Integrating Augmented Reality and SLAM for Accurate Navigation with Turtlebots

Joey van der Kaaij Arnoud Visser

May 2023

1 Introduction

The Turtlebot is a popular mobile robot platform that has been used for a variety of research applications.[12] One common use-case are navigation experiments, where the robot moves to a specific location autonomously [10]. To achieve this, the robot needs to have a map of the environment and use a localization system to determine its position. The combination is called Simultaneous Localization and Mapping (SLAM), which allows the robot to create a map of its environment and determine its location within that map at the same time [13].

Augmented Reality (AR) is a method of drawing a virtual environment over the real world environment in real-time [8]. Unlike virtual reality (VR), which creates an entirely simulated environment [9], AR enhances the user's existing reality by adding digital elements to it. AR can be used to create immersive experiences where virtual objects appear to interact with the real world [8] or to provide users with additional information about real-world objects [9].

In this technical report, we set out to investigate whether it is possible let an user point to a destination and let the Turtlebot find its way towards that position using SLAM. Specifically, we aimed to determine if it is possible to provide the robot with a destination point in an Augmented Reality environment, and have it autonomously navigate to that point using SLAM-based localization. This means that location and orientations in the virtual world and real world have to be synchronized. The overarching inquiry that we aim to address pertains to the constraints that exist within human-robot interaction, with both the human and the robot moving [11].

In order to create an Augmented Reality environment, we will employ the Unity engine for rendering purposes. We utilized the Meta Oculus Quest 2 as our AR device. The robot platform being used is the aforementioned Turtlebot 3 running Ubuntu 18.04 on an onboard Raspberry Pi 3b.

2 Methodology

2.1 Unity

Unity is a cross-platform game engine and development platform used for creating video games, simulations, and other interactive experiences. Unity possesses a variety of plugins for constructing AR and VR applications [2]. Among these plugins, OpenXR is particularly noteworthy. OpenXR serves as the open standard for facilitating communication between the VR headset, controllers, and the game engine, and is developed by the Khronos Group.

It's also possible to simulate the Turtlebot in Unity through tools provided in the Unity Robotics Hub [5].

2.2 Meta Oculus Quest 2

The Meta Oculus Quest 2 is a standalone VR device, meaning it does not require a connection to a PC or console in order to function [1]. The device is equipped with various sensors and cameras, enabling it to track the user's movements and the environment around them. Additionally, the Meta Oculus Quest 2 possesses passthrough capabilities that allow the user to view the outside world through the headset's external cameras. The Meta Oculus Quest 2 is primarily used for gaming but also has potential applications in education, training, and other fields that could benefit from VR or AR technology.

The Meta Oculus Quest 2 comes with two wireless controllers that are specifically designed for VR and AR experiences. These controllers feature a range of buttons and triggers that allow for intuitive interaction with virtual objects and environments. The controllers are also equipped with sensors that enable precise tracking of their movements and orientation in 3D space, which provides a more immersive and realistic experience for the user. The Meta Oculus Quest runs Android as the operating system.

2.3 Turtlebot & ROS

TurtleBot is a compact and modular robot that consists of a mobile base, a Raspberry Pi computer, sensors, and other hardware components [12]. In this case we have used the Turlebot 3 Burger configuration (see Fig. 1). It uses ROS (Robot Operating System), a popular open-source framework for robotics, for communication and control. ROS consists of a communication layer for different packages to communicate with each other. To be able to communicate with ROS from the Unity application, Unity provides a project containing a bridge for subscribing to ROS topics and receiving and sending ROS messages [4]. This bridge is running in a docker container with provides a ROS2 Galactic environment. In theory, this should make the integration between Unity and ROS a lot simpler.

Figure 1: The TurtleBot 3 Burger

2.4 Nav2

ROS2 contains the Nav2 package for autonomous navigation, which provides a framework for building and deploying navigation systems for mobile robots [6]. Nav2 can be integrated with the SLAM Toolbox (Simultaneous Localization and Mapping) [7]. Giving the robot the functionality to map the area while navigating. SLAM Toolbox is a set of tools for 2D mapping.

