Design of Decentralized Robot Architecture and Sim-to-Real Transfer Pipeline For Deep Reinforcement Learning Based Rough Terrain Navigation

by

Han Hu

A thesis submitted in conformity with the requirements
for the degree of Masters of Applied Science
Department of Department of Mechanical and Industrial Engineering
University of Toronto

Abstract

Design of Decentralized Robot Architecture and Sim-to-Real Transfer Pipeline For Deep Reinforcement Learning Based Rough Terrain Navigation

Han Hu
Masters of Applied Science
Department of Department of Mechanical and Industrial Engineering
University of Toronto
2021

Robots that autonomously navigate real-world 3D cluttered environments need to safely traverse terrain with abrupt changes in surface normal and elevation. This thesis presents a decentralized robot architecture for navigation and a novel sim-to-real pipeline for learning real-world navigation in simulation using deep reinforcement learning.

The decentralized robot architecture design avoids the requirement of a powerful central server and improves the robustness of the processes against hardware and software failures by isolating the computing units. A set of experiments were conducted to demonstrate the robustness of the architecture's mapping system against challenging environmental conditions.

The sim-to-real pipeline uses deep reinforcement learning to learn a navigation policy from data collected in simulation. It incorporated a combination of sim-to-real strategies to address the reality gap that uniquely exists for 3D navigation problems. A set of real-world experiments demonstrated that the pipeline successfully transferred the learned navigation policy into the real world.

Acknowledgements

I would like to express a special thanks to my supervisor Professor Goldie Nejat for her guidance and support throughout my masters degree. I would also like to thank my committee members for their valuable feedback and time. I would like to thank Dr. Silas Franco dos Reis Alves and Aaron Tan for all of their help and support throughout my master's degree that made me believed. I would like to thank my friends and colleagues from the Autonomous Systems and Biomechatronics Lab that worked on the project alongside me, Aaron Tan, Kaicheng Zhang, Michael Ruan, and Chris Agia for their hard work and support in conducting the experiments. I would like to thank Dr. Chris Mohammed for helping me define my thesis problem. I would also like to thank other members of the Autonomous Systems and Biomechatronics Lab, Mingyang Shao, Daniel Dworakowski, Gurtej Bhasin, and more for their friendly faces and support created an enjoyable work environment. I am extremely grateful to my parents, Ellie, and Elaine for their constant emotional support and encouragement. I could not have been the person today without them.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Autonmous navigation of a mobile robot in a real-world, cluttered, 3D rough terrain is a challenging issue with applications to Urban Search and Rescue (USAR) [1–3], hazardous material clean-up [4–6], mining [7, 8], and construction [9, 10]. In these applications, the terrain is often cluttered and consists of features that have abrupt changes in elevation and surface normals; uneven ground, steps, rocks of arbitrary shapes and sizes, and ramps. The complexity of the terrain poses a traversability challenge that existing 2D approaches do not consider. 2D environments, such as an office space, consist of obstacles with well-defined boundaries in the x-y plane; alls, desks, boxes, etc. In such environments, the terrain traversability at a location can be sufficiently represented as a binary value if an obstacle is present [11]. In contrast, the terrain traversability of a 3D environment may depend on numerous factors such as slope, object height, robot's velocity, and wheel positioning, etc. For instance, steps and road curbs can be traversable depending on their height relative to the robot. Slopes and ramps are only traversable from the front. Due to the complexity of the terrain traversability, paths planned using 2D approaches are not always feasible.

To date, most existing methods for navigating in 3D rough terrain focus on representing the terrain traversability of the environment. The computed traversability is then used to plan a path using a graph search algorithm as A* [12], Dijkstra [13], or Rapidly exploring Random Tree (RRT) [14]. The most common approaches in the literature are classical methods and learning-based methods. Most classical methods solely consider manually designed heuristics to map terrain features (e.g., height, slope, and roughness) into traversability representations [15–19]. However, manually designed heuristics cannot exhaustively describe the traversability of a complex 3D terrain. Furthermore, the manually selected input features often do not consider critical traversability factors, such as wheel terrain interaction, robot's intended action, and the robot's state [20]. Learning-based methods [21–28] focus on using machine learning models to learn the relationship between the input features and the traversability representation. The learning-based methods commonly use images and manually selected features as the input to learning the terrain traversability. However, many of these require a large amount of manually labeled training data, which can be a labor-intensive process. Both classical and learning-based methods produce a simplified traversability representation; binary [18,19,22], scalar value [15, 17, 24–26], probabilistic value [21, 27], or a categorical label [16, 23, 28]. Path planning based on these representations loses critical terrain information, such as robot-terrain interaction and wheel slippage effects. These information directly influences the robot's ability to successfully navigate through rough terrain.

The end-to-end approach is a promising method that avoids using handcrafted mapping heuristics and the simplification of the traversability representation. This approach trains a deep neural network policy to directly map an input to the desired output, such as pixels to robot motor command [29, 30] or a heightmap to a path [31]. Furthermore, end-to-end approaches can train in simulation. This avoids the reliance on a large amount of manually labeled training data, which can be expensive. The feasibility of using the end-to-end Deep Reinforcement Learning (DRL) approach for rough terrain navigation in simulation has been demonstrated by Zhang et al. [29] and Josef et al. [31]. However, this has yet to be achieved in the real world.

