UNIVERSITÄT ZU LÜBECK

See what the robot sees: enhanced Human-Robot Interaction through shared perception in Augmented Reality

Sehen, was der Roboter sieht: Verbesserte Mensch-Roboter-Interaktion durch geteilte Wahrnehmung in Augmented Reality

verfasst am
Institut für Technische Informatik

im Rahmen des Studiengangs
Robotik und Autonome Systeme
der Universität zu Lübeck

vorgelegt von
Jonas Jakobi

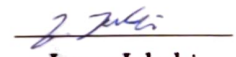
ausgegeben und betreut von
Prof. Dr.-Ing. Heiko Hamann

mit Unterstützung von
Julian Kaduk, M. Sc.

Lübeck, den 1. Dezember 2022

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.



Jonas Jakobi

Zusammenfassung

Um die Mensch-Roboter-Interaktion zu verbessern, müssen bessere Möglichkeiten zur Visualisierung von Roboterdaten geschaffen werden. Insbesondere bei der Programmierung und dem Debugging von Robotern benötigt der Programmierer eine Möglichkeit, komplexe und schwer verständliche Daten auf intuitive Weise zu visualisieren. Augmented-Reality-Brillen wie die Microsoft HoloLens 2 können die Lücke zwischen physischer und virtueller, visualisierter Welt schließen. So könnten Daten von einem Roboter so visualisiert werden, dass sie in die Welt des Trägers der Brille projiziert werden.

Diese Arbeit konzeptualisiert und implementiert eine Augmented-Reality-Plattform, die dieses Ziel erreichen soll, indem Daten von einem mobilen Turtlebot Roboter in der HoloLens visualisiert werden. Die Daten des Roboters, der herumfahren und mit einem Lasersensor seine Umgebung abbilden kann, werden direkt an der Position visualisiert, an der sie aufgenommen wurden. Über User Interfaces die als virtuelle Hologramme realisiert wurden, kann der Benutzer mit der Anwendung interagieren. Dabei werden die Interaktionsmöglichkeiten der Gestikerkennung und Sprachsteuerung der HoloLens genutzt, um instinktive Interaktionen zu ermöglichen.

Richtlinien und Ideen zur Weiterentwicklung und Verbesserung solch eines Systems auf Basis wissenschaftlicher Recherche und eines Interviews wurden erläutert. Dabei zeigte sich auch das Potenzial von Augmented Reality in anderen Bereichen der Robotik, wie der Robotersimulation.

Meine Forschung bildet die Grundlage für eine Augmented-Reality-Anwendung, die die Zusammenarbeit mit mobilen Robotern verbessert und exploriert Möglichkeiten, wie eine solche Anwendung weiter ausgebaut werden könnte.

Abstract

To improve the interaction between robots and humans, better means of visualizing robot data need to be created. Specifically, when programming and debugging robots, the programmer would highly benefit from a way to visualize complex and hard-to-understand data intuitively. Augmented Reality glasses like the Microsoft HoloLens 2 can bridge the gap between the physical and virtual world. This could be used to visualize data from a robot, superimposed into the world view of the user as holograms.

To achieve this goal, this thesis proposes and implements an augmented reality platform, using the HoloLens 2, to visualize data from a mobile Turtlebot robot. Data from the robot, which can drive around and map its environment with a laser sensor, is then directly shown at the position where it is recorded. Virtual hologram interfaces allow the user to interact with the application. The interaction possibilities of the HoloLens's gesture recognition and voice control are used to enable intuitive interaction.

Guidelines and ideas for further development and improvement of such a system based on scientific research and an interview were elaborated on. This also showed the potential of augmented reality in other areas of robotics, such as robot simulation.

This project forms the basis for an augmented reality application improving collaboration with mobile robots and suggests ideas on how such an application could be expanded.

Contents

1

Introduction

Robots and humans interact with the world around them in entirely different ways. Humans perceive the world with a variety of senses; they see in high-resolution images with full color and have a feeling for space and orientation. Humans interact with one another intuitively through spoken language and intricate gestures. They possess the cognitive ability of complex reasoning and have an innate feeling for their surroundings that encompasses all their senses.
Robots on the other hand rely on limited digital sensors to perceive the environment. Their vision is limited to different types of distance sensors or cameras providing them with orientation and position in the world. Through dedicated algorithms, these 'senses' can be combined, enabling the robot to navigate and memorize the world around it, similar to how a human would. However, they perceive and store this information in a different way from us humans. It is present in complex data types and long arrays of numbers which are hard to intuitively understand as a human.
This disconnect between the world the human and the one the robot perceives can make it difficult for the human side to understand the robot's knowledge and decisions toward its goal state. This is increased due to the fact that means of communication vary greatly between humans and robots. A simple robot might communicate through text logs or led status lights. Information is conveyed through digital signaling, unlike the typical ways humans communicate. Which is through speech, gestures, and body language. While user interfaces to get information on the robots' perception exist, they often lack the ability and richness to convey all necessary information or to present it in a way directly understandable by a human. A tool to enable a more effective and intuitive way for interaction and communication between humans and robots would be helpful in bridging the gap between both worlds. Besides the development of user interfaces (UI), there is also the diverse research field of human-robot interaction (HRI), which researches how to improve the interaction and collaboration between humans and robots.

One aspect of working with mobile robotics in which research in HRI could provide the tools for big improvement is the programming and debugging task [4]. Working on robot control software is a complicated task and requires significant debugging, as a robot has to interact with the real world and its many unpredictable variables and influences.

In most cases, debugging information is not directly displayed on the robot and requires the programmer to check log files or remotely log into the robot. This creates a disconnect between actually engaging with and observing the robot, and observing further debugging information through user interfaces on a computer.
Most tasks for mobile robots involve 3D space and portraying this complex, multi-dimensional data through traditional debugging means is ineffective. Therefore, a debugging tool, able to highlight this information, directly superimposed onto the real-world view of the human, could be a more effective and intuitive way of debugging robot programs. Fundamentally, it could provide a way of sharing the robot's perception of the world and its decision-making in a humanly intuitive way, tapping into the innate feeling humans have for space.

A fitting solution would be to have the programmer wear Augmented Reality (AR) glasses, which allow for this type of visualization. The Microsoft HoloLens 2[1] are industry-leading AR glasses enabling holograms to be shown and anchored into the physical environment. These virtual objects can be interacted with and manipulated in real time. It also allows for gesture and speech control, which can both be used to explore potential ways in which the human can interact with the robot. This makes it imaginable to introduce a layer of shared perception between the human and robot, in which they can mutually benefit from each other. Instead of writing commands into a console or developing and using a graphical user interface on a monitor, the programmer could link functions of the robot with voice commands or virtual objects like holographic buttons that do not need the programmer to leave the space of the robot and focus on a computer screen.

The goal of this bachelor thesis is to develop an AR tool that is able to communicate with a robot via ROS 2 [10] and visualize its data in wearable augmented reality glasses in order to improve the communication with the robot and build the foundation for a level of shared perception. The main focus of these visualizations lies in the use case of debugging a robot. This suggested concept should lay the ground for a potential AR platform, allowing more interactions and visualizations to interface with robots. With this solution, the question of how usable and intuitive AR is as a platform for Human-Robot Interaction could be explored. Furthermore, the more specific use case of how intuitive AR is as a means of debugging a robot will be explored.

[1] https://www.microsoft.com/en-us/hololens

2

Related Works

In order to further discuss the potential use cases and possibilities that AR offers, we first have to clarify what AR is. A common definition for AR by Azuma is that "3D virtual objects are integrated into a 3D real environment in real-time" [1, p. 1]. So the user is not fully emerged in a virtual world but instead has their real environment augmented by virtual objects. A more broad approach for defining the spectrum of reality to virtuality was designed by Milgram and Kishino in the form of the "virtuality continuum" [13], depicted in Figure 2. Here, the full span of mixed reality is shown, ranging from a real environment, all the way to a virtual environment for which the word Virtual Reality (VR) is most commonly used. The company, Microsoft, markets the HoloLens as a Mixed

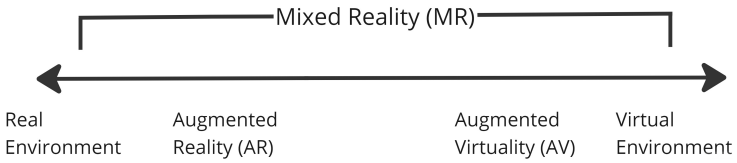


Figure 2.1: Visual representation of the "virtuality continuum"

Reality device but further specifies that they place the Microsoft HoloLens to the left of the virtuality continuum, where the area of augmented reality lies, [12] with devices like the Samsung Head Mounted Display Odyssey+ [2] lying in the space further right on the continuum. This distinction is mostly made on the fact that applications for the HoloLens will typically show virtual objects embedded into a real environment. Unlike devices like the Odyssey+, which instead shows a fully virtual environment with the representation of the real world embedded into the virtual world. Examples include showing a boundary grid of the real environment as a virtual grid of points to enable safe navigation through the augmented virtual environment or showing video of the headset's camera to be able to interact with the headset without taking it off.

