A Sim-to-Real Pipeline for Deep Reinforcement Learning for Autonomous Robot Navigation in Cluttered Rough Terrain

Han Hu, *Student Member, IEEE*, Kaicheng Zhang, *Student Member, IEEE,* Aaron Hao Tan*, Student Member, IEEE,* Michael Ruan, Christopher Agia, and Goldie Nejat, *Member, IEEE*

Abstract— **Robots that autonomously navigate real-world 3D cluttered environments need to safely traverse terrain with abrupt changes in surface normals and elevations. In this paper, we present the development of a novel sim-to-real pipeline for a mobile robot to effectively learn how to navigate real-world 3D rough terrain environments. The pipeline uses a deep reinforcement learning architecture to learn a navigation policy from training data obtained from the simulated environment and a unique combination of strategies to directly address the reality gap for such environments. Experiments in the real-world 3D cluttered environment verified that the robot successfully performed point-to-point navigation from arbitrary start and goal locations while traversing rough terrain. A comparison study between our DRL method, classical, and deep learning-based approaches showed that our method performed better in terms of success rate, and cumulative travel distance and time in a 3D rough terrain environment.**

Index Terms— **Autonomous Agents; Deep Learning for Robotics and Automation; Search and Rescue Robots**

I. INTRODUCTION

MOBILE robots need to autonomously traverse and navigate real-world 3D cluttered rough terrain in numerous applications including urban search and rescue [1]–[3], hazardous material clean-up [4], and mining and construction [5]. Traversability of such terrain can be complex due to the existence of uneven ground, ramps, steps, and rocks consisting of varying shapes and sizes. To successfully navigate such terrain, a robot must be able to find traversable paths within the environment to reach different goal locations.

To date, the most common approaches used for navigating rough terrain have focused on either representing terrain traversability by using handcrafted heuristics [6]–[10] or learning-based models [5], [11]–[13]. In general, these methods are optimized for terrain traversability by solely considering terrain features selected by experts. They do not consider a robot's intended actions and states in the environment, such as wheel positioning and angle of approach. Furthermore, these approaches represent terrain traversability as a categorical label or a scalar value. These simplifications can result in the loss of critical information for more challenging 3D terrain with abrupt changes in elevation and surface normals, where such robot and environment interactions can directly influence a robot's ability to successfully navigate through the rough terrain.

In our own previous work in [14], we introduced a deep reinforcement learning (DRL) approach for robots to learn how to traverse unknown rough terrain by explicitly considering a robot's pose in the environment along with 3D terrain information. However, similar to [15], [16], both training and testing were in simulated 3D environments.

Sim-to-real is an emerging research area that studies how to overcome the discrepancies between the simulated and real-world environments, known as the reality gap, as a result of factors such as sensory noise, lighting conditions, and unmodeled dynamics [17]. Sim-to-real has been used in a number of robotic applications including aerial vehicles [18], manipulators [19], and mobile robots [20]–[22]. For mobile robot navigation, the environments considered so far have been in buildings including hallways and offices, where the ground has always been flat. To the authors' knowledge, sim-to-real approaches have not yet been considered for robot navigation in 3D rough terrain environments, which has the added challenge of climbable terrain with varying shapes, sizes, and steepness. This can introduce additional simulation-reality discrepancies with respect to the modeling of the 3D terrain and a robot's direct interactions with the environment.

In this paper, we propose a novel sim-to-real pipeline and unique strategies to address the challenge of robot navigation in 3D cluttered real environments. Our main contributions are: 1) we are the first to present a sim-to-real approach and overall pipeline for autonomous navigation in cluttered 3D terrain, 2) we introduce strategies specific to 3D cluttered terrain to overcome the reality gap by uniquely considering robot and environment interactions when using DRL-based robot navigation in such 3D environments, 3) we successfully train our DRL policies in simulation and transfer them to a physical mobile robot which is able to traverse real-world rough terrain without further adaptation, and 4) we present a comparison study which highlights the superior performance of our navigation method with these strategies over standard classical

Manuscript received: February 22, 2021; Revised: May 24, 2021; Accepted: June 21, 2021.

This paper was recommended for publication by Editor Youngjin Choi upon evaluation of the Associate Editor and Reviewers' comments.

This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC), and the Canada Research Chairs program (CRC).

The authors are with the Autonomous Systems and Biomechatronics Laboratory, Department of Mechanical and Industrial Engineering, University of Toronto, Toronto ON, M5S 3G8 Canada (e-mail: {kc.zhang; aaronhao.tan christopher.agia; michael.ruan}@mail.utoronto.ca, {hanhu, nejat} @mie.utoronto.ca). *The first two authors contributed equally to this work.*

Digital Object Identifier (DOI): see top of this page.

and deep learning-based techniques in cluttered environments.

II. Related Works

Existing navigation methods that have been used for robots traversing rough terrain can be classified as: 1) classical approaches, 2) learning-based techniques, and 3) sim-to-real techniques used to transfer learned navigation behaviors from simulations to real-world environments.

A. Classical Approaches for Rough Terrain Environments

In general, classical navigation approaches for robots in rough terrain environments have utilized a combination of a terrain traversability classifier and a path planner. In [23], an extensive survey on classical terrain assessment approaches was presented. The most common approach utilizes geometric information with statistical processing techniques to extract traversability features [7]–[10]. In particular, statistical values such as terrain roughness, plane slope, and height were extracted from 3D point clouds to determine a traversability assessment measure based on heuristics, or a scalar cost. These measures were used in conjunction with 1) rapidly exploring random trees, 2) D*-Lite, or 3) probabilistic roadmap method for navigation.

Classical approaches utilized either heuristic rules or cost functions that were manually designed and tuned for traversability assessment. Such manual design can be time consuming, requires expert knowledge, and is difficult to transfer to other robots, since the design is specific to a robot's physical characteristics. Moreover, these approaches classify terrain into either a binary [7], [8], [10], or a scalar [9] measure. This simplification can result in the loss of critical information for representing traversability (e.g., robot's relative pose, angle of approach to the terrain, and previous navigation actions). As a result, these methods are not able to generalize well to varying real-world cluttered terrain.

B. Learning Approaches for Rough Terrain Navigation

Learning based techniques have also been used to determine traversability in rough terrain environments, e.g. [5], [11]–[13]. For example, in [5], a traversability cost function was learned from a manually-labeled cost map and human navigation demonstrations. In [11], a support vector machine was retrained online for binary ground classification in outdoor forest terrain using texture and color features from 3D point clouds. In [12], [13], a deep learning based approach used convolutional neural networks (CNNs) to classify terrain to produce binary traversability measures. Compared to the classical approaches, the learning-based terrain traversability classification methods can be applied to real-world terrain while avoiding the need for manually designed and tuned rules. However, many of these methods require a large amount of manually labeled data. Unfortunately, they also simplify traversability into discrete classes or numerical values, resulting in the potential loss of information.