The challenge with applying DRL to the real world is that policies learned in simulation suffer from the reality gap. The reality gap is the performance degradation during inference in the real world due to the discrepancies between the simulated training environment and the real-world inference environment [32]. This is because the learned policy can overfit to domain-specific features during training; sensor noise, visual textures, model dynamics, etc. These discrepancies cause the inputs to the policy to be in a different distribution than the ones seen during training [33, 34].

Sim-to-real is an emerging research area for strategies to overcome the reality gap. The main categories of the strategies include domain randomization, domain-invariant inputs, and improving simulation fidelity. Successful applications of these have been shown in numerous robotic applications for aerial vehicle obstacle avoidance [35], quadruped robot local motion [32], object manipulation [36–38], and 2D navigation [39–42]. Sim-to-real strategies have yet to be applied to 3D rough terrain navigation. The 3D navigation problem has unique challenges unseen in 2D navigations, such as robot terrain interaction and wheel slippages.

1.2 Thesis Objective

The objective of this thesis is to develop a method for a wheeled robot to navigate in a cluttered 3D rough terrain.

The navigation capability of mobile robots in 3D rough terrain is desirable by applications such as USAR, hazardous material cleanup, mining, and construction. Existing classical and learning-based rough terrain navigation approaches are unsuitable due to the reliance on manually designed heuristics and overly simplified traversability representations. The end-to-end DRL based method showed feasibility for 3D navigation in simulation. However, these methods cannot be directly applied to the real world due to the reality gap. Sim-to-real strategies are effective in addressing the reality gap in numerous robotic applications, including 2D navigation.

1.3 Proposed Thesis Methodology

This thesis achieves aims to address the 3D rough terrain navigation problem through a sim-to-real pipeline that will learn and transfer the DRL policy from simulation to the real world. This thesis will present:

- The development of a hardware and software robot architecture for navigation in 3D rough terrain.
- A sim-to-real pipeline for learning and applying an end-to-end DRL based navigation policy to real-world 3D rough terrain navigation.

1.3.1 Literature Review

Chapter 2 provides a detailed review of the literature as a baseline understanding of existing work. This also provides support to the claims and decisions made in the development of the mobile robot platform, sim-to-real pipeline, and sim-to-real strategies. The proposed mobile robot architecture needs to operate in a cluttered challenging 3D environment. Core modules, such as navigation and control, requires stable pose estimation to function. Therefore, the pose estimation module must be robust against challenges such as motion blur, lighting, and texture, which exist in the terrain of the expected operating environment (i.e., USAR). This chapter starts with a review of the literature that compared the accuracy and robustness of the state-of-the-art SLAM algorithms. Section 2.2 presents a detailed review and analysis on classical, learning-based, and deep-learning-based approaches as the baseline understanding of the existing work and their shortcomings. End-to-end papers are presented as supporting evidence for the feasibility of the DRL-based rough terrain navigation approach that this thesis uses. Section 2.3 presents literature that used or developed sim-to-real strategies as the supporting evidence and inspiration on the strategies developed in this thesis.

1.3.2 Decentralized Robot Architecture Design and Integration

Chapter 3 details the hardware and software design of a decentralized ROS based architecture. The hardware architecture consists of multiple computing units, sensors, and network communication devices which are integrated into the Jaguar4×4 wheeled robot. The software architecture uses the ROS framework for its modular architecture, well-supported robotic packages, and popularity within the field of robotics.

The robustness of the SLAM module is critical in autonomous navigation. It provides the pose estimation and mapping information that the navigation and control modules require. Most existing literature evaluates the robustness of SLAM using popular, or inhouse datasets [43–45] that have rich visual features and good illumination levels. However, there is a lack of understanding of the SLAM's robustness under challenging environmental conditions; sparse features, poor illumination, textureless surface, and long-range depth estimation.

This thesis presents a set of mapping experiments conducted to evaluate the feasibility of SLAM algorithms under challenging environmental conditions.

1.3.3 Sim-to-Real Transfer of Deep Reinforcement Learning Policy for Rough Terrain Navigation

Chapter 4 presents the proposed sim-to-real pipeline. The pipeline addresses the reality gap by 1) improving simulation fidelity, 2) using domain invariant observations, and 3) applying domain randomization to training parameters. The pipeline first constructs a geometrically accurate model of the real-world for training in the Gazebo [46] simulator. The geometric model minimizes the terrain discrepancies between the real world and the training environment. The Asynchronous Advantage Actor-Critic (A3C) DRL algorithm [47] is used to train a Convolutional Neural Net (CNN) for navigation in the simulation terrain. This CNN takes in the robot's position and an elevation map centered around the robot as domain invariant observations. This avoids overfitting to features that are environment-specific, such as texture, color, and lighting condition [39, 40]. During training, domain randomization is applied to terrain steepness, motion disturbance, and robot pose estimation. Domain randomization is a popular sim-to-real strategy used by [32,35–37]. The randomization exposes the policy to a wide range of simulation parameter variations during training to reduce overfitting to a specific parameter value. This would have the real-world value to appear to the policy as one of the variations [32, 35]. The trained policy is deployed to the physical robot without additional training needed in the real world.

