Simulated Forest Environment and Robot Control Framework for Integration with Cover Detection Algorithms

Avi Spector
Department of Mechanical Engineering
University of Maryland
College Park, MD
ajspec17@umd.edu

Wanying Zhu
Department of Computer Science
University of Georgia
Athens, GA
wanying.zhu@uga.edu

Jumman Hossain, Nirmalya Roy
Department of Information Systems
University of Maryland, Baltimore County
Baltimore, MD
jumman.hossain@umbc.edu, nroy@umbc.edu

***Abstract*—Simulated environments can be a quicker and more flexible alternative to training and testing machine learning models in the real world. Models also need to be able to efficiently communicate with the environment. In military-relevant environments, a trained model can play a valuable role in finding cover for an autonomous robot to prevent getting detected or attacked by adversaries. In this regard, we present a forest simulation and robot control framework that is ready for integration with machine learning or object recognition algorithms. Our framework includes an environment relevant to military situations and is capable of providing information about the environment to a machine learning model. A forest environment was designed with wooded areas, open paths, water, and bridges. A Clearpath Husky robot is simulated in the environment using Army Research Laboratory's (ARL) Unity and ROS simulation framework. The Husky robot is equipped with a camera and lidar sensor. Data from these sensors can be read through ROS topics and RViz configuration windows. The robot can be moved using ROS velocity command topics. These communication methods can be employed by a machine learning algorithm for use in detecting trees to attain maximum cover. Our designed environment improves upon the default ARL framework environments by offering a more diverse terrain and more opportunities for cover. This makes the environment more relevant to a cover-seeking machine learning model. Code, videos, and integration process available at : https://github.com/avispector7/Forest-Simulation**

***Index Terms*—ROS, rospy, Unity, semantic segmentation, object detection, Husky, ARL framework**

I. Introduction

It is not always practical to train and test machine learning models in real-world environments. This can especially be true when machine learning is used for robotics. Simulated environments can be a quicker and more flexible alternative, provided they are realistic and relevant to the machine learning model's purpose. This means they need to accurately represent the real-world environment they are attempting to replicate so that the transition from the simulated environment to the real world can be as seamless as possible. They also need to provide a diverse dataset for any potential conditions the model may encounter or tasks the model needs to accomplish.

Robot controllers, sensors, and other robot components are not usually integrated within a machine learning model, as this could make the model far too large and complex. Therefore, it is important to find the best methods for a machine learning model to communicate with an autonomous robot in an environment. This includes to receive information about the environment through sensors and to command and move the robot. These efficient methods are necessary for communication to be as seamless as possible.

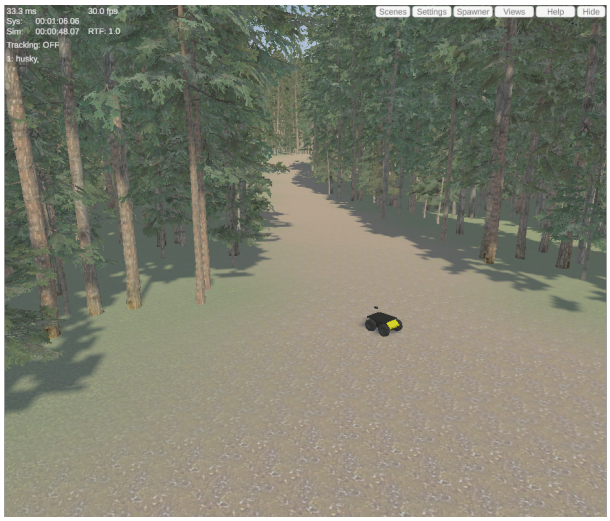

Fig. 1. Husky robot simulated in our environment using the ARL framework.

In military-relevant environments, autonomous ground vehicles often need to find cover to prevent getting detected or attacked by adversaries. A trained machine learning model can help find this cover after receiving information about the environment. One option is to detect trees as sources of cover.

We present a framework that attempts to address all of these problems. The key contributions of our framework include:

1) We designed a realistic forest environment that includes diverse terrain and extensive areas of cover. The diverse terrain includes grass, an open hiking path, water, and many hills. The cover is provided by trees. The diverse terrain and areas of cover are included to make the

environment relevant to a machine learning model that detects sources of cover from adversaries in a military situation.

