Validation of Robot Model with Mobile Augmented Reality

Chanapol Piyavichayanon
Department of Creative Informatics
Kyushu Institute of Technology
Fukuoka, Japan
piyavichayanon.chanapol402@mail.kyutech.jp

Masanobu Koga
Department of Intelligent and Control Systems
Kyushu Institute of Technology
Fukuoka, Japan
koga@ces.kyutech.ac.jp

2021 6th Asia-Pacific Conference on Intelligent Robot Systems (ACIRS) | 978-1-6654-4069-1/21/$31.00 ©2021 IEEE | DOI: 10.1109/ACIRS52449.2021.9519362

***Abstract*—Augmented Reality has excellent potential for many robotic applications, including the validation of robot models in working environments. However, a dedicated depth sensor is only available on a high-end mobile device. Hence, the realistic AR experience for robotic applications is not available for a wide range of mobile phones. This paper presents a way to use a robot control algorithm on a virtual robot model in AR without the need for a depth sensor while keeping the depth required feature such as an occlusion. The connection between the AR and the robot operating system allows the exiting control algorithm to be applied to the augmented robot model. The navigation of a mobile robot with the AR interface was used as an example of robot validation in real-world spaces. Despite some communication delays, the virtual robot can be controlled and navigated in the real world with a certain degree of accuracy. It has been shown that visualizing and controlling a robot in an AR scene can be done with this method. Future work on the interaction between the virtual robot and the real environment should be conducted to expand the application of robot validation with AR.**

Keywords—augmented reality, mobile AR, robotics

I. Introduction

Augmented Reality (AR) has become widely used in many fields, from the game industry to medical applications as well as robotic research [1]. One application of AR is visualizing robot models and the data from various sensors. The AR robot model, developed by Disney Research [2], demonstrated the possibility of presenting the virtual robot in real-world space using Hololens[1]. On the other hand, Augmented Reality can also be used as an interfacing tool for controlling robots. Some research has shown the application of AR in mobile robot navigation, such as the swarm robotic [3]. One key common feature introduced in these works is the improvement of Human-Robot Interaction (HRI) by using AR. HRI is challenging in the robot validation process since human interaction is difficult to predict [4]. The study from CARIS Lab [5] successfully used the virtual robot in AR to test and improve the operation between a human and robot. The operator can easily specify the trajectory of the manipulator in a 3-dimensional space with Hololens and sent it to the robot workspace. AR is used to preview the robot's motion in the working environment and increase the confidence of the user before executing the motion of the robot. Inspired by the previous works, we desire to use AR for validating robot models in a real-world environment. However, relying on expensive AR devices will limit the development of AR systems and the accessibility of AR solutions. Alternative devices such as an AR platform on mobile phones seem to be a more affordable choice but their lack of the time-of-flight sensor restricts the capability of an AR technology.

To overcome the limitation, we aims to propose a method for implementing the robot controlling system with ARCore[2] and Unity[3]. We use Depth API [6], a library that provides depth maps from a single RGB camera, to get over the missing depth information. By connecting to a robot operating system (ROS), the robot model in AR can be controlled with the exiting control algorithms. We apply the method to a Turtlebot3 [7] which is a widely used open-source mobile robot. After demonstrating the AR user interface on top of ROS Navigation Stack, we validate it by controlling the virtual mobile robot in the AR to the desired position in real-world space. Finally, we evaluate the performance of the proposed method in terms of navigation accuracy and execution time. These experimental results illustrate the capability of the proposed method in robotic utilization.

II. Related Work

The proposed method is built upon the ARCore toolkit together with the Unity 3D engine. ROS#, a Plugin for Unity, is used for interfacing ROS on Unity. Some additional scripts are required to operate ROS with AR application.

A. ARCore

ARCore is a platform for developing an augmented reality application on a mobile device introduced by Google. The major task of the ARCore is tracking where the mobile phone is in real-world space and finding the surface for placing the AR object. ARCore can also be used on another developing platform like Unity. Two SDKs tools are provided for building an ARCore application on Unity which are AR Foundation and ARCore SDK for Unity. AR Foundation is a cross-platform API that can be built on both Android and iOS devices. On the other hand, ARCore SDK for Unity can only be used on Android devices but provides all features of the ARCore. The ARCore SDK is used for developing the prototype because it supports the up-to-date Depth API.

[1]Microsoft HoloLens: https://microsoft.com/hololens.