Real-world navigation experiments were conducted using the mobile robot integrated with the architecture developed in chapter 3. In these experiments, the robot navigates between a pair of randomly selected start and goal locations. There are a total of 20 pairs of points. Each pair is repeated 3 times for a total of 60 trials. A comparison study is conducted between a classical method and a deep reinforcement learning method in the simulation environment using the 20 location pairs. The performance is measured in terms of success rate, cumulative travel distance, cumulative travel time, and replanning rate. A simulation-based ablation study was conducted to evaluate the importance of the proposed domain randomization parameters in sim-to-real transfer. This study measured the cumulative reward of the policies trained in the absence of one of the domain randomization parameters. During each set of evaluations, the simulation environment doubled the randomized training parameters value until they were 8 times the values used during training.

1.3.4 Conclusion

Chapter 5 highlights the contribution of the thesis on 1) the development of a robot architecture for rough terrain navigation, 2) a set of experiments that compared the robustness of SLAM algorithms under challenging environmental conditions, 3) a novel sim-to-real pipeline for 3D DRL rough terrain navigation policy, 4) three unique domain randomization parameters that address the unique challenges in 3D rough terrain navigation, 5) a set of experiments that demonstrated the first successful application of real-world 3D rough terrain navigation using end-to-end DRL, 6) a set of comparison experiment that showed the superior performance of DRL against existing classical methods, and 7) an ablation test that demonstrated the importance of the proposed domain randomization in generalizing to larger terrain variations. Finally, future recommendations of the work are discussed, followed by a concluding statement.

Chapter 2

Literature Review

2.1 Simultaneous Localization and Mapping Algorithms

Ibragimov et al. [48] compared the performance of SLAM algorithms based on different sensors in an indoor environment. Both visual and LiDAR based SLAM algorithms were investigated; monocular camera based Oriented FAST and Rotated BRIEF (ORB-SLAM2) [49] and Dense Piecewise Planar Tracking and Mapping from a Monocular Sequence (DPPTAM) [50], RGB-D camera based RTAB-Map [51], stereo camera based ZED-SLAM [52], and 2D-LiDAR based Hector SLAM [53]. The testing office environment contained features that are challenging to visual SLAM, such as monochrome, homogeneously painted walls, and glass walls. Camera images and 2D LiDAR measurement data were recorded as the robot was teleoperated along a pre-determined path. The recorded data was later passed into the SLAM algorithms to build the trajectory. The experiment showed that the Hector SLAM had the lowest average deviation from the ground truth amongst all the SLAM algorithms. ZED-SLAM had the lowest deviation from all visual SLAM algorithms.

Ragot et al. [54] compared the performance of ORB-SLAM2 against RTAB-Map in both indoor and outdoor. The ground truth was validated with a Vicon motion capture system. Comparison between RTAB-Map with RGB-D, ORB-SLAM2 with RGB-D and ORB-SLAM2 with stereo demonstrated that the ORB-SLAM2 algorithm performed better in terms of measuring total distance traveled. Whereas the RTAB-Map algorithm performs better in terms of trajectory estimation.

Giubilato et al. [55] compared stereo camera-based SLAM algorithms and visual odometry algorithms in a hallway environment for a mobile robot. The testing environment contained challenging visual features such as low texture walls, lens flare due to direct sunlight, and high contrast regions. The SLAM algorithms compared were ORB-SLAM2, RTAB-Map, and SPTAM [56]. The visual odometry algorithms compared were LibVISO2 [57] and ZED visual odometry. The trajectory produced by the 2D LiDAR based Hector SLAM was used as the ground truth. The experiment showed that motion blur due to sharp turns, low texture walls, and over-exposure due to direct sunlight resulted in a lack of discriminative visual features. Only ORB-SLAM2 and ZED-SLAM were robust against this issue and was able to estimate the full trajectory.

Filipenko et al. [58] compared 11 ROS based SLAM algorithms. 3 of which are LiDAR based, 3 were stereo sensor-based, and 6 were are monocular camera-based. The environment was a well-lit, large, open concept office environment with chairs and desks. The experiment teleoperated the robot along a closed-loop rectangular path which served as the ground truth. The experiment showed that RTAB-Map had the smallest absolute RMSE of $0.163m$ but lost tracking at monochrome walls. ORB-SLAM2 was more stable, with a similar absolute RMSE of $0.190m$.

Silva et al. [59] compared the accuracy of 5 ROS based SLAM algorithms that are compatible with RGB-D cameras. Two of which were 2D LiDAR based algorithms that used the RGB-D camera to imitate a laser scan. One of the algorithms was monocular vision-based, and the other two algorithms were stereo vision-based. The experiments were conducted on two custom datasets of an office environment and three publicly available datasets. The experiment showed that RTAB-Map is the most efficient 3D mapping method. ORB-SLAM2 has good accuracy but can have loop detection errors that caused visible drift when there is a lack of visual features.

