestalten.

1.2 Zielsetzung

Im Rahmen dieser Bachelorarbeit soll eine AR-Anwendung entwickelt werden, mit welcher Systeminformationen autonomer Systeme aus ROS visualisiert werden können. Dafür wird mithilfe von ARCore und der Unity-Game-Engine eine Anwendung für Android entwickelt. Mit dieser Anwendung sollen Informationen eines autonomen Roboters in Augmented Reality dargestellt werden.

Für die Visualisierung der Daten werden zwei verschiedene Arten von Anwendungsgeräten in Form eines Handys und eines Tablets verwendet. Anhand dessen soll abschließend die Nutzbarkeit der Anwendung auf den verschiedenen Geräten betrachtet werden. Als autonomes System dient in dieser Arbeit der Husky[1] aus dem TIQ-Projekt der HAW-Hamburg.

Die AR-Anwenung soll Aufschluss über den Aufbau und die einzelnen Komponenten des Huskys geben. Zusätzlich sollen über die Anwendung Informationen zum aktuellen Zustand des Huskys dargestellt werden. Dadurch soll die Interaktion und das Arbeiten mit dem Husky intuitiver und einfacher gestaltet werden.

1.3 Gliederung

Diese Arbeit ist in mehrere Kapitel unterteilt. Im zweiten Kapitel werden zunächst einige Grundlagen und relevante Technologien für diese Arbeit vorgestellt. Anschließend werden vergleichbare Arbeiten und Ansätze betrachtet und von dieser Arbeit abgegrenzt, um ein

genaueres Bild des geplanten Vorhabens zu schaffen und die Arbeit in den fachlichen Kontext einzuordnen. Daraufhin werden Anforderungen an der Anwendung noch einmal genauer definiert. Das dritte Kapitel befasst sich mit dem Entwurf des Systems. Dabei wird der grundlegende Aufbau des Systems, sowie der Aufbau einzelner Komponenten dargestellt. In Kapitel vier werden mehrere Versuche zur Überprüfung des Systems vorgestellt. Dabei wird zunächst der Aufbau der Versuche beschrieben. Im Anschluss werden die Ergebnisse der beschriebenen Versuche ausgewertet. In Kapitel fünf werden noch einmal die wichtigsten Inhalte der Arbeit zusammengefasst und ein Fazit gezogen. Abschließend wird in Kapitel sechs ein Ausblick auf möglich weiter Entwicklungsschritte und andere Einsatzmöglichkeiten der hier verwendeten Techniken gegeben.

2 Grundlagen und vergleichbare Arbeiten

2.1 Zentrale Begriffe

ROS

Das Robot-Operating-System ROS stellt eine Menge von Biblitheken und Werkzeugen zur verfügung, die das Entwicklung von Robotik-Systemen vereinfachen. Außerdem bietet ROS ein Kommunikations-Netzwerk, über welches Daten zwischen verschiedenen Systemen ausgetauscht werden können. Ein ROS-Netzwerk besteht dabei aus verschiedenen Nodes und einem ROS-Master. Die Nodes können Daten über das ROS-Netzwerk bereitstellen und empfangen. Die Kommunikation zwischen den Nodes basiert auf einem Publish/Subscribe-Netzwerk. Die verfügbare Daten werden als Topics dargestellt. Wenn ein Node Daten publishen möchte, stellt er diese über ein Topic mithilfe des Masters bereit. Um Daten zu empfangen kann ein verfügbares Topic subscribed werden [20].

iViz-ROS-API

Wie bereits erwähnt werden Android-Geräte standardmäßig nicht von ROS unterstützt. Für die Kommunikation mit anderen ROS-Systemen wird eine ROS-Schnittstelle benötigt, welche die Verbindungen zu den anderen Peer-Knoten handhabt und die ROS-Nachrichtenserialisierung umsetzt. Da Android-Geräte nativ nicht von ROS unterstützt werden, wird hier die „iviz ROS API" verwendet. Mithilfe der „iviz ROS API" können sich Android-Geräte mit ROS-Netzwerken verbinden. Anschließend kann die Liste aller im ROS-Netzwerk verfügbaren Topics ausgelesen werden. Über die Topics können ROS-Nachrichten wie üblich empfangen (subscribed) und gesendet (published) werden [23].

Die empfangenen ROS-Nachrichten werden von der „iviz ROS API" als C# Objekt bereitgestellt. Dabei gibt es für jeden Nachrichtentyp eine eigene Nachrichten-Klasse. Über

eine Nachrichten-Klasse kann auf alle Informationen einer ROS-Nachricht zugegriffen werden. Je nach Datentyp werden dementsprechend unterschiedliche Daten bereitgestellt. Die iviz-ROS API bietet eine Reihe an Standard ROS-Nachrichtentypen, es können aber auch eigene Nachrichtentypen für einen spezifischen Anwendungsfall ergänzt werden. Alle Nachrichtentypen, die in dieser Anwendung verwendetet wurden, sind bereits in der iviz ROS API umgesetzt [23].

ROS-tf2-Frames

Ein Robotik-System besteht üblicherweise aus mehreren Bauteilen, wie einem Gripper oder einer Kamera. Um die relative Position der Bauteile zueinander bestimmen zu können werden Frames verwendet. Diese Frames werden von der ROS Bibliothek tf2 in einer Baumstruktur gespeichert. Ein Frames stellen dabei das Koordinatensystem des zu gehörigen Bauteils dar. Beispielsweise stellt der Kamera-Frame das Koordinatensystem der Kamera dar, in welchem diese ihre Daten bereitgestellt. Typischerweise gibt es eine World-Frame, eine Base-Frame und die Frames der einzelnen Bauteile. Der World-Frame beschreibt den Ursprung des Weltkoordinatensystems und steht an oberster Stelle der Baumstruktur. Ein Base-Frame bildet die Basis für einen Roboter und ist dem World-Frame untergeordnet. Unter dem Base-Frame befinden sich alle Frames der Bauteile des Roboters. Mit der Bibiliothek tf2 lassen sich Punkte oder Vektoren zwischen verschiedenen Koordinaten-Frames transformieren [13].

ROS-PointCloud2

Durch Point-Clouds können Objekte oder ganze Umgebungen in Form von Tiefendaten dargestellt werden. Ein Point-Cloud besteht dabei aus vielen einzelnen Punkten, welche jeweils ein x, y und z Koordinaten haben. Die Punkte stellen dabei einzelne Messpunkte einer Tiefkamera dar. Setzt man alle aufgenommen Punkte zusammen, so erhält man eine Tiefendarstellung des Aufgenommen Objektes, oder der Umgebung [21].

ARCore

ARCore ist ein Framework für die Entwicklung von Augmented Reality Anwendungen auf Android. Mit ARCore können virtuelle Objekte realistisch in der Umgebung darstellt werden. Dafür bietet ARCore eine Reihe an nützlichen Funktionen und Werkzeugen, die

zur Entwicklung einer Augmented Reality Anwendung benötigt werden. Die wichtigsten Funktionen für diese Anwendung sind das „Motion-Tracking" und das „Environmental Understanding" [7].

Environmental Understanding: ARCore sucht in der Umgebung nach flachen Oberflächen, auf welchen virtuelle Objekte platziert werden können. Dafür wird nach Clustern von Feature-Punkten, welche auf einer ähnlichen horizontalen oder vertikalen Ebene liegen, gesucht. Diese ist beispielsweise bei Oberflächen von Wänden oder Tischen der Fall. Die gefundenen Cluster werden als geometrischen Ebenen dargestellt, auf diesen Ebenen können virtuelle Objekte platziert werden. Das Verständnis der Umgebung wird dabei stetig durch Erkennung weiterer Features und Flächen verbessert. Da ARCore Feature-Points verwendet um Oberflächen zu erkennen, werden Texturlose Flächen wie weiße Wände häufig nicht als Oberfläche erkannt [7].

Motion-Tracking: Um das Motion-Tracking umzusetzen und die Position des Gerätes in der Umgebung zu bestimmen, wird ein SLAM Algorithmus [12] verwendet. Durch das Motion-Tracking können virtuelle Objekte fest in der Umgebung platziert werden. Dadurch behalten sie ihre Position bei, wenn das Anwendungsgerät bewegt wird. Auch wenn die Objekte aus dem Sichtfeld der Kamera verschwinden bleiben sie fest an ihrer Position verankert [7].

Occlusion

Über die Depth-API von ARCore kann aus einem Kamerabild ein Tiefenbild erzeugt werden. Um die Features der Depth-API verwenden zu können muss das Gerät die Depth-API unterstützen. Ob ein Geräte die Depth-API unterstütz oder nicht kann der Liste der von ARCore unterstützten Geräte entnommen werden[4]. Das Teifbild wird anhand der Bewegung des Gerätes berechnet. Zusätzlich können Daten eines Tiefen-Sensor mit einbezogen werden, beispielsweise die Daten eines ToF-Sensors. Es wird aber keine Tiefensensor benötigt um die Depth-API zu unterstützen. Da die Depth-API viel Rechenleistung benötigt ist sie standardmäßig deaktiviert und muss manuell in ARCore aktiviert werden [5].

Mithilfe des Tiefenbildes können virtuelle Objekte realistischer dargestllt werden. Die Depth-API ermöglicht es virtuelle Objekte hinter realen Objekten darzustellen. Die vir-

tuelle Objekte werden dann von den realen Objekte verdeckt. In Abbildung 2.1 ist eine Android-Figur zu sehen, welche in der Szene platziert wurde. Auf dem linken Bild ist die Android-Figur ohne Objektverdeckung dargestllt. Auf der rechten Seite ist die Android-Figur mit Objektverdeckung zu sehen. Ohne die Verdeckung ist deutlich zu erkennen, dass es sich bei der Android-Figur um ein virtuelles Objekt handelt. Mit der Verdeckung fügt sich die Android-Figur deutlich realistischer in die Umgebung ein [5].

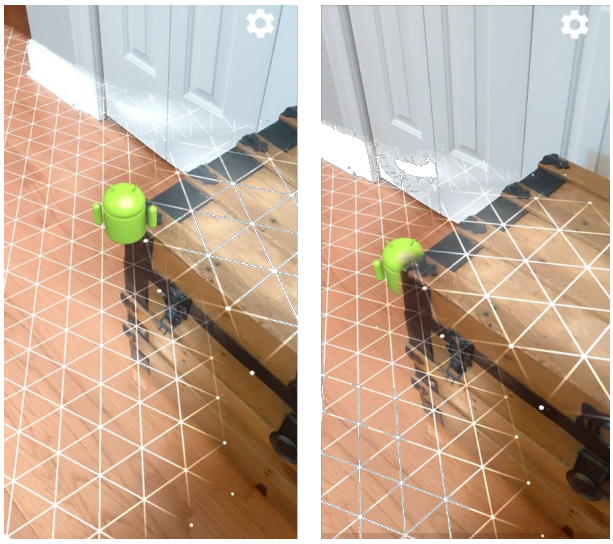

Abbildung 2.1: Die Darstellnug zeigt eine virtuelle Android-Figur die in der Umgebung platziert wird, im linken Bild ist Occlusion deaktiviert, im rechten Bild ist Occlusion aktiviert [5].

Raycast

Damit virtuelle AR-Inhalte realistisch wirken und sich wie reale Objekte verhalten, müssen die Inhalte in der betrachteten Szene richtig positioniert und skaliert werden. Das bedeutet beispielsweise, dass Objekte, die weiter weg sind, kleiner dargestellt werden müssen. Um die korrekte Position eines 3D-Objektes in der Umgebung festzustellen, kann ein sogenannter „raycast" oder auch „hit-test" durchgeführt werden. Der „hit-test" wird aber auch für die Interaktion mit virtuellen Objekten benötigt, um so das ausgewählte Objekt zu bestimmen.

Bei einem „hit-test" wird ein virtueller Strahl bestehend aus einem Ursprung und einer Richtung in die Szene projiziert. Das Ziel des „hit-tests" ist es, Überschneidungen zwischen Flächen oder Objekten und dem projizierten Strahl zu finden (Abbildung 2.2).

Wenn der Strahl eine Fläche oder ein Objekt schneidet, dann wird der Schnittpunkt in Weltkoordinaten zurückgegeben. Dabei kann es dazu kommen, dass der Strahl mehrere Flächen oder Objekte schneidet. Daher wird eine Liste von „hit results" zurückgegeben, welche alle gefundenen Schnittpunkte beinhaltet. Die Liste der Schnittpunkte ist dabei nach Entfernung der Schnittpunkte sortiert. Die „hit result" Liste beinhaltet für jeden Schnittpunkt zusätzlich die Information, welches Objekt getroffen wurde [8].

Als Ursprung der Hit Results wird häufig das Gerät des Anwenders verwendet, um Eingaben auf dem Touch-Display in reale Weltkoordinaten zu übertragen, wie in Abbildung 2.2 dargestellt.

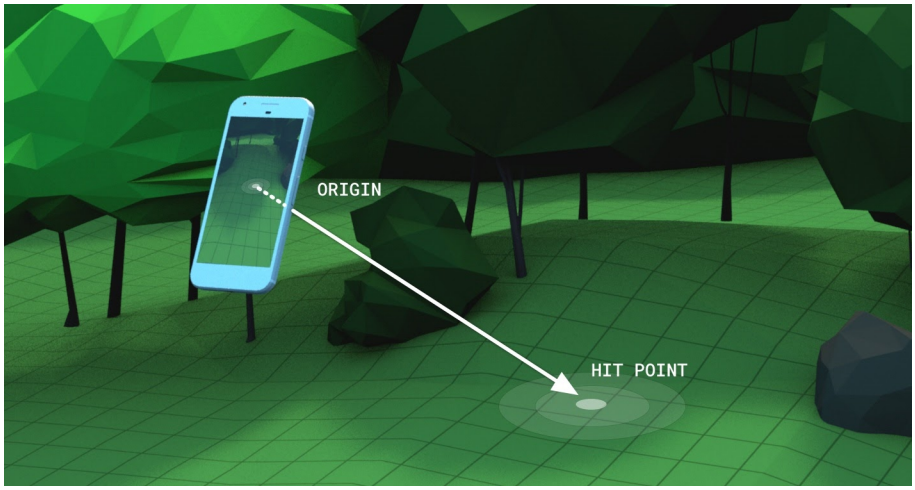

Abbildung 2.2: Raycast mit dem Anwendungsgerät als Ursprung, um die Toucheingabe in Weltkoordinaten zu übertragen [5]

Unity-Programmiermodell

Für die Entwicklung der in dieser Arbeit beschriebenen AR-Anwendung wurde die Unity-Game-Engine als Entwicklungsplattform verwendet. Mit Unity lassen sich graphische Anwendungen und 3D-Welten einfach modellieren und umsetzten, dabei können auch komplexere Funktionalitäten, wie beispielsweise Augmented Reality eingebaut werden. Das Programmiermodell von Unity bringt einige Besonderheiten mit sich, daher werden die grundlegenden Elemente des Unity-Programmiermodell im Folgenden kurz erläutert.