In the scope of this thesis, we will examine and develop applications that lie to the

[2] https://www.samsung.com/us/support/computing/hmd/hmd-odyssey/hmd-odyssey-plus-mixed-reality/

left of the continuum, enhancing the real environment with virtual objects.

An architectural application for human-robot collaboration that uses AR was explored before, explaining how AR technology is well suited for human-robot collaboration [6]. In its essence, an AR application allows a robot to visually show its internal state through the use of user interfaces directly into the real worldview of the human wearing the AR device. AR offers a seamless interaction between real and virtual environments: User interfaces should be made tangible and manipulate the shared 3D scene of the human and the robot. It enables spatial dialogue as visual cues are present in the same environment.
In the same paper it was proposed, that by combining gesture and speech, deictic gestures can be used to effectively communicate with a robot, like for example pointing to a place and saying ”here” to communicate a position in 3D space. Also, the importance of being able to reach into and interact with the virtual 3D world was stressed, as well as the tracking of the user's gaze. Also noted was the fact that AR enables the human to have a worldview of the space the robot navigates in, which enables spatial understanding of the robot in relation to its environment. The same could then be applied to any visualizations about/from the robot. All of these requirements and options are possible in the HoloLens, making it a suitable platform for an AR tool by these requirements.

Colett and MacDonald [3] have developed and tested an AR debugging platform for mobile robots using a screen to show AR visualizations. Their ARDev library acted as an extension to Player [5], which is a middle-ware software for robots, similar to ROS. Their case studies highlighted a multitude of advantages that AR can bring to robot debugging in particular. The validity of data is immediately apparent when viewed against the backdrop of the real world, ruling out common and basic errors like inverted, offset, or mirrored data that otherwise would be relatively hard to spot on more traditional visualizations (2d graphs, text logs).
Another advantage that was mentioned is the monitoring role offered by AR visualizations. Interviewed developers commented, that even after a feature was functioning, the continued visualization helped to monitor it. More traditional debugging means, like additional print statements, are usually removed after a feature was debugged, as they can lead to clutter. The trial also highlighted the importance of having matching visualizations for simulated and real robots and environments, as this allowed a seamless transition from simulation to real testing in the development process, without having to switch or rework debugging tools.
Potential areas of improvement include the need for 3D immersion, as the system used traditional computer monitors with camera footage that has the visualizations embedded into the image. The in-situ 3D visualizations can only really show their benefit if the viewing device supports three-dimensionality, like having a 3D display and being able to change camera perspective. This is an issue that is entirely solved and even expanded upon through the use of the HoloLens, as its AR display makes holograms appear spatially anchored in the real world and thus increases 3D immersion considerably. It also noted the need for visualizing various abstract data types, as their ARDev library did not support AR text.

An approach using AR with the HoloLens, specifically in the use case of debugging robot controllers, has already been outlined before by Ikeda & Szafir [8]. Their introduced concept is quite similar to the implementation in this thesis. The proposed system is capable of estimating the robot's position as well as visualizing a variety of information including sensor data, traveled path, planned path, etc. An experimental procedure is also outlined in which the participants, described as programmers with experience in robot programming, complete a detection and finders task and then evaluate the usefulness of the AR tool based on the System Usability Scale. [2].

In a follow-up paper by the same authors [7], the challenges and considerations of visualizations for robot debugging were further explored. Among these challenges, is how to visualize certain data types. For every data type, a different kind of visualization has to be considered. For example, the scalar data of a battery percentage could be shown as a number, or as a visual image of a battery that shows the current charge, superimposed into the world as a virtual object. The text could either be shown on a screen or spatially anchored into a scene where it is most convenient.

Potential ways of interacting with the data of a robot were also outlined. These include Analyzing multiple streams of data, finding unexpected patterns in data, and making predictions about the future state.
A qualitative evaluation of feedback was gathered by 24 roboticists comparing different tools, a 2D Graphical User Interface (GUI), a 3D GUI on a 2D Screen, and an AR application. From their feedback, a guideline for future tools was synthesized that summarizes their findings. Grouping visualizations and data by the task they inform about is one aspect that was found. Data that is frequently needed should be accessible through the primary interface. The ways in which data is visualized or transmitted should be dynamically alterable; Whether or not it is shown at all and if it is shown, its visual saliency should be easily changeable. This also helps in mitigating occluding visualizations, one of the frequently noted aspects in the feedback gathered in this paper.

Out of these aspects, we synthesized a list of features and design considerations, that should be present in the implementation of an AR app:

- Visualize the internal state of a robot superimposed into the real world.
- Find suitable means of visualizing each data type.
- Design tangible and manipulable User Interfaces.
- Consider the visual saliency of AR objects and User Interfaces and allow for dynamically altering transparency or visibility.
- Utilize gaze and voice input for a more broad range of interactions.
- Frequently needed data accessible in the primary interface.
- Offer the same visualizations for real and virtual robots.

3

Hardware

3.1 Robot: Turtlebot 3

The robot that should be used for an AR application should fulfill some basic criteria:

- Have sensor information complex enough to warrant being visualized (specifically in AR)
- Have enough processing power to run a ROS node and be able to send its sensor information to the augmented reality glasses

The robot chosen for this thesis is the TurtleBot 3 WafflePi[3]. It is a differential drive robot with two servo motors for wheel control. It possesses several sensors to perceive the environment and its own position in it. A Lidar (light detection and ranging) sensor is able to measure the distance to surrounding objects in 360° with a 1° angular resolution and a sampling rate of 1.8kHz[4]. This sensor provides suitable information to create a point cloud of surrounding objects. A Raspberry Pi camera is also attached, allowing for potential image recognition functionality, although the camera was not used in this thesis. The 9 Axis Inertial Measurement Unit and a precise encoder also allow for improved localization accuracy through odometry data. This data is used for moving the robot's relative holograms.

3.2 Augmented Reality: Microsoft HoloLens 2

The Augmented Reality glasses used in this thesis are the Microsoft HoloLens 2. Holograms are projected into the wearer's view through see-through holographic lenses. To ensure all holograms stay spatially anchored, the HoloLens has a variety of sensors that enable it to track its position in the world. Visible light sensors for head tracking, IR cameras for eye tracking and the 1-MP Time-Of-Flight Depth Sensor is capable of mapping the physical environment. Along with being able to keep the position of holograms consistent, this also allows for holograms to be anchored to surfaces in the physical world. An Inertial Measurement Unit, a microphone array and a 1080p camera are also included.

[3] https://www.turtlebot.com/turtlebot3/
[4] https://emanual.robotis.com/docs/en/platform/turtlebot3/features/#components

The various sensors allow for a plethora of interaction and input methods. Both hands of the user are fully tracked and allow for the detection of instinctive gestures for direct object manipulation. Possible gestures include pinching, grabbing, pointing and pressing. This allows the user to manipulate holograms by moving or resizing them. It also enables certain input objects like buttons and sliders to be interacted with like their real counterparts. Eye tracking can be used to recognize the user's gaze and voice recognition for natural language processing.

These features should allow an application made for the HoloLens to allow for a lot of potential interactions.