RTAB-Map [51] proposed by Labbé is a SLAM algorithm based on an incremental appearance-based loop closure detector. Since RTAB-Map is a loop closure based algorithm, it can take in different types of odometry inputs; visual, LiDAR, or wheel odometry, thereby providing support both to visual SLAM and LiDAR SLAM. Furthermore, RTAB-Map is well integrated into ROS. This versitility allows it to use various existing odometry approaches, such as RGBDSLAMv2 [60], which is robust against situations where the camera is obstructed, or there are not enough visual features to track [51].

ORB-SLAM2 [49] proposed by Mur-Artal is a visual-based SLAM algorithm that allows input from monocular, stereo, or RGB-D camera. This algorithm uses ORB features [61] for visual feature detection. Bundle adjustment is performed after loop closure to optimize the estimated camera frame position.

2.2 Rough Terrain Navigation Methods

Existing rough terrain navigation methods treat navigation as a multi-step problem. A representation of the environment is first extracted from sensory information, where the environment traversability is represented by a binary [18, 19, 22], scalar value [15, 17, 24–26], probabilistic value [21,27], or a categorical label [16,23,28]. A graph search algorithm is then applied to the environment to search for a path considering the traversability representation.

2.2.1 Classical Methods

In [15], path planning for a tracked, flipper robot in an indoor staircase environment was performed in a 3D grid map with D*-Lite [62]. A dense, 3D grid map environment representation was generated from sparse point cloud information using tensor voting. Tensor voting extracted fundamental geometries such as planes, lines, and spheres, as well as the saliency of the feature. D*-Lite was employed to search for a path using binary feasibility criteria and a traversability cost computed from the saliency, orientation, distance, and heading of the neighboring cells. Real-world experiments in a turning staircase showed that the robot could reach the goals; however, active control was needed for the robot flippers.

In [16], a planning system was developed which integrated different 2D planners for mobile robots navigating specific 3D terrains types. 3D polygon environment models were decomposed into 2D regions via a grid-based technique. The polygons were assigned semantic types by applying a handcrafted heuristic to their geometry. Different primitive planners were chosen to generate path trajectories based on the semantic type of the terrain in these regions. A* was used on unlabeled terrain. The single-query bidirectional sampling technique was used on flat ground. The robot was restricted to solely move in the direction of the slope on slopes and stairs. Simulated experiments showed that the planning system was able to generate 5m long trajectories in a varying terrain environment in less than 1 second. This time varies depending on factors such as terrain types and the obstacles in the terrain.

In [17], a global path planning method was presented that utilized: 1) initial trajectories generated on top of 3D point clouds via RRT, 2) its variant RRT for global optimization, and 3) fine-grained local optimization. Terrain traversability was measured as a weighted sum of robot orientation with respect to gravity and terrain roughness. The terrain roughness was evaluated by computing the residual distribution of a plane fitted to point cloud patches. Each local trajectory segment along the path was optimized by minimizing a handcrafted cost function which consisted of the weighted sum of steering curvature, traversability, and trajectory length. The least-cost trajectory was found using graph search with Dijkstra's shortest path algorithm. Waypoint navigation experiments in a sloping terrain and a two-story parking garage showed that the robot could navigate these environments with some human intervention needed in the first scenario.

In [18], RRT was used for ground vehicle navigation in conjunction with a heuristic-based traversability assessment approach. This method planned trajectories on a 3D point cloud map acquired by an aerial vehicle. Feasible trajectories were found by searching through the tree for a traversable path. The traversability was determined as a binary value using heuristic rules that measured the roughness, pitch, and roll of a plane fitted to point cloud patches. Navigation experiments on point clouds of different real-world terrain from the Sensefly elevation dataset showed that the method generated feasible trajectories in a mining quarry and a gravel pit type terrain.

Ruetz et al. [19] proposed a method for real-time 3D mesh representation of the environment using the General Hidden Point Removal Operator [63]. The proposed method first constructs a mesh from the local free space identified using the General Hidden Point Removal Operator. A binary traversability value is then computed for the mesh polygon by thresholding its geometric features such as surface angle, neighboring surface normal, and vertex height. Real-world experiments using paths planned with RRTconnect [64] with Reeds-Shepp path [65] showed that the proposed method allowed for long-range navigation in an urban environment.

2.2.2 Classical Learning-Based Methods

In [21], Sock et al. proposed a method of generating 2D traversability grid-map using multi-sensor probabilistic fusion. RGB color means, Lab color means, the entropy of intensity, and normalized positions acquired with a vision sensor were passed into a Support Vector Machine (SVM) to compute a traversability probability of the grid cells. Slope measured by a 3D LiDAR were mapped to a traversability probability using an empirically found exponential function. The probability value found by the vision sensor and the LiDAR were fused with the commonly used Bayesian principle [66]. Real-world experiments demonstrated that a robot autonomously traversed a trail environment using the proposed method.