End-to-end DRL approaches were presented in [14]–[16] to directly map sensory modalities to robot motion commands without simplifying the terrain traversability representation. In [15], zero to local-range sensing was used within a rainbow DRL architecture. In [16], 2D laser scans, RGB images, and 3D point cloud were fused using a three-branch network architecture. However, both training and testing were only conducted in 3D cluttered simulated environments, therefore these works do not address the reality gap.

C. Sim-to-Real Strategies for Robot Navigation

Sim-to-real strategies have been developed for robot navigation tasks [18], [20]–[22]. For example, in [18], domain randomization was applied to visual parameters such as texture, lighting, and furniture placement of a set of synthetic hallway environments during training for learning collision-avoidance actions for a quadcopter.

Lidar distance measurements or semantic features extracted from RGB images were used as inputs to learn robot velocity or navigation commands using Asynchronous Deep Deterministic Policy Gradient network [20], Probabilistic Road Map–Reinforcement Learning [21], or Asynchronous Advantage Actor-Critic (A3C) network [22].

The aforementioned learning-based navigation approaches demonstrate the feasibility of using DRL to learn 2D navigation behaviors in structured environments (e.g., hallways and offices) and using sim-to-real strategies to transfer the learned policy to real-world environments. In this paper, we propose a novel pipeline that utilizes a unique combination of sim-to-real strategies for a mobile robot to navigate real-world 3D cluttered rough terrain environments autonomously. To the authors' knowledge, we are the first to consider 3D rough terrain environments in sim-to-real applications. We have developed new sim-to-real strategies to directly address the reality gap for such 3D environments. Using this novel sim-to-real pipeline, we extend our A3C DRL-based rough terrain navigation method developed in [14] so that it can be directly applied to *real-world* 3D cluttered environments.

III. Methodology

Our proposed sim-to-real architecture for robot navigation in 3D cluttered rough terrain is presented in Fig. 1. Using RGB-D, 3D lidar, and IMU information as *Perception* inputs, a 3D point cloud map of the real environment is first generated by the *3D Mapping* module. This map is then reconstructed into a 3D mesh by the *3D Meshing* module to represent the surface geometry of the terrain. The simulated environment provides the simulated robot sensory data that is used as input to the *Deep Reinforcement Learning (DRL)* module.

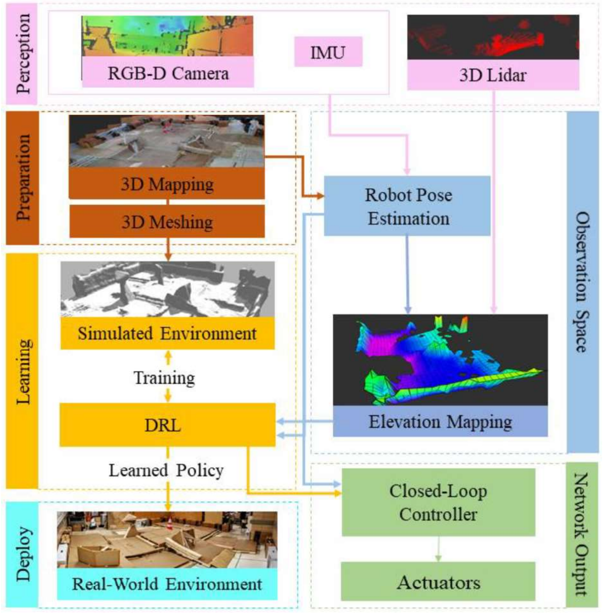

Figure 1. Proposed sim-to-real pipeline for rough terrain robot navigation.

The *DRL* model uses an elevation map, which provides a 2D representation of the surrounding terrain, and the robot's pose as inputs. The *DRL* module then learns a policy for the robot to navigate the environment. For real-world deployment, the *Perception* inputs are used in the *Robot Pose Estimation* module to determine the robot's 6 degrees-of-freedom (DOF) pose within the real world. The robot's pose and lidar information are then used by the *Elevation Mapping* module to generate the real-time elevation map. Similar to the *Simulated Environment* used for training, the pose and elevation map are used as inputs to the *DRL* module. Based on the trained policy, the *DRL* module determines the desired robot actions, which are then sent to the robot's *Closed-Loop Controller* to be executed as motion commands for the robot's *Actuators*. The main modules of the pipeline are discussed in details below.

A. 3D Mapping & 3D Meshing

A 3D map of the real environment is created using the RGBDSLAMv2 method [24] within the Real-Time Appearance-Based Mapping (RTAB-MAP) package in ROS [25]. It uses simultaneous localization and mapping (SLAM) to construct and update a 3D point cloud map of the environment using information from an RGB-D sensor with an onboard IMU [25]. This 3D map is reconstructed into a 3D continuous mesh surface using a Poisson surface reconstruction approach [25]. Meshlab [26] is also used to fill any holes in the mesh and filter out any floating surfaces. The mesh provides a continuous surface for a robot to interact with in the *Simulated Environment*.

B. Observation Space

The observation space represents the inputs to the *DRL* in both the *Simulated Environment* and the *Real-World Environment.* It consists of the elevation map from the *Elevation Mapping* module, the robot's distance d_t to the goal location, and the robot's orientation $(\alpha_t, \beta_t, \gamma_t)$, where the robot heading, γ_t, is relative to the target goal location on the terrain. d_t and $(\alpha_t, \beta_t, \gamma_t)$ are determined by the pose provided by the *Robot Pose Estimation* module. The elevation map, and the robot's orientation towards and distance to the goal location are domain invariant and do not include raw sensory data. This data representation prevents the DRL model from overfitting to features that are only present in the *Simulated Environment*, namely, color, texture, lighting conditions, and sensor noise patterns [17].

1) Elevation Mapping: An elevation map centered around the robot is generated using the ROS Elevation Mapping package [27]. The map consists of a $5m$ by $5m$ 2D grid, where each grid cell represents the average height of a squared area.

2) Robot Pose Estimation: In the *Simulated Environment*, the robot's pose is directly obtained from the ROS Gazebo simulator [28] which uses a transform tree structure to maintain the spatial relationship between objects and coordinate frames. For real-world deployment, RGB-D images, IMU, and the 3D map of the environment are used to estimate the robot pose p_t using RGBDSLAMv2 [24].