4

Software

4.1 Robot Operating System (ROS)

To exchange data between the robot and the HoloLens, we use the Robot Operating System (ROS).[5] [10] is a middleware that facilitates message transfer between various ROS processes using an anonymous publish/subscribe method. The ROS graph is at the center of any ROS system. It is the network of nodes in a ROS system and the interconnections that allow them to interact with one another. In this graph, a node is one entity that uses ROS to interact with other Nodes. The data exchanged comes in the form of messages, a ROS data type. A message is described in a .msg file and can contain several fields or constants. These must be either one of the built-in types (like bool, float64, string) or another ROS message that is defined on its own. Nodes communicate by exchanging messages, they do that by subscribing or publishing to topics. There are two other ways nodes can interface, those being services and actions, but we will not go further into these during this thesis. The automatic process through which nodes determine how to talk to each other is called 'Discovery'. ROS is language independent and mainly runs on python and c++, but for my software, ROS operates natively on the Universal Windows Platform, allowing it to run on Windows and on the HoloLens, using 'ROS2 for .NET'[6] in Unity.

4.2 Unity

The main implementation in this project was done with the Real-Time Development Platform Unity Engine[7]. It is commonly used for game development. Because of its user-friendliness, great cross-platform support and AR tools, more than ninety percent of applications on emerging AR platforms like the HoloLens are created with Unity [11]. Developing AR applications in Unity is supported by the Mixed Reality Toolkit[8]. It consists of a variety of components and features, enabling fast development of cross-platform

[5] Specifically, ROS 2 foxy is being used: https://docs.ros.org/en/foxy/
[6] https://github.com/ros2-dotnet/ros2_dotnet
[7] https://unity.com/
[8] https://learn.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk2/?view=mrtkunity-2021-05

Mixed Reality Applications. It supports a variety of Virtual and Augmented Reality devices, including the HoloLens. Building blocks for spatial interactions and User Interfaces are given and can be used to make a highly interactable application.

A Unity application itself consists of one or more scenes. In the most common application of video games, they represent individual levels of the game. Each scene contains a variety of GameObjects. In the presented project, only a single scene is used. GameObjects are the fundamental objects in Unity that can represent any object or fulfill any desired purpose. A GameObject itself does not accomplish much but instead, Components can be attached to GameObjects that implement specific functionalities. There are many built-in component types in Unity and one can also make custom component types using the Unity Scripting API and the programming language C Sharp (C#). The one component every GameObject inherently has is a Transform, which describes the object's position, rotation and scale.

Unity allows for packages to be installed through the use of the package manager. A package can include a variety of scripts and assets to extend a Unity application. In this thesis, several packages are utilized to implement certain functionality that is needed to properly communicate with the robot and the HoloLens.
One of these is the Mixed Reality Toolkit for ROS2[9] that extends the functionality of "ROS2 for .NET" and allows a Unity Mixed Reality application to communicate with other ROS nodes. ROS2 will natively run on the HoloLens and expose a node to the ROS graph. Using the package, one can write custom components that subscribe or publish to various ROS topics. Upon subscribing to a ROS2 topic, one can define a callback method that will be called whenever a message arrives on the subscribed topic. As a parameter to the callback function, an object will be passed, containing the variables of that ROS message type as fields. For publishing, one has to construct a new instance of the class of the ROS2 message and fill in all the information that is needed.

All my unity objects that require or publish ROS information therefore refer to the ROS2Listener class, creating a Subscription or a Publisher onto its ROS node. At creation, the topic, Callback method and Quality of Service (QOS) profile have to be set. With QOS profiles the reliability and speed of data transfer can be customized. As all the data shared in my project is sensor data, the SensorData QOS profile is used. It uses best-effort reliability and a smaller queue size to ensure that readings are received as timely as possible while keeping reliability as a lower priority.

[9] https://github.com/ms-iot/ros_msft_mrtk

5

Implementation

The application implemented in this thesis should be capable of visualizing various data types of the Turtlebot robot that are published through ROS. The following table contains a list of all messages published by the Turtlebot:

Table 5.1: List of messages sent by the Turtlebot.

Topic	Message Type
/battery_state	sensor_msgs/BatteryState
/cmd_vel_rc100	geometry_msgs/Twist
/diagnostics	diagnostic_msgs/DiagnosticArray
/imu	sensor_msgs/imu
/joint_states	sensor_msgs/JointState
/magnetic_field	sensor_msgs/MagneticField
/odom	nav_msgs/Odometry
/rosout	rosgraph_msgs/Log
/rosout_agg	rosgraph_msgs/Log
/rpms	std_msgs/UInt16
/scan	sensor_msgs/LaserScan
/sensor-state	turtlebot3_msgs/SensorState
/tf	tf/tfMessage
/version_info	turtlebot3_msgs/VersionInfo

Of these messages, /odom, /scan and /battery_state are implemented in this thesis. In order for the visualizations /scan and /batter_state to work, the coordinate frame of the robot data, that is published to ROS, has to be synchronized with the coordinate system the Unity application uses. The first step to achieving this synchronization is to determine the position of the robot in Unity's coordinate system.

5.1 Determining Robot Position

There are two primary approaches that were considered for determining the position of the robot: Either setting the start position of the robot manually and updating its position

through the use of odometry data, or by tracking the robot visually. The initial approach that was planned for this program was to use a fiducial marker system to track the robot visually. There are several ways of potentially achieving this, with two main variants.

The first is to use the HoloLens' built-in detection for QR codes. The HoloLens natively detects QR codes and allows the user to view their information when just using the HoloLens without an application open. We attempted to get the API working to get the position and rotation of the QR codes in our application, but for the Unity Version that was chosen for this project[10], it was not possible to get it to work. This could in part be because of Legacy XR support being dropped in Unity 2020 and onward.[11] We tried several potential workarounds. Porting the entire project to Unity 2019 was also tried, but this version of the project did not compile onto the HoloLens.

The second potential visual tracking method is to use the webcam output of the HoloLens to track a fiducial marker with it. For this, both the Vuforia Engine[12], as well as Aruco marker tracking[13] were tried. The Aruco marker tracking worked when running the project on the Windows Computer in the Unity Editor, but not on the HoloLens, even when the HoloLens Camera was calibrated several times, the marker was not recognized.

After various attempts, instead, the approach of using odometry data was implemented. For this, a semi-transparent cube hologram is to be present in the center of the robot and move and rotate along with the real robot, as shown in Figure 5.2. All sensor data and visualizations can then be placed relative to this hologram, while this hologram can also function as a virtual robot for testing and simulation purposes.

The RobotPose component subscribes to the "odom" topic, receiving odometry messages with the latest position and rotation that the robot reports.
In order to start the synchronization process, the user has to drag the robot hologram inside of the real robot and rotate it to match the robot's orientation. The hologram can be pinched or grabbed naturally with hand gestures, in order to move and rotate it to the correct position. In order to ease that process, the robot hologram's transparency can be altered. This should make it easier to place the hologram correctly inside the robot. Inside the cube hologram is a small coordinate frame with differently colored axes that can be used to ensure the hologram is oriented correctly. The red x-axis should be pointed towards the front of the robot, where its camera is located.

Once the hologram is placed correctly, the "Set Start Position" button on the main interface can be pressed, or the voice command "Set Start" can be said. The press of this button will call the *SetStartToCurrentPos*() method in the *RobotPose* component of the robot hologram. The current position and rotation of the robot hologram is saved as p_{start}

[10] Unity 2021.2.7f1
[11] Through the use of the Console Hologram 5.6 the error message was visible: "MissingMethodException: System.RuntimeType::GetGUID(System.Type,System.Byte[])". Installing and running a given example project for QR tracking by Microsoft resulted in the same error message.
[12] `https://developer.vuforia.com/`
[13] `https://github.com/doughtmw/ArUcoDetectionHoloLens-Unity`

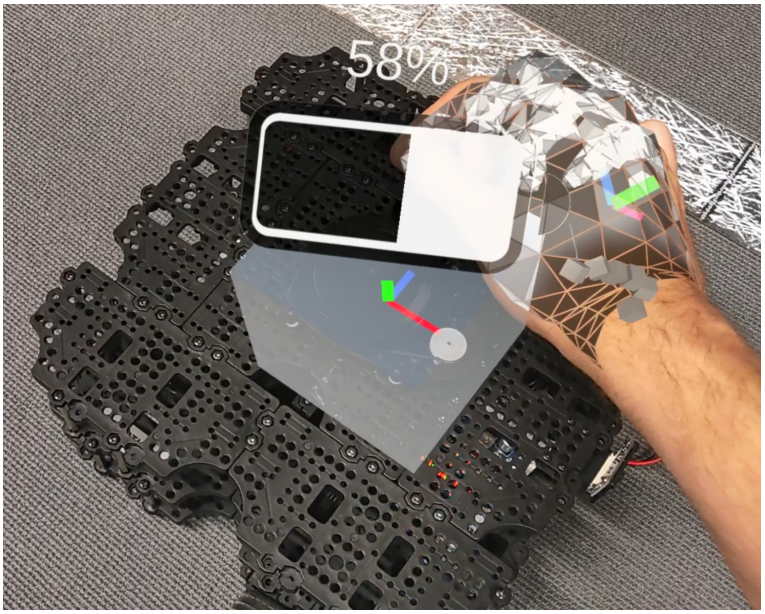


Figure 5.2: Robot Hologram currently being placed inside of the turtlebot robot.