2) We used the Army Research Laboratory's (ARL) framework to simulate a robot in our environment and find efficient communication methods. The ARL framework integrates environment and robot simulation with Robot Operating System (ROS) for robot control, sensor reading, and more. We determined the best method for sensor reading was through ROS Visualization (RViz) windows that converted the sensor data into an easily visualized format. We determined the best method for robot control was through ROS velocity command topics, and we created a Python script to do so more easily. These communication methods allow for a machine learning algorithm to more easily receive data about our environment and move the robot based on its computations.
3) Our framework is designed to be easily integrated with a machine learning model that detects trees as sources of cover in a military-relevant environment. This is made possible through our military-relevant simulated environment and our methods of communication.

II. Background Information

A. Robot Operating System (ROS)

Robot Operating System (ROS) [1] is a set of software frameworks for making robot software. It is used in this project to design a robot, configure the robot's joints and how to drive/control it, and to communicate with the robot. ROS is made up of packages, nodes, topics, messages, and services. Services are not used in this project.

Packages are groups of ROS files, nodes, messages, etc. with a similar purpose. Nodes are processes running in ROS that are able to publish and subscribe to topics to send and receive messages. Topics are containers of information. Some publish information to subscribers, like camera and sensor data or positions. Some are open to receive information from publishers, like controllers. Some are able to do both. Messages are the definition or format of the information that is passed between nodes and topics. Services are client/server

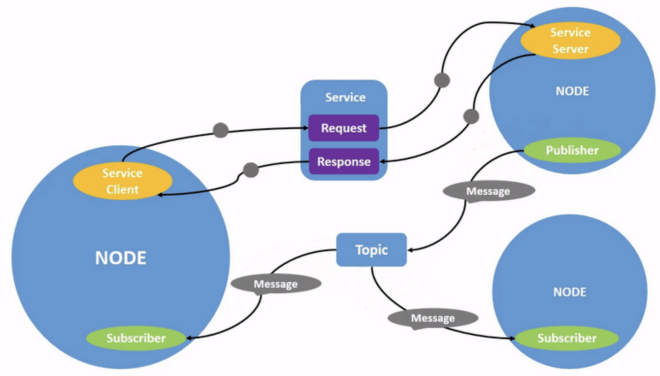

Fig. 2. Diagram of ROS and its parts. [3]

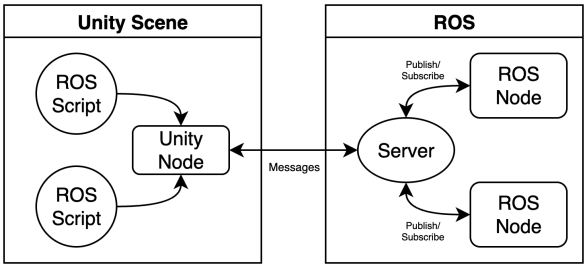

Fig. 3. Diagram of ROS/Unity integration.

systems made up of requests and responses for communicating actions between nodes.

One specific package used in this project is rospy, which is a package for using Python scripts to interact with ROS. It includes functions for creation of nodes, and publishers and subscribers to ROS topics.

B. Unity

Unity [2] is a software used for game creation and interactive simulation in 2D and 3D. It is used in this project to design the environment.

The files in Unity projects are called assets. Models are predesigned objects. Prefabs are templates for models and objects. Materials add colors, patterns, etc. to models, prefabs, and objects. Shaders control how materials appear. Textures are patterns for surfaces for objects like terrains and planes. Scripts can be attached to objects and have many different purposes, including user control. These are all different types of assets. There are many more types of assets, but these are the primary types used when making a project.

ROS/Unity integration allows users to control environments in Unity as well as objects that are connected to ROS and are simulated within Unity, like simulated robots. It works through scripts in Unity that create ROS nodes specifically for Unity. These nodes hold and publish information about the simulated world and are able to receive information from other ROS nodes. They can also be used for communication with ROS nodes for simulated robots within the environment, as well as for publishing information about the simulated robot, like its position within the world. Fig. 3 shows a visual representation of the integration process and how ROS and Unity interact.

C. Semantic Segmentation and Object Detection

Semantic segmentation and object detection are two important and fast-growing parts of machine learning.