[2]ARCore : https://developers.google.com/ar.
[3]Unity : https://unity.com/

Depth API is a programming interface that allows developers to build a depth awareness feature on an android AR application by using depth-from-motion algorithms. Depth API makes many AR features possible to be presented on a mobile phone without a time-of-flight sensor. The examples of these features are presented in the Depth Lab [8].

B. ROS#

ROS# is an open-source software library containing Unity plugin and ROS packages for the communication between two platforms [9]. The provided ROS packages are used to communicate the data to the Unity application via the RosBridge server and send the robot description read from the URDF resource files of the robot model. Oppositely, the provided plugin on Unity can send the data and the robot model from Unity to ROS as well.

ROS# provides a simple method for importing the robot URDF model to a Unity game object. The provided ROS node read through the robot description and send the corresponding mesh file to the computer used to develop the Unity application. After that, the given plugin creates a robot model in Unity using the hierarchy of the game object.

C. ROS and AR

Previous studies have shown the possibility of using ROS with the mobile AR device. The framework for communicating between ROS and the AR using ROS# library has been shown by Krupkeet et al. [10]. The other work demonstrates the use of AR for visualizing the information of the navigation stack on top of the physical space [11]. A similar implementation on mobile devices has been done by iVIZ [12].

Motivated by these works, we developed a new method for visualizing and controlling the robot model on a mobile device AR without the need for a time-of-flight sensor by using Depth API from ARCore.

III. System Design

As stated above, this paper aims to use the Augmented Reality developed on ARCORE to validate the robot by assigning the navigation in real-world space to a virtual robot. Fig. 1 shows the overview of the system in this paper. Starting by designing the developing environment, Unity3D is chosen for prototyping AR on a mobile phone since it supports the development toolkit of an ARCORE. We use Turtlebot3 Burger as a model of the mobile robot in this work. The robot was operated on ROS and simulated in GAZEBO resulted in the need for communication between unity apps and ROS Node. This can be done by using RosBridge which is a ROS package helping the non-ROS program to communicate with the ROS node. ROS#, an open-sourced project from Siemens, present an effortless way to use RosBridge with Unity together with the other useful tools for exporting robot URDF model to Unity game object. After setting up the communication and exporting the robot model, the transformation between different coordinate of ROS environment and AR application needs to be considered to make the virtual robot in AR move to the desired position in a real-world environment and also to give the desired goal in the real world to ROS environment correctly. Then, the

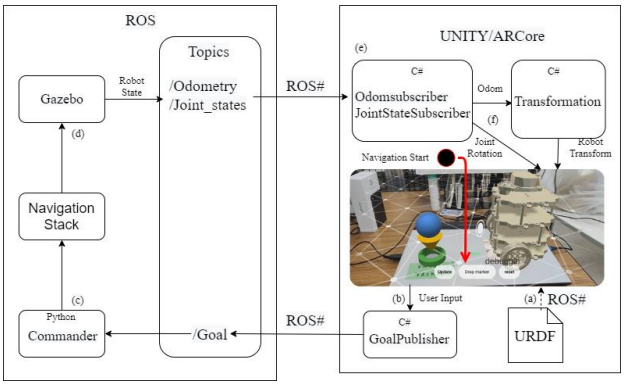

Fig. 1. Overview of The System. (a) Robot Model is imported from URDF with ROS# library. (b) User input data is published to ROS from a mobile device. (c) The desired goal is sent with an action command to the navigation stack. (d) The robot states are published to the ROS topic while the robot is moving toward the goal. (e) The subscribed joint states are sent to the robot joint and subscribed odometry is sent to the Transformation node. (f) The transformation node transforms odometry data and sent it to the robot model.

Depth API is applied to introduce the depth-required features including occlusion and depth cursor. Finally, we connected the AR interface to the navigation stack to validate the mobile robot in the workspace.

A. System Requirement

An ARCore compatible device is required for utilizing the proposed method. The summary of the hardware system specification together with the chosen version of the software development tools are listed in Table I.