Project: *Project* ist der Überbegriff für die Gesamte Anwendung und fasst somit alle Komponenten des Systems zusammen.

Scene-Graph: Ein Unity *Project* besteht aus mehreren *Szenen*. Eine *Szene* Repräsentiert dabei die Nutzeroberfläche der Anwendung. In der Regel setzt sich eine *Szene* aus mehreren *Game-Objects* zusammen, welche hierarchisch angeordnet sind. Die Anordnung der *Game-Objects* wird im *Scene-Graph* festgehalten. Innerhalb des *Scene-Graphs* herrscht eine „Eltern-Kind-Beziehung", das bedeutet, dass *Game-Objects* einer höheren Ebene, sogenannte „Eltern-Objekte", ihre Position und Orientierung an unterliegende „Kind-Objekte" vererben.

Game-Obejcts: *Game-Obejcts* stellen das zentrale Konzept von Unity dar. Jedes Element, was in einer Anwendung dargestellt oder verwendet werden soll, muss in Unity als *Game-Object* definiert werden. Dabei ist ein *Game-Object* lediglich einen Container ohne die eigentliche Funktionalität.

Components: In den *Components* wird die eigentliche Funktionalität der *Game-Objects* umgesetzt. Ein *Game-Object* kann dabei aus Verschiedenen *Components* bestehen. So können beispielsweise Animationen, Texturen oder ein bestimmtes Verhalten durch *Skripte* eingebaut werden.

Scripting: Um Funktionalitäten umzusetzen, welche sich nicht mit den vordefinierten Komponenten aus Unity abbilden lassen, gibt es *Skript* Komponenten. Darin kann das gewünschte Verhalten in den Programmiersprachen C# implementiert werden. *Skripte* werden nur ausgeführt, wenn diese an ein *Game-Object* gebunden sind.

Prefabs: *Prefabs* sind eine spezielle Art von *Game-Objects*. Sie können mehrfach in Form von Kopien wiederverwendet werden. *Prefabs* stellen dabei das Original für die Kopien des Objektes dar. Nimmt man Änderungen am Prefab vor, so werden die Änderungen automatisch in allen Kopien übernommen.
Die Entwicklung mit Unity hat den Vorteil, dass verschiedene Betriebssysteme und Geräte als Zielplattform gewählt werden können, ohne dafür große Änderungen im Code vornehmen zu müssen. Des Weiteren bietet Unity eine Vielzahl an nützlichen Plugins für die Entwicklung von 3D-Anwendung [3].

Unity bietet mit AR-Foundation ein Framework für die Entwicklung von Augmented Reality Anwendungen. AR-Foundation umfasst die Kernfunktionen von ARCore und bietet dazu noch weitere Funktionen für die Entwicklung von AR-Anwendungen [18].

2.2 Technischer Hintergrund

In diesem Abschnitt wird die verwendete Hardware vorgestellt und die Auswahl der Hardware begründet.

2.2.1 Auswahl der Android-Geräte

Bei der Auswahl der Anwendungsgeräte müssen verschiedene Aspekte betrachtet werden, um sicherzustellen, dass die entwickelte Anwendung auf dem gewählten Gerät nutzbar ist. Dazu wird zunächst beschrieben worauf bei der Auswahl der Anwendungsgeräte geachtet werden muss. Anschließend werden die in dieser Arbeit verwendeten Geräte vorgestellt. Dabei wird auch begründet, warum die Geräte gewählt wurden.

ARCore supported devices

Damit die Funktionen von ARCore verwendet werden können muss das Gerät von ARCore unterstützt werden. Um eine Unterstützung zu erhalten werden die Kamera, die Bewegungssensoren und die Rechenleistung des Gerätes Überprüft. Dadurch kann gewährleistet werden, dass Funktionen wie das Motion-Tracking, was durch die Kombination der Kamera- und der Bewegungssensor-Daten umgesetzt wird, wie erwartet funktioniert. Alle Unterstützten Geräte werden in einer Tabble aufgelistet [4].

AR-Required vs. AR-Optional

Eine AR-Anwendung kann entweder als AR-Required oder AR-Optional konfiguriert werden. AR-Required Apps können ohne ARCore nicht verwendet werden, da die Gesamte Funktionalität der Anwendung in Augmented Reality dargestellt wird. In AR-Optinal Anwendungen wird Augmented Reality als zusätzliches Feature genutzt, welches nicht

zwingend für die Ausführung benötigt wird. Typischerweise gibt es hier die Möglichkeit einen AR-Modus ein oder auszuschalten. Die Informationen können also auch ohne Augmented Rality dargestellt werden [6].

AR-Required AR-Required Anwendung sind nur auf Geräten verfügbar, welche von ARCore unterstütztes werden. Für die Ausführung der Anwendung müssen zusätzlich die „Google Play Services for AR" auf dem Gerät installiert sein. Die „Google Play Services for AR" werden bei der Installation von AR-Required Apps automatisch mit installiert. Um AR-Required Apps nutzen zu können muss das Gerät zusätzlich mindesten auf der Android-Version 7.0 sein [6].

AR-Optional Anwendungen die AR-Optional sind, können auch auf Geräten genutzt werden, welche nicht von ARCore unterstützt werden. Auch werden die „Google Play Services for AR" nicht zwingend benötigt. Um den AR-Modus verwenden zu können gelten allerdings die gleichen Bedingungen wie bei AR-Required Anwendungen. Die „Google Play Services for AR" werden bei der Installation einer AR-Optional Anwendung nicht mit installiert. Für die Ausführung einer AR-Optional Anwendungen ist die Android-Version 4.0 ausreichend [6].

Verwendete Geräte

In dieser Arbeit wird eine AR-Required Anwendung entwickelt. Daher müssen die gewählten Geräte von ARCore unterstützt werden. Zusätzlich verwendet die Anwendung die Depth-API (siehe Kapitel 2.1). Die Geräte müssen also auch genügend Rechenleistung bieten. Da die Anwendung auf alltäglichen Geräten nutzbar sein soll und diese meist keine Tiefensensoren beinhalten, wurden hier ebenfalls Geräte ohne zusätzliche Tiefensensoren gewählt.

Als Anwendungsgeräte wurden hier ein Google Pixel 6 und ein Samsung Galaxy Tab S7+ gewählt. Durch die Wahl eines Handys und eines Tablets, kann die Auswirkung der Displaygröße auf die nutzbarkeit der Anwendung untersuchen werden. Beide Geräte waren zum Zeitpunkt der Anschaffung das neuste und leistungsstärkste Modell des Herstellers. Die Geräte werden beide von ARCore unterstützt und unterstützen zusätzlich die Depth-API [4].

2.2.2 Autonomes System

Als Autonomes System wird der Husky Roboter des Forschungsprojektes „Testfeld Intelligente Quartiersmobilität“ (TIQ) [2] verwendet. Der Husky ist ein Industrieroboter welcher in der Lage sein soll sich autonom über den Campus der HAW zu bewegen. Er soll dabei kleine Aufgaben, wie das Ausliefern von Pakten übernehmen. Um autonom über den Campus zu navigieren und Aufgaben zu erledigen ist der Husky mit eine Reihe von Sensoren ausgestattet [14].

Wie in Abbildung 2.3 zu sehen ist besteht der Husky aus einer Grundplatform mit vier Rädern. Auf der Grundplattform befindet sich ein schwenkbarer Greifarm. Mit Hilfe des Greifarms kann der Husky mit seiner Umwelt interagieren und so beispielsweise Türen öffnen. An dem Greifarm ist eine intel-realsense-D435 angebracht. Diese liefert sowohl Bild, als Tiefen-Daten [1].

Für die Entwicklung des Huskys wird das software-Framework ROS (Robot-Operating-System) eingesetzt [20]. Die Steuerungssoftware des Huskys ist in C++ und Python umgesetzt [1].

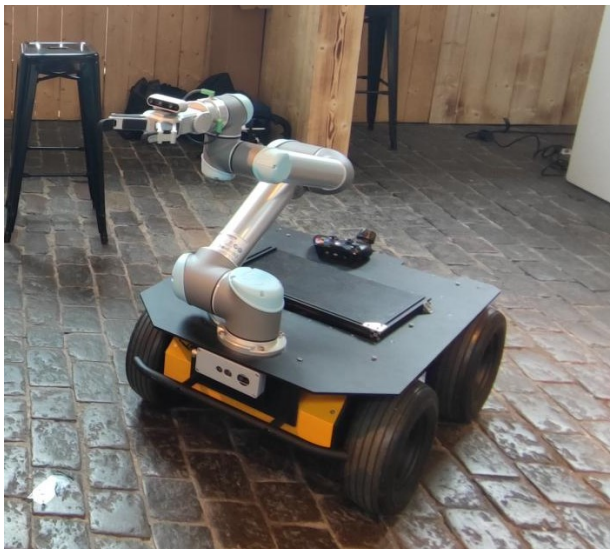

Abbildung 2.3: In der Abbildung ist der Husky Roboter des TIQ[2] zu sehen

2.3 Vergleichbare Arbeiten

Es bestehen bereits mehrere Möglichkeiten Daten aus ROS zu visualisieren, um einen genaueren Einblick in die Funktionsweise von Robotern zu ermöglichen. Die meisten Visualisierungen beschränken sich dabei aber auf eine Darstellung auf einem 2D Computer-Monitor. Es existieren jedoch auch einige Anwendungen, welche zur Darstellung der ROS Daten bereits Augmented Reality verwenden, ähnlich wie auch in dieser Arbeit umgesetzt.

Im folgenden Kapitel werden vergleichbar Arbeiten vorgestellt, welche bezüglich des Theams und der gewählten Ansätze Ähnlichkeiten zu dieser Arbeit aufweisen. Dabei wird das Grundlegende Konzept und die Funktionsweise der vergleichbaren Arbeiten erläutert. Anschließend findet eine Abgrenzung zu der in dieser Arbeit beschriebenen AR-Anwendung statt.

2.3.1 ARviz

ARviz ist einen stand-alone AR-Visualisierungs-Plattform für Robotik-Anwendung, welche mit dem Robot-Operating-System (ROS) entwickelt werden. Zusätzlich zur Visualisierung der Daten ist es mithilfe von ARviz auch möglich, mit den Robotern zu interagieren. Als Visualisierungs-Hardware dient dabei die Hololens 2, eine stand-alone AR-Brille von Microsoft. ARviz ist als eine erweiterbare AR visualisierungs Plattform aufgebaut, welche aus verschiedenen Plugins besteht. Es soll eine Reihe an standard Plugins geben, welche die standard ROS Nachrichten Typen abdecken um die Daten in AR zu visualisieren. Zusätzlich dazu soll es aber auch möglich sein eigene Plugins zu entwickeln, um das System so um neue Datentypen zu Erweitern. Somit können für spezifische Anwendungsfälle eigene ROS-Nachrichten Typen visualisiert werden und eigene Interaktionsmethoden definiert werden.

Plugins

Die Arviz Plugins können grundsätzlich in zwei verschiedene Kategorien unterteilt werden, die „Display Plugins” und die „Tool Plugins”.
Display Plugins sind Komponenten, welche für die Visualisierung der Daten aus ROS

zuständig sind. Damit können beispielsweise Navigations-Pfade und andere Systeminformationen visualisiert werden. Die zu visaualisierenden Daten aus ROS werden durch einen Subscriber entgegen genommen und gesammelt. Anschließend werden die aus den Daten entstehenden visuellen Informationen und Elemente mit einer Frequenz von 20Hz in das Sichtfeld des Anwenders gerendert.
Tool Plugins sind Komponenten, welche intuitive Eingabemöglichenkeiten Beispielsweise über Sprache und Gesten ermöglichen, um mit dem ROS System zu interagieren. Mithilfe von Tool Plugins können Befehle an den Roboter gesendet werden, um diesen beispielsweise im Raum zu bewegen oder eine Aufgabe ausführen zu lassen. Die Befehle werden über ROS mithilfe eines Publishers an den Roboter übermittelt.

Die Plugins sind dabei so aufgebaut, dass sie modular einsetzbar sind und in anderen ROS-Anwendung mit gleichen Datentypen einfach wieder verwendet werden können.

Für die Auswahl der Plugins und die Interatkion mit dem ROS System stellt ARviz ein Hand-Menü bereit, auf welches durch anheben der Hand zugegriffen werden kann. Durch das Hand-Menü kann ausgewählt werden welche Display Plugins angezeigt werden sollen und welche Tool Plugins (Eingabemöglichenkeiten) aktiv seien sollen. Außerdem können so einfache Konfigurationen an den Display Plugins und den Tool Plugins vorgenommen werden.