Semantic segmentation is dividing images by pixel into various distinct object classes, such as ground, building, sky, etc. It is most commonly used for autonomous driving. Terrain segmentation is a type of semantic segmentation used for classifying different types of terrains. This can be useful to distinguish between, for example, an paved path that is easy to walk on and uneven ground.

Object detection is identifying distinct instances of objects of certain classes within an image, such as humans, animals,

Fig. 4. Terrain segmentation e.g. [16]

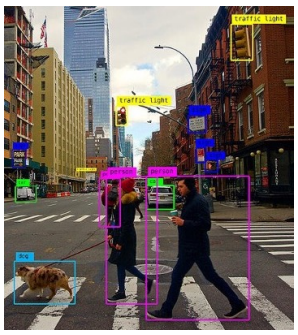

Fig. 5. Object detection e.g. [15]

furniture, cars, etc. It is also used for autonomous driving, as well as for surveillance.

D. Clearpath Husky

A Husky robot is an unmanned ground vehicle (UGV) designed by Clearpath Robotics. Clearpath makes robots designed for research and industry. The Husky robot is built for easy integration with different kinds of sensors. It is designed to be able to travel on multiple different terrains.

III. Related Work

A. Visually Realistic Multi-robot Simulation

The capabilities of game engines have been utilized to the benefit of the computer vision community in order to construct frameworks that are applicable in scientific applications where testing and evaluation of vision-based algorithms for detection, tracking, and navigation could be accomplished more effectively with inputs from several kinds of sensors and conditions of the environment. Ganoni and Mukundan [5] introduced a generic framework for simulating a fleet of robots or drones operating in a synthetic model of the natural world. The suggested simulation architecture makes use of the Unreal Engine4 to generate optical and depth sensor outputs at any location and in any direction inside the simulation environment.

B. Simulation for Agricultural Robotics

Because agricultural robots are complex systems, efficient task execution in contexts with unstructured crops and plants necessitates multidisciplinary collaborations between many study group. The design and development of agricultural robots could be sped up with the help of simulation software and virtual environments which can offer a cost-effective and trustworthy framework for experimenting with various sensing and acting mechanisms in order to confirm the performance capabilities of the robot in dynamic settings. Shamshiri,Hameed et. al. [6] provides a comprehensive overview of the various professional simulators and custom-built virtual environments that have been put to use in agricultural robotics.

C. ARL Unity and ROS Simulation Framework

This framework was designed by members of the Tactical Behaviors for Autonomous Maneuver Collaborative Research Program (TBAM-CRP) as part of the Army Research Laboratory's US Army Combat Capabilities Development Command. The framework includes multiple Unity environments: an outdoor nature/park area, a city, a basic plane with obstacles, and an outdoor storage facility area. It has the capability to simulate multiple different robots, including Husky. It has a package for full ROS integration, which can be used for loading environments, spawning robots in the environments, configuring and using sensors, and controlling the robots and environments. It also includes configuration for multiple different kinds of sensors, including inertial measurement unit (IMU) data, which reports forces, orientation, and more; a camera with RGB image, semantic segmentation, object detection, and depth capabilities; and a lidar sensor. [12]

The ARL framework was used as a primary backbone of this project. It was used to simulate a Husky robot in our designed environment and to communicate with our environment and the Husky robot through ROS.

With our framework, we aimed to improve upon the ARL framework and tailor it more towards our project's purposes and goals. In order for our framework to be military-relevant, it was important for the simulated environment to include extensive areas of cover so a machine learning model could be trained to find maximum cover from adversaries. It was also important for the simulated environment to have a diverse terrain with both potential unsafe hills and easily navigable areas so a machine learning model could be trained to find the safest path for a small autonomous robot. While the ARL framework includes many excellent environments, we did not feel that any of them met our desired qualities to be military-relevant. Therefore, we decided to design our own environment with these qualities in mind to ensure we could appropriately train a military-relevant machine learning model.

The ARL framework also provides several methods of communication with a simulated robot through its ROS integration. We wanted to find the most efficient and straightforward of those methods in order to make integration with a machine learning model as seamless as possible.

D. GA-Nav Terrain Segmentation

GA-Nav: Efficient Terrain Segmentation for Robot Navigation in Unstructured Outdoor Environments is a terrain segmentation algorithm created by Guan et. al. It is used for autonomous navigation. It is a semantic segmentation algorithm that is focused on classifying terrain based on its navigability. It includes classes such as smooth, rough, bumpy, and non-navigable. The classification is used to create a safe navigation path for a UGV [18].

