EAI Endorsed Transactions
on AI and Robotics

Research Article EAI.EU

Integrating Virtual Reality and Robotic Operation System (ROS) for AGV Navigation

Ata Jahangir Moshayedi[1*], KM Shibly Reza[1], Amir Sohail Khan[1], Abdullah Nawaz[2]

[1]School Of Information Engineering, Jiangxi University Of Science And Technology, No 86, Hongqi Ave, Ganzhou, Jiangxi, 341000, China
[2]Department of Electrical Engineering, University of Engineering and Technology, University campus, University Rd, Rahat Abad, Peshawar, Khyber Pakhtunkhwa, Pakistan

Abstract

The use of AGVs (Automated Guided Vehicles) is rapidly expanding in various applications and industries, meeting the growing demand for automated material handling systems. However, AGV control and navigation remain a challenge. To address this issue, robotics simulators such as ROS (Robot Operating System) have become widely used, reducing the cost and time of checking robot performance. Furthermore, the integration of virtual reality technology into the robotics field has facilitated the study of various robot behaviors in realistic environments, replicating the robot's real-life size and dimensions. In this study, the TurtleBot2i and RAZBOT AGV robot platforms were integrated into the 3D Unity environment and controlled using ROS. Using Unity as a simulator for the robot's working environment offers several benefits, including high-quality graphics and a detailed examination of the robot's behavior. The results of the study demonstrate the accurate simulation and control of the AGV platforms in both ROS and Unity environments.

Received on 24 March 2023; accepted on 11 April 2023; published on 21 April 2023

Keywords: AGV, automated guided vehicle, Robotics, Robot Operating System, ROS, Unity.

Copyright © 2023 AJ. Moshayedi *et al.*, licensed to EAI. This is an open access article distributed under the terms of the Creative Commons Attribution license, which permits unlimited use, distribution and reproduction in any medium so long as the original work is properly cited.

doi:10.4108/airo.v2i1.3181

1. Introduction

The use of robotics simulation in virtual environments has emerged as an essential element of robotics research and development [1]. It affords engineers and researchers the opportunity to fine-tune their algorithms, designs, and control systems before actualizing them in physical robots. This approach not only provides a secure and regulated testing environment but also mitigates the risks of damage to the robot and its surroundings. Two of the widely used tools for simulating and controlling robots are Unity and ROS (Robot Operating System) [2].
Unity has recently gained widespread popularity in robotics simulation due to its user-friendly interface, real-time rendering capabilities, and robust support for 3D environments [3]. Conversely, ROS has become the industry-standard robotic software development platform, offering various functionalities such as real-time communication, perception, and control [4]. The integration of these two tools can enable researchers and engineers to construct highly realistic and sophisticated simulations that accurately represent their robots' real-world behavior [5, 6].
As artificial intelligence, computer vision, and dynamic path planning advance [12], robots can now perceive, learn, and move autonomously in the world [13]. As the complexity of robotic tasks increases, the number of tests and validations required to ensure performance increases exponentially [14]. This presents a significant issue, as robots are expensive, and hiring someone to work with them is even costlier [15]. Finding an available space to set up a test environment is also potentially expensive and time-consuming. Fortunately, simulation can alleviate many of these issues [16]. Users can iterate much more rapidly on algorithm design, and they can also have a say in the robot's design decisions. Another significant advantage is that it permits testing over a vast array of varied environments [17]. Therefore, simulating robots in

*Corresponding author. Email: ajm@ jxust.edu.cn

virtual environments is considerably quicker and easier to prototype and iterate than in real life.
To simulate a robot, the robot's task must be defined, and an approach for how the robot will accomplish that task must be developed [18]. The user can then write code that will execute that approach and deploy it in a simulated robot and environment. The results from the simulation can then be evaluated. If the results align with the user's desired outcome, it will be implemented; otherwise, the user will try again until they achieve the acceptable performance metric [19]. Weak performance necessitates a real-world test, where the user can deploy it to the actual robot and evaluate it in a real-world scenario. If the test succeeds, the user can then proceed to production [20].
The simulation workflow can be distilled into three pivotal elements, comprising a mechanized agent navigating through a dynamic milieu, where a particular kind of regulatory algorithm shall be executed on the machine itself [21]. Unity boasts an extraordinary aptitude in the first two constituents, but its sole deficiency is the lack of software responsible for overseeing the robot's operation. Unity offers an editor equipped with a vast range of functionalities, amenities, and auxiliary tools, which are intentionally calibrated to cater to the exigencies of robotics [22].
Many teams have contributed unique approaches to automaton simulation and interaction research, based on different requirements. Game engines have commonly been employed in this context. Recently, all-purpose frameworks that combine game engines with ROS have been proposed [23]. A variety of robot simulators have been in use both before and after the release of ROS. Understanding the history of robot simulation provides context for the current state of the art. Using tools like Unity and ROS to simulate robots in virtual environments has become increasingly popular in recent years, for the purposes of testing and refining robotic algorithms, designs, and control systems [24].

The Unity platform proffers a highly intuitive user interface along with an array of real-time 3D rendering capabilities. In contrast, the Robot Operating System (ROS) furnishes an extensive gamut of functional utilities, encompassing the realms of real-time communication, perception, and control [25]. Through the harmonious integration of these two efficacious instruments, researchers and engineers can materialize impeccably authentic and intricately intricate simulations, mirroring the real-world behavior of their robotic counterparts with impressive verisimilitude [26].

The employment of virtual environments to simulate robotic systems entails an array of advantages, most notably the opportunity to scrutinize and optimize control mechanisms prior to their deployment on tangible robots [27]. This method serves to conserve temporal and material resources while diminishing potential hazards to both the robotic apparatus and its environment [28]. Furthermore, the simulation of robotic systems in virtual environments affords a well-ordered and replicable arena for testing, which is particularly significant for tasks necessitating exacting manoeuvres or for appraising the efficacy of disparate control algorithms [29, 33].

A plethora of investigations have incontrovertibly exhibited the efficaciousness of utilizing Unity and ROS for the purposes of stimulating and regulating robots. Schön and colleagues (2012) expounded upon the Unity robot simulator as an erudite platform for the development of robot software. Similarly, Chitta and co-authors (2012) elucidated the employment of Gazebo and ROS for the simulation of robots. Collectively, these empirical inquiries attest to the great potential that ensues from integrating Unity and ROS, resulting in the creation of intricate and verisimilar robot simulations [34, 38].

This paper main contribution can be listed as bellow:

1. Integration of virtual reality technology with ROS to simulate the robot's performance in realistic environments bwith a better graphical interface.

2. Using the TurtleBot2i and Razbot AGV robots platforms were integrated into 3D Unity environment and controlled using ROS.

3. To demonstrate the accurate simulation and control of AGV platforms in both ROS and Unity environments.

This paper is arranged as follows : The section 2 shows the Simulator Environments like Unity and ROS. The section 3 shows the robot platforms that used. The section 4 shows the modelling and simulations while the section 5 describes the results and discussions of this paper.

2. Simulator Environment

This study employed a duo of simulator environments, namely Unity and ROS. Each of these virtual realms was scrutinized in detail to yield a comprehensive account of their respective features and functionalities. Unity, an established game engine, offers a visually-rich and user-friendly interface that allows for the creation of interactive and immersive 3D environments. On the other hand, ROS (Robot Operating System) is a popular platform for robotic simulation and control, well-regarded for its robustness and flexibility in accommodating diverse robotic hardware and software configurations. The present study availed itself of the unique capabilities of both Unity and ROS, thus enabling a more nuanced exploration of the research questions at hand.

2.1. Unity

The Unity engine is a highly flexible and versatile platform that caters to the requirements of game developers striving to fashion elaborate and aesthetically pleasing games [39]. A salient feature of the Unity framework is its Game Simulation, which aptly satisfies the exigencies of game developers. Notably, it is imperative to acknowledge that Unity is not a simulation instrument but rather a visualization instrument [40]. This attribute renders it an exemplary resource for virtual reality (VR) endeavors on a diverse range of platforms, including OSX, Windows, MAC, and Android [41].

The Unity editor is endowed with an exceptional degree of configurability, which facilitates the availability of community-crafted bespoke user interfaces in the Unity Asset Store, a number of which are proffered gratis [42]. Unity proffers a comprehensive suite of extensions via a C# programming interface, thus empowering users to configure environments and user interfaces, tweak game physics and lighting, animate objects, debug, profile, and undertake a plethora of other functions [43].