and γ_{start}, as well as the position of the most recent odometry message is saved as o_{start}. Once the "Receive Odometry" button is pressed, or the word "odometry" is said, and the start position was defined at least once, the hologram starts updating its position and rotation according to the robot motion. The position and rotation received from ROS are transformed into Unity's coordinate frame and rotation as follows:

Observation 5.1. $o = \begin{bmatrix} r_x \\ 0 \\ r_y \end{bmatrix}, \alpha = \begin{bmatrix} 0 \\ -1 * \sigma_z \\ 0 \end{bmatrix}$

where o = saved odometry position, vector of meters[14]; r = position of the robot in ROS coordinate frame, a vector containing position in meters; α = saved odometry rotation, a vector containing rotations around x-,y- and z-axis in degree[15]); σ_z = rotation of robot around the z-axis in ROS coordinate frame, degree.
With these values given, the final position of the robot can be calculated based on all stored values:

Observation 5.2. $p = p_{start} + o - o_{start}$, $\gamma = \gamma_{start} * \alpha$

where p = position of robot hologram, m; p_{start} = start position of robot hologram, m; o = most recent odometry position, m; o_{start} = start odometry position, m; α = saved odometry rotation, quaternion; γ = rotation of robot hologram, quaternion and γ_{start} = start rotation of robot hologram, quaternion.
Only the rotation around the z-axis (in ROS coordinate frames) is saved as we are dealing with a robot on a plane. The robot hologram would therefore not be displayed correctly

[14] Meters are used interchangeably with Unity's unit of distance as 1 Unity meter = 1 meter in real life when deployed to the HoloLens.
[15] The data is initialized by taking the three degree values but internally stored as a Quaternion.

should the robot be on a slope.
Any other visualizations that make use of the robot's position will be calibrated to the moving robot hologram. The point cloud will be placed at the correct position and the battery level will be visualized floating above the robot.

5.2 Managing and Visualizing LaserScan data as a Point Cloud

Data from the robot's Lidar sensor should ideally be visualized where it is recorded, superimposed onto the objects from which the data stems. The most simple method of visualizing Laser Scan data is by using a point cloud, in which every data point is represented as a sphere hologram. The visual saliency of the point cloud should also be considered and should ideally reflect the quality of the data.
The data from the Lidar is published as a message of type *LaserScan* on the topic "scan" by the TurtleBot robot. The message contains a single scan of the planar data points captured by the laser range finder. Each data point can be triangulated by its angle (in radians) and a corresponding range (in meters). A start and end angle is given, and an angle increment dictates the increment of the angle for each entry in the array of ranges. A minimum and maximum range is given, any range values outside of these will be discarded.
The class *PointCloudManager* is responsible for receiving this data. It is a component that is attached to a GameObject of the same name. The *PointCloudManager* is responsible for subscribing to the "scan" message and instantiating the individual points as separate GameObjects, each having the component *PointInCloud* attached to them. It can also toggle on / off whether or not to receive any points and show/hide all current points. Each instance of the *PointInCloud* component will have a score value that determines the validity of a point. If the score falls below 0, the point will be destroyed. The higher a points score gets, the more opaque it is. This sets apart points that have been recorded frequently and seem to be of high quality, while also deleting points that are not valid anymore.
When a new *LaserScan* data arrives, the *ReceiveLaserScanData* callback method will be called. This method will iterate over all ranges given in the message and determine the final position of each data point in Unity space by the following equation:

Observation 5.3. $\gamma_i = \alpha_{min} + \alpha_{increment} * i - r_{y,\beta} * deg2rad$

Observation 5.4. $p_i = \begin{bmatrix} r_x \\ r_y \\ r_z \end{bmatrix} + \begin{bmatrix} +cos(\gamma_i) * l_i \\ \\ +sin(\gamma_i) * l_i \end{bmatrix}$

where p_i = position of new point, meter; α_{min} = minimum angle of incoming points, rad; $a_{increment}$ = increment of each data point, rad; $deg2rad = 0.01745$; i = current iteration index; $r_{x/y/z}$ = position of the robot, meter; $r_{y,\beta}$ = angle of rotation of the robot around the y-axis, degree and l_i = i-th recorded range in meters.

In Unity's coordinate system, the y-axis is pointing upwards and the coordinate system is left-handed. In ROS' coordinate system, on the other hand, rotations are right-handed and the z-axis is pointing upwards. The angle given by the *LaserScan* message is therefore

added to the negative robot rotation around the y-axis.
After determining the position where a point is supposed to land, the program will check for potential collisions with past points to build a map of the environment.
A Unity Ray is cast toward the target position, which will register a hit with every other point cloud point along the path from the robot position to the target position. For every hit, we check if the distance between the hit and the destination is smaller than a given *minDistance*, approximately the diameter of a point (scenario a). If it is, we increment the score of that point, which seems to be a correct data point and at the right spot. If this happens at least once, no new point will be created at the destination. if a hit does not fulfill this requirement and is therefore closer to the robot than our destination, we decrease its score (scenario b). If all hits were only in scenario b, we create a new point at our destination.

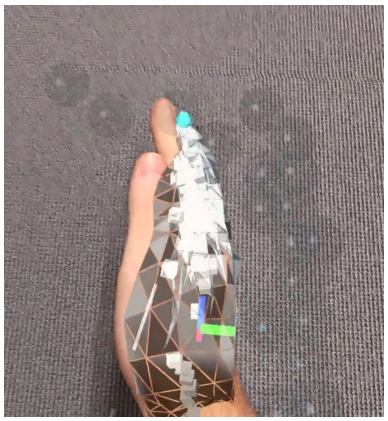


Figure 5.3: Recently created points with a lower score.

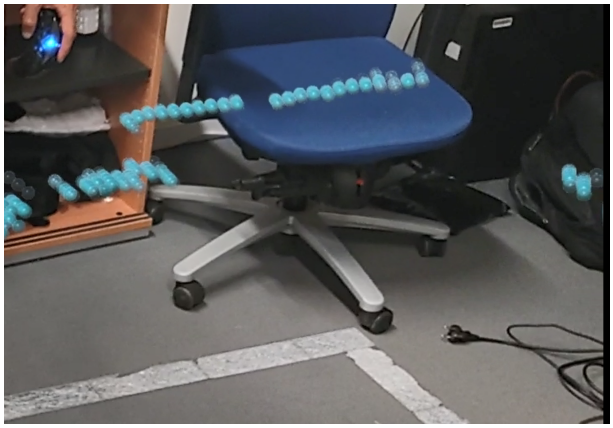


Figure 5.4: Points of static objects with high scores.

Overall, this implementation should result in the following behavior: A map is built, with the walls ideally filled with as few duplicate points as possible and points being very opaque, due to increased scores (as seen in Figure 5.4). Also, points of an object

that are no longer there or only in front of the sensor for a short period of time become transparent or entirely fade away (as seen in Figure 5.3). The score-based point cloud system therefore acts as a suitable means of visualizing sensor data that also highlights useful data and gets rid of obsolete data points.