Da ARviz mit der Unity Game Engine umgesetzt wurde, einer Plattform, die ROS-APIs zum Aufbau einfacher Kommunikationskanäle nicht nativ unterstützt, muss hier eine Lösung gefunden werden. Der Komunikationskanal zwischen ARviz und der ROS-Anwendung wird auf Unity Seite durch das Package "ROS#"[1] umgesetzt und auf ROS Seite kommt das Package "Rosbridge"[2] zum Einsatz. Dadurch kann ARviz mit jeder ROS-Anwendung über ein gemeinsames Standard-JSON-String-Format über TCP/IP kommunizieren. Um die übertragenen Daten korrekt in der Umgebung zu lokalisieren, werden zwei Koordinaten Systeme betrachtet. Das virtuelle Koordinaten System, in welches die Virtuellen Objekte gerendert werden und das reale Koordinaten System, in welchem physishe Entitäten, wie der Roboter exisistieren. Die Daten aus den Display Plugins müssen aus dem virtuellen Koordinaten System in das reale Koordinaten System transformiert werden. Dadurch werden die virtuellen Elemente und Daten an geeigneten Stellen in der realen Welt dargestellt [16].

[1] https://github.com/siemens/ros-sharp
[2] http://wiki.ros.org/rosbridge_suite

Abgrenzung

Sowohl bei ARviz, als auch in der hier entwickelten Arbeit wird eine Visualisierung von ROS-Nachrichten mithilfe von Augmented Reality umgesetzt. Dabei wird in beiden Projekten zur Entwicklung der AR-Anwendung die Unity Game Engine verwendet.

Während bei ARviz sowohl die Visualisierung der ROS-Nachrichten, als auch Interaktion mit den Robotern möglich sein soll, wird der Fokus dieser Arbeit auf die Visualsierung der Daten aus ROS gelegt. Eine Interaktion mit dem Roboter, wie bei ARviz wird hier nicht umgesetzt. Dafür werden hier zusätzlich zu den aus ROS bezogenen Daten auch Informationen zu der Hardware und dem Aufbau des Roboters in AR dargestellt. Des Weiteren wird bei ARviz für die Visualisierungs-Hardware auf teure AR-Brillen zurückgegriffen, in dieser Arbeit werden für die Visualisierung mobile Endgerät in Form eines Handys und eines Tablets genutzt.

2.3.2 Iviz

Iviz ist eine mobile Andwendung mit welcher Daten aus Robotik-Anwendung, die mit dem Robot Operating System (ROS) entwickelt werden, in Augmented Reality visualisiert werden können. Dabei ist diese Platform speziell auf Android und IOS Smartphones und Tablets ausgelegt. Mit Iviz soll es möglich sein viele verschiedene ROS Datentypen geeignet in Augmented Reality zu visualisieren.

Für die Entwicklung von iviz wurde die Unity Game Engine verwendet. Um eine Verbindung zwischen dem ROS Netzwerk und der Unity Engine aufzubauen, wurden einige der ROS Funktionalitäten für Unity neu implementiert. Diese Funktionalitäten umfassen unter anderem eine schnelle und effiziente Serialisierung und Deserilsierung von ROS Nachrichten, sowie eine Implementierung der ROS API. Die neuen Funktionalitäten wurden in externe Pagackes ausgelagert, damit diese in anderen Anwendung wiederverwendet werden können.

Abgrenzung

In iViz können die Daten auch ohne AR dargestellt werden, es handelt sich bei iViz also um eine AR-Optinal Anwendung. In dieser Arbeit hingegen eine AR-Required Anwednug entwickelt. Des Weiteren liegt der Fokus von Iviz nicht darauf die Daten im Arbeitsbereich

des Roboters zu visualiseren. Stattdessen soll die Daten Primär auf flachen Oberflächen wie einem einem Tisch dargestellt werden.

2.4 Anforderungen

Durch die Android Anwendung soll es möglich sein, Daten aus ROS in Augmented Reality auf einem mobilen Endgerät zu visualisieren. Die Anwendung soll auf verschiedenen Gerät-Typen nutzbar sein. Dabei soll die Visualisierung Aufschluss über den Aufbau und die Bauteile des Huskys geben. Zusätzlich sollen einige Sensordaten des Huskys dargestellt werden, um die Wahrnehmung seiner Umgebung verständlich zu machen. In der Anwendung soll dafür auswählbar sein, welche Daten angezeigt werden sollen. Um Informationen zum Aufbau des Huskys zu liefern, sollen in Augmented Reality Informationen auf die jeweiligen Bauteile projiziert werden. Sensor Daten sollen je nach Sensor Typ direkt am Sensor, oder in die Umgebung des Huskys dargestellt werden. Dafür müssen die Daten des Huskys aus dem Koordinatensystem des Huskys in das Koordinatensystem des Android-Gerätes transformiert werden. Um dies umzusetzen, muss die relative Position des Huskys zum Android-Gerät festgestellt werden. Damit sich die Daten nicht überlagern und besser analysiert werden können, ist es wichtig, dass verdeckte Bauteile/Sensorwerte nicht angezeigt werden.

3 Entwurf

In diesem Kapitel wird die Architektur und die Umsetzung der Android Anwendung beschrieben. Dazu werden alle Komponenten des Systems einzeln beschrieben. Dabei wird jeweils aufgezeigt welche Funktionen aus ARCore und ROS verwendet werden. Zusätzlich wird der Aufbau der Anwendung in Unity erläutert.

3.1 Systemübersicht

Das beschriebene System setzt sich aus mehreren Hardware und Softwarekomponenten zusammen. Der grundlegende Aufbau des Systems, die Verteilung der Hardware- und Softwarekomponenten und die Kommunikationswege zwischen den verschiedenen Hardwarekomponenten werden im Verteilungsdiagramm in Abbildung 3.1 dargestellt.

Das System besteht aus zwei Hardwarekomponenten, dem Husky-Roboter und einem ARCore fähigen Android-Gerät. Die Kommunikation zwischen den beiden Hardwarekomponenten erfolgt über Wi-Fi auf Basis von ROS.

Husky Der Husky-Roboter ist für die Erzeugung und Bereitstellung der Daten zuständig. Dafür „published" der Husky aufgenommene Sensorwerte, sowie Systeminformationen über seinen Aufbau und seinen aktuellen Zustand per ROS.

Android-Gerät Das Android-Gerät dient zur Visualisierung der Daten, die vom Husky bereitgestellt werden. Um die Daten des Huskys über ROS entgegen zu nehmen, wird die iViz-ROS-API verwendet (siehe Kapitel 2.1). Für die Darstellung der Daten in Augmented Reality werden verschiedene Funktionalitäten aus ARCore eingesetzt. Aufgaben wie die Auswahl der anzuzeigenden Daten und die Transformation der Daten in das korrekte Koordinatensystem werden von der „ROS AR-VIZ" Komponente übernommen. Der

Aufbau und die Umsetzung der „ROS AR-VIZ" Komponente werden anschließend noch einmal genauer betrachtet.

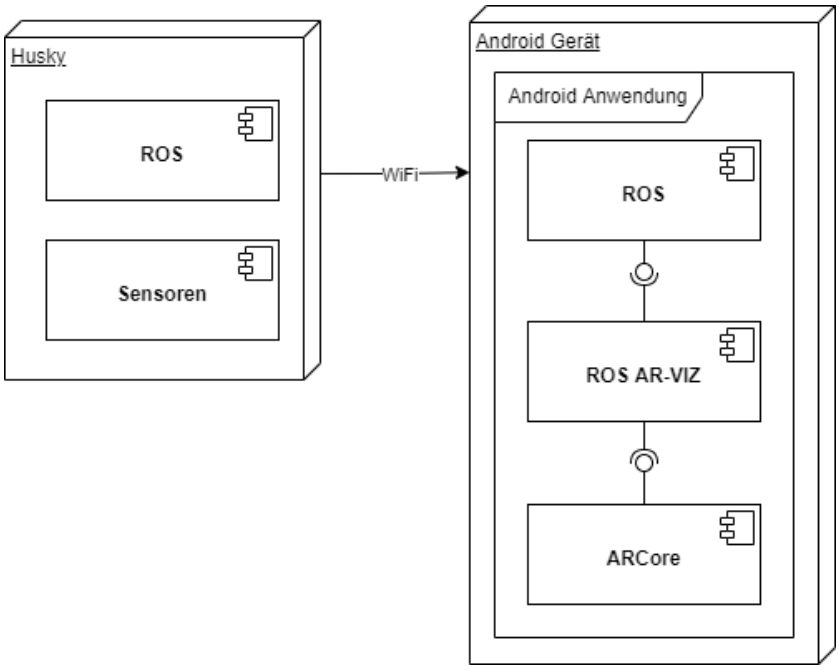

Abbildung 3.1: Der Gesamtaufbau des Systems dargestellt als Verteilungsdiagramm, ein Überblick über die eingesetzten Hardware- und Softwarekomponenten.

3.2 Komponentenübersicht

Im Folgenden werden die Funktionsweise und die Komponenten der Android-Anwendungen aus Abbildung 3.1 genauer erläutert. Die Anwendung ist in drei Hauptkomponenten unterteilt, „ROS" zum Empfangen der Daten, „ARCore" zur Unterstützung der Visualisierung der Daten und die „ROS AR-VIZ" Komponente, welche in weitere Teilkomponenten unterteilt ist. Der Fokus liegt darauf, wie die „ROS AR-VIZ" Komponente umgesetzt ist und wie ROS und ARCore eingesetzt werden, um die in Kapitel 2.4 gestellten Anforderungen zu erfüllen.

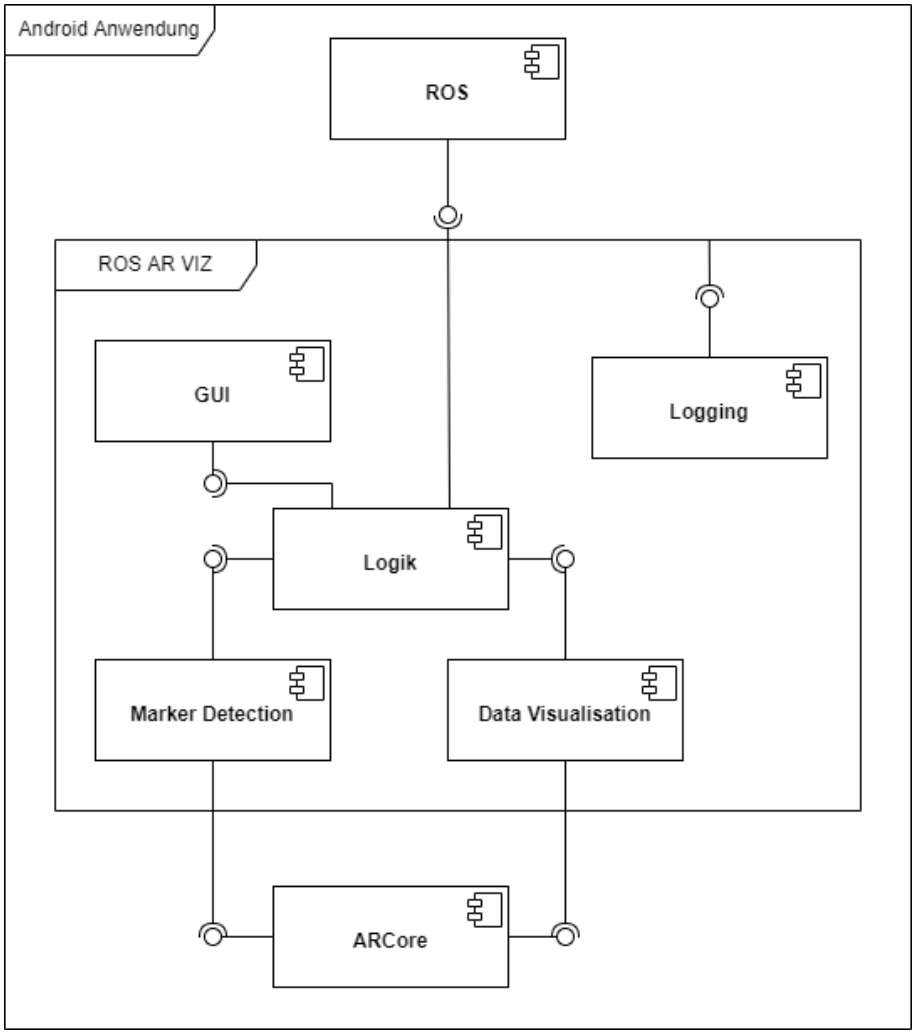

Abbildung 3.2: Der Aufbau der Android-Anwendung, dargestellt als Komponentendiagramm, der Fokus liegt dabei auf dem Aufbau der „ROS AR-VIZ" Komponente

Die Komponente „ROS AR-VIZ" lässt sich in fünf Teil-Komponenten unterteilen, GUI, Logik, Marker-Detection, Data-Visualisation und Logging. Die Daten des Huskys werden über ROS in Form der iViz-ROS-API entgegengenommen (sieh Kapitel 2.1). Über die GUI kann ausgewählt werden, welche Daten/Topics visualisiert werden sollen. Außerdem können über die GUI Einstellungen und Parameter der Anwendung zur Laufzeit angepasst werden. Die Logik Komponenten holt sich je nach ausgewähltem Topic die entsprechenden Daten aus der ROS-Komponente. Um die Visualisierung der Daten korrekt umzusetzen, müssen die Daten aus dem Koordinatensystem des Huskys, in dem sie be-

reitgestellt werden, in das Koordinatensystem des Android-Gerätes transformiert werden. Dafür muss die relative Position des Huskys zum Android-Gerät bestimmt werden. Der Husky kann anhand von Markern oder Bildern lokalisiert werden. Für die Lokalisierung des Huskys ist die Komponente Marker-Detection zuständig. Diese erkennt und lokalisiert bekannte Bilder oder Marker in der Umgebung. Die Logik-Komponente transformiert die Daten anschließend aus dem Koordinatensystem des Huskys in das Koordinatensystem des Android-Gerätes. Die transformierten Daten werden dann an die Data-Visulisation-Komponente übergeben. Diese ist für die Darstellung der Daten in Augmented Reality zuständig. Für die Erkennung der Marker, sowie zur Darstellung der Daten werden verschiedene Funktionalitäten aus ARCore verwendet. Die Logging-Komponente wird von allen Komponenten verwendet und stellt die Möglichkeit bereit, Log-Nachrichten in ein Log-Fenster auf dem Android-Gerät zu schreiben. Dadurch kann das Verhalten der anderen Komponenten während der Laufzeit besser nachvollzogen werden.