The Unity engine is a highly versatile platform capable of accommodating a diverse range of game genres, thereby fostering a work environment that is both streamlined and unencumbered [44]. Nevertheless, it is crucial to acknowledge the instrumental role played by community support in shaping the development of specific game genres and workflows. At the core of the engine's programming architecture lies Unity's GameObject, which is augmented by a Transform component that is hierarchically linked to it within the scene [45]. This component affords the entity in question a precise location, rotation, and scaling within the 3D environment, thereby enabling the seamless integration of ancillary components such as rendering, physics, and animation [46].

The unparalleled ease of use and extensibility that Unity proffers make it an invaluable tool in the context of robot development workflows, especially within the realm of AI simulation. The C# programming interface and other sophisticated tools that are offered by Unity have been specifically designed to aid roboticists in their endeavor to more readily utilize AI simulation [47]. This versatility that Unity offers is a direct consequence of the plethora of community-generated resources that are available to users, including an assortment of 3D and 2D models, AI scripts that enable pathfinding algorithms, and the provision of comprehensive environments or object sets [48]. Lastly, it is worth noting that Unity does not possess a traditional point of entry. Instead, it employs inherited methods that are activated by specific events [49].

How to Leverage Unity Using Robotics. Within the domain of advancing robotic applications, the salience of simulation as a fiscally judicious and temporally efficient modality for generating and scrutinizing applications has been rapidly gaining impetus. The development and evaluation of applications necessitating mechanical functionalities in real-world environments pose a formidable quandary and incur prodigious expenses. However, the assimilation of simulation technology within these realms can assuage the temporal exigencies for iterations and expedite the identification of incipient intricacies[50].
Furthermore, the utilization of simulation presents itself as a propitious sanctuary that enables a comprehense scrutiny and evaluation of peripheral circumstances or contingencies that may imperil genuine, real-world scenarios. In the domain of a masterfully constructed artificial intelligence simulation, the tangible features of the automaton, the operational milieu, and the computational algorithm that impels the mechanism all represent crucial elements that are essential for guaranteeing optimum efficiency and effectiveness [51]. Ensuring the veracity of assessments and instructional approaches is a crucial undertaking that necessitates the verification of congruity between the tripartite constituents and authentic contexts. Simulation technology offers a platform that emulates reality and engenders all-encompassing mechanisms to generate and scrutinize applications, thereby rendering them amenable to verification and validation prior to deployment. This modality exhibits the potential to curtail developmental expenditures, expedite time-to-market, and enhance the precision and dependability of the application. By enabling developers to scrutinize and refine their applications prior to their deployment, simulation technology affords a cost-effective and adroit technique for generating innovative and trustworthy applications [52].

2.2. ROS

Robot Operating System (ROS) is a comprehensive framework that comprises a suite of tools, libraries, and protocols aimed at facilitating the development of various robotic systems. The system manages the creation and control of communication between a robot's peripheral modules, such as sensors, cameras, and actuators, thereby enabling the seamless integration of these components. Initially, the

conception of ROS took place at Stanford's Artificial Intelligence Lab, and the development continued at Willow Garage. ROS is primarily designed to function optimally on Ubuntu, while only partially compatible with other operating systems, such as Windows and Mac. As an open-source project, ROS boasts an extensive community of contributors who work tirelessly towards enhancing its functionalities, thus making it more accessible and useful for various robotics applications.
The unparalleled adaptability and durability of the Robot Operating System (ROS) has established it as the top preference for both developers and researchers, who have effectively utilized it to create a diverse range of robotic systems spanning from industrial manipulators to self-governing drones. ROS's modular configuration enables effortless integration of other software and hardware components, resulting in a multi-faceted platform for experimentation and swift prototyping. The widespread adoption and versatility of ROS have instigated the development of numerous third-party packages and libraries that further enhance its capabilities. Overall, ROS is a pivotal tool for individuals seeking to develop robotic systems, providing an unwavering and flexible framework for the cost-effective and efficient construction and testing of robots.Figure 1 depicts the intricate and multifaceted communication flowchart within the Robot Operating System (ROS). This visual representation not only highlights the intricate interconnections among the various nodes but also accentuates the complex nature of the communication system. The portrayal of this intricate system within the purview of figure 1 serves to elucidate the manifold intricacies of the ROS architecture, providing a comprehensive and systematic understanding of the communication processes that underlie this sophisticated robotic system.

Figure 1 presents the communication flowchart of Robot Operating System (ROS) between two computers, wherein one computer houses a simulator world containing the Slm robot and sensor, and the other comprises custom code under ROS. This setup enables evaluation of the system in the simulator world, improving it via custom code, and finally deploying it to ROS for further use.

ROSBridge. Rosbridge is a cutting-edge application programming interface (API) that enables the seamless integration of non-ROS applications with the full range of capabilities offered by the Robot Operating System (ROS). This innovative technology leverages the JSON data format and is comprised of a set of state-of-the-art front-end tools, including a WebSocket server that is compatible with web browsers, the rosbridge package, and various front-end packages. The protocol of rosbridge comprises a meticulously defined set of guidelines that are specifically designed to facilitate the transmission of JSON-based commands to the ROS operating system, enabling communication with ROS in any language or transport that is proficient in transmitting JSON. Thus, RosBridge confers the ability to communicate with a ROS system over a network, rendering it an indispensable tool for both roboticists and software developers alike. Significantly, RosBridge can harness the power of ROS without modifying pre-existing architecture, thus constituting a valuable asset [53].

ROS#. The Ros# is known as a highly intricate apparatus that functions to extend the Unity editor's ability. as the primary it has the objective to enable a seamless exchange of data between the Robot Operating System (ROS) and Unity, through the implementation of the advanced RosBridge protocol.The comprehensive feature set of Ros# is characterized by a vast array of tools, with the robot description importer standing out for its exceptional proficiency in supporting URDF files, thereby promoting streamlined integration with ROS. This multifarious software tool finds heterogeneous and multifaceted applications, spanning from the visualization of robots and sensors to the facilitation of system simulation and the enabling of remote operations. The versatility of Ros# is supported by a plethora of research studies, as attested by reference to a substantial body of literature [54].
Figure 2 provides an overview of the communication procedures in the simulation process, using the ros-sharp/master repository with its three modules: Libraries, ROS, and Unity3D. The ROS module has three packages, with the file server package being important as it contains the file server service node.
The Libraries module is an abstraction of the RosBridge protocol, providing access to the file server service and URDF transfer capability. The Unity3D module expands the Unity Editor and 3D modelling and offers Unity run-time scripts for example applications. Some examples can be reused in future ROS-Unity applications. Two binary files, RosBridgeClient.dll and URDF.dll, are generated by the Libraries module and must be placed in the Unity project folder.

3. Robot Platforms

There are several robot platforms available for use in both ROS and Unity, but two platforms, in particular, stand out as being highly prominent.TurtleBot2i [55] is a highly acclaimed mobile robot platform that has been tailored for research, educational and experimental purposes. It is outfitted with an array of sensors, including a 3D camera and laser scanner, and can be flexibly adapted with supplementary

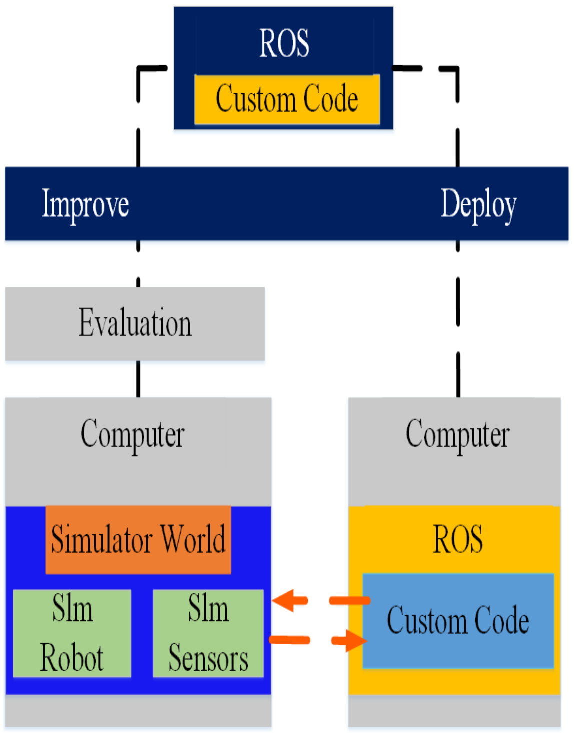

Figure 1. Communication Flowchart [59]