The GA-Nav algorithm was used as a basis for the final part of this project – implementing a semantic segmentation or object detection algorithm to use for tree detection to obtain maximum cover. We considered their methods when deciding how we would like to implement our own algorithm. We

resolved that their terrain segmentation algorithm could be used to differentiate between open paths, water, and grassy, covered areas in our environment.

E. Semantic Mapping

Real-time Semantic Mapping for Autonomous Off-Road Navigation is a semantic mapping algorithm designed by Maturana et. al. It is used for autonomous navigation for off-road all-terrain vehicles (ATVs). A semantic map is a model of the vehicle's surroundings that includes both semantic data and geometric data, including height, size, texture, and more. The algorithm combines semantic segmentation on image data with lidar point cloud data to create a semantic map, which is then used to create a safe navigation path for an ATV [8].

We considered the methods of this semantic mapping algorithm in addition to the GA-Nav algorithm for the final part of our project. We theorized that we could use the GA-Nav algorithm to classify the terrain, and we could use semantic mapping to identify sources of cover. We hypothesized that, by combining these two methods, we could create an algorithm for obtaining maximum cover in our environment.

IV. Methods

The goal of our project was to create a framework for simulating autonomous robots in military-relevant environments and communicating with those robots. The purpose of the framework was to be integrated with a machine learning model and assist in the training and implementation of that model. Our framework was more specifically designed for a model that seeks to obtain maximum cover to avoid being detected or attacked by adversaries in a military situation.

The first step in creating our framework was designing a realistic, military-relevant forest environment using Unity. There were two important parts to consider during the design process. First, it was important for the environment to have diverse terrain so that a machine learning model could adequately learn to find the safest and most navigable terrain for an autonomous ground vehicle. Second, the environment needed to include plenty of areas of cover, in this case trees.

The next step was to simulate an autonomous robot in an environment and determine the best method of communicating with the robot while it was in the environment. We decided to use the ARL Unity ROS simulation framework to do so and take advantage of the available premade environments, access to simulated robots, and full ROS integration. We used the framework to determine the quickest and easiest method of controlling the robot in the environment, what sensors would be most useful, and how to read from those sensors.

After determining the best communication methods, we had to add our designed environment to the ARL framework, simulate a Husky robot in our environment, and apply the methods in our environment.

Finally, we attempted to use our environment and methods of communication to implement a machine learning algorithm. We determined the best method of finding maximum cover in a military-relevant environment would be to detect trees as sources of cover. We concluded the best way to do so would be using a semantic segmentation or object detection algorithm.

V. Data and Results

A. Designing the Environment

The first step in creating our framework was to design the forest environment in Unity [19]. To do so, we first created a terrain object and sculpted the height of the terrain with Unity's terrain tool to be that of a forest with many hills and a creek running through it. We then painted the texture of the terrain using the terrain tool with different kinds of grass, an open hiking path, and water for the creek and a large lake using terrain texture assets [4] [14].

The purpose of the hills and different terrain textures are to have a diverse terrain to train a machine learning model. In the real world, it is common to encounter various kinds of terrain, many of which may be non-navigable for an autonomous robot. It is also common to encounter hills, some of which may be too steep for a small robot. By providing a diverse terrain, a machine learning model can be trained to find a path that is safe and can be easily navigated.

Next, we added trees densely throughout the entire terrain. We used a Unity asset that included four kinds of conifer trees of varying height, diameter, and leaf density [9]. We used Unity's terrain tool to automatically place trees, randomly selecting amongst the four, throughout the entire environment. Density is configured by inputting the number of trees for the tool to place. We inputted a large number to make sure the terrain was densely covered. We then used the terrain tool to manually add trees in certain places to account for less dense areas the automatic process may not have covered, and we removed the trees in the paths and water.