2.5 The AR application

In Unity, a scene is constructed including the OpenXR Rig. This rig contains all the standard functionality for head and controller tracking. The rig translates the position and orientation of the real-life headset and the 2 controllers to the virtual counterparts. Also, there is a floor object made which consists out of a plane mesh object with the Box Collider component attached.

As shown in Figure 2 a ray is cast in a virtual environment using Unity's build in ray casting functionality that points straight out of the front of the right controller. The ray-casting component interacts with collider components in the scene to detect possible collisions. Various geometric shapes can be used as the collider components depending on the type of collider component you choose. The Box Collider component draws a box around the mesh of the GameObject that the component is applied to. The constraints of the box can be further refined. When a collision is detected between a ray and a collider, the raycaster can be queried for the position of the occurred collision in world space. By pointing the controller at the floor object and letting the ray collide with the floor box collider it's possible to detect the position in the virtual environment. The headset tracking makes it so that the detected position directly corresponds to a position in the real world. It's important to understand that the real-world position is relative to the user and not to the Turtlebot.

By enabling the pass-through functionality, the images that the cameras on the outside of the headset record, become visible to the user. By projecting the earlier described virtual scene on top of the pass-through image you are enabling the user to point a virtual ray to a position in real space. See Figure 2:

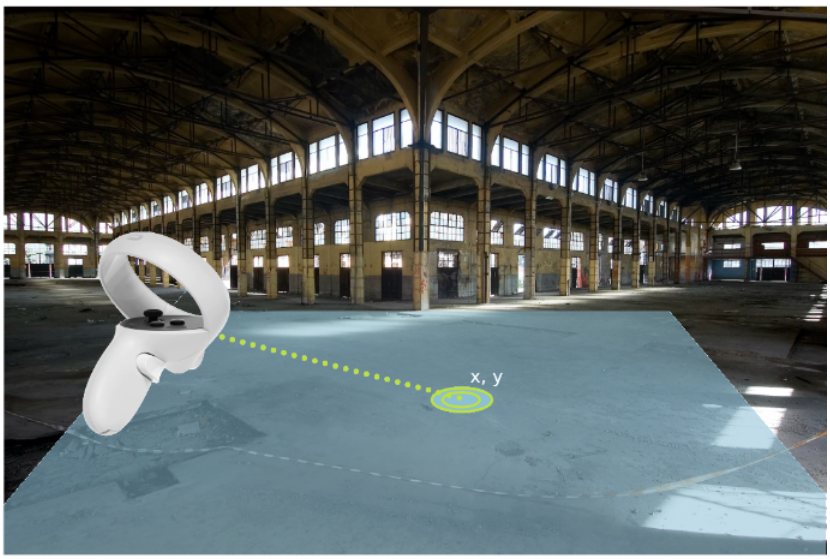

Figure 2: In the virtual scene a controller casts a ray that collides with the floor object and the world position can be queried. What the user of the VR headset would see as the virtual scene is projected onto the pass-through image.

2.6 AR Headset to Turtlebot communication

The AR application requires a means of establishing direct communication with the Turtlebot in order to effectively transmit the selected position. As an initial step, the aforementioned Unity bridge to ROS was tried. The package provides a way for Unity to communicate with a ROS device over the network. By sending a message to the *goal_pose* ROS topic, the Nav2 stack should be able to try and navigate the robot to that position.

The functionality was satisfactory when tested within the Unity editor. However, when attempting to deploy the application for the Android platform on an AR headset, complications emerged due to incompatibilities of certain libraries provided by the ROS package with the Android system. Consequently, an alternative approach had to be devised to establish communication between the AR headset and the Turtlebot.

A solution was found in the *rosbridge_suite* ROS plugin. This plugin allows you to send ROS messages directly over a websocket connection. This enabled the Android headset to send the *goal_pose* topic with positional parameters to the Turtlebot.