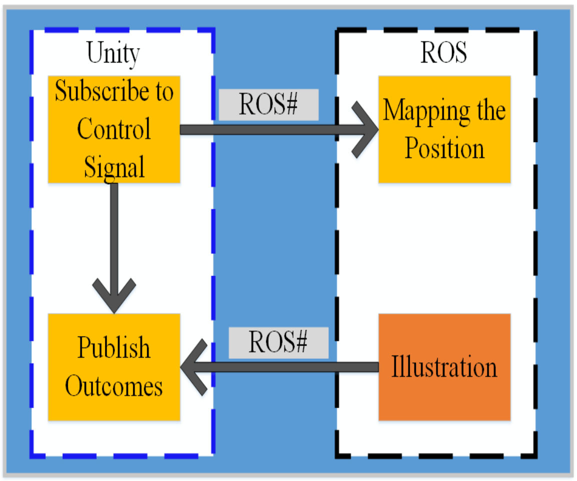

Figure 2. Communication Flow Chart

hardware and software. A notable advantage of TurtleBot2i is its seamless amalgamation with Razbot, a formidable open-source robot simulation environment. By leveraging Razbot, researchers and developers can meticulously scrutinize their algorithms and control strategies within a simulated environment prior to deploying them on a physical robot, resulting in notable reductions in time, and resources and enabling rapid prototyping and experimentation. in this section mentioned on platforms are described in detail.

3.1. Turtlebot2i

The TurtleBot2i embodies an advanced robotic platform, leveraging the Robot Operating System (ROS) and powered by the Intel Joule. This platform represents the groundwork for the development of cutting-edge robotic systems that promise to transcend the boundaries of contemporary technology. Recently, Interbotix Labs has announced the debut of the TurtleBot 2i Mobile manipulator robot, which is operated through an open-source Robotics Foundation (OSRF). By harnessing the potential of the ROS-based AI platform, the TurtleBot 2i [56] showcases a range of sophisticated features that are readily accessible, allowing for the swift prototyping of advanced robots. With its restructured framework, support for robotic arms, and integration with the Intel Joule 570x cipher module, the TurtleBot 2i has been crafted to simplify the development process of the next generation of robots. The architecture of the TurtleBot 2i is visible in figure 3, while the technical specifications of the TurtleBot are elaborated in table 1. The TurtleBot is an affordable personal robotics kit with open-source code that allows users to build a robot capable of navigating, perceiving 3D, and performing tasks with power. The TurtleBot 2i offers built-in support for robotic arms and a standardized chassis, making it a highly-capable mobile manipulator. The TurtleBot is used for Ambient Assisted Living, navigation research, mapping, and educational purposes,

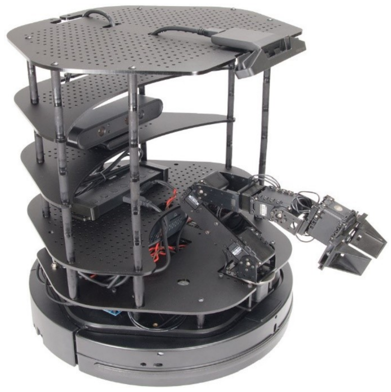

Figure 3. Turtlebot 2i structure

Table 1. The turtile bot robot specification

CPU	TurtleBot 2i Sensors
Intel Joule 570X Gumstix Nodana Carrier Board 4GB RAM 16GB eMMC Storage 802.11AC WiFi / Bluetooth 4.0 Ubuntu 16.04 / ROS Kinetic	SR300 RealSense 3D Camera ZR300 RealSense 3D Camera Accelerometer/Gyro/Compass Edge Detection & Bumper Sensors

as well as in multi-robot systems and mobile manipulation. Despite being budget-friendly, the platform has necessary certifications for household use, and it may constitute an integral part of a final service robot design. The TurtleBot's high functionality shows that even affordable solutions can be suitable for ranking analyses [57].

3.2. Razbot

The RAZBOT robotic platform is an exceptionally adaptable and configurable mechanism that has been specifically conceptualized for research and academic objectives. The said robotic platform has been developed on the bedrock of the Robot Operating System (ROS), which not only simplifies but also expedites the process of tailoring the RAZBOT to perform a wide spectrum of assignments ranging from rudimentary navigation to intricate manipulation and supervision [58]. Figure 4 shows the structure of razbot robot and specifications of the razbot are shown in table 2. The RAZBOT robotic platform is an adaptable and multifunctional tool that provides an array of benefits for scholars and educators. One of its key advantages is its modular design, which comprises interchangeable components that facilitate customization to suit specific

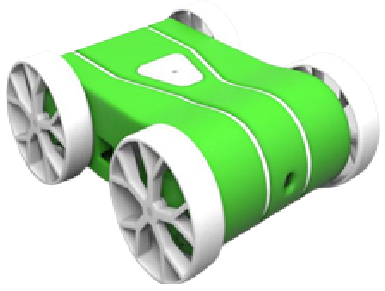

Figure 4. Razbot robot

Table 2. Razbot Specification

Features	Quantity
Raspberry Pi B, B+, 2	1
Raspberry Pi Camera	1
Turnigy 3S 2200mAh Lithium Polymer Pack	1
25mm 100rpm 12V DC Gearmotor	4
PCB and Electronics Components or OTS motor controller & voltage regulator / Bluetooth 4.0	1
SD Card Image with Raspbian Jessie, ROS, and razbot packages installed/ ROS Kinetic	1
3D printed components	12
Wires and connectors	1
Socket cap screw hardware	1

requirements.
Moreover, the platform is compatible with ROS, a middleware that furnishes tools and libraries for the efficient programming and control of robots. This compatibility enables users to exploit pre-existing code and tools, thereby fostering a collaborative and cooperative research and educational milieu.
Nonetheless, the RAZBOT platform has potential drawbacks. The complete kit's cost may be prohibitive for individuals on a limited budget, and the modular design can render the platform more intricate to operate and maintain, necessitating a profound understanding of programming and robotics.
Overall, the RAZBOT platform presents numerous advantages, but it may not be suitable for all users due to its intricacy and expense. Nevertheless, for those equipped with the requisite expertise and resources, the platform constitutes an extraordinary tool for research and education, capable of accomplishing a diverse range of tasks.

3.3. Gazebo, overview and advantage

Gezbo, an advanced and high-fidelity physics-based robot simulator, constitutes a viable and versatile solution that can be leveraged within the Robot Operating System (ROS) to devise, assess, and authenticate robotic algorithms and systems. By means of a virtual environment that closely emulates the physical attributes of tangible objects, Gezbo furnishes a simulated setting that affords users the capability to scrutinize numerous robot configurations, and evaluate them across different scenarios and under various conditions.
As an open-source platform, Gezbo seamlessly integrates with ROS, endowing a comprehensive suite of features that encompasses physics-based sensor models, motion planning, and visualization [59]. Its modular design enables facile assimilation with extant ROS packages, thereby empowering users to leverage pre-existing code and tools. Gezbo constitutes a highly valuable tool for robotics researchers and developers, delivering an efficacious and cost-efficient mechanism for verifying and validating their algorithms and systems prior to deploying them in the physical realm.
Gezbo is an innovative software application that facilitates the creation of virtual environments for simulating robotic systems. This tool bridges the gap between the Robot Operating System (ROS) and Unity, thereby enabling the development of more accurate and effective algorithms for simulating realistic robotic systems. Specifically, Gezbo allows users to create ROS nodes and topics that can interact with Unity in real-time, thereby simplifying the process of data transfer between the two platforms. The Unity plugin associated with Gezbo further enhances this functionality, enabling users to import ROS messages and sensor data into their Unity projects. This capability is particularly significant because it facilitates the development of more sophisticated and realistic algorithms for simulating robotic systems. Overall, Gezbo is a vital tool for researchers and practitioners seeking to enhance the accuracy and effectiveness of algorithms and systems by connecting ROS to Unity and enabling seamless data transfer. Its open-source nature, customization capabilities, and integration with other robotics tools make it a popular choice in the field of robotics.

4. Modelling and Simulation

This section is aimed to simulate a Unity scene using Turtlebot2i and Razbot Urdf to create a near-realistic environment and move the robots with different methods [7]. the modelling can be divided into 3 steps; which are described as follows:

(a) Step 1 is the general step used for both platform

(b) Step 2: Setting up turtlebot2i Scene and razsbot Scene.