3.2.1 ROS

Der Husky arbeitet bereits mit ROS und stellt seine aufgenommenen Daten per ROS zur Verfügung. Daher bietet es sich an, die Kommunikation zwischen dem Husky und dem Android-Gerät über ROS laufen zu lassen. So müssen keine Anpassungen am Husky vorgenommen werden. Die Anwendung kann also einfach an das System angebunden werden und so Systeminformationen in Augmented Reality darstellen.

Um eine Kommunikationskanal zwischen dem Husky und dem Android-Gerät aufzubauen, müssen sich beide Systeme im gleichen Wi-Fi Netzwerk befinden. Zusätzlich müssen sich beide Systeme im gleichen ROS-Netzwerk befinden und dementsprechend mit dem gleichen ROS-Master verbunden sein. Der ROS-Master wird auf dem Husky ausgeführt, das Android-Gerät muss also zunächst mit dem Husky verbunden werden. Um sich mit dem ROS-Master zu verbinden, wird die Master-URI benötigt. Diese setzt sich aus der IP-Adresse des Huskys und dem Standard ROS-Master Port „11311" zusammen. Außerdem muss das Android-Gerät seine eigene URI angegeben, diese setzt sich aus der eigenen IP-Adresse und einem beliebigen Port zusammen, hier wird der Port „7613" verwendet. Mit diesen Informationen kann sich über die iviz-ROS-API mit dem ROS-Master verbunden werden (sieh Kapitel 2.1).

Wenn sich der Husky und das Android-Gerät im selben ROS-Netzwerk befinden, kann das Android-Gerät über die iViz-ROS-API auf alle Topics, die im System gepublished werden,

zugreifen. Um die Sensordaten des Huskys auf dem Android-Gerät zu empfangen, muss lediglich das entsprechende Topic subscribed werden. Die Daten der empfangenen ROS-Nachrichten werden von der iviz-ROS-API als C# Objekt bereitgestellt (sieh Kapitel 2.1).

3.2.2 Logging

Um während der Ausführung der Anwendung direktes Feedback zu erhalten, gibt es eine Logging-Komponente. Durch diese ist es möglich, Log-Nachrichten auf dem Android-Gerät während der Laufzeit einzusehen.

Die Log-Nachrichten, die auf dem Android-Gerät angezeigt sind, werden über eine eigene Logging-Klasse geloggt. Die Standard Unity-Debug-Logs werden auf dem Android-Gerät nicht angezeigt. Für die Lognachrichten der Logging-Klasse, gibt es ein eigenes Dialog-Fenster in der Anwendung. Die Lognachrichten werden in ein Textfeld im Log-Dialog geschrieben. Der Log Dialog ist in Abbuldung 3.3 dargestellt. Im Logdialog werden beispielhaft Lognachrichten der Marker-Detection-Komponente angezeigt. Den Lognachrichten wird jeweils der aktuelle Timestamp hinzugefügt. Der Timestamp wird dabei wie in Abbildung 3.3 zu sehen ist Fettgedruckt dargestellt. Zusätzlich werden die Lognachrichten der Logging-Klasse auch in der Unity-Konsole ausgegeben.

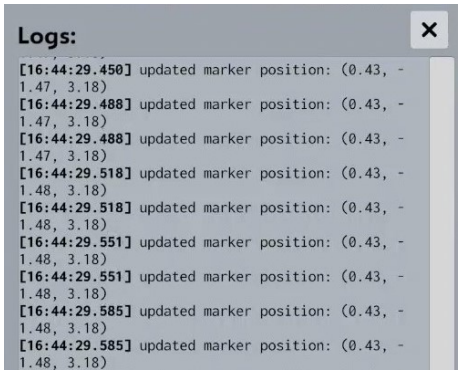

Abbildung 3.3: Darstellung des Log-Dialogs, die Lognarchichten stammen dabei aus der Marker-Detection-Komponente

3.2.3 Marker-Detection

Um den Husky zu lokalisieren und somit die ROS-Daten des Huskys korrekt ausrichten zu können, wird hier der in Abbildung 3.4 dargestellte QR-Code als Marker verwendet. Der QR-Code wird mittig auf der Ladefläche das Huskys platziert und stellt den Ursprung des Koordinatensystem des Huskys dar.

Für die Umsetzung der Marker-Detection wurden verschieden Funktionen aus ARCore verwendet. Die Erkennung und das Tracking des Markers werden mit dem „AR Tracked Image Manager" [19] umgesetzt. Der „AR Tracked Image Manager" wird von Unitys AR-Foundation-Framework [18] bereitgestellt und bietet eine Möglichkeit, Bilder zu erkennen und zu lokalisieren. Um festzulegen, welche Bilder erkannt und lokalisiert werden sollen, sind die Bilder in einer „reference image library"[19] hinterleget. Die Position der erkannten Marker wird über das Delegate Pattern [15] an die Logik weiter geleitet.

Als Marker kann hier ein beliebiges Bild verwendet werden, es muss lediglich genug Merkmale bieten, damit der „AR Tracked Image Manager" in der Lage ist das Bild zu erkennen. Der Huskys kann sowohl mit als auch ohne Marker lokalisiert werden. Für die Lokalisierung mit Marker wird der QR-Code aus Abbildung 3.4 als Marker verwendet. Bei der Erkennung des Huskys ohne Marker wird ein Bild der Ladfläche des Huskys verwendet, um diesen zu lokalisieren. Das Bild der Ladefläche ist in Abbildung 3.5 zu sehen.

Der Marker ist bei der Analyse der Daten oft nicht im Kamerabild zu sehen. In diesem Fall kommt es manchmal vor, dass der Tracking-Punkt des Markers nicht selben Position bleibt und die Daten somit verrutschen. Daher wird versucht die Position des Markers auf eine von ARCore erkannte Oberfläche zu projizieren. So kann ein stabileres Tracking des Markeres realisiert werden. Die Ladefläche des Huskys besteht aus einer wagerechten Oberfläche, daher wird hier in der Regel eine Fläche von ARCore erkannt. Auf dieser Fläche können virtuelle Objekte an einer festen Position im Raum platziert werden (siehe Kapitel 2.1). Durch das Motion Tracking (Kapitel 2.1) von ARCore bleibt die Position des erkannten Marker so auch erhalten, wenn der Marker nicht mehr im Kamerabild zu sehen ist.

Um den gefundenen Marker auf die ARCore Fläche zu projizieren, wird ein „raycast" (Kapitel 2.1) verwendet. Der Strahl wird dabei senkrecht auf den Marker projiziert, sodass der Ursprung des Strahls auf dem Mittelpunkt des Markers liegt und die Richtung senkrecht nach oben zeigt. Wird ein Schnittpunkt mit einer Oberfläche gefunden, so wird

dieser als Marker-Koordinate verwendet. Sollte es mehrere Schnittpunkte geben, wird der Schnittpunkt mit der geringsten Entfernung zu der gefundenen Marker-Koordinate gewählt. Wenn keine Oberfläche gefunden wurde oder es keinen Schnittpunkt gibt, wird die ursprüngliche Position als Marker-Koordinate verwendet.

Abbildung 3.4: Der QR-Code, der als Marker für die Lokalisierung des Huskys verwendet wird.

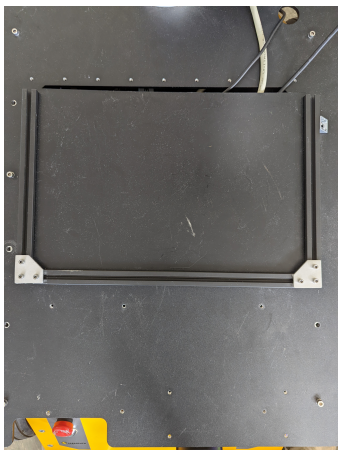

Abbildung 3.5: Bild der Ladefläche des Huskys, welches zur Lokalisierung des Huskys, ohne Marker verwendet wurde

3.2.4 GUI

In der GUI können Einstellungen und Parameter der Anwendung angepasst werden. Außerdem kann ausgewählt werden, welche Daten visualisiert werden sollen. Die Daten werden dabei als virtuelle Objekte in der Umgebung dargestellt. Mit diesen virtuellen Objekten kann ebenfalls über die GUI interagiert werden. Die GUI Komponente stellt dabei lediglich das Interface zum Benutzer dar. Sie ist für die Darstellung der Buttons und Panel zuständig und stellt fest, ob ein Button gedrückt wurde.

Um die Topics auszuwählen und Einstellungen anzupassen, gibt es an der linken Seite ein Side-Panel mit einer Reihe an Knöpfen. (sieh Abbildung 3.6) Einige der Knöpfe öffnen ein eigenes Dialog-Fenster. Auf der rechten Seite in Abbildung 3.6 ist beispielhaft das Dialog-Fenster für die Topic-Auswahl zu sehen. Neben dem Side-Panel, mittig auf Abbildung 3.6, ist das ROS-Connection-Info-Panel zu sehen, hier werden Informationen zur aktuellen ROS-Verbindung dargestellt.

Die Knöpfe an der Seite des Bildschirms, sowie die zusätzlichen Info-Panel können über Toggle-Button ein- und aus-geklappt werden, damit die Daten beim Analysieren nicht verdeckt werden.

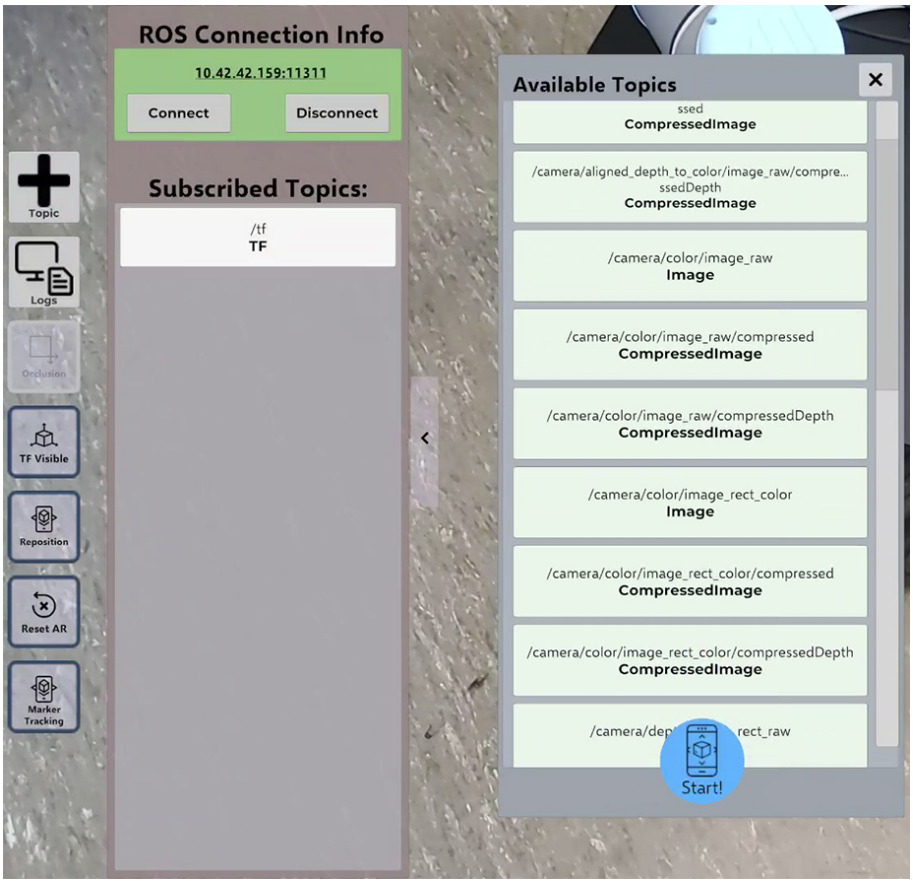

Abbildung 3.6: In der Abbildung ist links das Button-Side-Pannels, mittig das Connection-Info-Interface und rechts der Available-Topic-Dalog dargestellt

ROS-Connection-Info-Pannel

Im oberen Bereich des ROS-Connection-Info-Pannels werden Informationen zur Verbindung mit dem ROS-Netzwerk dargestellt. (Abbildung 3.6) Die Farbe des Feldes signalisiert, den Status der Verbindung. Ein grünes Feld bedeutet, dass erfolgreich eine Verbindung mit einem ROS-Netzwerk hergestellt werden konnte und das Gerät aktuell verbunden ist. Durch ein graues Feld wird dargestellt, dass keine Verbindung aufgebaut werden konnte. Zusätzlich wird angezeigt, mit welcher Master-URI das Gerät verbunden ist,

also in welchem ROS-Netzwerk sich das Gerät befindet. Durch Klicken auf die Master-URI öffnet sich ein Dialog Fenster, darüber können die Master-URI und die eigene URI angepasst werden.

Unter der Master-URI gibt es eine Connect- und einen Disconnect-Button. Mit diesen kann ein Verbindungsaufbau mit dem ROS-Master gestartet oder eine aktive Verbindung beendet werden. Unter den Connection-Infos werden die aktuell subscribten Topics dargestellt (Abbildung 3.6).

Button-Side-Pannel

Über die Knöpfe im Side-Pannel, welches auf der linken Seite von Abbildung 3.6 zu sehen ist, kann das Verhalten der Anwendung angepasst werden. Die beiden oberen Knöpfe, Topic und Logs, öffnen jeweils ein eigenes Dialog-Fenster mit entsprechenden Inhalten. Über die unteren 5 Knöpfe können Änderungen an den Einstellungen und Parametern der Anwendung vorgenommen werden.

Arten von Knöpfen Es gibt zwei verschiedene Arten von Buttons normale Button und Toggle-Button. Beim Drücken eines normalen Button wird immer die gleiche Aktion ausgeführt. Toggle-Button können ein- oder aus-geschalteten werden. Je nach Zustand werden unterschiedliche Aktionen ausgeführt. Der Zustand eines Knopfes wird über seine Farbe dargestellt. Ein ausgeschalteter Knopf wird wie der Occlusion-Button in Abbildung 3.6 ausgegraut.