C. Simulated Environment

We developed a 3D simulated environment in Gazebo using the 3D mesh of the real environment. We used the 3D physics model for the wheeled mobile platform developed in [14] to represent the real robot. The cluttered real-world 3D terrain consists of a combination of changes in elevation including ramps and steps, sharp corners, and climbable and unclimbable obstacles of different shapes and sizes.

D. DRL Module

The *DRL* module uses the A3C DRL method [29] to learn the policy and the state-value function for rough terrain navigation to determine the optimal robot navigation action under a given reward. At each discrete time step t, the robot in state s_t executes a navigation action a_t according to the policy $\pi(a_t|s_t, \theta)$. This action transitions the robot to state s_t' to maximize expected future rewards. The policy $\pi(a_t|s_t; \theta)$ is determined by a neural network parameterized with weights θ. The navigation actions consist of the robot moving forward or backward for a travel distance L, or turning right or left for a yaw rotation angle θ_Y. A state-value function $V(s_t; \theta_v)$, which is the expected future cumulative reward starting from state s_t is estimated by a state-value network parameterized with weights θ_v to help update the policy [30]. Rewards are used to compute the gradients with respect to θ and θ_v at every step. Stochastic gradient descent (SGD) is used to update the θ and θ_v as follows [29], [31]:

$$\Delta\theta = \nabla_\theta \log\pi(a_t|s_t; \theta)A(s_t, a_t; \theta_v) + \zeta\nabla_\theta H(\pi(s_t; \theta)), \quad (1)$$

$$\Delta\theta_v = A(s_t, a_t; \theta_v)\nabla_{\theta_v} V(s_t; \theta_v), \quad (2)$$

where,

$$A(s_t, a_t; \theta_v) = \sum_{i=0}^{n-1} \Gamma^i r_{t+i} + \Gamma^n V(s_{t+n}; \theta_v) - V(s_t; \theta_v), \quad (3)$$

and where $A(s_t, a_t; \theta_v)$ is the advantage function and Γ is the discount factor. n is the number of steps to the end of a training episode. H is the entropy term to encourage the robot to explore different navigation actions, and the hyperparameter ζ determines the strength of H.

Our developed A3C structure is presented in Fig. 2. A long short-term memory (LSTM) recurrent layer [32] is used to capture information from previous robot states. The output of the network consists of both the navigation policy $\pi(a_t|s_t; \theta)$ and the state-value function $V(s_t; \theta_v)$.

Rewards: Positive rewards are given to encourage the following actions: 1) move closer to the goal location, 2) explore surrounding terrain for alternative routes to the goal, when an existing route is not traversable, and 3) reach the state s_{goal} by arriving at the navigation goal within a certain distance tolerance d_g. Negative rewards are given to discourage the robot from reaching an undesirable terminal state s_{fail}. Undesirable terminal states include the robot getting stuck on obstacles, flipping over, and/or reaching a time-out limit.

After executing a navigation action at time step t, the robot receives a reward r_t determined as follows:

$$r_t = \begin{cases} -\mu_1 \Delta d_{min,t}, & \Delta d_{min,t} < 0 \\ \mu_2 \Delta a_{expl,t}, & \Delta a_{expl,t} > 0 \\ 1, & s_{goal} \\ -0.5, & s_{fail} \\ 0, & elsewhere \end{cases}, \quad (4)$$

where distance $d_{min,t}$ is the closest distance that the robot has navigated to with respect to the target goal location up to the current time step t. $a_{expl,t}$ is the explored area of the environment at time step t. To keep track of the explored area, the robot maintains a 2D binary grid map where each cell represents a $5m \times 5m$ square area with values that represent whether the robot has visited the cell. All grid cell values initiate with 0, representing unexplored cells. Cells that are explored have a value of 1. $\Delta d_{min,t}$ and $\Delta a_{expl,t}$ represent the changes in $d_{min,t}$ and $a_{expl,t}$, with respect to the previous time step, $t-1$. $\mu_1 = 0.2$ and $\mu_2 = 0.005$ are two hyperparameters that adjust the strength of rewards related to $\Delta d_{min,t}$ and $\Delta a_{expl,t}$, such that the robot learns to navigate to the goal location safely and efficiently. We empirically determined that μ_1 should be between 0.1-0.3, and μ_2 should be between 0.001-0.01. Empirical analysis found that if μ_1 is too high (above the range), the robot learns to move aggressively towards the goal location without considering obstacle-free routes. If u_2 is too high (above the range), the robot first tries to unnecessarily explore the full environment before heading towards the goal location. On the other hand, if μ_1 and μ_2 are too low (below these ranges), the robot does not receive enough rewards for training.

1) Closing the Reality Gap

To address the reality gap [17], we uniquely incorporate three sim-to-real strategies during training. The first two strategies were developed to address challenges that are unique to 3D terrain navigation, while the third focuses on robot pose estimation errors.

a) Varying Terrain Steepness: The accumulation of depth camera measurement errors from 3D Mapping can result in the height of the 3D mesh deviating from its real-world counterpart. Consequently, the steepness and the traversability of the simulated terrain may not reflect the real-world accurately. Therefore, robots trained in the simulated terrain may execute inappropriate navigation actions in the real-world such as attempting to climb steep slopes or not following traversable paths. To alleviate this issue, we vary terrain steepness by scaling the mesh height h by a uniform distribution $U_h(e_{lh}, e_{uh})$ with lower bound e_{lh} and upper bound e_{uh}. Uniform distribution is used to expose the CNN to a broad distribution of randomized parameters to increase the likelihood of the real-world values being present during training, as the terrain steepness measurement error cannot be accurately modeled. This allows the CNN to cover a wide range of variations during training in order for it to be generalizable to the real-world, as the real-world may appear to the model as such a variation [17], [18]. This new height of the mesh is represented as:

$$h \cong h_o U_h(e_{lh}, e_{uh}), \tag{5}$$

where h_o is the original height of the mesh obtained *from 3D Meshing*. If the height of the terrain is scaled by a factor larger than one, the steepness across the terrain is increased, while a smaller number decreases the steepness.

b) Robot Motion Disturbance: When a robot navigates over rough terrain, its interactions with the environment may not result in the intended actions. For example, slippage could occur when climbing an obstacle or rotating without sufficient traction. To improve the robustness of our DRL navigation system to such interactions with the environment, movement disturbances are applied to both the robot's travel distance L and yaw rotation angle θ_γ. These movement disturbances are represented by uniform distributions $U_L(e_{lL}, e_{uL})$ and $U_{\theta_\gamma}(e_{l\theta_\gamma}, e_{u\theta_\gamma})$, where e_{lL} and $e_{l\theta_\gamma}$ are the lower bounds and e_{uL} and $e_{u\theta_\gamma}$ are the upper bounds, respectively.