(c) Step3: robot platform navigation.

Step 1: The General Step Used For Both Platform There are 6 basic Footsteps for the simulation which are shown in table 3.

Table 3. Simulation Basic Steps

Simulation General basic steps
Footstep 1: Enabling Hyper- V
Footstep 2: Installing Ubuntu on Hyper-V in Windows 10
Footstep 3: Installing ROS Melodic on Ubuntu
Footstep 4: Installing Unity 2019.4.32f on Windows 10
Footstep 5: Making ROS Environment
Footstep 6: Making Unity Environment

All of the above six steps are explained in detail below.

- **Footstep 1: Enabling Hyper- V**

Hyper-V constitutes Microsoft's hardware virtualization offering, which facilitates the creation and execution of virtual machines, compand uter simulations realized in software. Hyper-V confers to each virtual machine a distinct and self-contained setting, thereby empowering users to operate multiple virtual machines concurrently on identical hardware. In order to activate Hyper-V through PowerShell, a series of sequential actions must be pursued, as tabulated in table 4.

Table 4. Simulation Basic Steps

Steps for Hyper-V Enabling
A. Open a PowerShell console as an administrator
B. Type "Enable-Windows Optional Feature -Online -Feature Name Microsoft-Hyper-V -All" in the terminal.

- **Footstep 2: Installing Ubuntu 18.04 on Hyper-V in Windows 10**

Ubuntu 18.04 represents a prospective Long-Term Support (LTS) iteration, imbued with the capacity to receive updates and technical assistance from Canonical until the April of 2023. This version's advent, nearly seven years following Canonical's egress from the GNOME 2 sphere in favor of the Unity desktop, connotes the latter's somber disintegration, inciting Canonical's decision to concentrate on its server and IoT systems. To initiate the installation of Ubuntu in the Hyper-V environment, users are implored to meticulously adhere to the instructions delineated in table 5.

Figure 5 shows the whole footstep 2 by step.

Table 5. Installing the Ubuntu 18.04 version on Hyper-V in windows 10, step by step

Installing Ubuntu 18.04
A. Open Hyper-V from the Start menu.
B. "Click on new from the actions tab in the Hyper-V GUI and select Virtual machine".
C. "Select generation and click on next button".
D. Assign "memory size in MB in the startup memory box and click next".
E. "Select default network switch from the Connection drop-down box" for network connection.
F. "Create the virtual hard disk and assign the size and the machine's location."
G. The user needs to Choose an OS Iso image to install it on the virtual machine. To do so the user should download the Iso from the link releases.ubuntu.com/18.04/ and select the downloaded iso.
H. "Click on the finish button" to finish the installation.
I. Start the virtual machine and start the OS installation process. To do so the user needs to select the virtual machine and click on start.
J. Select any language from the list and select Install Ubuntu.
K. Select the keyboard layout.
L. In this step, User needs to specify the installation of the start-up software. To do that user needs to select normal installation.
M. Select Erase disk and install Ubuntu.
N. Confirm the installation space and click on the continue button.
O. Select the location and click next.
P. In this step, user needs to make the user account and the login password.
Q. When the installation is done restart the system.

- **Footstep 3: Installing ROS Melodic on Ubuntu Virtual machine**

To expedite the simulation process, it is imperative to leverage a sophisticated software suite referred to as Robot Operating System (ROS) that functions as an interface with autonomous machines. Despite its nomenclature implying an operating system, ROS is, in fact, a complex application that enables seamless interaction with robotic systems. As a Linux-centric software, ROS mandates its installation onto a Linux-based operating system, necessitating the user to comply with meticulous directives outlined in table 6 to ensure proper installation.

- **Footstep 4: Installing Unity 2019.4.32f on Windows 10**

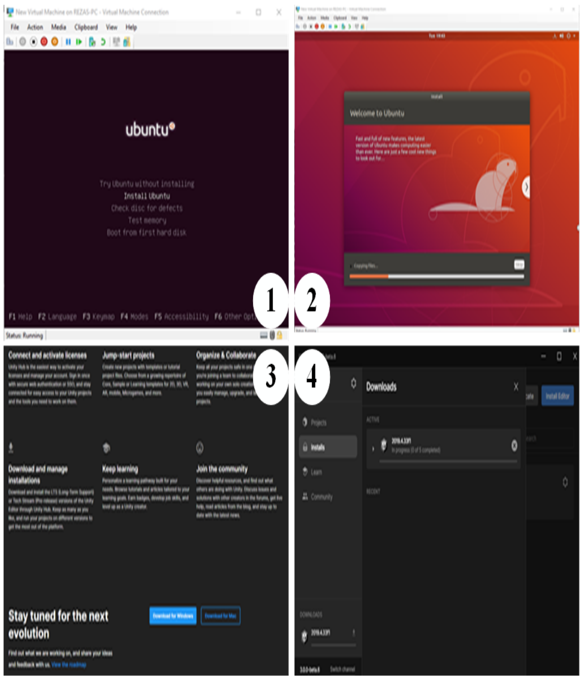
ubuntu
Try Ubuntu without installing
Install Ubuntu
Check disc for defects
Test memory
Boot from first hard disk
Welcome to Ubuntu
1
2
3
4
Downloads
Stay tuned for the next evolution

Figure 5. Installation of Ubuntu, Step by Step. 1: Shows the installation of Ubuntu. 2: Shows the installing in process for Ubuntu. 3: shows downloading of Unity Hub. 4: Shows the installation of editor.

Table 6. Installing ROS Melodic on Ubuntu Virtual Machine

Installing ROS Melodic
A. Open a new terminal by pressing $Ctrl + Alt + t$ and type the following command " sudo sh -c 'echo "deb packages.ros.org/ros/ubuntu $(lsb_release -sc) main">/etc/apt/sources.list.d/ros-latest.list'
B. Type the command in the terminal " sudo apt install curl"
C "curl -s raw.githubusercontent.com/ros/rosdistro /master/ros.asc | sudo apt-key add.
D. Type the command in the terminal. "sudo apt update"
E. Type the command "sudo apt install ros-melodic-desktop-full" in a terminal to install ROS melodic.

Unity is a potent and versatile game engine that enables users to construct dynamic and interactive environments for game development, facilitating the creation of immersive and engaging simulations. The installation process for this software necessitates a rigorous adherence to a prescribed set of steps, as enumerated in table 7.

- **Footstep 5: Making ROS Environment**

In order to leverage the full capabilities of the Robot Operating System (ROS), it is imperative to establish

Table 7. Installing Unity 2019.4.32f on Windows 10

Installing Unity 2019.4.32f
A. Unity hub is needed to install the unity editor. To do so the user needs to go to the unity website and download Unity Hub from the link unity.com/unity-hub
B. Select the Install tab and choose the editor version.
C Select the module they need.
D. Click on the install button, The installation of the editor will start.

an efficient and streamlined ROS configuration, as explicated in prior research. The establishment of a robust ROS environment is a prerequisite for optimal system functionality and successful execution of requisite tasks. In order to craft an effective ROS environment, users are advised to meticulously adhere to the procedural steps delineated in table 8. Such a methodical approach towards ROS setup will guarantee seamless and efficient functioning of the system in accordance with user requirements.

Table 8. Making of the ROS Environment

Making ROS Environment
A. Press “Ctrl+Alt+t” to open a new terminal and type “source/opt/ros/melodic/setup.bash”
B. cd ~/catkin_ws/src
C. Open a browser and go to the link github.com/siemens/ros-sharp to download the ros-sharp-master file.
D. Extract the zip file by right-clicking the mouse and selecting Extract here.
E. Navigate inside to `ros-sharp-master > ROS` and select the "file_server" and "unity_simulation_scene" folders.
F. Navigate inside to `catkin_ws > src` and paste the "file_server" and "unity_simulation_scene" folders.

Figure 6 shows the whole footstep 5 step by step.

- **Setting up ROS Environment**

For experimenting ROS Melodic is used on Ubuntu 18.04 LTS. to set up the ROS a user should follow the steps below in table 12:

- **Footstep 6: Making Unity Environment**

To commence the experimental phase, the user must first establish the UNITY environment, specifically utilizing version **2019.04 L. T. S**. The process to achieve this objective involves a series of sequential steps that are elaborated in table 9, which must be meticulously followed to ensure optimal results.