Topic-Button Über den Topics-Knopf kann der in Abbildung 3.6 dargestellt Topic-Dialog geöffnet werden. Für jedes verfügbare Topic wird dabei der Name und der Datentyp des Topics angeziegt. Über diesen Dialog kann ausgewählt werden, welche Topics subscribed werden sollen. Die entsprechenden Daten des Topics werden dann in Augmented Reality dargestellt. In der Anwendung werden nur Daten vom Typ tf 2.1 und PointCloud2 2.1 dargestellt. Topics mit anderen Datentypen werden daher nicht beachtet.

Log-Button Durch den Logs Knopf wird der in Abbildung 3.3 dargestellte Log Dialog geöffnet. Im Log-Dialog können Log Nachrichten des Systems eingesehen werden.

Occlusiuon-Button Der Occlusion-Button ist ein Toggle-Button. Über den Occlusiuon-Button kann die Objektverdeckung von ARCore aktiviert oder deaktiviert werden (siehe Kapitel 2.1).

TF-Visible-Button Der TF-Visible-Button ist ebenfalls ein Toggle-Button. Mit dem TF-Visible-Button können die Frames das Huskys sichtbar beziehungsweise unsichtbar gemacht werden.

Marker-Tracking-Button Bei dem Marker-Tracking-Button handelt es sich ebenfalls um einen Toggle-Button. Wenn das Marker-Tracking aktiviert ist, bleibt der Frame des Huskys fest an der erkannten Position des Markers. Wird das Marker-Tracking deaktiviert, so kann die Position des Frames über eine Reihe von Reglern angepasst werden. Der in Abbildung 3.7 dargestellte Regler-Dialog öffnet sich bei drücken des Marker-Tracking-Buttons.

Reposition-Button Sollte der Frame des Huskys verrutschten, kann dieser mit dem Reposition-Button neu Platziert werden, dafür wird die Marker-Detection erneut gestartet. Darauf hin erscheint das in Abbildung 3.8 dargestellt Marker-Dialog-Fenster. Flächen und Feature-Punkte die von ARCore gefunden wurden bleiben für das erneute Platzieren erhalten.

Reset-AR-Button Wird der Marker mehrmals an der falschen Stelle positioniert, weil beispielsweise die Oberflächenerkennung von ARCore falsche Werte liefert kann das gesamte AR-Tracking über den Reset-AR-Button zurückgesetzt werden.

Frame-Mover

Durch die in Abbildung 3.7 dargestellten Regler kann die Position des Husky-Frames korrigiert werden, falls sich diese über die Zeit verschoben hat. Dafür gibt es jeweils eine Regler, um den Frame in X, Y und Z Richtung zu verschieben. Zusätzlich gibt es einen Regler mit welchem die Orientierung angepasst werden kann.

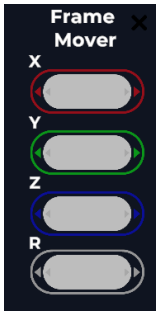

Abbildung 3.7: Regler Dialog um den Husky-Frame zu verschieben, es gibt jeweils einen Regler für die x, y und z Richtung und einen Zusätzlichen regler um die Rotation des Huskys anzupassen.

Interaktion mit virtuellen Objekten

Mithilfe der GUI kann außerdem mit den Daten des Huskys, welche in Form von virtuellen Objekten in der Umgebung dargestllt sind, interagiert werden. Um mit einem virtuellen Objekt zu interagieren, kann auf das Objekt gedrückt werden. (siehe Abbildung 4.8) Mithilfe eines Raycast (Kapitel 2.1) wird dann festgestellt, welches Objekt vom Nutzer ausgewählt wurde. Als Ursprung des Strahls wird dafür das Android Gerät verwendet. Die Richtung wird aus der Orientierung des Gerätes abgeleitet.

Marker Detection Dialog

Beim Start der Anwendung wird zunächst versucht den Huksy zu lokalisieren. Der Status der Marker-Suche wird dabei in ein Dialog-Fenster dargestellt, welches in Abilldung 3.8 zu sehen ist. In diesem wird der Nutzer dazu aufgefordert, die Kamera des Gerätes auf den Husky zu richten. Wenn der Husky erfolgreich lokalisiert wurde, wird dies in dem Dialog-Fenster dargestellt und im unteren Bereich des Bildschirms erscheint ein Start-Button, wie in Abbildung 4.1 zu sehen ist. Über diesen Start-Button kann der Frame des Huskys platziert werden. Beim drücken des Start-Button wird das Dialog-Fenster zum Finden des Markers wird geschlossen.

Die gefundene Position des Markers wird dabei durch den Ursprung eines Koordinatensystem dargestellt. Die Frames werden auf die gleiche Weise dargestellt. Die Visualisierung wird in der Data-Visulizatiuon-Komponente, Kapitel 3.2.6, nocheinmal genauer beschrieben.

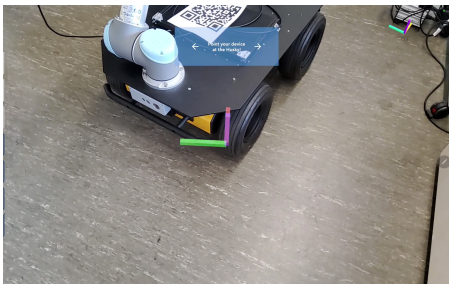

Abbildung 3.8: Dialog-Fenster zum lokalisieren des Huskys
Im oberen Bereich der Abbildung ist das Dialog-Fenster zum lokalisieren des Huskys zu sehen.

Abbildung 3.9: Dialog-Fenster Husky wurde erfolgreich lokalisiert
Im oberen Bereich der Abbildung ist ein Dialog-Fenster zusehen, im unteren Bereich der Abbildung wird ein Start-Button angezeigt

3.2.5 Logik

Die Logik-Komponente bildet den Mittelpunkt der Anwendung. Alle Input-Daten des System werden an die Logik weitergereicht und von dieser verarbeitet und die entsprechenden Funktionen umgesetzt. Dazu zählen die ROS-Daten des Huskys, der Input des Nutzer und die Position des Huskys. Der Umgang mit den unterschiedlichen Input Daten des Systems wird im folgenden genauer betrachtet. Anhand der verschiedenen Input Daten wird ermittelt welche Daten in Augmented Reality dargestellt werden. Die darzustellenden Daten werden von der Logik-Komponente an die Data-Visualisation-Komponente weitergegeben.

Verarbeitung der ROS-Daten

Um die Daten aus den ROS-Nachrichten-Klassen darstellen zu können, müssen diese zunächst in das Unity-Programmiermodell übertragen werden. Dabei wird jeder Nachrichten-Typ, gesondert behandelt. In dieser Anwendung werden tf-Frame- und PointCloud2-Daten umgesetzt.

tf-Frame-Daten Die tf-Frame-Daten des Huskys stellen den Aufbau des Huskys in Frames dar. Ein einzelner Frame repräsentiert dabei ein Bauteil des Huskys, wie beispielweise das linke Vorderrad oder die IMU. Jeder Frame wird durch ein eigenes Objekt

dargestellt, welches Informationen über den Frame, wie den Namen und die Pose, bestehend aus Position und Orienteirung, enthält. Die Frame Objekte werden dabei mit den Daten aus den Nachrichten-Klassen der iViz ROS API befüllt (siehe Kapitel 2.1).

Alle Bauteil-Frames des Huskys sind einem Base-Frame Objekt untergeordnet. Die Position und Ausrichtung der Bauteil-Frames ist dabei relativ zur Position des Base-Frames. Um die Frames des Huskys korrekt zu platzieren, muss also lediglich die Position des Markers aus der Marker-Detection verwendet werden, um den Base-Frame entsprechend daran auszurichten. Somit werden die Position und Orientierung aller Frames in das Weltkoordinatensystem umgewandelt.

Die Frame-Daten des Huskys werden immer benötigt, da diese die Koordinatensystem der verschiedenen Sensoren darstellen. Die Sensoren stellen ihre Daten in diesen Koordinatensystemen bereit(siehe Kapitel 2.1). Da die Frame-Daten nicht entfernt werden können, gibt es einen Button um die Sichtbarkeit der Frames festzulegen.

Wenn Frames nicht mehr angezeigt werden sollen, müssen sie trotzdem noch subscribed werden, da sie als Ursprung für die PointCloud Daten benötigt werden.

Point-Cloud-Daten Die Daten werden von der iViz-ROS-API in Form eines Point-Buffers bereitgestellt, dieser Enthält die Koordinaten aller Punkte der Point-Cloud. Zusätzlich enthält der buffer für jeden Punkt Farbinformationen. Die Koordinaten der Point-Cloud-Daten werden im Koordinatensystem des Kamera-Frames bereitgestellt. Wenn der Fame des Huskys korrekt ausgerichtet ist werden die Daten also automatisch in Weltkoordinaten umgerechnet. Die transformierten Punkte werden zusammen mit den Farbinformationen an die Data-Visualisation weitergelietet.

Verarbeitung der GUI-Input-Daten

Die Eingaben des Nutzer werden in der GUI-Komponente registriert und an die Logik weitergegeben (siehe Kaptel 3.2.4). In der Logik-Komponente sind die Funktionen der entsprechenden Buttons umgesetzt.

ROS-connectio-Info Um eine Verbindung mit einem ROS-Netzwerk aufzubauen werden die Master-URI und die eigene URI in einer Verbindungs-Klasse gespeichert. Anschließend werden die Daten der Verbindungs-Klasse an die iViz-ROS-API übergeben, um sich mit dem ROS Netzwerk zu verbinden (siehe Kapitel 2.1). Die iViz-ROS-API stellt Funktionen für den Verbindungs-auf und -abbau bereit. Wird der Connect- oder Disconnect-Button gedrückt, so wird die entsprechende Funktion der iViz-ROS-API aufgerufen. Wenn die Verbindung zum ROS-Netzwerk getrennt wurde, müssen die Verbindungs-Daten für eine erneute Verbindung nicht noch einmal eingegeben werden, da diese in der Verbindung-Klasse gespeichert werden.

Buttons Im folgenden wird beschrieben wie die Funktion der verschieden Buttons umgesetzt wurde.

Topics Wenn der Topics-Button gedrückt wird, werden alle Topics, die im ROS-Netzwerk gepulished werden, angeziegt. Die Liste der Topics wird dafür aus der iViz-ROS-API ausgelesen (sieh Kapitel 2.1). Wenn ein Topic ausgewählt wird, wird dieses Topic mithilfe der iViz-ROS-API subscribed. Die Verarbeitung der ROS-Daten wird im Abschnitt 3.2.5 genauer beschrieben.

tf-Visible Über die Toggle-Buttons können Funktionen aktiviert oder deaktiviert werden. Die Sichtbarkeit des tf-Frames kann an der Data-Visualisation-Komponente gesetzt werden. Wenn sich der Zustand des tf-Visible-Buttons ändert, wird diese Änderung in die Data-Visualisation-Komponente übertragen.

Occlusion Die Erkennung, ob ein viruelles Objekt verdeckt ist oder nicht wird von ARCore zur Verfügung gestellt (sieh Kapitel 2.1). Die Objektverdeckung kann über den AROcclusionManager ein- oder aus-geschaltete werden. Je nach Zustand des Knopfes ist die Objektverdeckung entweder aktiviert oder deaktiviert. ARCore bietet verschiedene Qualitäts Stufen für die Verduckungserkennung, hier wurde die Qualitätsstufe medium verwendet.

Reposition Um den Frame des Huskys neu auszurichten muss der Husky erneut lokalisiert werden. Dafür wird die alte Position des Huskys gelöscht und die Marker-Detection erneut gestartet (siehe Kapitel 3.2.3).

Reset-AR Beim zurück setzten aller AR-Parameter wird wie schon beim Reposition-Button die Position des Huskys gelöscht und versucht durch die Marker-Detection erneut zu finden. Zusätzlich wird die ARSession in Unity zurückgesetzt. Dadurch werden alle von ARCore gefundenen Features und Flächen für die Orientierung in der Umgebung gelöscht (siehe Kapitel 2.1).

Marker Tracking Der Marker-Tracking-Button legt fest, ob der Frame des Huskys verschoben werden kann, oder ob dieser fest an die erkannten Positon der Marker-Detection gebunden ist. Standardmäßig ist das Marker Tracking aktiviert, der Frame bleibt also fest an der Position des Markers.

Regler-Dialog Über die Regler kann der Frame des Huskys in x, y und z Richtung verschoben werden. Dabei wird aber nicht die Position des Markers, also der Tracking-Punkt von ARCore verschoben. Stattdessen wird der Frame relativ zur Position des Markers verschoben. Die Regler-Werte der x, y und z Achse werden ausgelesen und in einen Verschiebungs-Vektor geschrieben. Dabei ist der maximale Wert der Regeler 1, im Ruhezustand ist der Regler-Wert 0. Um den Frame zu verschieben wird der Verschiebungs-Vektor auf dessen aktuelle Koordinaten aufaddiert. Die Verschiebung kann dabei sowohl positiv als auch Negativ sein. Die Drehung der Orienteirung des Frames wird genauso umgesetzt. Hier wird die Drehung auf den Rotationswinkel des Frames aufaddiert. Die Drehung kann dabei ebenfalls Positiv oder Negativ sein. Die Joysticks sind mit dem „Joystick Pack" [22] umgesetzt, worüber die Werte der Joysticks einfach ausgelesen werden können.

Interaktion mit virtuellen Objekten Die GUI-Komponente stellt über einen Raycast fest, welches virtuelle Objekt vom Nutzer ausgewählt wurde. Der Raycast liefert eine Liste an hit-results zurück, welche die getroffenen Unity-GameObjects beinhaltet. Dadurch kann festgestellt werden, welches Objekt ausgewählt wurde und welche Daten entsprechend angezeigt werden sollen. Die Daten werden dann durch die Data-Visualisation-Komponente dargestellt (siehe Kapitel 3.2.6).