5.3 Visualizing Battery Level

Knowing the current charge of the battery of a robot is an important piece of information that should always be at hand. Following the design principle of superimposing the internal state of the robot into the real world, ideally in the most relevant place/context, the battery should be visualized on the robot. For this, a hologram was designed that constantly shows the current charge superimposed on top of the robot.
The battery level of the robot is transmitted over ROS as the message type *BatteryState* on the topic "battery_state". It includes a multitude of potential information from voltage to temperature but the only one visualized in this project is the percentage charge of the battery.
The battery charge is visualized as both a percentage number, as well as a graphical bat-

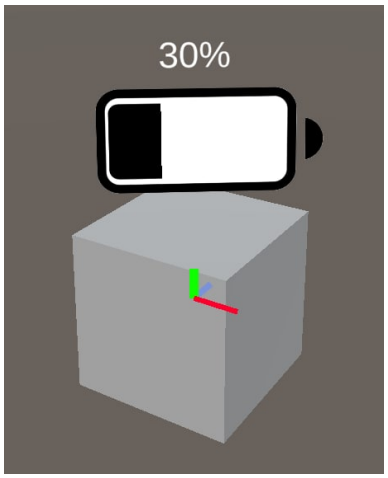


Figure 5.5: Visual representation of battery state.

tery image that changes its infill width according to the battery's percentage. The battery visualization has the component *BillBoard* attached to it, which will seek out the Main Camera in the Scene (which will be the head of the person wearing the HoloLens). It will then orient this GameObject to always face toward the target GameObject.
When a new battery charge message is received, it is only applied to the visualizations when the change to the last visualized battery percentage is higher than 2%. Otherwise, the display would be stuttering, as the turtlebot alternates between two close battery percentages.

5.4 Robot Simulation and Virtual Interaction

Following the design principle of offering the same visualizations for real and virtual robots, this application needs a virtual robot, to begin with. A virtual robot also allowed an easier time writing and debugging this application, and was therefore needed for development. A way of simulating robot data in order to check all functionality is working as intended is needed, as the time to compile and run the project on the HoloLens is long, and a robot is also not always accessible. For this, a virtual robot was designed, able to send messages over ROS2, simulating the messages of the real turtlebot robot.

In some configurable time interval, the virtual robot will publish one message of each data type used/subscribed to in this project. *LaserScan* data is, for testing purposes, published as 64 points, evenly spaced out as a circle around the robot. For *Odometry* data, only the Position of the current pose is updated, by taking the current position and adding some set amount to the x-axis. The purpose of this is, to be able to see that movement is working correctly and points are placed relative to the updated position. *BatteryState* message is simply updated with a lower percentage up until the percentage reaches 0. The virtual robot will take control of the same robot hologram that is placed inside the real robot. This means, the real and the virtual robot function in the same way and can therefore be used interchangeably.

5.5 User Interface

One of the main advantages of AR is the tangibility of holograms and the intuitive user interfaces it enables. The design consideration of making a tangible and manipulative user interface was therefore of great importance.
Several of the design principles went into consideration for the main interface design.

Primarily, all functionality should be accessible in this interface.
The six buttons can roughly be grouped into three groups: The first row controls aspects around the robot, the second controls the point cloud and the third row the odometry/synchronization of the robot.
The *Activate Fake Robot* button of the first row enables or disables the robot simulation. The virtual robot starts publishing messages onto the ROS network that can be picked up by any ROS node, including our own visualization components. The second button on the first row is for hiding or showing the battery hologram.
The buttons in the second row serve the purpose of controlling the point cloud. The top button shows or hides the entire point cloud while the second button dictates whether or not points are accepted.
The third row is necessary for allowing the robot to move around. As detailed in the section 5.1, the "Set Start Position" will calibrate the robot to the hologram, enabling the use of the "Receive Odometry" which will make the robot hologram move along with the real robot by using ROS odometry messages.
Underneath the buttons is a slider that can be moved by pinching it and moving it along its axis. The slider controls the transparency of the cube robot hologram, allowing the

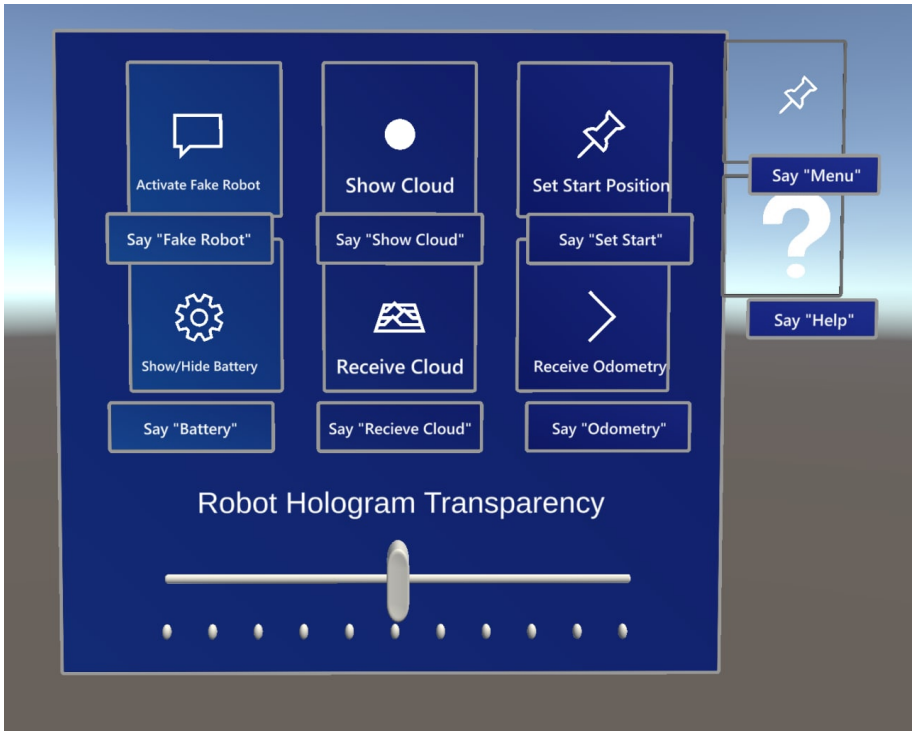


Figure 5.6: The primary user interface of the application.

user to decrease the transparency to see the robot more clearly or other things the hologram might be blocking the view of.
This menu can be interacted with instinctively to move, rotate and scale it to the user's need. The top-right semi-transparent button with a pin symbol will un-pin the menu, making it follow the user as he moves around the area. A second control button is also present, that will open another window, containing explanations of all buttons of the interface. This should enable the program to be usable with no prior introduction and also act as a refresher if the function of a button is forgotten.

All buttons of the main interface are controllable by voice commands. There are two options for voice commands in the Mixed Reality Toolkit: Voice commands that require focus and those that do not require focus. Voice commands that require focus will only be registered if the user's eye gaze falls onto the object which will register the voice input. Due to the limited number of buttons and their distinct identities, it was decided that the buttons do not require focus to be usable. Instead, each one has one unique keyword that can be said to activate the functionality, as if the button has been pressed, no matter where the user is looking. This allows the user to call the buttons without currently having access to the menus. For example, if their hands are occupied or they are in a physical position where the buttons are too far away to reach. Because the button for anchoring or making the menu follow the user also has a voice command, the user can say "menu" whenever it should be out of reach to make the menu float to the user.
Underneath each button, visible in figure 5.7, floats a small label indicating the voice

command that has to be said to activate the button via voice. These labels are hidden at run-time and appear when a user fixates their gaze on one of the buttons for a short time. This implementation should remove excessive clutter while also enabling a way to view the voice commands.

5.6 Console Hologram

To enable this program to also utilize traditional debugging means, in the form of reading console logs, a custom console was implemented as part of the user interface.