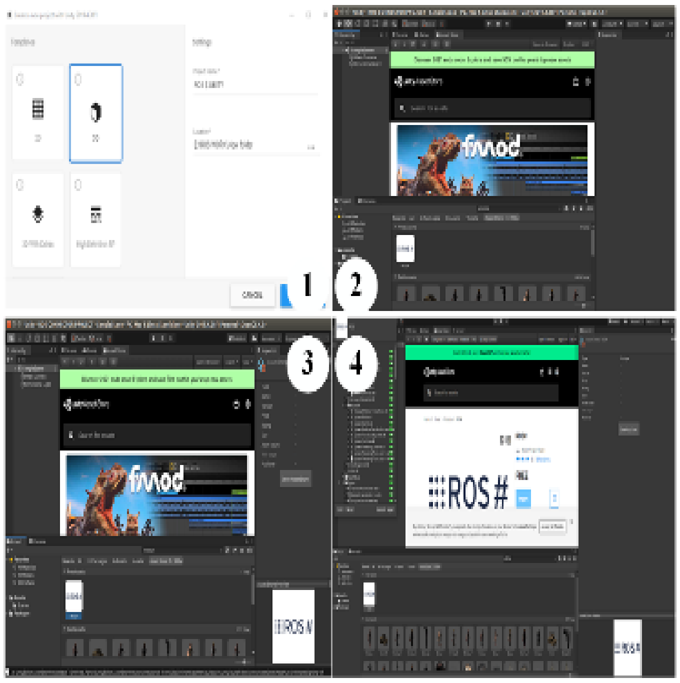

Figure 6. The ROS Environment Setup. 1: Shows the creation of Unity Scene. 2: Shows the Asset store for ROS#. 3: Shows the view of asset store. 4: Shows importing the ROS# asset.

Table 9. Setting up the ROS Environment step by step

ROS Environment Set up
A. Open a browser and go to the link github.com/siemens/ros-sharpto download the ros-sharp-master file. (Figure 9) in Ubuntu operating system.
B. Extract the zip file by right-clicking the mouse and selecting Extract here. (Figure 10)
C. Navigate inside to "ros-sharp-master > ROS" and select the "file_server" and "unity_simulation_scene" folder. (Figure 10)
D. Navigate to the "catkin_ws" workspace and paste the 2 folders inside "catkin_ws > src".
E. Open a terminal by pressing Ctrl+Alt+T
F. Type the command "cd catkin_ws"
G. Build the workspace by typing “catkin_make”

Step 2: Setting up turtlebot2i /Razbot Scene

To initiate the movement of the robotic system, the user is advised to adhere to a prescribed sequence of actions. Specifically, for the present inquiry, the turtlebot2i URDF model has been implemented. The aforementioned steps are delineated in a meticulous and orderly fashion within the confines of table 10.

Table 10. Making Unity Environment

Unity Environment Steps
A. Open the UNITY and create a new 3D unity scene.
B. Type ROS# in the search box in the Asset store tab.
C. Import the ROS# asset from the unity asset store. Or directly download the asset from the GitHub link. assetstore.unity.com/packages/tools/physicss/ros-107085

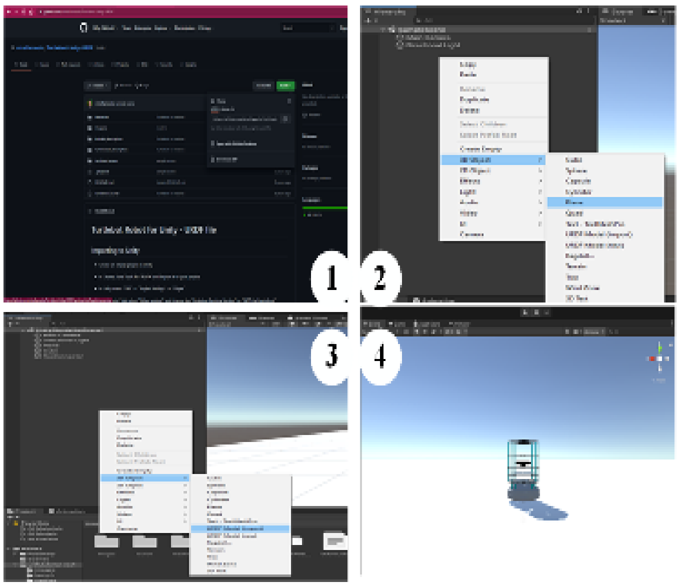

Figure 7. Turtlebot Setup step by step. Fig 7(1); Download page of Turtlebot-Unity-URDF, Fig 7(2); Creating a plane, Fig 7(3); Importing Turtlebot URDF, Fig 7(4); Turtlebot2i Outlook in simulation

Table 11. Setting up ROS Environment for Unity Import

ROS Environment Set up for Unity
A. Move the razbot_discription folder to the src folder of the catkin_ws.
B. Type the command "cd catkin_ws" in a new terminal.
C. Type the command "catkin_make" to build the workspace.
D. Type "cd src/razbot_discription/urdf" in the same terminal.
E. Run the command "rosrun xacro xacro razbot.urdf.xacro > razbot.urdf" in the same terminal.
F. Copy or move razbot.urdf into Assets/Urdf folder in Unity project.

Table 12. Setting up the turtlebot 2i/RAzbot Scene step by step

Turtlebot Setup Step by Step
A. Download the Turtlebot-Unity-URDF file from the link: github.com/mirellameelo/Turtlebot-Unity-URDF and import the Turtlebot2i_discription file into the asset folder of UNITY Scene.
B. Create a plane from the Hierarchy tab and put the x y z scale to 100 in the inspector tab. (Figure 7(1))
C. Right-click in the Hierarchy tab and import URDF from the create 3D object menu and import turtle bot URDF.
D. Right-click on the Hierarchy tab > create an empty game object > rename it as ROSConnector. (Figure 7(2))
E. In the inspector tab add a component called ROSConnector Script.
F. In the inspector tab select ROSConnector Script > change the protocol as web socket.NET. (Figure 7(3))
G. In the inspector tab add a component named Joint State Patcher script.
H. From Hierarchy tab select turtlebot URDF > drag and drop > URDF Robot Box of Joint State Patcher script in the inspector tab.
I. Click on the Enable button of Publish Joint State to create the Joint State Publisher script.
J. in Joint State Publisher Script, Rename the topic as "/joint_states".
K. Add another Script named Joy Subscriber and rename the topic as "/joy".
L. Add another Script name Pose Stumped Publisher and rename the topic as "/odom".
M. From the Hierarchy tab, navigate inside the turtlebot URDF and select the base_link and drag and drop it in the Publish Transform box of Pose Stumped Publisher Script.
N. Add a Script name Image Publisher and rename the topic as "/unity_image/compressed".
O. In the Hierarchy tab navigate inside the turtlebot URDF and select camera_rgb_frame and in the inspector, the tab add a camera component.
P. Select the ROSConnector from the Hierarchy tab and in the Inspector tab, drag and drop the camera_rgb_frame in the image camera box of the Image Publisher script as shown in the (figure 7(4))
Q. Select the turtlebot from the Hierarchy tab and in the inspector tab in URDF Robot Script Enable the Gravity and Convex button.
R. Select both of the wheels from the turtlebot URDF in the Hierarchy tab in the Inspector tab Joy Axis add Joint Motor Writer and set the max velocity to 2000 by the user.

S. Under the Hinge Joint tab check the use motor box and set force to 0.01 and Connected Mass Scale to 240.
T. Selected both of the caster_link from the Hierarchy tab and in the Inspector tab under Fixed Joint change the Connecter Mass to 240.
U. Select the ROS Connector from the Hierarchy tab and in the Joy Subscriber set Joy Axis Writer size to 2 and drag and drop the left and write wheel in the Element 0 and 1 box.

Footstep1: Setting up ROS Environment
The **turtlebot2i** robot configuration employs the xacro framework as the primary method for defining its structural properties. However, due to the incompatibility of the xacro format with Unity, the user is compelled to initiate the generation of a URDF file from the xacro format table 11 to enable the rendering of the robot model in Unity. The figure 8 shows the whole ROS environment setup for Unity step by step from downloading the ros-sharp master to navigating the source folder.