Zhou et al. [22] proposed training a SVM visual classifier from a hand-labeled dataset for identifying ground regions in outdoor forested terrains. Textures and color acquired from a vision sensor and geometric features such as height and ray tracing score from LiDAR point clouds were used as input features to the SVM. The SVM classifier was retrained online using additional samples generated using flood filling and hole filling operations. Classification experiments in outdoor, forested environments showed that the retrained classifier had increased accuracy at identifying ground regions.

Happold et al. [23] proposed a method to classify types of terrain under insufficient range information. A shallow Multi-Layer Perceptron (MLP) classifies the terrain type of proximal grid cells based on geometric features such as height variation and slope. The classification data generated from the MLP classifier was then used to build a set of 3D histograms to learn the mapping from color to terrain geometry. The terrain geometry predicted from the MLP and the set of 3D histograms were combined into an overall 2D traversability grid-map. Classification experiments in an open environment with obstacles showed that the proposed method fully identified the obstacles at long range.

In [24], a multi-objective particle swarm optimization algorithm (MOPSO) was proposed for global path planning of a known rough terrain. LiDAR, RGB-D, and stereo cameras were used to obtain the contour map and the height map of the terrain, which were then discretized into 2D grids. From there, the minimum and maximum pixel values in each grid cell were used to indicate the terrain's roughness and presence of obstacles. Costs measure based on terrain roughness and trajectory length were used to evaluate the global trajectory found by each particle. The optimal trajectories were then found using the Pareto dominance method. MOPSO compared against NSGA-II, TV-MOPSO, and SPEA2, in a simulated environment demonstrated that MOPSO found feasible paths that were more Pareto optimal.

Silver et al. [25] proposed to use LEArning to seaRCH(LEARCH) for learning a non-linear cost function to map features into a scalar traversability cost in a 2D grid-map. The proposed method learned from manually annotated traversal costs maps and human demonstration of desired navigation behavior. Input features such as max object height and object density were attained from point clouds and vision sensors. Navigation experiments in rugged terrain compared the trajectories found from a manually annotated cost map and the learned cost map demonstrated that the learned cost resulted in more aggressive navigation behaviors.

Shan et al. [26] proposed to use Bayesian Generalized Kernel (BGK) for computing a dense 2D binary traversability grid-map under sparse point cloud data. The traversability score of each known cell is computed as a weighted linear combination of height, slope, and surface roughness [67]. The traversability of unknown cells was inferred using the known cells as training data. BGK regression is first applied to unknown cells to infer the elevation, then BGK classification is applied to the unknown cells using the inferred elevation as input. Experiments in a simulated urban environment showed that the proposed method produced a traversability grid-map for the full environment in real-time when 50% of the environment is directly observed.

2.2.3 Deep Learning-Based Methods

Chavez-Garcia et al. [27] proposed to use a CNN architecture trained from a synthesized height map dataset to output a probabilistic traversability graph. The CNN outputs a binary traversability classification for a grid cell given a terrain heightmap at a robot orientation. All traversability maps produced for each robot orientation are overlayed to attain the overall traversability probability. The grid cell forms the nodes of the traversability graph and the cell connectivity forms the edges. The training data was synthesized by directing a simulated robot with a random initial pose to move straight on a procedurally generated map. The terrain patch under the robot is labeled traversable if the robot could navigate through, otherwise, non-traversable. Classification experiments conducted on 3D point clouds of real-world terrains (e.g. mining quarry and town environments) demonstrated that the classifier performed well on real-world features, such as slopes, holes, and steps. A real-world experiment on sloped terrain showed that the robot successfully reached the target location when teleoperated along the maximum traversable path.

F. Schilling et al. [28] proposed using fused semantic segmentation and terrain geometry features as inputs to Random Forest (RF) for multi-class terrain category classification. Pixel-wise semantic segmentation features was attained using the VGG-16 [68] network architecture pre-trained on the Imagenet [69] dataset transferred onto the CityScapes dataset [70]. Geometric features such as height, slope, and roughness were calculated with point cloud plane fitting. The semantic and geometric features were fused via a point cloud projection onto the image plane. The RF classifier used the fused feature vectors to classify each pixel into safe, risky, or obstacle. The RF classifier was trained from a hand-labeled dataset obtained through a depth sensor and a LiDAR. Classification experiments using real images showed that the proposed method achieved a high AUC score. However, the real-world experiment was unsuccessful due to motion blur.

2.2.4 End-to-End Learning Based Methods

Kai et al. [29] proposed using the A3C [47] DRL algorithm to learn to navigate in an unknown 3D rough terrain in simulation with a CNN and Long Short-Term Memory (LSTM) architecture. The CNN network takes depth image, elevation map, and the robot's relative pose as inputs to output linear and angular velocity motion commands. A LSTM layer was used to capture the underlying state of the partially observable environment. A positive reward was given as the robot gets closer, or reaches the goal location. A negative reward was given when the robot reached an undesirable terminal state. Random environments with hills and valleys were generated using Gaussian noise. Navigation experiments in simulation with unknown terrain showed that the robot was able to perform point-to-point navigation.