In addition to disturbances from 3D terrain interactions, robot movements are also affected by the network latency and the latency due to obtaining measurements from visual odometry when executing control commands. These disturbances are also represented as uniform distributions, $U_L'(e_{lL}', e_{uL}')$ and $U_{\theta_\gamma}'(e_{l\theta_\gamma}', e_{u\theta_\gamma}')$, with e_{lL}' and $e_{l\theta_\gamma}'$ as lower bounds and e_{uL}' and $e_{u\theta_\gamma}'$ as upper bounds.

The main difference between these two types of disturbances is that the interaction disturbances happen at random instances during rough terrain navigation, and the latency disturbances occur at every time step. Therefore, the robot's motion during training is represented as:

$$L = L \pm I U_L(e_{lL}, e_{uL}) + U_L'\left(e_{lL}', e_{uL}'\right), \tag{6}$$

$$\theta_\gamma = \theta_\gamma + I U_{\theta_\gamma}\left(e_{l\theta_\gamma}, e_{u\theta_\gamma}\right) + U_{\theta_\gamma}'\left(e_{l\theta_\gamma}', e_{u\theta_\gamma}'\right), \tag{7}$$

where I follows a Bernoulli distribution, $I \sim Bern(P)$, to represent that movement disturbance from 3D terrain interactions can happen at random instances. P is the probability of successfully applying this disturbance.

c) Robot Pose Estimation Error: Robot pose errors exist due to occurrences of both image errors and feature association errors. The former are a result of lens distortion effects and biases in the image rectification process, while the latter is a result of the inclusion of ambiguous and spurious features [33]. Inaccurate association of visual features for localization can result in the estimated robot 6 DOF pose p_t $(x_t, y_t, z_t, \alpha_t, \beta_t, \gamma_t)$ deviat from the true values [33]. These errors are reflected in the inputs to the DRL network, as $(\alpha_t, \beta_t, \gamma_t)$ and d_t are obtained directly from p_t and the elevation map uses p_t. To ensure the robustness of our DRL navigation to real-world pose estimation errors, we represent pose errors by adding a Gaussian distribution $N_p(0, \sigma_p)$ with standard deviation σ_p to the robot's pose in simulation:

$$p_t = p_o + N_p(0, \sigma_p), \tag{8}$$

where p_o is the current 6 DOF robot pose.

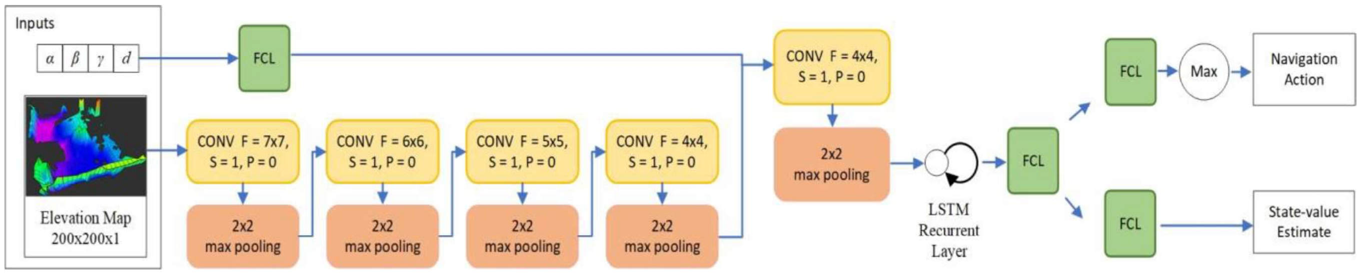
Inputs
α
β
γ
d
Elevation Map 200x200x1
FCL
CONV F = 7x7, S = 1, P = 0
CONV F = 6x6, S = 1, P = 0
CONV F = 5x5, S = 1, P = 0
CONV F = 4x4, S = 1, P = 0
2x2 max pooling
CONV F = 4x4, S = 1, P = 0
LSTM Recurrent Layer
FCL
Max
Navigation Action
State-value Estimate

Figure 2. Neural network structure with CNN hidden layer configuration details. CONV represents a convolutional layer with filter size F, stride S, and padding P. FCL represents a fully connected layer.

2) *Training*

The training was conducted by randomly generating different robot starting locations and headings, and goal locations within the simulated environment. Nine agent threads were simultaneously used to run nine independent simulations. A robot executed a series of navigation actions in each of the simulation environments based on the latest policy determined by the A3C network and received rewards to update the weights of the A3C network.

The general parameter values used in training are as follows: $L = 10\,cm$; $\theta_\gamma = 15°$; $\zeta = 0.01$; $\Gamma = 0.99$; $d_g = 0.3\,m$; $\mu_1 = 0.2$; $\mu_2 = 0.005$; and $t_{max} = 5$. The learning rate was set to 0.0001 [31]. The parameter values utilized in the three sim-to-real strategies are detailed below:

a) Terrain Steepness Parameter: A $\pm 5\%$ scaling was added to the mesh height, i.e., $e_{lh} = 95\%$ and $e_{uh} = 105\%$. This scaling was determined based on the translational errors obtained along the height direction of the environment in RTAB-Map [25].

b) Robot Motion Disturbance Parameters: $e_{lL} = 20\,cm$, $e_{uL} = 40\,cm$, $e_{l\theta_\gamma} = -45°$, $e_{u\theta_\gamma} = 45°$ and $P = 0.017$ were used to apply a large enough disturbance to knock the robot off its original trajectory at expected intervals of 60 time steps based on P. e_{lL} and e_{uL} were chosen based on 50% and 100% of the robot length. $e'_{lL} = 0\,cm$, $e'_{uL} = 3\,cm$, $e'_{l\theta_\gamma} = 0°$ and $e'_{u\theta_\gamma} = 3°$ were used to model movement errors caused by latency. They were measured on the physical robot in an arbitrary environment. The deviations of the actual travel distance L and yaw rotation angle θ_γ from the desired values after the robot finished executing an action were obtained using visual odometry [25].

c) Robot Pose Estimation Errors: We used $\sigma_p = 6\,cm$ and $\sigma_p = 5°$. These error values were selected based on the translation and orientation errors that can be expected when mapping similar-sized environments using Visual SLAM [24], [25].

An AMD Threadripper 2990wx CPU was used to train the *DRL* network. The total training time was about 16.4 days. Figure 3 presents the cumulative rewards per episode averaged across 2,000 episodes. As can be seen by the figure, the A3C network started to converge after 200,000 episodes for an average cumulative reward of 7.5.