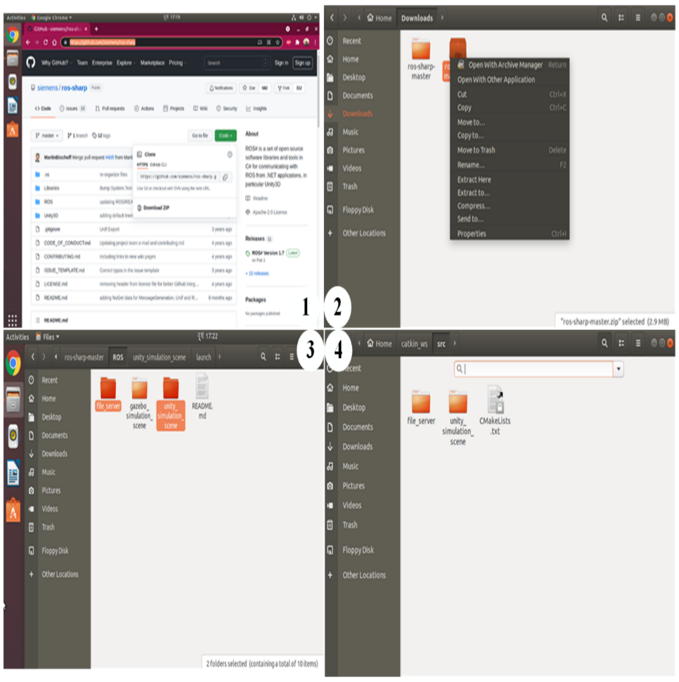

Figure 8. ROS Environment Setup Step by Step. 1: Shows Downloading the ros-sharp-master, 2: Shows Extracting the Zip, 3: Shows Copying files from the Zip, 4: Shows Navigating to src folder

Footstep2: Import to Unity
The subsequent segment is exclusively dedicated to the transfer of the project onto the Unity platform. This endeavor necessitates a scrupulous method towards the fastidious achievement of two cardinal phases, specifically the customization of the Robot Operating System (ROS) and the configuration of the Unity . By adhering to these imperative protocols, the seamless amalgamation of the project into Unity can be warranted, thus hastening the efficacious accomplishment of the envisaged objectives. **Step 3: Robot Navigation**
The pursuit of robot navigation was undertaken with assiduity, whereby the utilization of multifarious input modalities, including but not limited to the computer mouse, keyboard, and command line, were executed with meticulousness and precision.

(a) **Turtlebot2i Control by Mouse**

In order to manipulate the movements of the turtlebot2i through the utilization of a mouse, it is incumbent upon the operator to adhere scrupulously to the sequence of actions delineated in table 13.

Table 13. Turtlebot2i Controlling by Mouse

ROS Environment setting
A. Navigate to the "catkin_ws" workspace and paste the file_server and unity_silulation_scene folder inside catkin_ws > src. The flowchart for these process are shown in figure 9
UNITY Environment setting
B. Open the Scene and click on the play button.
C. If the mouse is moved in ROS, the turtlebot2i will start to move in the Unity scene.

Figure 9: The code flowchart for using the mouse to joy.

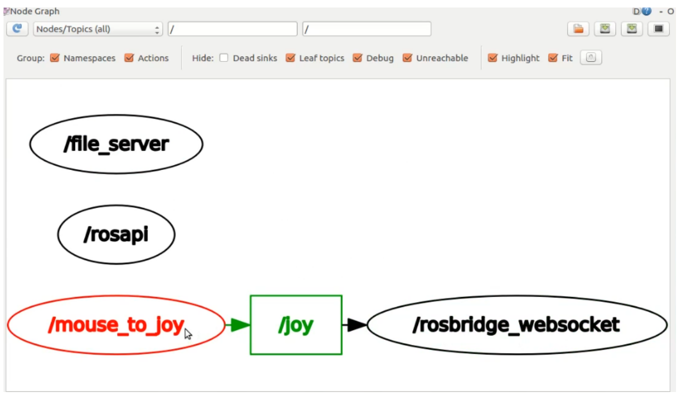
Node Graph
Nodes/Topics (all)
Group: ✓ Namespaces ✓ Actions
Hide: ☐ Dead sinks ✓ Leaf topics ✓ Debug ✓ Unreachable
✓ Highlight ✓ Fit

Figure 9. The code flowchart for using the mouse to joy

(b) **RAZBOT Control by Keyboard:**

Table 14 provides a thorough and meticulous exposition of the methodology involved in manipulating Razbot through the use of keyboard-based control. Each stage of this procedure is presented in a systematic and exacting fashion, affording readers a comprehensive and perspicuous synopsis of the complex intricacies inherent to the keyboard-operated functionality of the Razbot.

(c) **RAZBOT Control by command Terminal:** The governance of the platform via the keyboard teleportation plugin, obtainable in the Unity plugin section, may be accomplished by executing a series of meticulously planned actions. In order to navigate the robot through the use of the keyboard, the prescribed steps are explicated in detail in table 15.

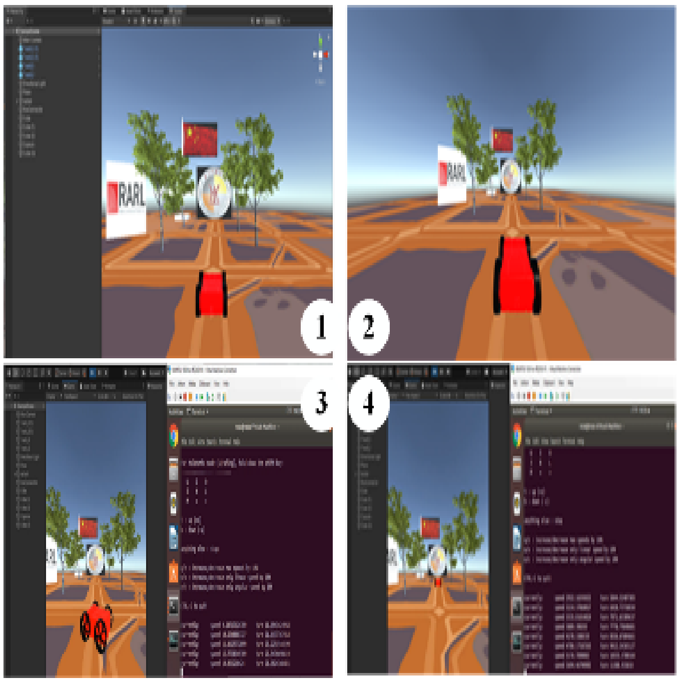

Figure 10. RAZbot Simulations Results

The results obtained from the simulation process are presented graphically in figure 10. This figure provides a clear and concise representation of the outcome, allowing for a comprehensive understanding of the simulated phenomena. The outlook results depicted in fig 10 offer an in-depth and cogent elucidation of the underlying trends and patterns that emerged from the simulation exercise. The figure demonstrates a high degree of accuracy and precision in portraying the simulation outcome, thereby facilitating a profound comprehension of the intricacies of the simulated phenomena.

Table 14. RAZBOT controlling by Keyboard

RAZBOT controlling by keyboard step by step
A. Open the Razbot scene in Unity.
B. Open the unity scene and navigate to plugins inside the razbot in the hierarchy tab.
C. Add Urdf plugin script.
D. Add the following codes inside the plugin text box `<gazebo>` `<plugin name="skid_steer_drive_controller" filename="libgazebo_ros_skid_steer_drive.so">` `<updateRate>100.0</updateRate>` `<robotNamespace>/</robotNamespace>` `<leftFrontJoint>front_left_wheel_joint</leftFrontJoint>` `<rightFrontJoint>front_right_wheel_joint</rightFrontJoint>` `<leftRearJoint>back_left_wheel_joint</leftRearJoint>` `<rightRearJoint>back_right_wheel_joint</rightRearJoint>` `<wheelSeparation>0.4</wheelSeparation>` `<wheelDiameter>0.215</wheelDiameter>` `<robotBaseFrame>base_link</robotBaseFrame>` `<torque>20</torque>` `<topicName>cmd_vel</topicName>` `<broadcastTF>false</broadcastTF>` `</plugin>` `</gazebo>`
E. In ubuntu Open a new terminal by pressing Ctrl+Alt+T.
F. Type the command `sudo apt-get install ros-melodic-teleop-twist-keyboard`
G. Open a new terminal by pressing Ctrl+Alt+T.
H. Install ROS Bridge server by typing the command `sudo apt-get install ros-melodic-rosbridge-server`
I. "After installing the ROS bridge type the following command" `roslaunch rosbridge_server rosbridge_websocket.launch`
J. Open a new terminal by pressing Ctrl+Alt+T.
K. Run the python script of the keyboard teleportation. `rosrun teleop_twist_keyboard teleop_twist_keyboard.py`
L. The terminal will show this information.