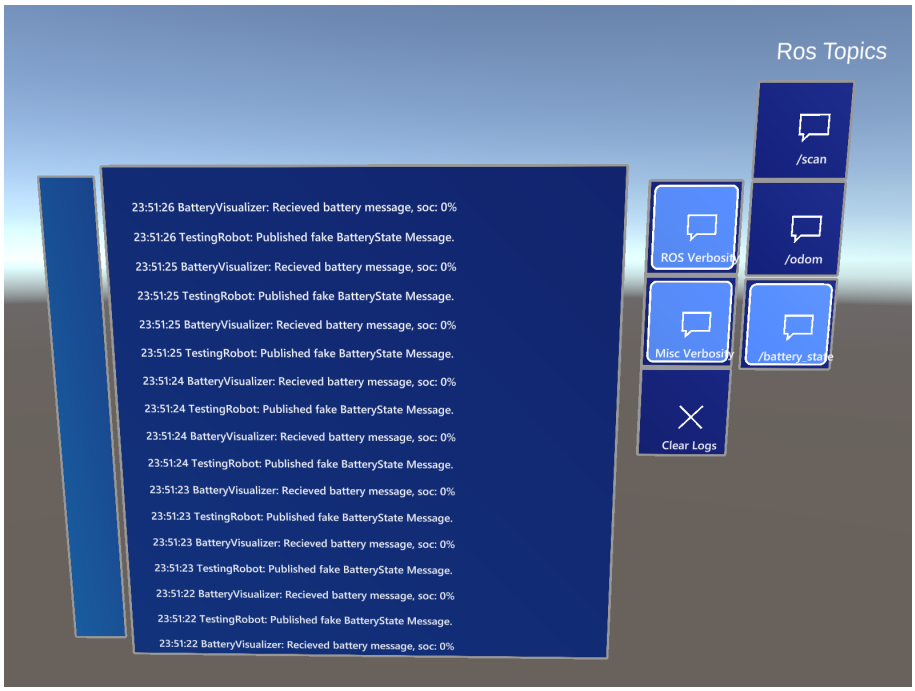


Figure 5.7: Custom holographic console for debugging purposes.

The console consists of a blank slate on which all logs are displayed. To the left of the slate is a bar, which can be grabbed to move, rotate and scale the console. To the right of the slate is a selection of buttons to control the console's behavior. A button for clearing the console of all logs is present, the same functionality can also be activated by saying "Clear". Several buttons for controlling the verbosity of the console are given: one button, *ROS Verbosity*, activates or deactivates whether or not ROS messages are logged. After activating ROS verbosity, the user can press one or more of the three buttons adjacent to the ROS verbosity buttons to receive messages of specific ROS message types. The three buttons are *scan*, *odom* and *battery_state*, each corresponding to the three message types visualized in this application. When a new message is received or published, a log with basic information about this data type is written to the console.
The other verbosity button is *Misc Verbosity* which enables or disables logging of other related message types that may or may not want to be logged. These include for exam-

ple the calculating time in seconds it took to process one batch of laser scan data, or the odometry data saved when a start position is set. A potential robot debugger could use this custom flag to tag more console messages that will then show up.

All these console logs are sent to the regular unity engine console. This hologram console mirrors the logs received on the Unity console. it does so by subscribing the method *ConsoleCallBack* to the Unity Event *Application.logMessageReceived*, which will make Unity call it whenever any log arrives at the console. Three input parameters are received: The actual string of the log, a string of the stack trace, which could be used for more precise error tracking in the future, and a *LogType* enumerator which the custom console uses to color each log according to its type. Regular logs stay white, while warnings are colored yellow and errors red.
A log is instantiated as a GameObject and added to a collection, sorting the logs by their time of arrival, with the newest log being on top. If an old log reaches the bottom of the slate, it will be erased. Having the logs as GameObjects allows for more expansion of functionality, as the logs could then be interacted with.
This interface should therefore be able to expand the debugging means beyond just the data types that are visualized in the application, as well as identify all potential errors that could come up in the Unity application.

This interface allows us to monitor ROS communication as well as see potential errors that come up in the Unity application. Considerations in the implementation should allow the console to easily be expanded by more functionality.

6

Evaluation

6.1 Success of incorporating design considerations

The initial goal of this thesis was to create an AR tool able to communicate with a robot and to offer tools to aid in the use case of debugging robots. Most of all, this application should offer a foundation for future tools and applications to build upon. In order to achieve these goals, a list of design considerations were gathered and implemented with varying degrees of success.

The first of these was to ”Visualize the internal state of a robot superimposed into the real world”. Of all topics that the turtlebot publishes that could be seen as internal messages (that do not directly relate to or visualize the environment), only the battery state was visualized. Other internal states that might be useful include /cmd_vel_rc100 , the velocity/movement of the robot, which could for example be visualized as an arrow pointing in the direction of movement and scaled according to the speed. Also, a window containing more debugging information from various topics could have been implemented.

The goal of ”Finding suitable means of visualizing each data type” was achieved. The point cloud data is visualized at the location where it is recorded, and it should be quite clear how the robot perceives the world.”Visualization” of odometry data is hard to define. It could be seen as having a suitable robot hologram syncing up with the real robot. The shape of the robot hologram could therefore be improved to more closely resemble the actual turtlebot. This could also aid in placing the hologram for the initial manual synchronization. The path that the robot has taken so far could also be visualized, along with some sort of indicator of where the robot is moving. For the battery, the percentage is both visualized as a number for exact reading and a graphical representation.

Designing tangible and manipulable User Interfaces was made achievable through the use of the Mixed Reality Toolkit. All methods of interaction with this application happen through buttons and sliders from the toolkit, which add visual and auditory feedback to every interaction. Every window can also be moved and resized, made to follow the user or stay in the specified place. Gaze and voice input was considered and implemented to improve the interaction with the user interface.

The saliency of visualizations has also been taken into consideration. As elaborated on before, the transparency of the robot's hologram can be altered and the points in the point cloud dynamically alter their transparency based on their quality.

The remaining design considerations of offering the same visualization for real and virtual robots and having frequently needed data accessible in the primary interface are expanded upon in more detail in sections 7.2 and 7.3 respectively.

6.2 Performance Issues of Point Cloud.

The performance of this application is a detail that leaves room for improvement. When running on the HoloLens and a lot of points in the point cloud are recorded, the frame rate starts dropping below comfortable levels. As each point in the point cloud is a GameObject, the hardware load can start becoming high when a lot of these points exist in a scene.

The algorithm elaborated on in Section 5.2, of calculating destination points and checking for collisions, is executed in a Unity Coroutine[16]. Unlike a regular method call, a Coroutine does not have to be completed in a single frame, which would halt the program every time data arrives, but instead can be spread across multiple frames.
This approach does not use multi-threading though, only running on the main thread and therefore not increasing performance. While one Coroutine is still executed, a flag will be set and all data points arriving while calculation still takes place will be discarded. This will result in a loss of data but ensures the program runs smoothly. Newly arriving points will be dealt with fast enough before they become obsolete, as the robot hologram keeps moving while the points are waiting. This could be prevented by saving the position the robot was at while the point was being recorded so that when it is visualized, it can be placed relative to that position. This would negatively affect the performance even more and would only fix one of the many issues the current implementation has, therefore it would be better to rebuild the point cloud entirely.

One possible way is to use Visual Effect Graphs[17] in Unity for visualizing the cloud. A new algorithm for checking collisions with prior points would then have to be written, as Unity Raycasts could not be used for this anymore.[18]

Another solution would be to write a shader that visualizes the points. Potentially also a compute shader could be used to calculate collisions. Due to the fact that the rest of this application does not put a high load on the graphics processing unit, this processing power could instead be used to optimize the point cloud.

If the visualization using GameObjects is maintained, the handling of new points could be realized by using actual multi-threading instead of just "emulated threads" in the form of coroutines.

[16] https://docs.unity3d.com/Manual/Coroutines.html
[17] Node-based system to create visual effects: https://unity.com/visual-effect-graph
[18] Unity Raycasts register hits with colliders, which have to be attached to a GameObject

6.3 Connecting ROS with Unity

Connecting ROS with Unity has no definitive ideal solution either. The initially chosen approach was to use the ROS TCP Endpoint[19] and ROS TCP Connector[20] to form the connection between Unity and ROS. Both of these technologies are part of the Unity Robotics Hub and are extensively documented, containing additional features like a message Generator to allow for additional ROS message types. The ROS TCP Endpoint is a ROS package that can receive and send ROS messages to the ROS TCP Connector, which is a package in Unity. The ROS node would therefore not run entirely inside the Unity application, but instead will be started separately and connected to the ROS TCP Connector in Unity. For this approach to work on the HoloLens though, the ROS TCP Endpoint would have to be started on a computer that has ROS installed. But as this application should ideally work standalone, entirely on the HoloLens, needing an additional topside computer for communication is undesirable. The ROS TCP Endpoint could theoretically be run on the Turtlebot robot, but when the initial approach for the ROS connection was decided on, the potential of this application supporting swarm robotics was taken into consideration, in which case it would also be undesirable to have a "master robot" that runs the connection to the HoloLens. Furthermore, this application should work with any robot that publishes the same message types without needing additional ROS packages on the robot to get the communication working.