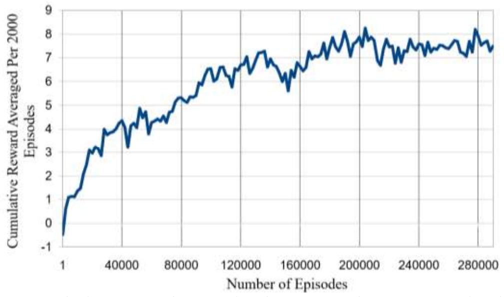

Figure 3. Cumulative reward per episode averaged per 2000 episodes.

E. *Closed-Loop-Controller*

A two-layer closed-loop controller has been designed consisting of a bang-bang controller [34] for position control using visual odometry, and a proportional controller [35] to control the wheel-spin rate using wheel encoders. This two-layer system compensates for wheel slippage caused by terrain surface roughness. The bang-bang controller outputs a constant linear and angular robot velocity until the robot has traveled a distance L and rotated by yaw angle θ_γ. The linear and angular velocities are converted into individual wheel spin rates using the kinematic model of skid steering robots [36]. The spin rates are then sent to the proportional controller for each wheel to track the setpoint spin rate.

F. *Real-World Environment*

The learned navigation policy is transferred to the *Real-World Environment* from the *DRL* module into a wheeled robot without requiring any training in the real-world.

IV. Experiments

The experiments consist of: 1) physical robot experiments in the real-world environment to determine the success rate of reaching goal locations autonomously, 2) a comparison study of our proposed DRL method for 3D rough terrain with respect to both a classical navigation approach and a learning-based terrain traversability classification method, and 3) a sensitivity analysis on the effect of the sim-to-real strategies.

A. *Real-World-Experiments*

1) *Mobile Robot*

The Jaguar 4x4 skid-steered mobile robot is equipped with a Velodyne VLP-16 3D LiDAR, and a Stereo Labs ZED Mini sensor containing a stereo camera and IMU, Fig. 4. The robot has two computing units. The primary unit is an Intel Mini PC (NUC) with a Core i5 CPU and the secondary unit is an Nvidia Jetson TX2 mobile GPU. The Jetson TX2 mobile GPU is used to obtain RGB images and IMU data collected from the ZED Mini sensor for obtaining robot poses. The NUC uses these poses, the 3D Lidar point clouds, and encoder data to obtain an elevation map and execute robot navigation actions via the *Closed-Loop Controller* module. A server equipped with an Intel i7-7700K CPU runs the *DRL* module which generates the navigation action commands sent to the NUC, using the elevation maps and robot poses.

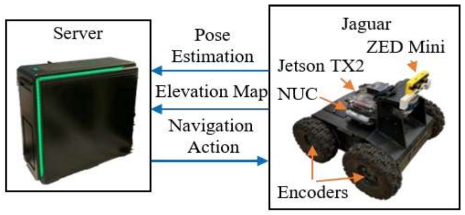

Figure 4. The Jaguar robot sends pose estimation and elevation map data to the server and receives navigation action commands over Wi-Fi.

2) *Cluttered 3D Environment*

We developed a 36m^2 cluttered rough terrain environment in our lab with multiple levels and ramps made from wooden pallets, with scattered objects such as pylons, boxes, and rocks, Fig. 5(a). The environment was divided into 22 separate regions with different types of transitions between neighboring regions. These transitions included full steps, half steps, and ramps, Fig. 5(b) and (c). Full steps had a height of 0.13m. A half step had a height between 0.05 to 0.08m. The slopes and lengths of the ramps ranged between 20°-40° and 0.3-0.9m, respectively. The traversability of the terrain was dependent on the robot's current position and angle of approach, as well as its intended action.

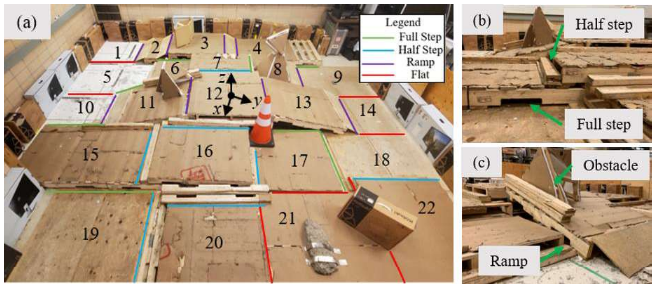

Figure 5. Real-world environment with: (a) its different regions (1 to 22) and transitions between regions labeled; zoomed in views on two pallets stacked with (b) full and half steps, and (c) wooden obstacles on a ramp.

3) *Procedure*

For each experiment, random start and goal locations were selected for the robot to traverse. Each trial was considered successful only if the robot navigated autonomously to the goal, within a position tolerance of 0.3m and a timeframe of 10 minutes. Twenty combinations of start and goal locations were randomly chosen across the environment, and repeated three times, for a total of 60 trials. The trajectories for these location pairs are shown in Fig. 6.

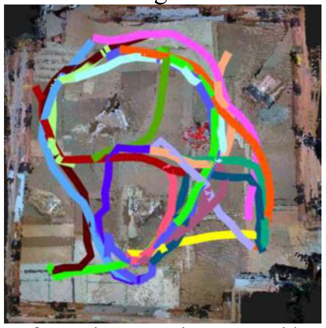

Figure 6. Top view of experiment environment with trajectories from 20 start and goal locations represented by color-unique lines.

4) *Results*

The success rate for all 20 location pairs is presented in Table I, where the percentage is the average across three trials per test (60 trials in total). Overall, the robot achieved a success rate of 86.67% in the environment. In general, the robot was able to traverse the multi-leveled cluttered scene using what it had learned during training. Some of the navigation strategies learned include making use of adjacent half steps to climb full steps with better success and keeping a minimum distance from walls to avoid collisions, even at the cost of having to travel on uneven climbable portions of the terrain. When becoming stuck in a portion of the environment, the robot would also learn to try a different heading direction. A video of our robot navigating rough terrain in simulation and in the real-world with the trained policy is presented at https://youtu.be/dtYlNWvK-7k.