B. Communication

To communicate between ROS and the android device, a WebSocket Server is used on top of the ROS# plugin. The Android device, which is connected to the same local network as ROS operating computer, can subscribe to the joint state and

TABLE I. System Specification

Mobile Phone	
Module	Pixel 4a
Chipset	Qualcomm Snapdragon 730G
Operating System	Android 11
Computer	
Module	Alienware m15 R4
CPU	Intel® Core™ i7
GPU	GeForce RTX 2070 Super
Operating System	Ubuntu 20.04.2.0 LTS
Network	
Protocol	802.11ac
Software Development Tools	
Unity	2019.14.18f1
ARCore SDK for Unity	1.22.0
Gazebo	11.3.0
ROS	Noetic

odometry data from the simulation software to move the virtual robot in AR. Before sending the data from ROS to the mobile device, serialization is required. ROS# provides the interface for JSON serialization tools where .Net from Microsoft can be selected.

C. Placing Virtual Robot in AR

A properly coordinate system alignment needs to be considered to ensure that the virtual robot in AR is presented correctly on the real-world scene. Most of the AR-related robotic research uses an image marker to reference the position in physical space. However, we found that using the marker is not very intuitive especially for using on the mobile robot since the reference position may need to change frequently. Thus, we use the anchor module for placing the virtual object on the object tracked by ARCore. ARCore updates the position of the trackable object when the understanding of the environment changes. This helps the referencing object stay in the same place in the real scene even when the mobile phone is moved around in the real scene.

Before changing the robot's transformation to the desired place, the odometry data of the robot in ROS has to be transformed to the world reference frame in ARCore. Fig. 2 demonstrates the corresponding coordinate frame in ROS workspace and ARCore workspace. The odometry in ROS is the same value as the transformation between the robot frame and the anchor frame in the AR scene, but the robot's input transformation is relative to the world frame in the AR scene. Hence, the desired robot position in the world coordinate frame can be found from:

$$ {}^{W}_{R}\mathbf{T} = {}^{W}_{A}\mathbf{T}\,{}^{A}_{R}\mathbf{T} \tag{1} $$

where ${}^{W}_{R}\mathbf{T}$ is a transformation matrix of the robot relative to world frame in AR scene, ${}^{W}_{A}\mathbf{T}$ is transformation matrix of the anchor frame relative to the world frame in AR, ${}^{A}_{R}\mathbf{T}$ is a transformation matrix of the robot frame relative to the anchor frame in the AR scene which is equal to the odometry in the ROS workspace. It should be noted that the coordinate system in Unity uses the left-handed coordinate system, which is different from the ROS coordinate system. Hence, additional transformation is required before using (1).

D. Depth Map Implementation

In reference to Depth API Samples for Unity, accessing depth information can easily be done by adding a game object which provides DepthSource class to the scene. This per-pixel depth data is crucial for applying a depth effect likes the occlusion effect on the AR object. Fig. 3 displays the AR model of the Turtlebot3 with the depth effect provided by the Depth API. In addition to the occlusion effect, we found that this depth information could be used to localize and navigate the robot model in real-world space.

E. AR User Interface

In the example of DepthLab, the navigation of virtual avatars in the AR scene has been shown. It uses the oriented reticle as a depth-aware cursor to locate the goal for the avatar to move. Although it is an intuitive user interface, the error between the desired target and robot position is noticeable. We

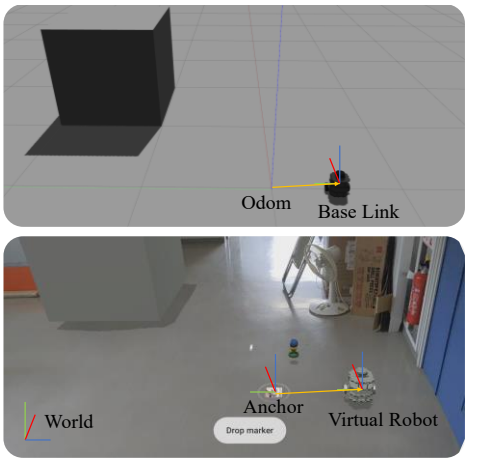

Fig. 2. Relevant Coordinate System where the virtual anchor was used as a reference of the odometry frame in real world. (a) shows the coordinate frame in Gazebo. (b) shows the coordinate frame in AR scene.