Josef et al. [31] proposed navigation using Deep-Q Reinforcement Learning with Rainbow network architecture [30] and a self-attention module. The Rainbow network took different types of range sensor inputs to learn forward, left, and right motion commands in a simulated environment generated with gaussian noise. A zero-range sensor such as IMU was used to sense the robot's position to the goal. Immediate-range sensing was used to perform a one-step look ahead binary traversability estimation. Local-range sensing produced an elevation 3m by 3m sized elevation map. A round shooting target reward structure was employed, where the robot is positively rewarded every time it enters inner rings that encircled the goal position. Navigation experiments in simulation comparing the different sensor inputs showed that local-range with zero-range sensing input attained the highest success rate. During the experiment, the self-attention module was shown to focus on hazardous areas such as holes to adjust the robot position to avoid falling.

Nguyen et al. [71] proposed the Navigation Multimodal Fusion Network architecture that uses end-to-end supervised learning for mapping multimodal sensory information to steering commands. The proposed architecture fused 2D laser measurements, RGB images, and 3D point clouds using ResNet8 [72] and T-Net [73]. The training data was collected from a Gazebo [74] simulation by manually driving a robot in a manually constructed disaster scene. Domain randomization was applied to the environment textures during training to improve generalization. Experiments against state-of-the-art network architectures used in autonomous navigation showed that the proposed architecture with domain randomization had the lowest error. Further experiments comparing the different sensor inputs showed that the fused multimodal input achieved the lowest error value.

2.3 Sim-to-Real Strategies for Deep Reinforcement Learning

Sadeghi et al. [35] applied domain randomization to a simulated environment during DRL training to learn a real-world collision avoidance policy of a quadcopter. Visual parameters of the training environment (i.e., texture, lighting, and furniture placements) were highly randomizing during training such that the network learns representations that are invariant to surface appearances. Learning agents were trained in a simulated indoor environment constructed using Blender [75], which had a low degree of realism. The VGG-16 [68] network was used to learn a Q-function that mapped an RGB image to a discrete set of motion commands. Real-world collision avoidance experiments showed that the policy learn using only simulated images could generalize well to a variety of real-world structures, including the ones that were not present during training.

Peng et al. [36] used domain randomization to learn non-prehensile manipulation of a puck with a 7-DOF robotic arm. Eight parameters, such as link mass, controller gains, and puck mass, were randomized during training. A recurrent network trained using the Recurrent Deterministic Policy Gradient [76] algorithm and Hindsight Experience Replay [77] were used along with the randomized parameters to learn target joint angles in simulation from a 52D state space. The state-space included the arm's position and velocity, gripper velocity, as well as the puck's position, orientation, and velocities. Object manipulation experiments in the real world showed that a policy learned with an LSTM architecture can reach a 91% success rate and can cope with a variety of real-world dynamics.

Tan et al. [32] used a combination of domain randomization, high-fidelity simulator, and compact observation space to reduce the reality gap when learning the local motion controller policy of a quadruped robot. Physical parameters were randomized during training; mass, motor friction, and latency, etc. The randomization value range was found via system identification. Random force perturbations were also applied during training to learn to recover balance. A high-fidelity simulation environment was constructed with a fine-tuned Universal Robot Description File (URDF), accurate motor dynamics, and a controller signal latency model. The observation space was limited to the roll, pitch, and angular velocities of the robot and the motor angles to avoid using noisy sensor measurements that may drift. The local motion controller policy was formulated as a Partially Observable Markov Decision Process (POMDP) and solved using the policy gradient method. The policy and the value functions were represented by shallow MLP. Physical experiments conducted on a quadruped robot demonstrated that the learned gait patterns were able to move at the same speed with less power consumption when compared to manually engineered motor gait patterns. Comparison with policy without the sim-to-real strategies demonstrated that the policy learned using high-fidelity simulation and random perturbations can transfer to real life, while others cannot. Additionally, it was found that applying random force perturbations had a similar effect to domain randomization.

Tobin et al. [37] explored the hypothesis: sufficient variation in simulation allows models trained in simulation to be generalized to the real world without additional training in the real world. This hypothesis was tested in an object localization task, using a VGG-16 [68] network trained with stochastic gradient descent on the L2 loss between the estimated object position and the ground truth. Training data were from simulated randomized RGB images. All training samples were randomized in position, texture, camera orientation, added noise, etc. Experiments in the real world demonstrated that the trained locator had comparable performance against traditional techniques for object pose estimation. An ablation study showed that a large amount of training data and variability is important to successfully transfer the policy learned in simulation to the real world.

In [39], domain invariant features were used to learn linear and angular robot velocity commands for 2D navigation using Asynchronous Deep Deterministic Policy Gradient (ADDPG) network. A 10-dimensional vector consisting of evenly sampled 2D LiDAR measurements was used as the domain invariant observation input. This vector was appended with the previous navigation actions and relative target position for a 14-dimensional input vector to the ADDPG network. Real-world experiments were conducted in an office space setting with a differential drive robot. It showed that the learned policy allowed the robot to navigate longer travel distances using observation inputs that were more sparse when compared to the ROS Move Base methods [78].