A successful trial in the cluttered environment is shown in Test 10 as presented in Fig. 7(a) in yellow. While navigating from the start location in region 10 to the goal location in region 9, the robot was able to: 1) climb a ramp while avoiding an obstacle on it (Fig. 7(b)), 2) use an adjacent ramp to aid in climbing over a full step (Fig. 7(c)), and 3) descend a ramp without falling (Fig. 7(d)). In Tests 3, 5 and 7, the robot successfully navigated to the goal in some but not all trials. Unsuccessful trials in these tests were due to repeated vibrations along trajectories with sudden drops and bumps between the wheels and the pallets which led to variability in robot decisions. Another reason was due to the deviation between the simulated and real-world maps caused by limited resolution from both the RTAB-Map and the 3D mesh. This deviation caused certain gaps to appear smaller in simulation which led the DRL algorithm to learn incorrect traversability. In Test 20, the simulated robot trajectory (purple) was able to reach the goal; however, the robot failed to complete this path during the real experiment (orange) because the angle of approach caused the ramp to be unclimbable in the real world. This ultimately stopped the robot at the terminal pose seen in Fig. 7(a) after it timed out.

TABLE I. SUCCESS RATES OF ALL TRIALS IN THE REAL ENVIRONMENT

Test	Start x, y, yaw (m,m,rad), Region	Goal x, y (m,m), Region	Success %
1	[-0.18, 0.27, -3.14], 12	[-0.33, -1.86], 10	100
2	[-0.45, -1.9, 1.57], 10	[-0.16, 0.22], 12	100
3	[-2.56, -1.65, 0], 1	[-0.17, 2.47], 14	66.67
4	[-0.34, 2.62, 3.12], 14	[-2.28, -1.85], 1	100
5	[-2.22, 1.39, -1.46], 4	[1.1, -0.2], 16	66.67
6	[1.33, -0.27, 0.13], 16	[-2.3, 1.42], 4	100
7	[-1.85, 0.25, -0.12], 7	[-0.17, 2.47], 14	33.33
8	[-0.20, 2.77, 3.05], 14	[-1.3, 0.1], 7	100
9	[-1.45, 2.71, -1.76], 9	[-0.33, -1.86], 10	100
10	[-0.38, -2.06, 1.47], 10	[-1.2, 2.31], 9	100
11	[-1.74, 2.51, -1.23], 9	[1.99, -1.49], 19	100
12	[2.22, -1.67, 2.98], 19	[-1.2, 2.31], 9	100
13	[-0.45, -1.76, 0.14], 10	[1.99, -1.49], 19	100
14	[2.16, -1.48, 1.78], 19	[-0.33, -1.86], 10	100
15	[1.38, 0.35, -1.57], 16	[-0.33, -1.86], 10	100
16	[-0.45, -1.96, 1.88], 10	[1.1, 0.2], 16	100
17	[-0.39, 0.46, -0.11], 12	[1.99, -1.49], 19	100
18	[2.26, -1.49, 2.57], 19	[-0.16, 0.22], 12	100
19	[-0.25, 0.38, 1.55], 12	[1.66, 1.96], 22	100
20	[1.95, 2.11, 3.12], 22	[-0.16, 0.22], 12	0
Average Total Success Rate			**86.67%**

We also implemented the 20 pairs of locations in the simulated environment and compared the total average success rates. The simulation and real-world experiments had a success rate of 90.0% and 86.67%, respectively.

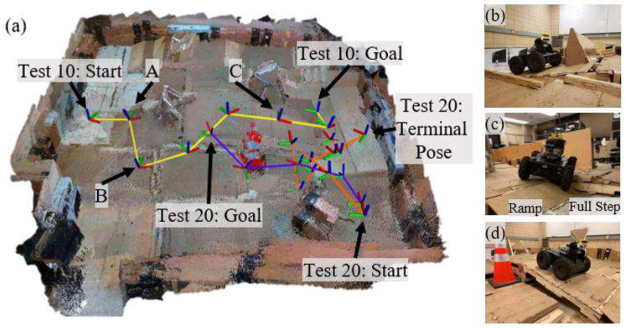

Figure 7. (a) 3D map of the environment with 3 different trajectories shown; Test 10 real trajectory (yellow), Test 20 real trajectory (orange), Test 20 simulated trajectory (purple). A, B and C are the zoomed in views in (b)-(d).

B. *Comparison Study*

We conducted a comparison study between our DRL method, a classical binary traversability method [10], and a deep learning-based terrain traversability classification method [13] within our 3D cluttered environment. The following performance metrics were used: 1) success rate, 2) cumulative travel distance to account for terrain and path variability, 3) cumulative travel time for successful trials, and 4) replanning rate. The same 20 location pairs were used in this experiment.

1) Comparison Methods

Classical Method: We choose a common heuristic rule-based method for comparison, e.g. [7], [8], [10]. This method computes a binary traversability value using a linear combination of height, slope, and roughness of the terrain, which is then compared against a set threshold.

Deep Learning (DL) Method: We choose the DL method in [13] as it incorporates the standard CNN architecture for terrain traversability estimation. The CNN architecture consists of a 60 px by 60 px image input layer, two consecutive 3×3 CONV layers, a 2×2 max-pooling layer, a 3×3 CONV layer, a 128 output FCL layer, a 2 output FCL, and a softmax output.

The output of both methods is a 2D traversability map. To ensure a fair comparison, the heuristic parameters in [10] and the hyperparameters in [13] were used. A* was used on the traversability maps to search for paths to the 20 goal locations. The robot executed the navigation plans for all three methods using the *Closed-Loop Controller* module.

2) Comparison Results

The performance metrics for all three methods are presented in Fig. 8. The classical method had the lowest success rate of 18%, followed by the DL method at 43%. Our DRL approach had the highest success rate of 87%. Amongst the three methods compared, our DRL method had the shortest cumulative distance traveled and travel time among the successful trials with a distance of 14.51m in 227s versus 15.26m in 379s for the DL method and 15.47m in 451s for the classical method, respectively. In general, both the classical and DL methods had lower performance than our DRL method as they use a multi-stage approach, while our DRL method uses a single stage approach that directly maps the observation space to an action output. As the multi-stage approaches first estimate traversability, then use a search algorithm to find a path, they are known to suffer from cascading errors [37], where errors from the traversability estimation are transferred to the path planning stage. Namely, since the traversability estimation stage produces a simplified representation of traversability (e.g., categorical label or scalar value), it does not consider the robot-terrain interactions that are present when navigating 3D rough terrain environments. However, our DRL method directly considers these interactions.

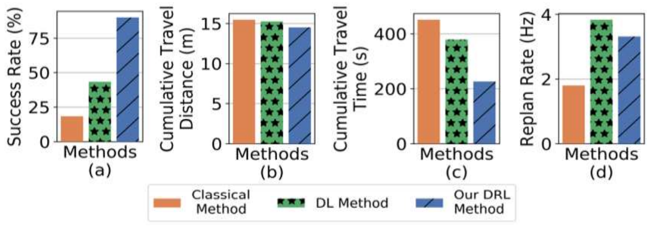

Figure 8. Performance metrics: (a) success rate, (b) cumulative travel distance, (c) cumulative travel time, and (d) path replanning rate.