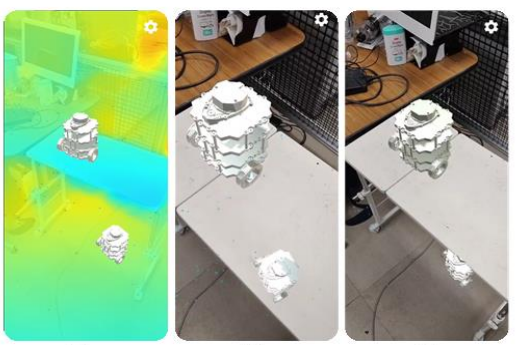

Fig. 3. Visualization of Turtlebot3 on a real-world space. (a) shows a visualization of depth map generated by Depth API. (b) shows a robot model without occlusion effect. (c) shows a robot with occlusion effect

customize the stated method to be more suitable for a mobile robot by ensuring the plane alignment between the robot and desired target. Also, designing a way to specify the desired orientation of the robot by using the angle calculated from the current position of the cursor.

Fig. 4 illustrates the described user interface used in this paper. Since the Turtlebot3 is moving on a flat plane, aligning each reference frame on the same plane is important to increase the accuracy of localization. A raycast function is used to find where the imaginary ray from the camera hit on the imaginary plane created by ARCore. The reticle depth cursor was used to help the user recognize where the raycast would be hit. After placing a referenced frame object and desired goal position, the destination in the world coordinate frame needs to be transformed to the referenced frame before publishing to the robot in ROS. The interaction between human and the navigation of the AR robot is shown in Fig. 5.

For an orientation of the robot, the angle between the depth cursor and the specified goal is used as an input value visualized by the guiding line. The difference of the desired goal and depth cursor is then transformed to the reference frame before publishing the desired rotation to the robot.

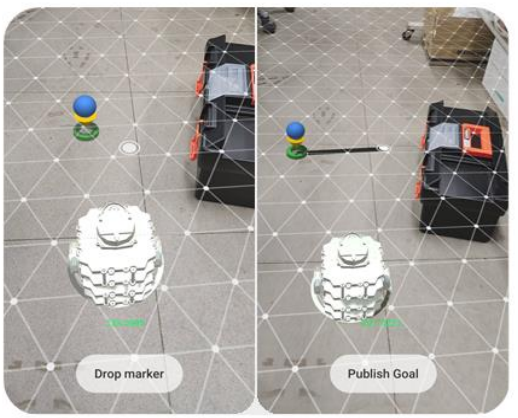

Fig. 4. An AR navigation interface of our system with virtual Turtlebot3 model sitting on the virtual plane. (a) shows a depth cursor for placing a marker on the goal position. (b) shows an orientation input line for determining the goal rotation.

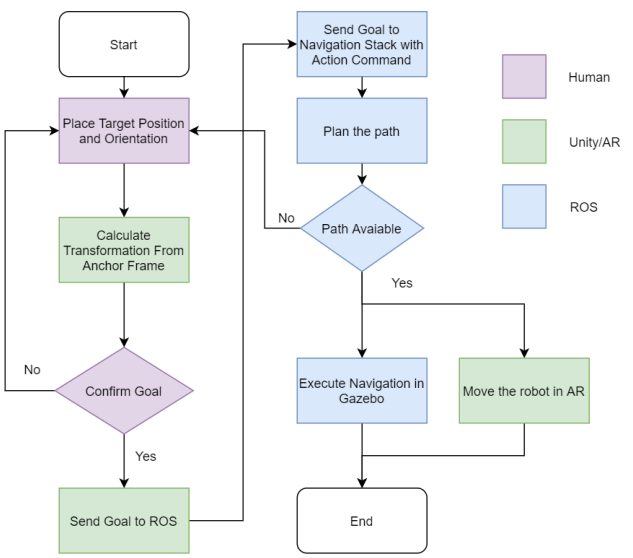

Fig. 5. The navigation procedure of human-robot interaction with AR.

F. *Robot Model for Validation with AR*

We use Turtlebot3 Burger as a demonstration of the robot validation with AR. Instead of using rviz, a typical 3D robotics visualization tool for ROS, the kinematic of the robot can be seen on top of the real environment with a mobile AR. In case of the dynamic model, we use GAZEBO to simulate the mobile robot since it has a ROS interface. After the simulation is calculated, the robot joint state and odometry data are published to the ROS workspace for moving the virtual robot in AR.

The navigation of Turtlebot3 is relying on the ROS navigation stack [13]. The navigation is done on the odometry frame of the robot by publishing the goal derived from the above method to the subscriber node in ROS. After the navigation stack receives the action command from the subscriber node, the planner in navigation would plan the path for the robot in simulation to follow.

IV. Performance Evaluation

In order to evaluate the performance of the proposed method, we evaluated the positioning accuracy and the execution time before the robot starts to move.