Table 15. Razbot Control by Command Terminal

Razbot Control by Command Terminal
A. Open a new terminal by pressing Ctrl+Alt+T.
B. Type `roslaunch rosbridge_server rosbridge_websocket.launch`
C. Type the command `rostopic pub -r 1 /cmd_vel geometry_msgs/Twist "{linear:{x:200.0,y:0.0,z:0.0}, angular:{x:0.0,y:0.0,z:0.0}}"` in a new terminal.
D. Open another terminal by pressing Ctrl+Alt+T.
E. To move the robot to a specific location, user should type the command `rostopic pub -r 1 /pose geometry_msgs/PoseStamped'{header:{frame_id:"Unity"}, pose:{ position:{x:5.2,y:2.7,z:0␣}, orientation:{x:1,y:5,z:2,w:1} }, }'` in a new terminal

5. Results and Discussion

This research paper aimed to demonstrate the feasibility of conducting robot simulation using the Robot Operating System (ROS) and Unity game engine. Various Automated Guided Vehicle (AGV) platforms were tested, and two robot platforms, RAZBOT and TURTLEBOT2i URDF, were used to explore the performance of the simulation. The research showed that ROS is a comprehensive operating system for robotics, and no other system can match its capabilities. During the research, ROS was installed on Ubuntu 18.04 L.T.S. Linux, and Unity hub was installed, and Unity Editor version 2019.4.32f was obtained from the hub. Specific ROS packages, such as ROS#, were downloaded to connect ROS and Unity. The research demonstrated the potential for ROS and Unity to be leveraged in robot simulation. While ROS is better suited for large-scale industrial robotics projects, Unity is a potent tool for gaming, design, and industry. The research also highlights the importance of proper sourcing of the workspace to avoid issues during the simulation process. In summary, the use of Unity and ROS for AGV simulation and modeling in Unity provides developers with a new pathway, and these tools have potential applications in various fields such as medicine, construction, and factories.The present study endeavours to explore the performance of alternative autonomous guided vehicle (AGV) platforms and conduct a comprehensive evaluation of the discernible disparities between their actual and simulated executions. To this end, in future work a meticulous analysis will be carried out to assess the efficacy of diverse AGV models vis-à-vis one another, in order to gain a nuanced understanding of their respective merits and drawbacks. In doing so, the study aims to provide an empirical basis for drawing reliable inferences about the relative efficacy of AGV platforms, both in real-world scenarios and in simulation environments.

References

[1] Hu, Y., & Meng, W. (2016). ROSUnitySim: Development and experimentation of a real-time simulator for multi-unmanned aerial vehicle local planning. Simulation, 92(10), 931-944.

[2] Sita, E., Horváth, C. M., Thomessen, T., Korondi, P., & Pipe, A. G. (2017, December). Ros-unity3d based system for monitoring of an industrial robotic process. In 2017 IEEE/SICE International Symposium on System Integration (SII) (pp. 1047-1052). IEEE.

[3] Krupke, D., Starke, S., Einig, L., Zhang, J., & Steinicke, F. (2018). Prototyping of immersive HRI scenarios. In Human-Centric Robotics: Proceedings of CLAWAR 2017: 20th International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines (pp. 537-544).

[4] Moshayedi, A. J., Roy, A. S., Taravet, A., Liao, L., Wu, J., & Gheisari, M. (2023). A Secure Traffic Police Remote Sensing Approach via a Deep Learning-Based Low-Altitude Vehicle Speed Detector through UAVs in Smart Cites: Algorithm, Implementation and Evaluation. Future Transportation, 3(1), 189-209.

[5] Das, S. K., & Pasan, M. K. (2016). Design and methodology of automated guided vehicle-a review. IOSR Journal of Mechanical and Civil Engineering (IOSR-JMCE), 29-35.

[6] Reveliotis, S. A. (2000). Conflict resolution in AGV systems. Iie Transactions, 32(7), 647-659.

[7] Feledy, C., & Schiller Luttenberger, M. (2017). A State of the Art Map of the AGVS Technology and a Guideline for How and Where to Use It.

[8] Pedan, M., Gregor, M., & Plinta, D. (2017). Implementation of automated guided vehicle system in healthcare facility. Procedia engineering, 192, 665-670.

[9] Moshayedi, A. J., Zanjani, S. M., Xu, D., Chen, X., Wang, G., & Yang, S. (2022, December). Fusion BASED AGV Robot Navigation Solution Comparative Analysis and Vrep Simulation. In 2022 8th Iranian Conference on Signal Processing and Intelligent Systems (ICSPIS) (pp. 1-11). IEEE.

[10] Kato, H., Hirano, D., Mitani, S., Saito, T., & Kawaguchi, S. (2021, March). ROS and cFS System (RACS): Easing Space Robotic Development. In 2021 IEEE Aerospace Conference (50100) (pp. 1-8). IEEE.

[11] Chi, H., Zhan, K., & Shi, B. (2012). Automatic guidance of underground mining vehicles using laser sensors. Tunnelling and Underground Space Technology, 27(1), 142-148.

[12] Miljković, Z., Vuković, N., Mitić, M., & Babić, B. (2013). New hybrid vision-based control approach for automated guided vehicles. The International Journal of Advanced Manufacturing Technology, 66, 231-249.

[13] Moshayedi, A. J., Abbasi, A., Liao, L., & Li, S. (2019, June). Path planning and trajectroy tracking of a mobile robot using bio-inspired optimization algorithms and PID control. In 2019 IEEE International Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA) (pp. 1-6). IEEE.

[14] Shengfang, L., & Xingzhe, H. (2006, October). Research on the AGV based robot system used in substation inspection. In 2006 International Conference on Power System Technology (pp. 1-4). IEEE.

[15] Xu, G., Khan, A. S., Moshayedi, A. J., Zhang, X., & Shuxin, Y. (2022). The Object Detection, Perspective and Obstacles In Robotic: A Review. EAI Endorsed Transactions on AI and Robotics, 1(1).

[16] Whitney, D., Rosen, E., Ullman, D., Phillips, E., & Tellex, S. (2018, October). Ros reality: A virtual reality framework using consumer-grade hardware for ros-enabled robots. In 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (pp. 1-9). IEEE.

[17] Ferreira, T., & Gorlach, I. A. (2016). Development of an automated guided vehicle controller using a model-based systems engineering approach. South African journal of industrial engineering, 27(2), 206-217.

[18] Moshayedi, A. J., Kolahdooz, A., & Liao, L. (2022). Unity in Embedded System Design and Robotics: A Step-by-step Guide. CRC Press.

[19] B. Gerkey et al., “The robot operating system (ROS): A platform for robot software development,” in Proc. of the ICRA workshop on Open source software, vol. 3, 2004, pp. 5–53.

[20] R. W. Goodrich et al., “ROS: An open-source Robot Operating System,” in Field and Service Robotics, 2007, pp. 15-20.

[21] S. R. Chitta et al., “Simulating robots with Gazebo and ROS,” in Robotics Automation Magazine, IEEE, vol. 19, no. 3, 2012, pp. 56-66.

[22] Tomic, T., Schmid, K., Lutz, P., Domel, A., Kassecker, M., Mair, E., ... & Burschka, D. (2012). Toward a fully autonomous UAV: Research platform for indoor and outdoor urban search and rescue. IEEE robotics & automation magazine, 19(3), 46-56.

[23] Qiu, W., & Yuille, A. (2016). Unrealcv: Connecting computer vision to unreal engine. In Computer Vision–ECCV 2016 Workshops: Amsterdam, The Netherlands, October 8-10 and 15-16, 2016, Proceedings, Part III 14 (pp. 909-916). Springer International Publishing.

[24] Moshayedi, A. J., Xu, G., Liao, L., & Kolahdooz, A. (2021). Gentle Survey on MIR Industrial Service Robots: Review & Design. J. Mod. Process. Manuf. Prod, 10(1), 31-50.

[25] Lee, J., Hyun, C. H., & Park, M. (2013). A vision-based automated guided vehicle system with marker recognition for indoor use. Sensors, 13(8), 10052-10073.