Figure 9(a) shows the paths that the robot traveled for all three methods for Test 3 in Table I. The robot's path using the classical method abruptly ended at location A, Fig. 9(b). This was due to the method misidentifying traversable paths surrounding the robot as non-traversable in region A. Therefore, the robot was not able to determine a long-range traversable path to the goal location and became stuck. Using the DL method, the robot flipped over at location C due to it not being able to execute its planned path. Specifically, the robot's interactions with the terrain at location B did not result in its intended action due to an undesirable lateral shift that caused it to overshoot the rotation, Fig. 9(c). Subsequent motion errors also caused the robot to diverge further from the plan. Alternatively, the robot using our DRL method successfully reached its goal location. Our DRL method inherently accounts for the robot-terrain interactions during the learning process via its single stage end-to-end approach and the aforementioned reality gap strategies.

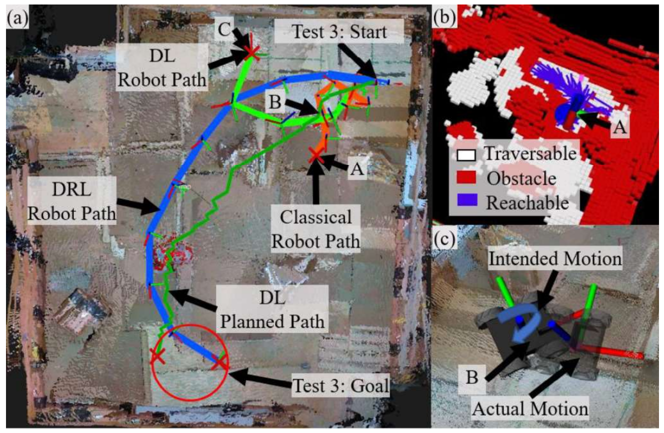

Figure 9. Test 3: (a) robot path using classical method (orange), DL method (light green), and DRL method (blue), and DL planned path (dark green), (b) Traversability map for classical method for region A, and (c) Robot-terrain interactions at region B.

The overall replanning rate of our method was 3.2Hz versus 1.8Hz for the classical method, and 3.8 Hz for the DL method. Our replanning rate was slightly slower than the DL method due to its requirement of larger input images (i.e., 200px by 200px vs 60px by 60px) and a larger network architecture with additional CONV and FLC layers (Fig. 2).

C. Sensitivity Analysis

We conducted a sensitivity analysis of the hyperparameters of the sim-to-real strategies in generalizing to increasingly complex terrain in simulation. We evaluated the performance of models that were trained 1) with all three strategies, and 2) in the absence of one of these strategies by increasing the hyperparameter values until they were 8 times the values in Section III.D.2, Table II. No external movement disturbance was applied to the robot during evaluation.

The results are presented in Fig. 10. The E+MD+P (elevation steepness + motion disturbance + pose estimation errors) model was trained with all three strategies. As seen from the figure, this model was able to consistently maximize the cumulative reward for each set of hyperparameter values. The E+MD and MD+P models performed worse under larger parameter variations, emphasizing the importance of both the E and P strategies in improving performance with respect to reward maximization, and generalizing to large terrain variations and robot pose estimation errors.

TABLE II. EVALUATION PARAMETERS

	BASELINE	**X2**	**X4**	**X8**
STEEPNESS	±5%	±10%	±20%	±40%
LATENCY	0~0.03m, 0°~3°	0~0.06m, 0°~6°	0~0.12m, 0°~12°	0~0.24m, 0°~24°
POSE ERROR	0.06m, 5°	0.12m, 10°	0.24m, 20°	0.48m, 40°

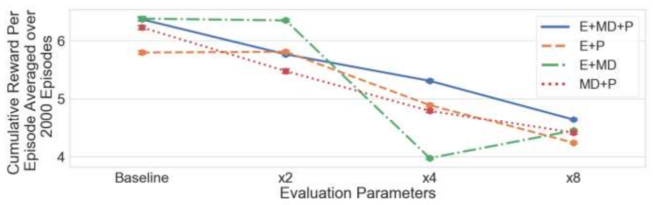

Figure 10. Sensitivity analysis of model performances. E: trained with elevation steepness, MD: trained with motion disturbance, and P: trained with pose estimation errors.

V. Conclusion

In this paper, we present the sim-to-real pipeline that teaches a robot to navigate 3D cluttered real-world terrains which can have abrupt changes in surface normals and elevations. Namely, we use a unique combination of sim-to-real strategies to allow for the transfer of the DRL navigation policies learned in simulation to be successfully deployed to the real-world environment without additional training in the real-world. Experiments with a mobile robot showed that the robot had a high success rate in navigating the 3D cluttered environment. Furthermore, a comparison study against a classical and a deep learning method verified that an end-to-end DRL approach performed better in terms of success rate as well as cumulative travel distance and time. DRL has the potential limitation of overfitting to a training environment. Thus, it can have difficulty generalizing to other real-world unstructured environments that contain dissimilar terrain from the training environment. To overcome this limitation would require: 1) representing the vast variability in terrain within the training stage, or 2) extending the CNN architecture to include techniques such as regularization and dropout layers. This is currently a part of our future work. Additionally, we will improve the robustness of the sim-to-real pipeline for the failure cases observed. We will also incorporate our navigation system with our previously developed exploration method [38], where the exploration method can provide goal locations to our proposed rough terrain navigation system.

References

[1] Y. Liu and G. Nejat, “Robotic urban search and rescue: A survey from the control perspective,” *J. Intell. Robot. Syst. Theory Appl.*, vol. 72, no. 2, pp. 147–165, 2013.

[2] Y. Liu and G. Nejat, “Multirobot Cooperative Learning for Semi-autonomous Control in Urban Search and Rescue Applications,” *J FIELD ROBOT*, 33(4) pp. 512–536, 2016.

[3] B. Doroodgar, Y. Liu, G. Nejat, “A Learning-Based Semi-Autonomous Controller for Robotic Exploration of Unknown Disaster Scenes While Searching for Victims,” *IEEE Trans. Cybern. Part B*, 44(12), 2014.

[4] M. K. Habib and I. Doroftei, *Using Robots in Hazardous Environments*. Elsevier, 2011.

[5] D. Silver, J. A. Bagnell, and A. Stentz, “Learning from demonstration for autonomous navigation in complex unstructured terrain,” *Int J ROBOT RES*, 29(12), pp. 1565–1592, Oct. 2010.