Francis et al. [40] used 2D LiDAR measurements as domain invariant features to help transfer navigation and obstacle avoidance behaviors learned in simulation. A hierarchical kinodynamic planner was presented, which combined Probabilistic Road Map [79] and Automated Reinforcement Learning (AutoRL) [80] for long-range indoor navigation. AutoRL was used to perform hyperparameter optimization over a Deep Deterministic Policy Gradient (DDPG) [81] network to learn short-range point-to-point navigation and obstacle avoidance using a compact 2D LiDAR measurement as input. The learned agent guides the roadmap construction by connecting two points only if the agent consistently navigated between the two points without collision. Evaluations conducted in two different real-world environments on a differential drive robot demonstrated a high success rate with a dense roadmap.

Pan et al. [41], proposed Realistic Translation Network (RTN) to improve the simulation fidelity of reinforcement learning based road navigation for self-driving cars. The RTN transformed raw images from a simulation that had unrealistic texture and color into realistic images by first extracting semantic meanings of the image, then synthesize realistic images from the semantic means using a conditional Generative Adversarial Network (cGAN). SegNet [82] trained on the CityScapes [70] dataset was used for the semantic meaning extraction. cGAN was used to synthesize the realistic images that had the same semantic representation. The synthesized images were used as the input to A3C algorithm using a CNN based network to train a driving policy that gave the actions of turning left, right, and go straight. Navigation experiments on a real-world driving dataset showed that the learned model using the RTN network improved performance compared to models trained in the same environment by only using domain randomization or images from the simulator.

Hong et al. [42], proposed using semantic segmentation as domain invariant feature inputs to an A3C algorithm to learn a target following and obstacle avoidance policy. Semantic segmentation was performed using DeepLab [83] trained on the ADE20K [84] and ICNet [85] trained on the [70] dataset. The A3C algorithm used the semantic segmentation images as inputs to learning linear and angular robot speeds in a simulated environment. The real-world experiment showed that policy learned using the proposed method maintained the same level of performance as in simulation. Comparison study against other sim-to-real strategies in simulation demonstrated that the policy learned using semantic labels performed better against methods such as domain randomization, depth image input, and RGB image input.

Ruse et al. [38] used the progressive neural networks architecture [86] to transfer robotic arm controller policy learned in simulation to the real world. A task of locating objects with the manipulator was trained in simulation using an image as the input to the A3C algorithm and the progressive neural network. The learned policy was transferred to the real-world by retraining on the physical manipulator with new columns added to the progressive network and freezing the parameters learned in simulation. Real-world experiments comparing progressive nets against fine-tuned parameters showed that the progressive net method attained higher reward.

Chapter 3

Decentralized Robot Architecture Design and Integration

This chapter presents the hardware and software design of a decentralized ROS based architecture for a wheeled mobile platform.

A decentralized design avoids the need for a central server and allows computation processes to run in parallel on different machines. This feature reduces the hardware requirement of the onboard computing units. It also modularizes each computing unit such that the processes are isolated, thereby becomes robust against issues such as unstable connection to the central server or crashing of unrelated processes. This architecture adapted the ROS framework for its modular architecture, well-supported robotic packages, and popularity within the field of robotics research.

The architecture is integrated into a Jaguar4x4 wheeled mobile robot from Dr.Robot Inc. for autonomous navigation in 3D terrain. It also served as the experiment vehicle for the sim-to-real pipeline presented in chapter 4.