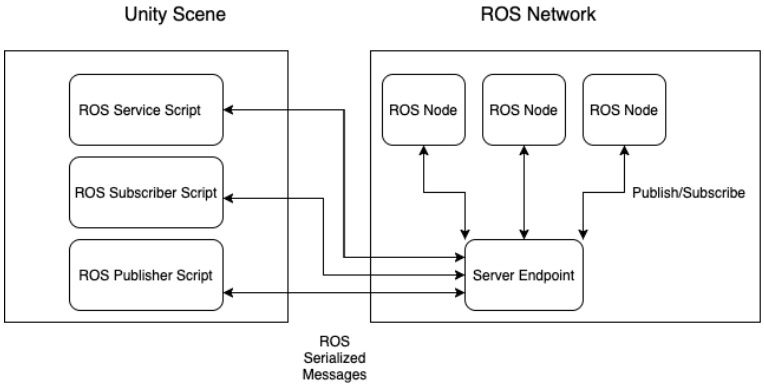


Figure 6.1: ROS TCP Endpoint and Connector flowchart.

Despite this approach having great features and documentation, it was therefore decided to instead use the "Mixed Reality Toolkit for ROS2" [21] due to the fact it runs natively on the HoloLens. This allows the application to work out of the box with any robot publishing data the application is able to visualize and also allows for the potential expansion to swarm robotics.

[19] https://github.com/Unity-Technologies/ROS-TCP-Endpoint
[20] https://github.com/Unity-Technologies/ROS-TCP-Connector
[21] https://github.com/ms-iot/ros_msft_mrtk

6.4 Interview

A semi-structured interview was conducted with a student of the University of Lübeck, who has experience with programming mobile robots in ROS. The interviewee was given questions about his experience with programming and debugging robots, ROS and augmented reality. After the initial conversation, the interviewee was given access to the HoloLens with an explanation of the various features, being encouraged to explore the interfaces and visualizations. After some experimentation, a follow-up interview was conducted. Here, the interviewee was asked for his opinions on the work realized in this thesis, as well as his general thoughts about augmented reality, ROS, human-robot interaction and debugging of mobile robots. The working environment that the interviewee used before with ROS was revisited again shortly for the sake of comparison. This includes a regular ROS console output, as well as visualizations by Rviz[22], a 3D visualization software tool for ROS.

The interviewee only had basic experience with ROS and experienced some frustrations with its complexity. His traditional approach to debugging would be, to view the console for error messages and to use specific additional logs to debug specific issues. After trying out the augmented reality application and comparing it again to visualizations from Rviz, the interviewee noted a better grasp of Lidar data in comparison to 2D visualizations like Rviz. Although he saw great potential for the intuitiveness of AR, he felt the HoloLens did not always register inputs or gestures correctly which led to some frustrations. He also noted that the buttons on the user interface should have more visual differences like clearer icons, to give them more distinct identities.
When asked what further functionality the application should have, the most emphasized addition was, to implement a way to get additional information that is not yet visualized as a simple text output. In its current iteration, the application does not offer any way of seeing more information about the robot than the one specifically visualized. So if additional information, like motor rotation values, is needed, there is no way to see it. The addition of virtual walls and other virtual obstacles that the robot is not able to pass through was also proposed. Especially being able to place them manually in AR. Along with a feature that allows the user to specify a position the robot should move to.

Overall, the interview gave a second opinion on how well the visualizations and interfaces were implemented but more importantly, offered ideas and insights into possible improvements to the system.

[22] http://wiki.ros.org/rviz

7

Discussion

This thesis aimed to develop a tool that uses AR to visualize data from a robot and to lay a foundation for further AR tools for human-robot collaboration. The visualization of the data types covered work well and allow for an intuitive way of engaging with the robot's data. The capabilities of the HoloLens are used to make an application that offers a wide range of interactions and visualizations. The considered design approaches should allow for the improvement of the debugging application, as well as expansion to other areas of robotics.

7.1 Expansion to Swarm Robotics

As ROS runs natively on the HoloLens, the application could be expanded to support a swarm of robots. In swarm robotics, more robots are being used, so there naturally is a larger quantity of debugging information that has to be conveyed. Getting information about the state of the swarm, as well as information about each individual robot, superimposed onto them, could be of great help in the debugging process. Seeing for example battery percentages of each robot directly floating above it, or seeing the ranges for the avoidance of other robots can make configuration, managing and debugging a lot easier.

In the current implementation, all visualizations are bound to the robot hologram that is placed inside the real robot. Through the use of several robot holograms, each applying to a different robot, the application could allow for multiple robots with some alteration to the code. Mainly, the singleton architecture would have to be reworked. A possible solution would be, to check to which robot arriving messages belong, and if it is not known yet, create a new instance of all classes that visualize data from the robot and link them to this robot.
The user interfacing with multiple robots would also necessitate a new interface design, as just duplicating the entire current primary user interface for every new robot would result in a very cluttered environment if many robots were used. Global voice commands in their current design would also not be possible, as some way of specifying which robot to apply the command to would have to be implemented as well.

Although all robots would expand the same point cloud, as it is calibrated to Unity's coordinate system, the robots would scan each other too. This would create a lot of

anomalies in the point cloud and is another problem that would have to be tackled if the point cloud visualization was kept.

7.2 Virtual Robots and Environments

For the wearer of a HoloLens, the real world and the virtual holograms superimposed through AR form one seamless environment. Similarly, a real robot and its real environment and a virtual robot with a virtual environment could be merged together to form one seamless continuum of a robot and its environment.

This expands the idea proposed by Colett and MacDonald of having identical visualizations for virtual and real robots and environments, which is one of the design considerations we proposed for the implementation.

Instead of just sharing visualizations for the two pairs ('real robot and real environment' and 'virtual robot and virtual environment'), these two worlds themselves could merge, in the sense that a real robot should be able to view the virtual environment and a virtual robot the real environment. For a human user wearing an AR display, this would solidify the idea of one seamless robot-environment, as both visually and functionally, there would be no clear distinction between real and virtual. On the basis of this, 4 possible combinations of robot-environment pairs could be modeled.

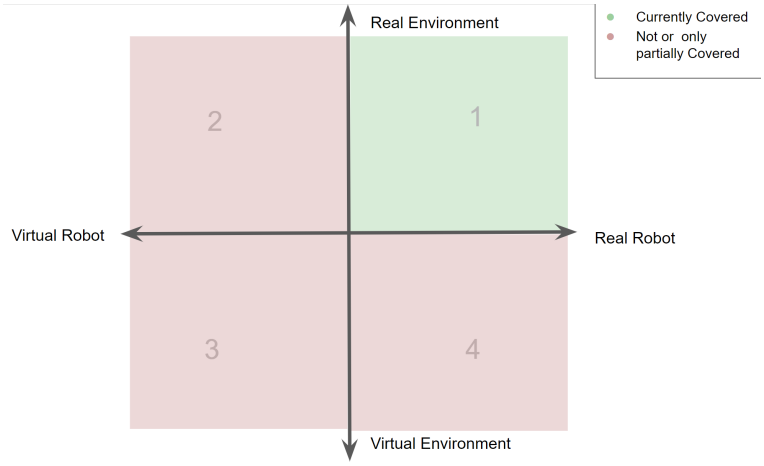


Figure 7.1: Visual representation of the 4 possible combinations of real and virtual robots and environments.

Quadrant 1: Real robot in a real environment. It can be observed without the need for screens or AR, but both the human and the robot have to be physically present in order for the human to see the robot interact with its environment. AR can still be used to visualize data from the real robot in the real environment.

Quadrant 2: Virtual robot in a real environment. This scenario necessitates some form of AR and it could be realized by constructing a virtual representation of the real environment. The inbuilt sensors for the AR glasses, like the HoloLens' Time-Of-Flight Depth Sensor, or the prior scans of a real robot could map the real environment and make it "visible" for a virtual robot. This would allow the virtual robot to navigate and sense the

representation of the real world. To the wearer of the AR glasses, it would seem as if the virtual robot is navigating the real world.
Quadrant 3: Virtual robot in virtual environment. This scenario is the typical robot simulation and can be observed on any regular computer screen. The same setup can also be shown on a device like the HoloLens, by showing both the virtual environment and the virtual robot as holograms. Although no relation to the real world is given or needed, AR could still serve the purpose of having better 3D immersion.
Quadrant 4: Real robot in a virtual environment. This scenario is only observable with an AR device. A real robot will react to and scan virtual environments. A basic implementation of this could be, as proposed by the interviewee, being able to create virtual walls to constrict the movement of the robot. Other use cases could include testing a robot's behavior in scenarios that are not physically buildable, like randomized mazes to test the robot's proficiency in navigation. Other options include simulating a robot in a replica of a real environment that is not accessible for the robot programmer and the robot. A scenario like this Quadrant has been modeled before by Jang et al. [9]. in their system, a swarm of real robots can interact with a virtual environment that is constructed by a person, acting as an "omnipotent virtual giant". This person can perceive the robots and the environment combined fully in virtual reality, with the robots being represented by virtual objects.