A. *Positioning Accuracy*

The accuracy evaluation was done by placing an object in a reference point and another one in a different position. The test positions were select on the principal axis of the reference point 0.5 m away from each other as shown in Fig. 6. The orientation accuracy was measured at 1.5 m away from the reference point at an angle of ±0º, ±45º, ±90º, ±135º, and ±180º. Then, the distances between these two objects were estimated and compared with the measured distance in real-world space. Fig. 7 demonstrates the deviation of the measured distance in the real world and the estimated distance with AR. The accuracy of the positioning was 17.14 ± 3.56 mm, and the accuracy of the input rotation was 0.77 ± 0.01º. The statistical tests were done at the 5% significance level.

The experimental result shows the possibility of using an AR interface to navigate robots in real-world space. The accuracy is compatible with the time-of-flight method [14] which reports 30 mm for static positioning. A similar test on the Hololens [5] can achieve an accuracy of 7 mm on the horizontal plane localization. Although the accuracy of this method is inferior to the one obtained from using the devices with a depth sensor, this level of accuracy can navigate the virtual robot to move to the desired position in physical space as shown in Fig. 8.

B. *Execution Time*

The execution time was recorded after the user published the command until the virtual robot in AR starts to move. In order to investigate the effect of the wireless communication on the delay of the operation, the result was compared with the execution time commanded on the same computer running the robot simulation.

Fig. 9 shows the time required for the virtual robot in AR to start to move. The rise in execution time on the mobile device was about 60 ms. The increase could be due to the additional communications between the programming modules and devices. The increase in execution time might be a severe issue for a specific type of robot application and should be considered when applying the proposed method.

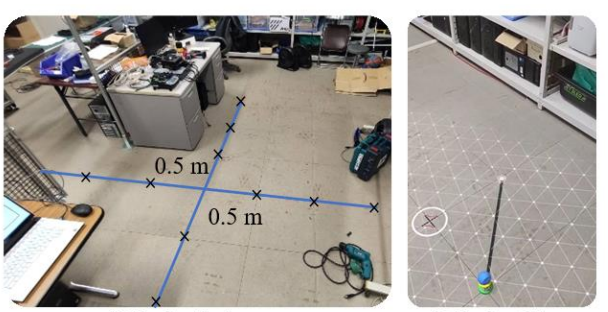

Fig. 6. Scene for testing localization accuracy. (a) shows the range of the testing environment. (b) Shows the virtual object which were used to get the input position and orientation of the robot.

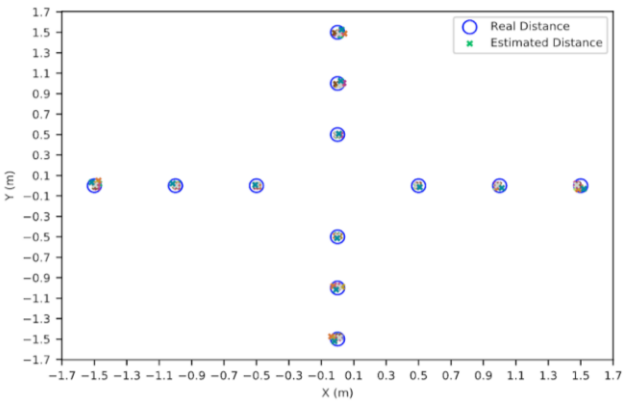

Fig. 7. The positioning of the virtual object estimated by AR and the measured distance in real world