[6] R. B. Rusu *et al.*, “Leaving Flatland: Effcient real-time three-dimensional perception and motion planning,” *J. FIELD. Robot.*, 26(10), pp. 841–862, Oct. 2009.

[7] F. Colas, S. Mahesh, F. Pomerleau, M. Liu, and R. Siegwart, “3D path planning and execution for search and rescue ground robots,” in *IEEE Int. Conf. on Intell. Robots and Syst.*, 2013, pp. 722–727.

[8] R. Fedorenko, A. Gabdullin, and A. Fedorenko, “Global UGV Path Planning on Point Cloud Maps Created by UAV,” in *IEEE Int. Conf. on Intell. Transp. Eng.*, 2018, pp. 253–258.

[9] P. Krüsi, P. Furgale, M. Bosse, and R. Siegwart, “Driving on Point Clouds: Motion Planning, Trajectory Optimization, and Terrain Assessment in Generic Nonplanar Environments,” *J FIELD ROBOT*, 34(5), pp. 940–984, Aug. 2017.

[10] T. Shan, J. Wang, B. Englot, and K. Doherty, “Bayesian Generalized Kernel Inference for Terrain Traversability Mapping,” in *CoRL*, 2018.

[11] S. Zhou *et al.*, “Self-supervised learning to visually detect terrain surfaces for autonomous robots operating in forested terrain,” *J FIELD ROBOT*, 20, pp. 277–297.

[12] F. Schilling, X. Chen, J. Folkesson, and P. Jensfelt, “Geometric and visual terrain classification for autonomous mobile navigation,” in *IEEE Int. Conf. on Intell. Robots and Syst.*, 2017, pp. 2678–2684.

[13] R. O. Chavez-Garcia, J. Guzzi, L. M. Gambardella, and A. Giusti, “Learning Ground Traversability from Simulations,” *IEEE Robot. Autom. Lett.*, 3(3), pp. 1695–1702, 2018.

[14] K. Zhang, F. Niroui, M. Ficocelli, and G. Nejat, “Robot Navigation of Environments with Unknown Rough Terrain Using deep Reinforcement Learning,” in *IEEE Int. Symp. on Saf, Secur., and Rescue Robot.*, 2018, pp. 1–7.

[15] S. Josef and A. Degani, “Deep Reinforcement Learning for Safe Local Planning of a Ground Vehicle in Unknown Rough Terrain,” *IEEE Robot. Autom. Lett.*, 5(4), pp. 6748–6755, 2020.

[16] A. Nguyen *et al.* Tran, “Autonomous Navigation in Complex Environments with Deep Multimodal Fusion Network,” *arXiv Prepr. arXiv2007.15945*, 2020.

[17] J. Tan *et al.*, “Sim-to-Real: Learning Agile Locomotion For Quadruped Robots,” in *Robot: Sci and Syst*, 2018.

[18] F. Sadeghi and S. Levine, “CAD 2 RL: Real Single-Image Flight Without a Single Real Image,” in *Robot: Sci and Syst*, 2017.

[19] L. Pinto, J. Davidson, R. Sukthankar, A. Gupta, “Robust Adversarial Reinforcement Learning,” in *arXiv:1703.02702 [cs.LG]*, 2017.

[20] L. Tai, G. Paolo, and M. Liu, “Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation,” in *IEEE Int. Conf. on Intell Robots and Syst*, 2017, pp. 31–36.

[21] A. Francis *et al.*, “Long-Range Indoor Navigation with PRM-RL,” in *arXiv:1902.09458 [cs.RO]*, 2019, pp. 1–19.

[22] Z.-W. Hong *et al.*, “Virtual-to-Real: Learning to Control in Visual Semantic Segmentation,” in *Int. Joint Conf. on Artif. Intell.*, 2018.

[23] P. Papadakis, “Terrain traversability analysis methods for unmanned ground vehicles: A survey,” *Eng. Appl. Artif. Intell.*, 26(4), pp. 1373–1385, Apr. 2013.

[24] F. Endres *et al*, “An Evaluation of the RGB-D SLAM System,” in *IEEE Int. Conf. on Robot. and Automat.*, 2012, pp. 1691–1696.

[25] M. Labbé and F. Michaud, “RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation,” *J FIELD ROBOT*, 36, 2019.

[26] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia, “MeshLab: an Open-Source Mesh Processing Tool,” in *Eurographics Italian Chapter Conf.*, 2008, pp. 129–136.

[27] E. Fankhauser, M. Bloesch, and M. Hutter, “Probabilistic terrain mapping for mobile robots with uncertain localization,” *IEEE Robot. Autom. Lett.*, 3(4), pp. 3019–3026, 2018.

[28] “Gazebo,” 2019. [Online]. Available: http://gazebosim.org/. [Accessed: 12-Sep-2019].

[29] V. Mnih *et al.*, “Asynchronous Methods for Deep Reinforcement Learning,” in *Int. Conf. on Machine Learning*, 2016, pp. 2850–2869.

[30] E. Alpaydin, *Introduction to machine learning*. The MIT Press, 2010.

[31] A. Gruslys *et al.*, “The Reactor: A fast and sample-efficient Actor-Critic agent for Reinforcement Learning,” in *IEEE Int. Conf. on Robot. and Automat.*, 2018.

[32] M. Hausknecht and P. Stone, “Deep Recurrent Q-Learning for Partially Observable MDPs,” in *arXiv:1507.06527 [cs.LG]*, 2015.

[33] D. C. Herath, K. R. S. Kodagoda, and G. Dissanayake, “Stereo Vision Based SLAM Issues and Solutions,” in *Vision Systems: Applications*, I-Tech Education and Publishing, 2007.

[34] I. Flugge-Lotz, *Discontinuous Automatic Control*. Princeton: Princeton University Press, 1953.

[35] K. J. Åström and R. M. Murray, “Feedback Systems (10. PID Control),” in *Feedback Systems: An Introduction for Scientists and Engineers*, Princeton University Press, 2018.

[36] F. Solc, J. Sembera, “Kinetic Model of a Skid Steered Robot,” in *Int. Conf. on Signal Process., Robot. and Automat.*, 2008, pp. 1087–1097.

[37] M. Shah *et al.*, “LiRaNet: End-to-End Trajectory Prediction using Spatio-Temporal Radar Fusion,” *arXiv Prepr. arXiv2010.00731*, 2020.

[38] F. Nirouri, K. Zhang, Z. Kashino, and G. Nejat, “Deep Reinforcement Learning Robot for Search & Rescue Applications: Exploration in Unknown Cluttered Environment,” *IEEE Robot. Autom. Lett.*, 4(2), pp. 610–617, 2019.