TII Digital Twin Assignment

Abdul Rehman

May 16, 2024

1. Summary

Following the instructions provided in the assignment, I focused on creating a digital twin for a Unity-based example referenced in the assignment outline. This example contains a Unity-based simulation of Turtlebot3 on which the ROS2-NAV2 stack has been deployed for SLAM-based navigation. The example includes several scenes, such as an empty world, an obstacle course (basic scene), and a warehouse. Given the time constraints, I chose a simplified version of the basic scene by converting it into a 4.5m x 4.5m walled room with primitive objects spread around. This setup was then replicated in a Gazebo-based simulation. The final application is multi-containered, where one container contains the ROS2-NAV2-Gazebo setup running on a predetermined ROS_DOMAIN_ID, and the other contains the ROS2-NAV2 setup with Unity running on the host, communicating through an exposed port. The two containers are integrated using Docker Compose, which serves the full application. I have written a bash script for publishing a PoseStamped message simultaneously in both containers in an asynchronous fashion to compare the two setups.

2. Implementation Details

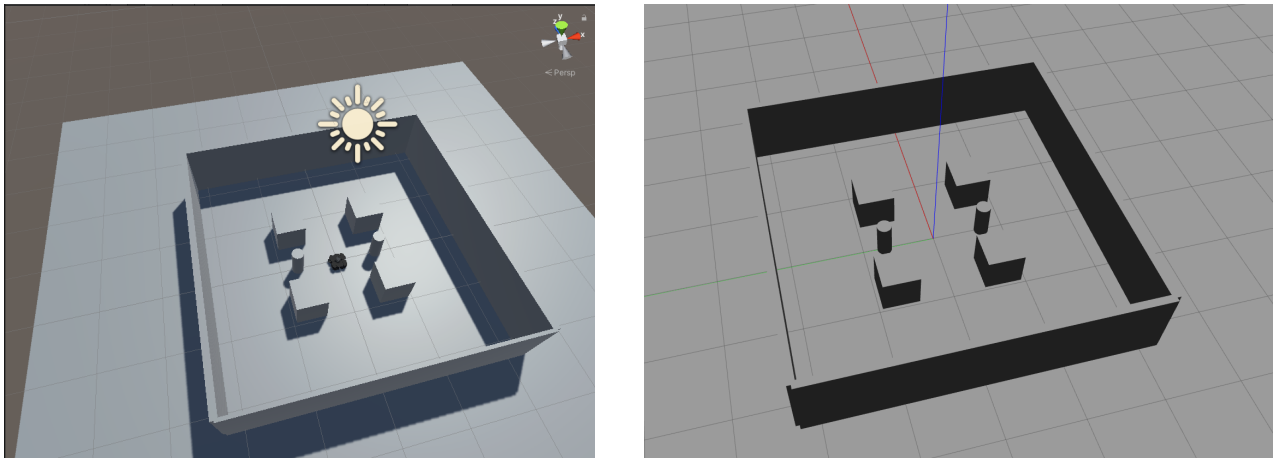
2.1. Unity Simulation

The Unity example implements a simulation of ROS2 NAV2-SLAM deployed on Turtlebot3. It utilizes the standard NAV2 stack with custom controllers, a LiDAR sensor, and a clock to manage the Unity-based simulation. Since the NAV2 stack is widely adopted in both industry and academia, my task involved replicating the environment, controllers, and sensors in a Gazebo-based simulation while utilizing the same NAV2 stack for navigation. The Unity-based implementation includes a perfect laser scanner and a differential drive controller with a maximum linear speed of 2 m/s and an angular speed of 1 rad/s.

The implementation includes several scenes, including a complex warehouse, a relatively simpler obstacle course, and an empty world. Given the time constraints, I modified the simpler obstacle course into a walled room with primitive shapes spread around, and used it as my ground truth world model to create a digital twin for.

2.2. Gazebo Simulation

The simpler world in the Unity simulation helped me replicate it using standard primitive shapes in SDF to write the world file for Gazebo. The two simulations are shown in Figure 2.1. To replicate the Turtlebot3 setup, I utilized the *turtlebot3_simulations* package by Robotis, which includes standard TurtleBot URDF models and Gazebo plugins for LiDAR scanners and differential drive controllers. However, the stack from Robotis required modifications to perfectly mirror the Unity simulation. I cloned the *turtlebot3_gazebo* package locally and made the necessary adjustments to the nodes and plugins, including setting minimal noise in the LiDAR scanner to match Unity's perfect laser, clipping the maximum linear and angular speed to match that of the Unity simulation, and adjusting the frequency of published topics.



(a) Unity World (b) Gazebo World

Figure 2.1: Comparison of Unity World and Gazebo World

2.3. Choice of ROS2 Version

I initially started with ROS2 Foxy but encountered significant issues due to breaking changes in the NAV2 stack for Foxy. Given its LTS status, I initially chose ROS2 Foxy over Galactic. I faced challenges upgrading to Ubuntu 22.04 on my personal machine, making it difficult to run a local setup of the more stable ROS Humble. To expedite development, I switched to ROS2 Galactic.

2.4. Docker Containerization

I created self-contained Docker containers for both the digital twin and the Unity simulation. The goal was to deploy them together using Docker Compose. However, Unity's discontinuation of manual activation for personal licenses posed a challenge. Although I considered creating an image with Unity Hub for login and activation, the large size of Unity images and

time constraints led me to abandon this approach. The final setup includes one container for ROS2 NAV2 with Gazebo and another for ROS2 NAV2 with ROS TCP, exposing port 10001 for communication with Unity hosted locally.

2.5. Deployment and Execution

The containers are deployed using Docker Compose. A script was written to publish goal-stamped messages in both containers simultaneously in an asynchronous fashion. This setup can be visualized in the supplementary video attached.