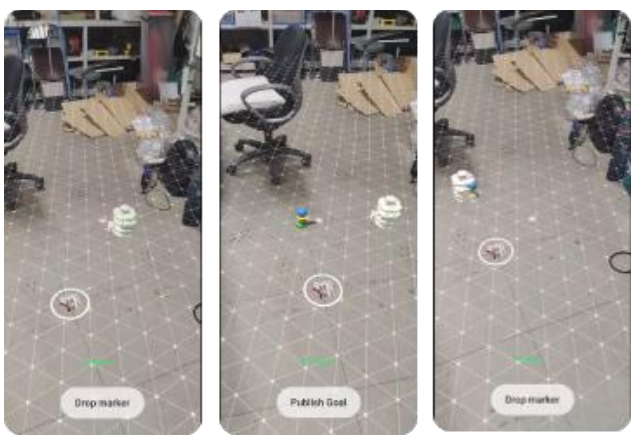

Fig. 8. Navigation of the Turtlebot3 in real environment

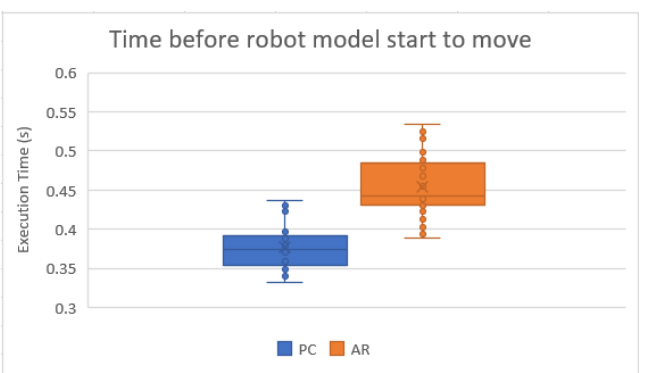

Fig. 9. Time to movement execution for a local-device commander and a mobile-device commander.

V. Conclusion and Future Work

]In this paper, we present the method of visualizing the virtual robot in Augmented Reality for validating the robot in a real-world environment. The integration with ROS is presented to be a shortcut for controlling the virtual robot in AR. As an example of robot validation, we validated the augmented reality localization and user interface with the ROS navigation stack. While the virtual robot model in AR can successfully move to the desire location with a certain degree of accuracy, the proposed method does have limitations.

The accuracy is limited due to the lack of a time-of-flight sensor. In case a higher accuracy is required, using the smartphone with a time-of-flight sensor, or using the additional range finding sensor might resolve the problem. These improvements could be simply made since ARCore support the usage of a time-of-flight sensor on some mobile phone and the implementation of the other sensor is straight forward on ROS.

This paper mainly focuses on the visualization of the robot model in real-world space. However, the robot validation might require the interaction between the augmented robot and the object in physical space, such as collision-free path planning. Hence, the study on other physical properties of the robot model should be conducted to wider the application of AR for robot validation.

References

[1] Z. Makhataeva and H. Varol, “Augmented reality for robotics: A review,” Robotics, vol. 9, no. 2, p. 21, 2020.

[2] G. Cimen, Y. Yuan, R. W. Sumner, S. Coros, and M. Guay, “Interacting with intelligent characters in AR,” 2018.

[3] S. Batra, J. Klingner, and N. Correll, “Augmented reality for human-swarm interaction in a swarm-robotic chemistry simulation,” arXiv [cs.RO], 2019.

[4] M. Webster et al., “A corroborative approach to verification and validation of human–robot teams,” Int. J. Rob. Res., vol. 39, no. 1, pp. 73–99, 2020.

[5] C. P. Quintero, S. Li, M. K. Pan, W. P. Chan, H. F. Machiel Van der Loos, and E. Croft, “Robot programming through augmented trajectories in augmented reality,” in 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018, pp. 1838–1844.

[6] J. Valentin et al., “Depth from motion for smartphone AR,” ACM Trans. Graph., vol. 37, no. 6, pp. 1–19, 2019.

[7] W.Son, “Turtlebot3.” [Online]. Available: https://github.com/ROBOTIS-Will/turtlebot3. [Accessed: 22-Mar-2021].

[8] R. Du et al., “DepthLab: Real-time 3D interaction with depth maps for mobile augmented reality,” in Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology, 2020.

[9] M.Bischoff, “RosSharp.” [Online]. Available: https://github.com/siemens/ros-sharp. [Accessed: 22-Mar-2021].

[10] D. Krupke, F. Steinicke, P. Lubos, Y. Jonetzko, M. Gorner, and J. Zhang, “Comparison of multimodal heading and pointing gestures for co-located mixed reality human-robot interaction,” in 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018.

[11] L. Kästner and J. Lambrecht, “Augmented-Reality-based visualization of navigation data of mobile robots on the Microsoft Hololens -- possibilities and limitations,” arXiv [cs.RO], 2019.

[12] A. Zea and U. D. Hanebeck, “iviz: A ROS Visualization App for Mobile Devices,” arXiv [cs.RO], 2020.

[13] “navigation-ROS Wiki,” Ros.org.[Online]. Available:http://wiki.ros.org/navigation. [Accessed: 22-Mar-2021].

[14] X. Li, Z. Yan, L. Huang, S. Chen, and M. Liu, “High-accuracy and real-time indoor positioning system based on visible light communication and mobile robot,” Int. J. Opt., vol. 2020, pp. 1–11, 2020.