In our environment, trees function as the main source of cover. It was important when designing the environment to include both highly covered areas and open areas. This allows for a cover-seeking machine learning model to learn to find cover from an open area in addition to learning how to maintain maximum cover. For example, if the autonomous

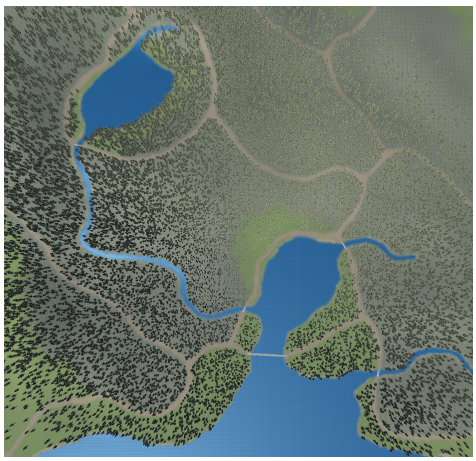

Fig. 6. Overview of designed environment.

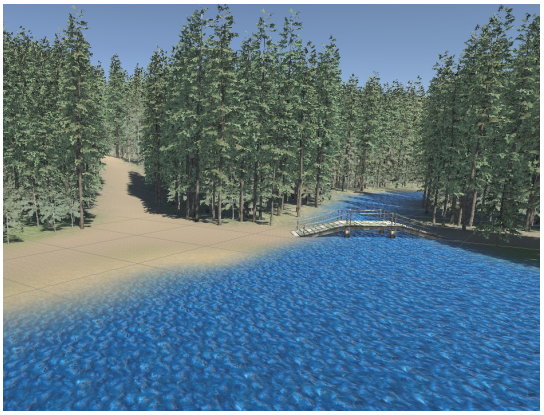

Fig. 7. Close-up of paths, trees, and bridges in designed environment.

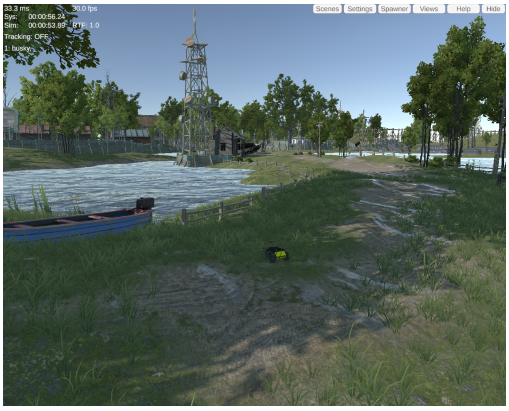

Fig. 8. Husky robot simulated in default ARL environment.

robot is in an open area, such as on a hiking path, the model can learn to find and move towards the nearest trees. Finally, we added bridges from the paths over the creeks and water using a bridges object asset [13].

Compare Figs. 1, 7, and 10, which show our designed environment, with Fig. 8, which shows the ARL framework's default environment. The differences between these two environments illustrate why we felt it was necessary to design our own environment. The ARL framework's default environment does have some trees, but we felt that it was not enough to provide sufficient cover for an autonomous robot and to train a tree detection algorithm. Because of this, we decided to design our own environment with much more trees and cover.

B. Finding a Communication Method

The next step was to find an efficient method of communication with a simulated Husky robot. We used the ARL framework's default environments to do so. We installed the framework and simulated a Husky robot in their default environment, shown in Fig. 8.

We then used their premade environments to investigate how to read from the sensors, and how to control and move the robot. For the sensors, we had configured the Husky robot with an RGB camera with semantic segmentation and object detection capabilities and a lidar sensor. We found that those sensors could be read through an RViz (ROS Visualization) configuration window. These windows communicate directly with the ROS topics that are publishing the camera and lidar sensor data and converts their messages into an easily visualized format, shown in Fig. 9.

For robot control and movement, the Husky robot is configured with a controller that can be used in the RViz window. The robot can also be controlled by publishing to the Husky velocity command topic. This method uses a ROS topic publishing command and publishes a linear and angular velocity, each with x-, y-, and z-components.

We concluded that using publisher commands would be the best method for controlling the robot and using RViz windows would be the best way to read sensor data.

C. Applying Communication Methods in Our Environment

After learning and deciding upon an efficient communication method, we applied those methods to our designed environment. First, we had to add our designed environment to the ARL framework. To do this, we downloaded ARL's original Unity project where the creators made the simulation interface and designed all of the default environments. We then exported our Unity project's assets and saved them in the ARL framework's ROS workspace. Finally, we created a launch file configured specifically for our environment and loaded it in the ARL framework through their Unity project, shown in Fig. 10.