By using an AR device like the HoloLens, a platform could be imagined in which all these Quadrants are seamlessly used together. Several robots could interact with one another and their respective environments without physically being in the same space. For development and debugging, this would allow seamless transition from working in a full simulation to slowly replacing the robot or parts of the environment with their real counterparts.

The application realized in this thesis mainly covers visualizations for Quadrant 1. It also allows a virtual robot to be superimposed onto a real environment, not being able to interact with or perceive it, so it does not fully cover Quadrant 2. In order for Quadrant 2 to be accounted for, either the depth sensor of the HoloLens or prior readings by the Turtlebot need to be converted to geometry and interacted with by the virtual robot. The application could cover Quadrant 3 if the virtual robot could scan virtual objects around it, and publish *LaserScan* data based on that. The application currently does not cover Quadrant 4. One could come up with a concept of publishing scan data of an invisible "virtual robot" that scans the virtual environment and tracks the exact movements of the real robot, thereby extending the real robot's perception to virtual objects.

7.3 Generalized and Universal ROS Message Visualizations

Currently, only a limited number of ROS messages are visualized in the application, and unlike using a desktop application that visualizes ROS, the other messages can not even be read or seen at all. This can lead to situations, where a programmer might unexpectedly want access to some information that they initially did not think they needed. Having this information accessible in a primary interface was one of our design consider-

ations, and was also noted to be important by our interviewee. Although our application is mostly concerned with presenting information in the most effective or intuitive way, it should also consider offering as much information as possible, even if some of that information will not be presented optimally.

An imagined solution to this could be, to have a window listing all ROS topics currently being published to, each of which can then be pressed to open new windows for each topic. This new window could default to being a simple console logging all messages on this topic as text. If a more complex visualization has been developed, this could override the default window and instead contain elements (buttons, sliders, text, etc.) that visualize or control the visualization of that ROS message. For example, opening the /scan topic could open a window containing the buttons for turning on and off the point cloud and additional information and settings regarding this visualization. opening the /diagnostics topic, a topic currently not visualized, would then instead only open a console logging all messages received.

This implementation would offer at least some insight into all topics of the robots without diminishing the current visualizations. It would also help with the problem of cluttered interfaces if more visualizations were added, as the hierarchical structure of the windows would hide functions and information not currently needed. If this application is to be used as a stand-alone tool for debugging robots, features like these have to be implemented for the tool to be of significant use.

8

Conclusion

Robots produce a lot of complex data which has to be viewed by people working with the robot. Especially the task of debugging a robot is made even more challenging when working with mobile robots in changing environments and complex 3D worlds that they navigate in. Tools to enable humans, working with robots, to more easily do so, could therefore be of great benefit.
Prior work has shown that augmented reality is a powerful tool for visualizing complex data in situ, superimposed into the real environment, in which a robot is navigating. Taking design considerations from these works, an application was developed on the HoloLens platform that visualizes data as holograms. The data is sent over the Robot Operating System and comes from a mobile robot.
The application tracks a robot's position through the use of its odometry data. Once the position of the robot is determined, a variety of data, like the points recorded by a Lidar sensor, are visualized. A holographic user interface allows the user to configure these visualizations and also see a console logging errors and ROS messages that are received by the HoloLens. The user interface can be moved, resized and interacted with by hand gestures to allow for intuitive controlling. Voice recognition is also used as an alternative means of input.
The information that is visualized in the application is presented in an effective way, that being, allowing a human to view where they intuitively expect this data. But the scope of how much is visualized is quite limited, with quite a few ROS message types not being considered. The performance of the system also suffers from the large amount of data visualized as a point cloud, even with a system in place, that tries to eliminate obsolete or wrong data points. A semi-structured interview showed the potential AR can have for improving the debugging process of robots and also how intuitive the visualizations could feel. Highlighted through the interview were several shortcomings of the application. One of the most important takeaways of the evaluation was the need for adding ways to include data that is not shown at all, but that might become relevant and necessary to view. Regardless of how well this data is shown, it would be helpful to have access to it at all. Especially, if this application is meant to be used as the only debugging tool, and not alongside a traditional computer.

Potential ways of expanding this application to other use cases outside of traditional

debugging of a single mobile robot were also considered. AR visualizations could prove effective in other areas of robotics, like swarm robotics, where information about the relation between the robots could be visualized more clearly.

Further expanded upon was the idea of merging augmented and real robots and environments. As AR enables applications to blur the line between simulation and reality, many possibilities arise. Among them, are the exploration of robot behavior in virtual environments that can not physically be modeled, as well as remote interaction between real robots and environments that are not physically in the same place.

Overall, this application is not yet capable of being a stand-alone debugging tool for robotics programming. Instead, it shows the potential for the intuitiveness AR visualizations and interactions offer and lays the ground for a potential platform of tools for debugging, visualizing, or otherwise improving human-robot interaction through the use of Augmented Reality.

Bibliography

[1] Azuma, R. T. A Survey of Augmented Reality. In: *Presence: Teleoperators and Virtual Environments* 6(4):355–385, Aug. 1997. DOI: 10.1162/pres.1997.6.4.355. eprint: https://direct.mit.edu/pvar/article-pdf/6/4/355/1623026/pres.1997.6.4.355.pdf. URL: https://doi.org/10.1162/pres.1997.6.4.355.

[2] Brooke, J. SUS - A quick and dirty usability scale. In: *Usability evaluation in industry* 189(3), 1996.

[3] Colett T.H.Jand MacDonald, B. An Augmented Reality Debugging System for Mobile Robot Software Engineers. In: *Journal of Software Engineering for Robotics*, 2010.

[4] Collett, T. H. J. and Macdonald, B. A. An augmented reality debugging system for mobile robot software engineers. In: 2010.

[5] Gerkey, B., Vaughan, R., and Howard, A. The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. In: *Proceedings of the International Conference on Advanced Robotics*, 2003.

[6] Green, S. A., Billinghurst, M., Chen, X., and Chase, J. G. Human-robot collaboration: A literature review and augmented reality approach in design. In: *International journal of advanced robotic systems* 5(1):1, 2008.

[7] Ikeda, B. and Szafir, D. Advancing the Design of Visual Debugging Tools for Roboticists. In: HRI '22:195–204, 2022.

[8] Ikeda, B. and Szafir, D. An AR Debugging Tool for Robotics Programmers. In: *HRI 2021I Workshop VAM-HRI Submission*, 2021.

[9] Jang, I., Hu, J., Arvin, F., Carrasco, J., and Lennox, B. Omnipotent Virtual Giant for Remote Human-Swarm Interaction. In: *CoRR* abs/1903.10064, 2019. arXiv: 1903.10064. URL: http://arxiv.org/abs/1903.10064.

[10] Macenski, S., Foote, T., Gerkey, B., Lalancette, C., and Woodall, W. Robot Operating System 2: Design, architecture, and uses in the wild. In: *Science Robotics* 7(66):eabm6074, 2022. DOI: 10.1126/scirobotics.abm6074. URL: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.

[11] Matney, L. *With new realities to build, unity positioned to become Tech Giant*. 2017. URL: https://techcrunch.com/2017/05/25/with-new-realities-to-build-unity-positioned-to-become-tech-giant.

[12] Microsoft *What is mixed reality? - Mixed Reality | Microsoft Learn*. 2022. URL: https://docs.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/?view=mrtkunity-2021-05 (visited on 10/14/2022).

[13] Milgram, P. and Kishino, F. A Taxonomy of Mixed Reality Visual Displays. In: *IEICE Trans. Information Systems* vol. E77-D, no. 12:1321–1329, Dec. 1994.