2.7 Turtlebot ROS Nav2

Running the ROS2 *nav2_bringup* package enables the Turtlebot to keep track of its own position and orientation on a virtual map and navigate to a positional point on that map, avoiding obstacles in the real world. To avoid obstacles, Nav2 needs a point cloud representation of the surroundings. The map can either be loaded into Nav2, or - in our case - it can create the map in real-time through a LiDAR sensor.

To create this map another ROS2 package is needed to be running called the SLAM Toolbox. This package takes in the LiDAR data and generates a map of the surroundings.

With both these packages running the Turtlebot is now capable of finding a position it receives through the *goal_pos* topic while mapping its surroundings and avoiding obstacles.

The ROS2 commands to run all packages:

```
ros2 launch turtlebot3_bringup robot.launch.py use_sim_time:=False
ros2 launch nav2_bringup navigation_launch.py use_sim_time:=False
ros2 launch SLAM_toolbox online_async_launch.py use_sim_time:=False
ros2 launch rosbridge_server rosbridge_websocket_launch.xml
```

3 Results

At first, the application was built as a simulation only in Unity [3]. Here the functionality of ROS2 and NAV2 were used but not on a real Turtlebot. The simulation would move the Turtlebot and pass back simulated LiDAR readings from Unity ray casts. Here everything worked smoothly. The VR controller would pass in a position and the virtual Turtlebot would move towards this position, finding its way autonomously around obstacles with the help of Nav2 and SLAM.

Moving this to the real-life Turtlebot was more problematic. The correct positional data was coming from the VR headset and made its way into the Turtlebot. This was verified with RViz. The orientational and positional representation of the Turtlebot in RViz started off correctly (see Figure 3). The SLAM readings of the surroundings would also be drawn correctly into the point cloud map. This would work initially but as soon as the Turtlebot started moving it would quickly veer off course or just stop altogether.

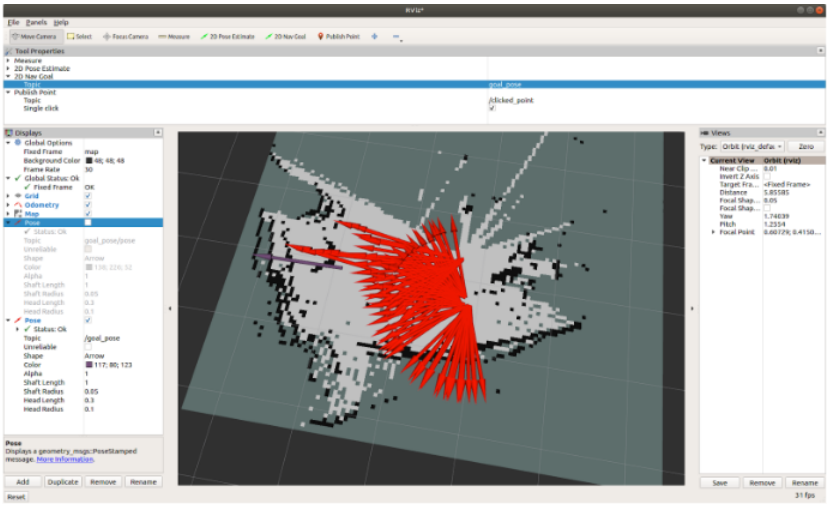

Figure 3: RViz Turtlebot orientation and SLAM point cloud

Trying to troubleshoot this behavior it was found - by looking at RViz - that the reference frame that the Turtlebot had of itself would move off the map in a seemingly random way. This likely caused the pathfinding to get confused about how to get to the target point - either being blocked by the point cloud map or just being too far away from the goal position.