Once we had loaded our environment in the ARL framework, we simulated a Husky robot in our environment and applied the communication methods. We used the ROS topic publisher command to move the robot around, and we used RViz to read the sensor data. As hoped, the methods worked as efficiently as they had in the default ARL environments. We also created a Python script using rospy to control the robot. The script creates its own ROS node, creates a publisher, and publishes to the same ROS topics as the ROS command. We created the script to make control more intuitive and to provide

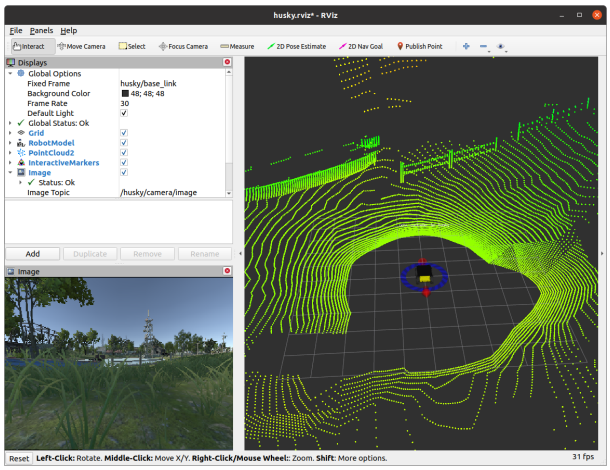

Fig. 9. RViz configuration window with camera image and lidar sensor data.

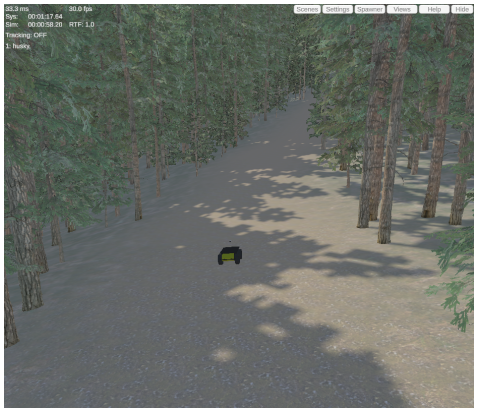

Fig. 10. Husky robot simulated in designed environment.

easy integration with a machine learning algorithm.

VI. Conclusion

We created a framework for simulating an autonomous robot in a military-relevant environment and communicating with that robot. Our framework is ready for integration with a machine learning model that seeks to obtain maximum cover through tree detection.

We designed a forest environment in Unity that provides a diverse terrain and both open and highly covered areas. The environment is designed to provide cover for an autonomous robot from adversaries in a military situation. We used the ARL Unity ROS simulation framework to simulate a Husky robot and find methods of communicating with the robot and getting environment data through a camera and lidar sensor. We determined ROS topic publisher commands were the most effective method of robot control and RViz configuration windows were the most effective method of reading sensor data. We added our designed environment to the ARL framework, simulated a Husky robot in our environment, and applied the communication methods with creating a Python script to control the robot more easily through rospy.

In the future, we would like to successfully implement a machine learning algorithm and determine a method of finding tree cover for use in military-relevant environments. We propose a solution that combines semantic segmentation and object detection. We propose applying semantic segmentation on our image data to classify the different types of terrain in our environment, as well as classify objects such as trees, bridges, and any other obstacles. We propose applying object detection on our image data, with use of data from the LiDAR sensor if necessary, to more concretely identify where the trees are. In order to obtain maximum cover, we propose using the semantic segmentation algorithm to find the general location and direction of nearby trees so a machine learning model could rotate an autonomous robot towards the trees. We then propose using the object detection algorithm to find the distance to the nearest tree so a machine learning model could move the robot to a set distance from the tree.

Once a tree detection algorithm is implemented, we plan to combine this framework with a reinforcement learning (RL) algorithm [17]. RL is a type of machine learning that trains models based on positive or negative rewards for each action. The RL algorithm we aim to combine with our framework is used for trajectory planning in unstructured outdoor environments. It identifies trajectories that have the most cover while remaining navigable and low-cost, meaning there are not too many elevation changes and hills, ensuring the robot can stay less visible. The RL algorithm can be combined with the tree detection algorithm to obtain maximum cover. Both can be tested in our environment using the developed simulated framework.

Acknowledgment

The work is funded by NSF Research Experiences for Undergraduates (REU) grant #CNS-2050999 and the U.S. Army grant #W911NF2120076.the U.S. Army grant #W911NF2120076

References

[1] Quigley, Morgan, et al. "ROS: an open-source Robot Operating System." ICRA workshop on open source software. Vol. 3. No. 3.2. 2009.