3.2.6 Data Visualisation

Die zu visualisierenden Daten wurden von der Logik bereits in Weltkoordinaten transformiert. Diese Komponente übernimmt also lediglich die Darstellung der Daten. Für jeden Daten Typ der visualisiert werden soll, gibt es dafür eine Klasse, in welcher festgelegt wird, wie die Daten visualisiert werden. In dieser Anwendung sind zwei Datentypen umgesetzt, die in Augmented Reality dargestellt werden können: der tf-Frame des Huskys, welcher Informationen über den Aufbau liefert und die PointCloud2 Daten der RealSense-Kamera, welche ein Tiefenbild der Umgebung repräsentieren.

tf Frames Frames werden in ROS üblicherweise als Ursprung eines Koordinatensystems mit x-, y- und z-Achse dargestellt. Dabei ist die x-Achse rot gefärbt, die y-Achse grün und die z-Achse blau. Diese Darstellungsform wurde hier ebenfalls übernommen (sieh Abbildung 4.8). In Unity werden die Achsen durch jeweils ein längliches Rechteck dargestellt, welches entsprechend der Achse gefärbt ist. Wenn die Frames nicht mehr angezeigt werden sollen, können die GameObjects der Frames über die Methode SetActive deaktiviert. Daraufhin sind die Achsen nicht mehr sichtbar.

Wenn ein Frame ausgewählt wird sollen, Informationen über den Frame dargestellt werden. Dabei wird der ausgewählte Frame vergrößert. Dafür werden die GameObjects der einzelnen Achsen vergrößert. Zusätzlich werden das Label und die Koordinaten des Frames in einem kleinen Dialog-Fenster über dem Frame angezeigt (siehe Abbildung 4.8). Die Informationen eines ausgewählten Frames werden immer für 2 Sekunden angezeigt.

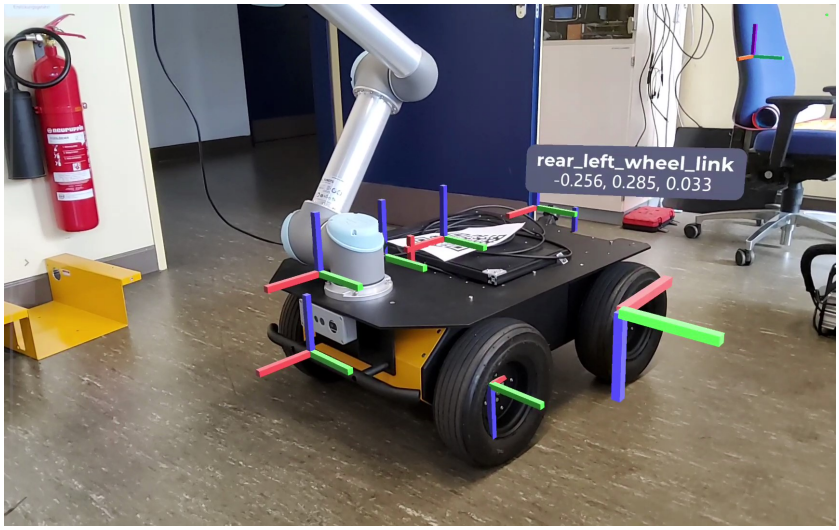

Abbildung 3.10: Darstellung des Husky Frames, hinterer linker Reifen ausgewählt

PointCloud2 Die Point-Cloud-Daten werden von der Logik in Form eines Point-Buffers entgegengenommen. Für die Visualisierung wird der Point-Buffer in einen Unity-ComputeBuffer geladen [9]. Der Unity-ComputeBuffer enthält dabei die gleichen Daten wie der Point-Buffer der Logik. Anschließend wird der Unity-ComputeBuffer an ein Unity-MaterialPropertyBlock übergeben. Unity-MaterialPropertyBlock werden verwendet um viele Objekte des gleichen Typs, in der gleichen Form, mit leicht veränderten Eigenschaften darzustellen. Dabei kann beispielsweise die Farbe der einzelnen Objekte geändert werden [10]. Als Objekte werden hier Punkte dargestellt. Die Position und Farbe der einzelnen Punkte sind im Unity-ComputeBuffer hinterlegt. Mithilfe von Unitys Graphics-Features kann der Unity-MaterialPropertyBlock anschließend visualisert werden. Dafür wird die Methode Graphics.DrawProcedural verwendet [11]. Die Punkte sind dabei folgendermaßen gefärbt: Blaue Punkte befinden sich im Vordergrund des Tiefenbildes, Rote Punkte befinden sich im Hintergrund und sind folglich weiter vom Husky entfernt.

3.3 Übertragung in Unity

3.3.1 Frame-Daten in Unity

Die Frame-Objekte sind jeweils an ein Unity-GameObject gebunden. Für jeden Frame gibt es also ein entsprechendes Unity-GameObject. Die Frames werden von ROS als

Baumstruktur, wie in Kapitel 2.1 beschrieben, bereitgestellt [13]. Die Baumstruktur der Frames wird in Unity durch eine hierarchische Anordnung der GameObjects im Scene Graph übertragen (Kapitel 2.1). Das bedeutet wie in der Baumstruktur der ROS-Frames gibt es auch in Unity ein GameObjekt für den Base-Frame. Unter diesem GameObject werden die GameObjects aller anderen Frames des Huskys angeorndet. Dadurch ist die Position aller Bauteil-Frames des Huskys relativ zur Position des Base-Frames. Das hat den Vorteil, dass die Koordinaten der Bauteil-Frames einfach aus den ROS-Daten übernommen werden können und nicht einzeln transformiert werden müssen. Durch die „Eltern-Kind-Beziehung" zwischen den GameObjects reicht es aus den Base-Frame richtig zu platzieren. Die Position und Orientierung der einzelnen Bauteil-Frames wird entsprechend angepasst [3].

3.3.2 Buttons in Unity

Jeder Knopf wird in Unity durch ein GameObject representiert. Dabei sind das Bild und der Text des Knopfes, wie in Abbildung 3.11 dargestllt, jeweils Kind-Objekte des GameObjects. An jedes Button-GameObject ist eine Button-Skript-Komponente gebunden. Über dieses kann der Button in C# referenziert, um einen OnClick-Listener zu setzten und den Button gegebenenfalls auszugrauen. Wenn ein Knopf gedrückt wird, wird das Clicked-Event über das Delegate-Pattern [15] an die Logik weitergereicht. Alle Knöpfe sind Kind-Objekte des Side-Panel GameObjects. Dadurch kann die Sichtbarkeit aller Knöpfe gleichzeitig auf false gesetz werden, um die Daten analysieren zu können.

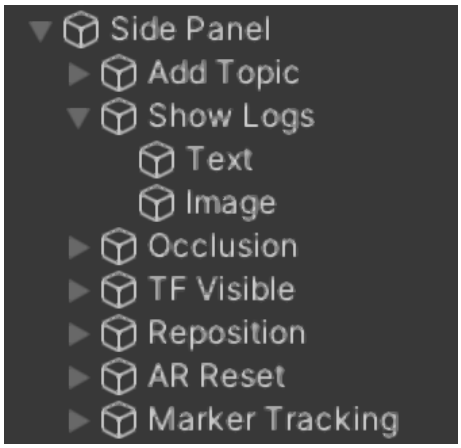

Abbildung 3.11: Die Abbildung zeigt den Szene-Graph des Button-Side-Panels aus Unity.

4 Evaluation

In diesem Kapitel wird die Funktion der Anwendung evaluiert und geprüft, ob die in Kapitel 2.4 gestellten Anforderungen erfüllt wurden. Dafür wurden vier verschiedene Versuche durchgeführt, in welchen jeweils eine oder mehrere Anforderungen getestet werden. Im ersten Versuch wird die Lokalisierung des Huskys geprüft. In den folgenden beiden Versuchen wird die Darstellung der Daten des Huskys überprüft und die Aussagekraft bewertet. Im letzten Versuch wird die Ausführbarkeit und Nutzbarkeit der Anwendung auf verschiedenen Geräten überprüft.

4.1 Lokalisierung des Husky-Roboters

In diesem Versuch soll die Lokalisierung des Huskys getestet werden. Die Lokalisierung wird dabei sowohl anhand eines Markers durchgeführt.

Versuchsaufbau

Um zu prüfen ob die Position des Huskys korrekt erfasst wird, wird der in Abbildung 3.4 dargestllt QR-Code als Marker auf der Ladefläche des Huskys platziert. Die Position des Huskys soll mithilfe des Markers erkannt werden. Dafür wurde der Husky freistehend im Raum platziert. Um Validieren zu können, ob die Marker-Detection-Komponente den Husky korrekt lokalisiert hat, wird der TF-Frame des Huskys visualisiert. So kann geprüft werden, ob die Position der Frames mit den Bauteilen übereinstimmen.

Für die Visualisierung der Daten wird das Samsung Galaxy Tab S7+ verwendet. Die Kamera wird zunächst neben den Husky gerichtet, sodass der Marker nicht vollständig im Bild zu sehen ist. Anschließend wird die Kamera schrittweise in Richtung des Markers bewegt, bis dieser von der Marker-Detection erkannt wird. Bei Erkennung des

Markers wird der Start-Knopf gedrückt. Der Versuch wurde mehrmals durchgeführt, um die Zuverlässigkeit der Erkennung zu überprüfen.

Auswertung

In der Abbildung 4.1 ist im oberen Bereich des Bildes im Marker-Detection-Dialog zu sehen, dass der Husky erfolgreich lokalisert wurde. Beim drücken des Start-Knopfes wird der Frame des Huskys in die Szene projiziert (siehe Abbildung 4.2). Dabei fällt auf das die Frames nicht korrekt platziert wurden, das liegt zum einen daran, dass die z-Koordinate des Markers nicht korrekt ist. Dadurch werden die Frames zu klein dargestellt. Des Weiteren wurde der Marker nicht gerade auf der Ladefläche ausgerichtet, daher ist die Orientierung Daten ebenfalls nicht korrekt.

Abbildung 4.1: Gerät auf Husky zeigen

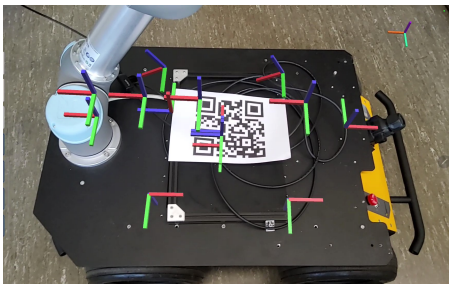

Abbildung 4.2: Husky erkannt

4.2 Darstellung von Bauteil Informationen des Huskys

Im folgenden Versuch soll die Darstellung der Bauteil Informationen getestet werden. Die Bauteile werden dabei durch die tf-Frames des Huskys repräsentiert. Die tf-Frames sollen auf die Bauteile des Huskys projiziert werden. Es soll geprüft werden ob die Bauteile an der richtigen Position dargestellt werden und ob Informationen über die Bauteile visualisiert werden können. Zusätzlich wird geprüft wie sich die Darstellung der Daten mit und ohne Occlusion verhält.

Versuchsaufbau

Für diesen Versuch wurde der Husky freistehend im Raum platziert. Für die Lokalisierung des Huskys wird der QR-Marker auf die Ladefläche des Huskys gelegt. Die Frame-Daten

sind bereits am Husky ausgereichtet. Dafür wurde der Husky von der Marker-Detection lokalisiert und die gefundene Position mit dem Frame-Mover leicht angepasst.

Für die Visualisierung der Daten wurde hier das Samsung Galaxy Tab S7+ verwendet. Der Husky wird von Verschiedenen Seiten betrachtet, indem das das Tablet um den Husky herum bewegt wird. So kann geprüft werden, ob die virtuellen Objekte korrekt platziert sind und die Objekt-Verdeckung sich wie erwartet verhält. Dabei werden Verschiedene Bauteile des Huskys ausgewählt, zu denen Informationen angezeigt werden sollen.

Auswertung

Zunächst wird die Darstellung der Frame-Daten des Huskys mit und ohne Occlusion betrachtet. Anschließend wird die Visualisierung der Bauteilinformationen mit und ohne Occlusion gezeigt.

Occlusion deaktiviert In Abbildung 4.3 sind die Frame Daten des Huskys ohne Objektverdeckung dargestellt. In der Abbildung sind alle Frames des Huskys zu sehen. Die Frame-Daten überlagern sich dabei zum Teil, wodurch die Daten unübersichtlich wirken. Das erschwert die Auswertung, da so nicht ohne weiteres erkennbar ist, welcher Frame zu welchem Bauteil gehört. Dafür können aber auch Bauteile, welche sich im inneren des Huskys befinden betrachtet werden.

Occlusion aktiviert In Abbildung 4.4 werden die Frame Daten mit aktivierter Objektverdeckung visualisiert. Dabei sind nur die Frames der ausliegenden Bauteile sichtbar. Frames die vom Husky verdeckt werden, werden nicht dargestllt. Durch die Objekt-Verdeckung ist besser zu erkennen welcher Frame zu welchem Bauteil gehört. Die virtuellen Objekte fügen sich hier deutlich realistischer in die Szene ein.

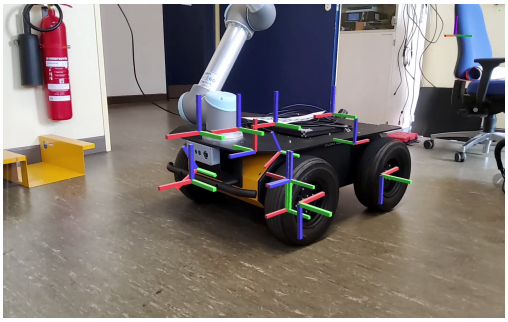

Abbildung 4.3: Darstellung der Husky Frames ohne Occlusion

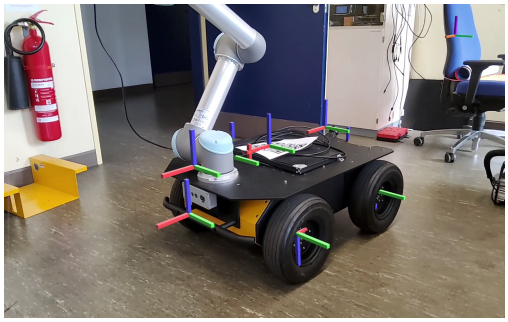

Abbildung 4.4: Darstellung der Husky Frames mit Occlusion

Darstellen von Bauteilinformationen ohne Occlusion Wie in Abbildung 4.5 zu sehen ist können bei deaktivierter der Objektverdeckung auch Bauteiele, die sich im inneren des Huskys befinden, betrachtet werden. Dadurch können der Aufbau und die verwendeten Sensoren des Huskys analysiert werden. In Abbildung 4.6 fällt dabei auf, dass die Position des Kamera-Frames nicht korrekt ist. Die ROS-Koordinaten des Kamera-Frames, die vom Husky bereitgestellt werden, stimmen nicht mit der realen Position der Kamera überein.