[26] Lavrenov, R., Zakiev, A., & Magid, E. (2017, May). Automatic mapping and filtering tool: From a sensor-based occupancy grid to a 3D Gazebo octomap. In 2017 International Conference on Mechanical, System and Control Engineering (ICMSC) (pp. 190-195). IEEE.

[27] Krupke, D., Einig, L., Langbehn, E., Zhang, J., & Steinicke, F. (2016, November). Immersive remote grasping: realtime gripper control by a heterogenous robot control system. In Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology (pp. 337-338).

[28] Moshayedi, A. J., Li, J., Sina, N., Chen, X., Liao, L., Gheisari, M., & Xie, X. (2022). Simulation and validation of optimized pid controller in agv (automated guided vehicles) model using pso and bas algorithms. Computational Intelligence and Neuroscience, 2022.

[29] Christopher, C., Graylin, J., Sarah, O., & Odest, C. J. (2016). Rosbridge: Ros for non-ros users. Springer Tracts in Advanced Robotics, 493-504.

[30] Craighead, J., Burke, J., & Murphy, R. (2008, September). Using the unity game engine to develop sarge: a case study. In Proceedings of the 2008 Simulation Workshop at the International Conference on Intelligent Robots and Systems (IROS 2008) (Vol. 4552).

[31] Unity Technologies. (2021). Unity-Manual: Open-source repositories [Web page]. Unity. Retrieved December 17, 2021, from https://docs.unity3d.com/Manual/OpenSourceRepositories

[32] Bartneck, C., Soucy, M., Fleuret, K., & Sandoval, E. B. (2015, August). The robot engine—Making the unity 3D game engine work for HRI. In 2015 24th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN) (pp. 431-437). IEEE.

[33] Mizuchi, Y., & Inamura, T. (2017, December). Cloud-based multimodal human-robot interaction simulator utilizing ros and unity frameworks. In 2017 IEEE/SICE International Symposium on System Integration (SII) (pp. 948-955). IEEE.

[34] Codd-Downey, R., Forooshani, P. M., Speers, A., Wang, H., & Jenkin, M. (2014, July). From ROS to unity: Leveraging robot and virtual environment middleware for immersive teleoperation. In 2014 IEEE International Conference on Information and Automation (ICIA) (pp. 932-936). IEEE.

[35] December17, 2021,from https://blog.unity.com/unity-2019-1

[36] Moshayedi, A. J., Roy, A. S., Sambo, S. K., Zhong, Y., & Liao, L. (2022). Review on: the service robot mathematical model. EAI Endorsed Transactions on AI and Robotics, 1(1).

[37] Udemy Blog. (2021, February 12). What is the Unity Game Engine? [Blog post]. Udemy Blog. Retrieved December 10, 2021, from https://blog.udemy.com/unity-game-engine/

[38] Christopher, C., Graylin, J., Sarah, O., & Odest, C. J. (2016). Rosbridge: Ros for non-ros users. Springer Tracts in Advanced Robotics, 493-504.

[39] Roldán, J. J., Peña-Tapia, E., Garzón-Ramos, D., de León, J., Garzón, M., del Cerro, J., & Barrientos, A. (2019). Multi-robot systems, virtual reality and ROS: developing a new generation of operator interfaces. Robot Operating System (ROS) The Complete Reference (Volume 3), 29-64.

[40] Lee, U. G., Choi, K. J., & Park, S. Y. (2021, August). The Design and Implementation of Autonomous Driving Pallet Robot System using ROS. In 2021 Twelfth International Conference on Ubiquitous and Future Networks (ICUFN) (pp. 372-374). IEEE.

[41] Gerkey, B., Vaughan, R. T., & Howard, A. (2003, June). The player/stage project: Tools for multi-robot and distributed sensor systems. In Proceedings of the 11th international conference on advanced robotics (Vol. 1, pp. 317-323).

[42] Moshayedi, A. J., & Gharpure, D. C. (2017, May). Evaluation of bio inspired Mokhtar: Odor localization system. In 2017 18th international carpathian control conference (ICCC) (pp. 527-532). IEEE.

[43] Ma, X., Fang, F., Qian, K., & Liang, C. (2018, May). Networked robot systems for indoor service enhanced via ROS middleware. In 2018 13th IEEE Conference on Industrial Electronics and Applications (ICIEA) (pp. 852-857). IEEE.

[44] Anggraeni, P., Mrabet, M., Defoort, M., & Djemai, M. (2018, October). Development of a wireless communication platform for multiple-mobile robots using ROS. In 2018 6th International Conference on Control Engineering & Information Technology (CEIT) (pp. 1-6). IEEE.

[45] Kerr, J., & Nickels, K. (2012, March). Robot operating systems: Bridging the gap between human and robot. In Proceedings of the 2012 44th Southeastern Symposium on System Theory (SSST) (pp. 99-104). IEEE.

[46] Huang, A. S., Olson, E., & Moore, D. (2009). Lightweight communications and marshalling for low latency interprocess communication. Computer Science and Artificial Intelligence Laboratory Technical Report, Massachusetts Institute of Technology, MA.

[47] Alberri, M., Hegazy, S., Badra, M., Nasr, M., Shehata, O. M., & Morgan, E. I. (2018, September). Generic ros-based architecture for heterogeneous multi-autonomous systems development. In 2018 IEEE International Conference on Vehicular Electronics and Safety (ICVES) (pp. 1-6). IEEE.

[48] Dosoftei, C. C., Popovici, A. T., Sacaleanu, P. R., & Budaciu, C. (2021, October). Real-Time Motion Control of an Electric Driven OMR using a ROS to Matlab Bridged Approach. In 2021 25th International Conference on System Theory, Control and Computing (ICSTCC) (pp. 160-165). IEEE.

[49] Guzman, R., Navarro, R., Beneto, M., & Carbonell, D. (2016). Robotnik—Professional service robotics applications with ROS. Robot Operating System (ROS) The Complete Reference (Volume 1), 253-288.

[50] Wang, T., Zhao, Y., Zhu, L., Liu, G., Ma, Z., & Zheng, J. (2020, November). Design of robot system for radioactive source detection based on ROS. In 2020 Chinese Automation Congress (CAC) (pp. 1397-1400). IEEE.

[51] Nurrohmah, E. A., Wibowo, I. K., Bachtiar, M. M., & Lathief, M. M. (2021, September). Improvement of the Processing Speed of The Robot's Vision System Using Robot Operating System. In 2021 International Electronics Symposium (IES) (pp. 482-487). IEEE.

[52] Hazem, Z. B., Ince, R., & Dilibal, S. (2022, September). Joint Control Implementation of 4-DOF Robotic Arm Using Robot Operating System. In 2022 International Conference on Theoretical and Applied Computer Science and Engineering (ICTASCE) (pp. 72-77). IEEE.

[53] Moshayedi, A. J., & Gharpure, D. C. (2012, May). Development of position monitoring system for studying performance of wind tracking algorithms. In ROBOTIK 2012; 7th German Conference on Robotics (pp. 1-4). VDE.

[54] Aagela, H., Al-Nesf, M., & Holmes, V. (2017, September). An Asus_xtion_probased indoor MAPPING using a Raspberry Pi with Turtlebot robot Turtlebot robot. In 2017 23rd International Conference on Automation and Computing (ICAC) (pp. 1-5). IEEE.

[55] Stan, A. C. (2022, June). A decentralised control method for unknown environment exploration using Turtlebot 3 multi-robot system. In 2022 14th International Conference on Electronics, Computers and Artificial Intelligence (ECAI) (pp. 1-6). IEEE.

[56] Li, K., & Tu, H. (2021, October). Design and implementation of autonomous mobility algorithm for home service robot based on turtlebot. In 2021 IEEE 5th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC) (Vol. 5, pp. 1095-1099). IEEE.

[57] Alajami, A. A., Pous, R., & Moreno, G. (2022, September). Simulation of RFID Systems in ROS-Gazebo. In 2022 IEEE 12th International Conference on RFID Technology and Applications (RFID-TA) (pp. 113-116). IEEE.

[58] Kim, Y., Lee, S. Y., & Lim, S. (2020, September). Implementation of PLC controller connected Gazebo-ROS to support IEC 61131-3. In 2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA) (Vol. 1, pp. 1195-1198). IEEE.

[59] Moshayedi, A. J., Jinsong, L., & Liao, L. (2019). AGV (automated guided vehicle) robot: Mission and obstacles in design and performance. Journal of Simulation and Analysis of Novel Technologies in Mechanical Engineering, 12(4), 5-18.