[2] "Unity Simulation Pro," *Unity*. [Online]. Available: https://unity.com/products/unity-simulation-pro.

[3] "ROS 2 Documentation," *ROS 2 Documentation: Foxy*. [Online]. Available: http://docs.ros.org/en/foxy/index.html.

[4] A dog's life software, "Outdoor Ground Textures," *Unity Asset Store*,11-Jan-2021.[Online]. Available:https://assetstore.unity.com/packages/2d/textures-materials/floors/outdoor-ground-textures-12555.

[5] Ganoni, Ori, and Ramakrishnan Mukundan. "A framework for visually realistic multi-robot simulation in natural environment." arXiv preprint arXiv:1708.01938 (2017).

[6] Shamshiri R R, Hameed I A, Pitonakova L, Weltzien C, Balasundram S K, Yule I J, et al. Simulation software and virtual environments for acceleration of agricultural robotics: Features highlights and performance comparison. Int J Agric Biol Eng, 2018; 11(4): 15–31.

[7] A. Gupta, "ML | Reinforcement Learning Algorithm: Python Implementation using Q-learning," *GeeksforGeeks*, 2019. [Online]. Available: https://www.geeksforgeeks.org/ml-reinforcement-learning-algorithm-python-implementation-using-q-learning/.

[8] D. Maturana, P.-W. Chou, M. Uenoyama, and S. Scherer, "Real-time Semantic Mapping for Autonomous Off-Road Navigation," in *Proceedings of 11th International Conference on Field and Service Robotics (FSR '17)*, 2017, pp. 335–350.

[9] forest, "Conifers [BOTD]," *Unity Asset Store*, 26-Jul-2021. [Online]. Available:https://assetstore.unity.com/packages/3d/vegetation/trees/conifers-botd-142076.

[10] "Gazebo Tutorials," *Gazebo*. [Online]. Available: https://classic.gazebosim.org/tutorials.

[11] J. Fink, G. Gremillion, S. Nogar, and A. Schang, "ARL Robotics Simulation," *GitLab*, 2019. [Online]. Available: https://gitlab.sitcore.net/arl/robotics-simulation/.

[12] J. Rogers, D. Asher, L. Frerichs, and D. Baran, "Tactical Behaviors for Autonomous Maneuver Collaborative Research Program (TBAM-CRP)," *US Army Combat Capabilities Development Command – Army Research Laboratory*, 29-Apr-2022. [Online]. Available: https://www.arl.army.mil/business/collaborative-alliances/current-cras/tbam-crp.

[13] Laxer, "Modular Wooden Bridge Tiles," *Unity Asset Store*, 07-Aug-2017. [Online]. Available: https://assetstore.unity.com/packages/3d/props/exterior/modular-wooden-bridge-tiles-29501.

[14] LowlyPoly, "Stylize Water Texture," *Unity Asset Store*, 11-Sep-2019. [Online]. Available: https://assetstore.unity.com/packages/2d/textures-materials/water/stylize-water-texture-153577.

[15] *Object detection in a city*, "Object Detection Powered by Computer Vision," *alwaysAI*. [Online]. Available: https://learn.alwaysai.co/object-detection.
[16] P. Jiang, P. Osteen, M. Wigness, and S. Saripalli, *Data example*, "RELLIS-3D: A Multi-modal Dataset for Off-Road Robotics," 2020. [Online]. Available: https://unmannedlab.github.io/research/RELLIS-3D.
[17] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA: The MIT Press, 2018.
[18] T. Guan, D. Kothandaraman, R. Chandra, A. J. Sathyamoorthy, K. Weerakoon, and D. Manocha, "GA-Nav: Efficient Terrain Segmentation for Robot Navigation in Unstructured Outdoor Environments," 2021.
[19] UGuruz, "How to Make Beautiful Terrain in Unity 2020 | Beginner Tutorial," *YouTube*, 15-Jul-2020. [Online]. Available: https://www.youtube.com/watch?v=ddy12WHqt-M.
[20] Unity Technologies, "Unity Robotics Hub," *GitHub*, 2020. [Online]. Available: https://github.com/Unity-Technologies/Unity-Robotics-Hub.