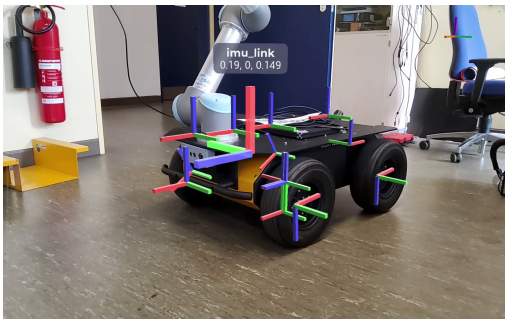

Abbildung 4.5: Darstellung der Husky Frames ohne Occlusion, IMU ausgewählt

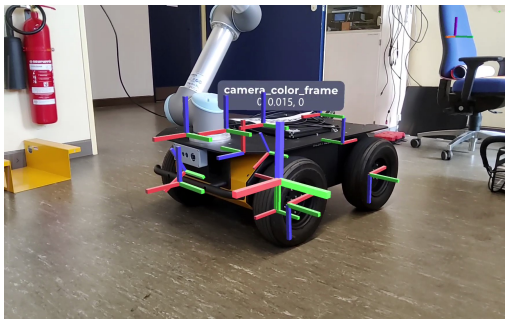

Abbildung 4.6: Darstellung der Husky Frames ohne Occlusion, Kamera ausgewählt

Darstellen von Bauteilinformationen mit Occlusion In den Abbildungen 4.7 und 4.8 ist die Visualisierung der Frame-Daten des Huskys aus zwei verschiedenen Blinkwinkeln dargestellt. In Abbildung 4.7 wird die rechte Seite des Huskys betrachtet, in Abbildung 4.8 die linke. Die Objekt-Verdeckung verhält sich dabei wie erwartet. Je nach

Blinkwinkel werden unterschiedliche Daten visualisiert. An den Abbildungen ist zu erkennen, dass die tf-Frame-Daten des Huskys mit dem tatsächlichen Aufbau des Huskys übereinstimmen. Die Position der virtuellen Bauteil-Frames stimmen mit der Position der realen Bauteile überein.

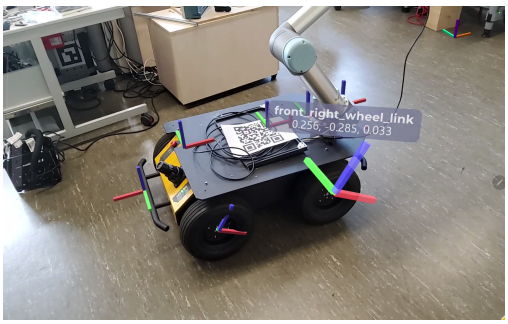

Abbildung 4.7: Darstellung der Husky Frames mit Occlusion, rechtes Vorderrad ausgewählt

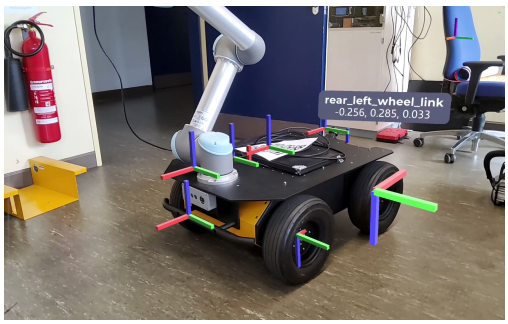

Abbildung 4.8: Darstellung der Husky Frames mit Occlusion, linkes Hinterrad ausgewählt

4.3 Darstellung von Point-Cloud-Daten

In diesem Versuch werden Sensordaten des Huskys visualisiert. Hierfür werden die Point-Cloud-Daten der RealSense Kamera des Huskys verwendet. Diese Liefern Informationen darüber, wie der Husky seine Umgebung wahrnimmt. Die Point-Cloud-Daten werden in die Umgebung des Huskys projiziert. Der Versuch ist in mehrere Teilversuche unterteilt. In den Teilversuchen wird der Husky in verschiedenen Umgebungen platziert. Dabei wird jeweils analysiert, wie der Husky seine Umgebung wahrnimmt und wie akkurat die Daten in die Umgebung projiziert wurden. Im letzen Teilversuche wird zusätzlich die Objektverdeckung der Point-Cloud-Daten untersucht.

Versuchsaufbau

Da der Frame der Kamera an der falschen Position bereitgestellt wird, muss dieser manuell an die Position der Kamera angepasst werden. Die Sichtbarkeit des Frames wird anschließend deaktiviert, um die Daten besser auswerten zu können. Der Husky wird in allen Versuchen an einer festen Position platziert. Die Point-Cloud-Daten werden immer aus verschieden Perspektiven betrachtet, um ein Gefühl für die Dreidimensionalität der Daten zu erzeugen und die Positionierung der Daten besser Überprüfen zu können.

Aufbau der Teilversuche:

1. Der Husky wurde so platziert, dass er durch eine Tür hindurch guckt. Dabei wird die Visualisierung der Daten bei geöffneter und geschlossener Tür betrachtet. Als Anwendungsgerät wurde das Pixel 6 verwendet.

2. Vor dem Husky wurde ein Stuhl Platziert, hier wird betrachtet wie akkurat freistehende Objekte im Vordergrund erkannt werden. Als Anwendungsgerät wurde das Samsung Galaxy Tab S7+ verwendet.

3. In diesem Versuch wurde der gleiche Aufbau wie in Versuch eins verwendet. Hier wird allerdings die Darstellung der Point-Cloud-Daten mit und ohne Occlusion verglichen. Als Anwendungsgerät wurde das Samsung Galaxy Tab S7+ verwendet.

Auswertung

Zunächst werden die einzelnen Teilversuche ausgewertet. Anschließend wird eine generelle Auswertung der dargestellten Point-Cloud-Daten vorgenommen.

Die Point-Cloud-Daten sind Farbig hinterlegt. Blaue Punkte befinden sich im Vordergrund des Tiefenbildes, Rote Punkte befinden sich im Hintergrund und sind folglich weiter vom Husky entfernt.

Sicht des Huskys durch eine Tür In den Abbildungen 4.9, 4.10 und 4.11 blickt der Husky durch eine Geöfnnete Tür. In den Point-Cloud-Daten ist dabei zu sehen, das sich der Türrahmen und ein Teil der Tür im Vordergrund befinden. Die Point-Cloud-Daten werden dabei so auf die realen Objekte projiziert, dass sowohl die Tür, als auch der Türrahemen erkannt werden können (siehe Abbildung 4.9 und 4.10). Wie auf Abbildung 4.11 zu sehen ist, wird die Wand des Flurs korrekt, hinter der Tür, dargestellt. In Abbildung 4.12 ist eine Datenaufnahme aus der gleich Position, mit geschlossener Tür zusehen. Hier ist lediglich eine Vertikale Fläche und die Ecke des Schranks auf der rechten Seite des Bildes zu erkennen.

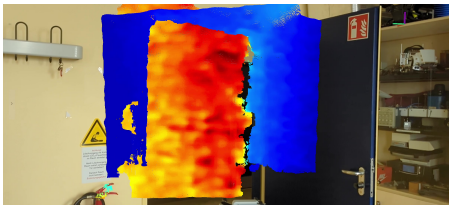

Abbildung 4.9: Point-Cloud-Daten: Frontalansicht auf eine geöffnete Tür

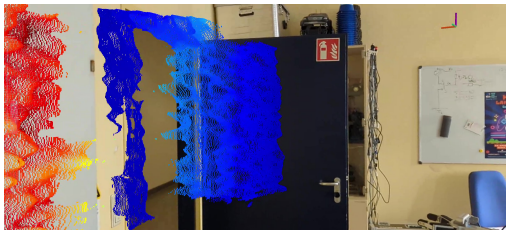

Abbildung 4.10: Point-Cloud-Daten: Seitenansicht auf eine geöffnete Tür

Abbildung 4.11: Point-Cloud-Daten: Sicht aus dem Flur auf eine geöffnete Tür

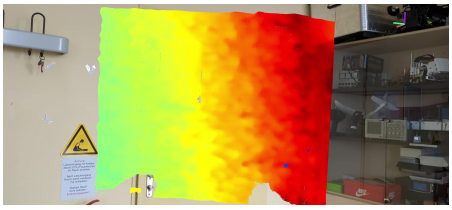

Abbildung 4.12: Point-Cloud-Daten: Frontalansicht auf eine geschlossene Tür

Sicht des Huskys auf ein Objekt im Vordergrund In Abbildung 4.13 und 4.14 ist die AR-Visualisierung der PointCloud-Daten der RealSense Kamera dargestellt. Vor dem Husky ist hier ein Stuhl platziert. In beiden Abbildungen ist der Stuhl in den PointCloud-Daten deutlich zu erkennen. In Abbildung 4.13 guckt die Kamera aus der Perspektive des Huskys, Frontal auf die Daten. Der Anwender steht dabei links neben dem Husky. Hier ist zu sehen, dass auch komplexere Strukturen, wie die Armlehne des Stuhls in den PointCloud-Daten deutlich erkennbar sind. An dem Stuhl kann erkannt werden, dass die Position der projizierten PointCloud-Daten der Objekte im Vordergrund recht genau mit der Position der reale Objekt übereinstimmt. Bei Objekten, die weiter vom Husky entfernt sind, wird der Unterschied zwischen projizierten und realer Position des Objektes jedoch größer. Das ist beispielsweise am Mülleimer in Abbildung 4.13 zu erkennen. In Abbildung 4.14 werden die Daten von der rechten Seite aus betrachtet. Dabei ist zu sehen, dass die Entfernung der Daten zum Husky korrekt dargestellt wird.

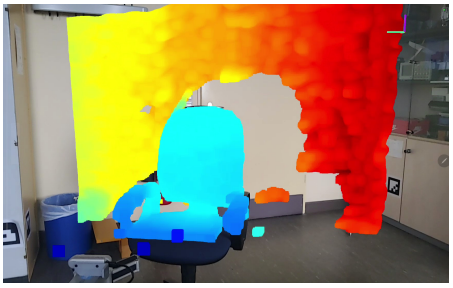

Abbildung 4.13: Point-Cloud-Daten: Fron-talansicht auf einen Stuhl

Abbildung 4.14: Point-Cloud-Daten: Seitenansicht auf einen Stuhl

Vergleich der Point-Cloud-Daten mit und ohne Objekt Verdeckung In Abbildung 4.15 sind die Point-Cloud-Daten des Huskys mit aktivierter Occlusion dargestellt. Der Husky guckt dabei durch eine geöffnete Tür. Die Daten werden aus der Seitenansicht betrachtet, sodass ein Teil der Flurwand auf den Point-Cloud-Daten verdeckt wird. In Abbildung 4.16 ist eine Datenaufnahme aus einer ähnlichen Perspektive mit deaktivierter Occlusion zu sehen. Hier ist zu sehen, dass alle Point-Cloud-Daten visualisiert werden und sich aufgrund der Perspektive überschneiden. In Abbildung 4.15 hingegen werden keine Daten überlagert. Die Point-Cloud-Daten des verdeckten Teil der Wand werden nicht dargestllt. Allerdings wird auch ein Teil der sichtbaren Wand ausgeblendet. Dennoch ist die Darstellung der Daten in Abbildung 4.15 übersichtlicher, sodass beispielsweise der Türrahmen deutlich besser, erkannte werden kann.

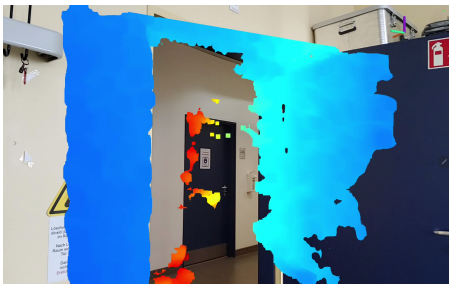

Abbildung 4.15: Point-Cloud-Daten: Fron-talansicht auf einen geöffnete Tür mit Occlusion

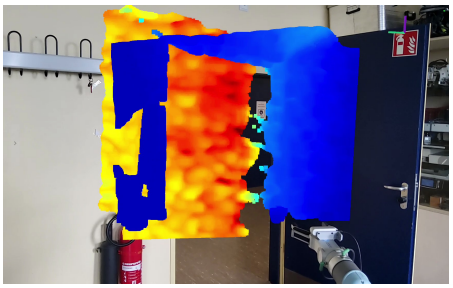

Abbildung 4.16: Point-Cloud-Daten: Frontalansicht auf einen geöffnete Tür ohne Occlusion

Generelle Auswertung der Point-Cloud-Daten Die Darstellung der Point-Cloud-Daten ist auf Grund der Anzahl an Punkten, die Dargestellt werden müssen sehr rechenaufwendig. Daher kam es bei der Darstellung immer wieder zu Fehlverhlaten der

Anwendung. Bei Aktivierung der Occlusion traten dabei besonders häufig Fehler auf. Die Tiefendaten wurden teilweise nicht mehr aktualisiert, oder wurden stark verzögert dargestellt. Außerdem wurden immer wieder falsche Tiefenbilder dargestellt, wie beispielhaft in Abbildung 4.17 und 4.18 zu sehen ist. Des Weiteren ist die Anwendung während der Analyse der Tiefendaten mehrfach abgestürzt. Die Anwendung ist dabei besonders häufig abgestürzt, wenn das Gerät durch den Raum bewegt wurde. Das liegt vermutlich am Motion-Tracking, da die Daten bei Bewegung des Gerätes in die neu Perspektive transformiert werden müssen. Mit der Anwendung können die Tiefendaten meist für einen kurzen Zeitraum betrachtet werden, nach einiger Zeit stürtzt die Anwendung entweder ab, oder die Tiefendaten werden nicht mehr korrekt dargestellt.