Why the reference frame would move off the map is something that still remains unanswered. At the moment of running, there were a couple of messages given by the Turtlebot that seemed interesting but were maybe unrelated.

```
[async_SLAM_toolbox_node-10] [WARN] [1664535117.594713708]
[SLAM_toolbox]: maximum laser range setting (20.0 m) exceeds
the capabilities of the used Lidar (3.5 m)

Control loop missed its desired rate of 20.0000hz
```

At the time of trying to develop this, there was a suspicion that the problem arose from the limits of the processing power provided by the Raspberry Pi board in the Turtlebot. The message of the control loop missing its desired rate of 20hz might point in that direction. Another option could be - as reported later by another user of the Turtlebot - that there might be problems with the odometry sensor. Further investigation would be needed to understand if that is indeed what threw off the navigation.

An aspect that has yet to be explored pertains to the positioning of the Turtlebot with respect to the user. The root position for both the AR environment and the Nav2 environment should be common. This works correctly when the VR user recenters standing closely to the Turtlebot at the start up of both applications. It's unclear how this would function when the experience would go on for a while and possible drift in both of the applications could occur.

A possible solution here is to point at the Turtlebot with the controllers from within the AR application from time to time. This would locate the rough position of the Turtlebot within the frame of the AR application. With that location, it might be possible to detect and account for any discrepancies.

Another option to be explored is to upgrade the Ubuntu and ROS version on the Rasberry controlling Turtlebot, which allow to communicate directly from the ROS2 docker running on the Unity desktop to the ROS2 packages running on the robot.

4 Discussion

Augmented Reality (AR) technologies hold substantial potential to enhance human-robot interaction. By leveraging AR, the communication process between the user and robotic systems can be significantly enriched, enabling a more comprehensive understanding of the situational context and providing more effective control over the robots.

Unity serves as a robust platform for integrating with AR hardware. Moreover, it also offers sophisticated solutions for conducting simulations and facilitating communication within the robotic domain. This dual functionality underscores its importance in the realm of AR and robotics.

However, the functionality could be further improved if the ROS–Unity Integration toolkit were capable of running on Android systems. Currently, rosbridge and C# websockets facilitate direct message transmission from the Oculus Quest 2 to the robot. Yet, the ROS-Unity Integration Toolkit offers a more robust method for defining messages, as well as publishing and subscription endpoints. This is largely attributable to the type system inherent in C#.

The Turtlebot platform, despite occasional setbacks in its odometry capabilities, remains an invaluable resource for testing and developing Simultaneous Localization and Mapping (SLAM) systems.

References

[1] https://developer.oculus.com/get-started-device, 2023.

[2] https://docs.unity3d.com/manual/vroverview.html, 2023.

[3] https://github.com/jvanderkaaij/vrrobocontroller, 2023.

[4] https://github.com/unity-technologies/robotics-nav2-slam-example, 2023.

[5] https://github.com/unity-technologies/unity-robotics-hub, 2023.

[6] https://navigation.ros.org, 2023.

[7] http://wiki.ros.org/slam_toolbox, 2023.

[8] Yunqiang Chen, Qing Wang, Hong Chen, Xiaoyu Song, Hui Tang, and Mengxiao Tian. An overview of augmented reality technology. *Journal of Physics: Conference Series*, 1237(2):022082, jun 2019.

[9] Pietro Cipresso, Irene Alice Chicchi Giglioli, Mariano Alcañiz Raya, and Giuseppe Riva. The past, present, and future of virtual and augmented reality research: A network and cluster analysis of the literature. *Frontiers in Psychology*, 9, 2018.

[10] Chandran Nandkumar, Pranshu Shukla, and Viren Varma. Simulation of indoor localization and navigation of turtlebot 3 using real time object detection. In *2021 International Conference on Disruptive Technologies for Multi-Disciplinary Research and Applications (CENTCON)*, volume 1, pages 222–227, Nov 2021.

[11] Thomas Sheridan. Human-robot interaction: Status and challenges. *Human factors*, 58, 04 2016.

[12] Diksha Singh, Esha Trivedi, Yukti Sharma, and Vandana Niranjan. Turtlebot: Design and hardware component selection. In *2018 International Conference on Computing, Power and Communication Technologies (GUCON)*, pages 805–809, Sep. 2018.

[13] Sebastian Thrun. *Robotic Mapping: A Survey*, page 1–35. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.