Abbildung 4.17: Point-Cloud-Daten: Frontalansicht auf einen Stuhl, fehlerhaft Darstellung

Abbildung 4.18: Point-Cloud-Daten: Seitenansicht auf einen Stuhl, fehlerhaft Darstellung

4.4 Nutzbarkeit der Anwendung auf verschiedene Android Geräten

Hier soll die Nutzbarkeit der Anwendung auf den beiden verschiedenen Anwendungsgeräte betrachtet werden. Dabei werden insbesondere die Vor und Nachteile der unterschiedlichen Displaygrößen und der unterschiedlichen Rechenleistung verglichen. Da es hier um einen Generellen Vergleich der Nutzbarkeit geht und nicht um die Auswertung der Daten wurden keine neuen Versuche durchgeführt. Stattdessen werden die Ergebnisse aus den vorherigen Versuchen miteinander verglichen. Hier wird zusätzlich auf den Installationsprozess auf den verschiedenen Geräten eingegangen.

Vergleich der Beiden Geräte

Um die Nutzbarkeit der Anwendung auf den beiden Geräten zu vergleichen, werden verschiedene Aspekte der Anwendung einzeln betrachtet.

Installation der Anwendung Die Anwendung kommt leider nicht ohne externe Abhängigkeiten aus, da die ARCore-Funktionallitäten ohne die „Google Play Services for AR" nicht nutzbar sind. Die ROS-Komponente kann ohne externe Abhängigkeiten verwendet werden. Die Installation der Anwendung gestaltet sich auf beiden Geräten dennoch sehr einfach. Da es sich um eine AR-Required-Anwendung handelt (sieh Kapitel 2.2.1) werden alle benötigten Abhängigkeiten automatisch mit installiert.

Darstellung der Point-Cloud-Daten Beide Geräte liefern bei der Verwendung der Anwendung ähnliche Ergebnisse. Wie auf den Abbildungen 4.10 und 4.11 zu sehen ist, werden die Punkte der Point-Cloud-Daten auf dem Google Pixel 6 feiner dargestellt. In den Daten sind eher einzelne Punkte erkennbar. Auf dem Galaxy Tab S7+ hingegen wirken die Punkte der Point-Cloud-Daten eher wie eine zusammenhängende Fläche, wie Abbildung 4.13 und 4.14 zu sehen ist. Die Unterschiedlichen Darstellungen sind vermutlich auf die verschiedenen Kamera und Display Auflösungen zurückzuführen.

Beidenbarkeit über die GUI Durch das deutlich größere Display lässt sich die Anwendung auf dem Galaxy Tab S7+ einfacher bedienen. Außerdem können die Daten auf dem Größeren Bildschrim besser Analysiert werden, als auf dem Display des Pixel 6.

Stabilität der Ausführung Das Pixel 6 bietet eine bessere Rechenleistung als das Samsung Galaxy Tab S7+. Dadurch läuft die Anwendung auf dem Pixel 6 stabiler und stürzt seltener ab. Auf dem Samsung Galaxy Tab S7+ kommt es häufiger dazu, dass die Point-Cloud-Daten einfrieren, oder verzögert dargestellt werden. Das Pixel 6 stellt die Point-Cloud-Daten allerdings auch nicht immer fehlerfrei dar. Die tf-Frame Daten hingegen können auf beiden Geräten Problemlos Dargestellt werden. Auch bei der Lokalisierung des Huskys gibt es keinen Unterschied zwischen den Geräten.

5 Fazit

Im Rahmen dieser Arbeit wurde eine Android Anwendung für die Visualisierung von Systeminformationen autonomer Systems in Augemnted-Reality entwickelt. Die Ergebnisse der Versuche zeigen, dass die Anwendung grundlegend in der Lage ist ROS-Daten in Augmented Reality darzustellen. Die Daten können dabei sowohl in der Umgebung des Roboter, als auch direkt am Roboter selbst visualisiert werden.

Um die Daten zu visualisiern müssen dabei keine Änderungen an dem bestehenden ROS-System des Roboters vorgenommen werden. Die Anwendung kann einfach mit dem ROS-Netzwerk des Roboters verbunden werden und so die bereitgestellten Daten in Augmented Reality visualisieren. Die Daten werden dabei direkt in die Umgebung projiziert, in der sie aufgenommen werden. Dabei ist die Position der Daten in der Umgebung verankert, sodass sich in den Daten frei bewegt werden kann und die Daten aus unterschiedlichen Perspektiven analysieren werden können. Die Wahrnehmung des Roboters kann durch die Darstellung der Point-Cloud-Daten aus verschiedenen Blickwinkeln nachvollzogen werden. Dabei wurde gezeigt, dass so ein gutes räumliches Verständniss für die Daten hergestellt werden kann. Objekte in der Umgebung können so einfach in den Daten identifiziert werden, da die Sensoren-Daten direkt auf die realen Objekte projiziert werden. Des weiteren können über die Anwendung Informationen zum Aufbau des Roboters, in Form der tf-Frames des Roboters visualisiert werden. Durch Interaktion mit den virtuellen tf-Frames kann der Aufbau des Roboters über die Anwendung genauer analysiert werden. Durch die Objektverdeckung von ARCore können die Daten des Systems ohne Überlagerung betrachtet werden, wodurch sie übersichtlicher und verständlicher werden. Die tf-Frames können durch die Objektverdeckung beispielsweise besser zu den zugehörigen Bauteilen zugeordnet werden. Durch die Deaktivierung der Objektverdeckung kann ein Blick ins innere des Systems geworfen werden.

Die automatische Lokalisierung des Roboters und die Ausrichtung der Daten an der Position des Roboters funktionierte nur teilweise. Häufig musste die Position der Daten nachträglich manuell korrigiert werden. In seltenen Fällen ist das Motion Tracking von

ARCore fehlgeschlagen, sodass die Position der Daten nach einer Bewegung des Gerätes verschoben waren und dementsprechend manuell korrigiert werden mussten.

Für die Nutzung der Anwendung ist ein Tablet deutlich besser, als ein Smartphone geeignet, da die Daten auf dem größeren Display besser analysiert werden können und die Anwendung sich über die GUI leichter bedienen lässt. Besonders die Darstellung der Point-Cloud-Daten ist sehr Rechenintensiv weshalb ein leistungsstarkes Gerät benötigt wird. Die in dieser Arbeit verwendeten Geräte boten leider nicht genug Leistung, weshalb die Anwendung oft abgestürzt oder eingefroren ist. Eine Analyse der Daten war deswegen immer nur für einen kurzen Zeitraum möglich. Mit der Anwendung lassen sich Daten autonomer Systeme anschaulich und leicht verständlich darstellen, für die Darstellung Komplexer Datentypen wäre mehr Rechenleistung, oder eine effizientere Visualisierung jedoch von Vorteile.

6 Ausblick

In dieser Arbeit wurde gezeigt das mithilfe von ARCore und der iviz-ROS-API Systeminformationen eines Roboters in Augmented Reality anschaulich visualisiert werden können. Um die Anwendung tatsächlich im Entwicklungsalltag nutzen zu können müsste die Fehler des Marker-Trackings behoben werden. Dafür müssten entweder Marker Fest am Roboter befestigt werden, oder Markante Punkte des Huskys gefunden werden, welche auch ohne Marker erkannt werden können. Zusätzlich müsste die Visualisierung der Daten effizienter gestaltet werden, sodass Daten auch über einen längeren Zeitraum und auf Geräten mit weniger Rechenleistung visualisiert werden können.

Momentan können in der Anwendung nur zwei Arten von ROS-Daten dargestellt werden. Über die iViz-ROS-API kann aber bereits ein Vielzahl an weiteren Datentypen entgegen genommen werden. Ein sinnvoller nächster Schritt wäre daher die Visualisierung um weiterer ROS-Datentypen zu erweitern. Dadurch könnte ein noch besseres Einblick in das System und den aktuellen Zustand des Systems geboten werden. Als eine weiterer Möglichkeit mehr Informationen über das System bereitzustellen, könnten mehr Details zu den Bauteilen des Roboters dargestellt werden. Aktuell wird zu jedem Bauteil lediglich der Name des entsprechenden Bauteils visualisiert. Hier könnten aber relativ einfach beliebige Informationen zu den verschiedenen Bauteils hinzugefügt werden.

Die Darstellung der Sensordaten eines Robotik-Systems in Augmented Reality hat sich als nützliche Funktion erwiesen, um so einen genaueren Einblick in die Funktionsweise des Systems zu erhalten. Die Anwendung lässt sich einfach an bestehende ROS-Systeme anbinden, um so die Daten des System zu visualisieren. Sie könnte also auch in anderen ROS-Systeme eingesetzt werden, um Informationen darzustellen. Je nach Anwendungsfall müssen dafür zusätzliche Datentypen hinzugefügt werden.

In dieser Arbeit wurde dargestellt, dass Augmented Reality viel Potential beitet Systeminformationen autonomer Systeme intuitiv und einfach zugänglich zu machen. Dementsprechend gibt es viele Ansätze diese Arbeit fortzuführen.

Literaturverzeichnis

[1] AUTOSYS: *autosysHusky*. – URL https://autosys.informatik.haw-hamburg.de/platforms/2020husky/. – Zugriffsdatum: 2022-07-06

[2] AUTOSYS: *Test Area Intelligent Quartier Mobility (TIQ)*. – URL https://autosys.informatik.haw-hamburg.de/project/smartmobility/. – Zugriffsdatum: 2022-07-06

[3] CHRISTIAN GEIGER, Patrick P.: Unity-Tutorial: Erste Schritte mit der Game Engine. In: *iX* 2 (2014), S. 36–46

[4] DOCUMENTATION, ARCore: *ARCore supported devices*. – URL https://developers.google.com/ar/devices. – Zugriffsdatum: 2022-07-06

[5] DOCUMENTATION, ARCore: *Depth adds realism*. – URL https://developers.google.com/ar/develop/depth#unity-ar-foundation. – Zugriffsdatum: 2022-07-06

[6] DOCUMENTATION, ARCore: *Enable ARCore*. – URL https://developers.google.com/ar/develop/java/enable-arcore. – Zugriffsdatum: 2022-07-07

[7] DOCUMENTATION, ARCore: *Fundamental concepts*. – URL https://developers.google.com/ar/develop/fundamentals. – Zugriffsdatum: 2022-07-05

[8] DOCUMENTATION, ARCore: *Hit-tests place virtual objects in the real world*. – URL https://developers.google.com/ar/develop/hit-test. – Zugriffsdatum: 2022-07-05

[9] DOCUMENTATION, Unity: *ComputeBuffer*. – URL https://docs.unity3d.com/ScriptReference/ComputeBuffer.html. – Zugriffsdatum: 2022-07-06

[10] Documentation, Unity: *Graphics.DrawProcedural*. – URL https://docs.unity3d.com/ScriptReference/MaterialPropertyBlock.html. – Zugriffsdatum: 2022-07-06

[11] Documentation, Unity: *Graphics.DrawProcedural*. – URL https://docs.unity3d.com/ScriptReference/Graphics.DrawProcedural.html. – Zugriffsdatum: 2022-07-06

[12] Durrant-Whyte, H. ; Bailey, T.: Simultaneous localization and mapping: part I. In: *IEEE Robotics Automation Magazine* 13 (2006), Nr. 2, S. 99–110

[13] Foote, Tully: tf: The transform library. In: *Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on*, April 2013 (Open-Source Software workshop), S. 1–6. – ISSN 2325-0526

[14] Hamburg, HAW: *Industrieroboter „Husky“ für die Autonomone Quartiersmobilität*. – URL https://www.haw-hamburg.de/hochschule/technik-und-informatik/departments/maschinenbau-und-produktion/forschung/forschungsgruppen/elektrische-mobilitaet/tiq-fahrplattform-husky/. – Zugriffsdatum: 2022-07-06

[15] Hilyard, Jay ; Teilhet, Stephen: C 3.0 Cookbook, 3rd Edition. (2007), S. 316–343. ISBN 9780596516109

[16] Hoang, Khoa C. ; Chan, Wesley P. ; Lay, Steven ; Cosgun, Akansel ; Croft, Elizabeth A.: ARviz: An Augmented Reality-Enabled Visualization Platform for ROS Applications. In: *IEEE Robotics Automation Magazine* 29 (2022), Nr. 1, S. 58–67

[17] Makhataeva, Zhanat ; Varol, Atakan: Augmented Reality for Robotics: A Review. In: *Robotics* 9 (2020), 04, S. 3

[18] Manual, Unity: *About AR Foundation*. – URL https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@4.2/manual/index.html. – Zugriffsdatum: 2022-07-05

[19] Manual, Unity: *AR tracked image manager*. – URL https://docs.unity3d.com/Packages/com.unity.xr.arfoundation@4.0/manual/tracked-image-manager.html. – Zugriffsdatum: 2022-07-06

[20] Quigley, Morgan ; Conley, Ken ; Gerkey, Brian P. ; Faust, Josh ; Foote, Tully ; Leibs, Jeremy ; Wheeler, Rob ; Ng, Andrew Y.: ROS: an open-source Robot Operating System. In: *ICRA Workshop on Open Source Software*, 2009

[21] Ros.org: *PCL Overview*. – URL http://wiki.ros.org/pcl/Overview. – Zugriffsdatum: 2022-07-06

[22] Store, Unity A.: *Joystick Pack*. – URL https://assetstore.unity.com/packages/tools/input-management/joystick-pack-107631#description. – Zugriffsdatum: 2022-07-05

[23] Zea, Antonio ; Hanebeck, Uwe D.: iviz: A ROS Visualization App for Mobile Devices. (2020). – URL https://arxiv.org/abs/2008.12725

Erklärung zur selbstständigen Bearbeitung

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original