3.1 System Design

3.2 Hardware Architecture

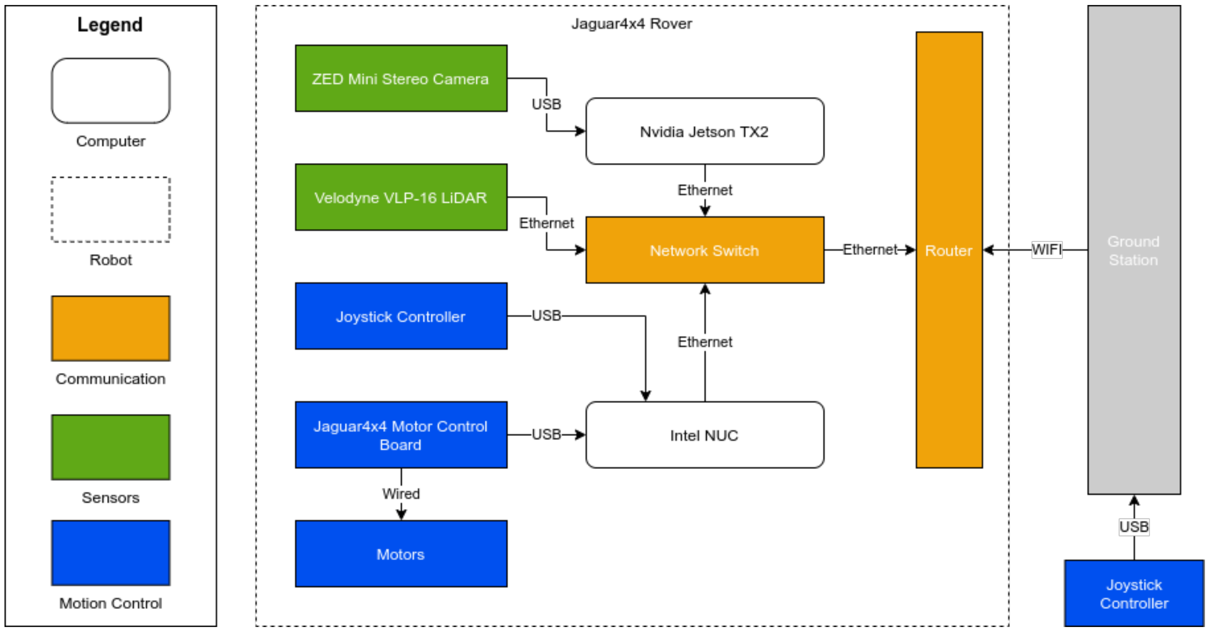


Figure 3.1: Hardware architecture diagram of Jaguar4×4.

The hardware architecture design is shown in Figure 3.1. It consists of multi-model sensors, computing units, and communication hardware. The robot shown in Figure 3.2 is the Jaguar4×4 robot integrated with the hardware architecture.

The multimodal sensors include the Stereo Labs ZED-Mini stereo camera with a built-in Inertial Measurement Unit (IMU) and Velodyne VLP-16 3D LiDAR. The ZED-Mini stereo camera is used to provide robust VIO odometry for pose estimation. The LiDAR is used to build an accurate spatial representation of the terrain surrounding the robot.

Two onboard computing units are used separately for CPU heavy tasks and GPU heavy tasks. In the hardware integrated into the robot, the Intel NUC with a Core i5 CPU and the Jetson TX2 were used. Both computing units use the Ubuntu 16.04 operating system and the ROS Kinetic framework for interprocess communication. Because of its higher processing capabilities, the Intel NUC is responsible for CPU-heavy computations;

pose estimation using RTAB-Map, robot motion control, map building, robot motion control, VLP-16 driver, and the mobile platform driver. The Jetson TX2 is used to provide processing capabilities for the ZED-Mini, as the camera requires Nvidia GPU Compute Unified Device Architecture (CUDA) cores.

The data transfer done between the computing units that are on the robot is achieved through a high bandwidth Network switch for high speed and reliable information transfer to maximize interprocess communication bandwidth.

A ground station computer for remote control and visualization is connected to the robot via Wi-Fi. This ground station can launch processes, teleoperate the robot, as well as visualize the data processing and collection by the robot's onboard computers.

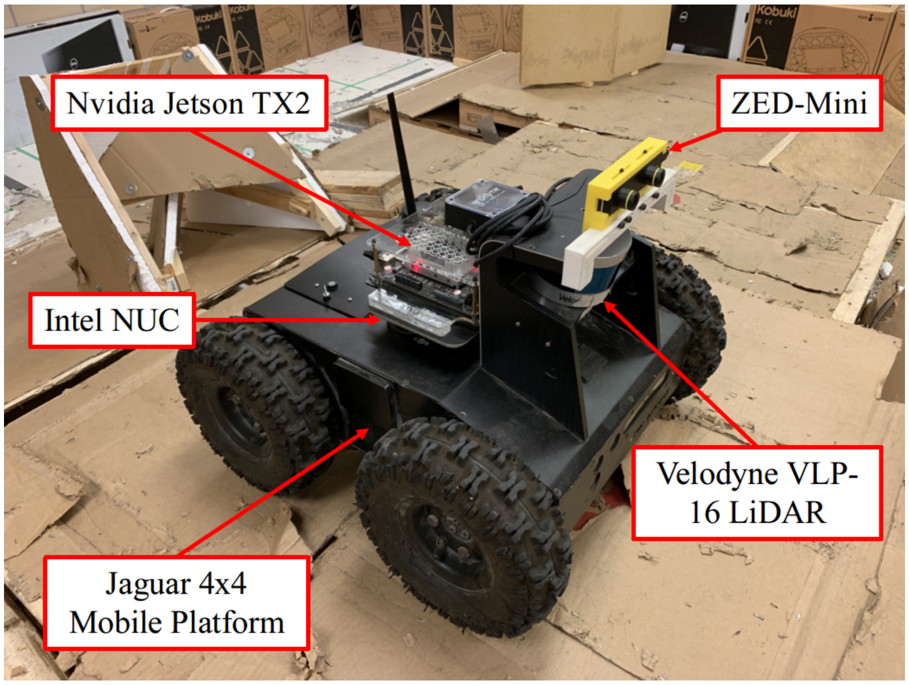


Figure 3.2: Jaguar4×4 physical appearance.

3.3 Software Architecture

The software architecture adapted the ROS framework because of its modular architecture, well-supported robotic packages, and popularity within the field of robotics research.

Figure 3.3 shows the software architecture integrated into the robot shown in Figure 3.2.

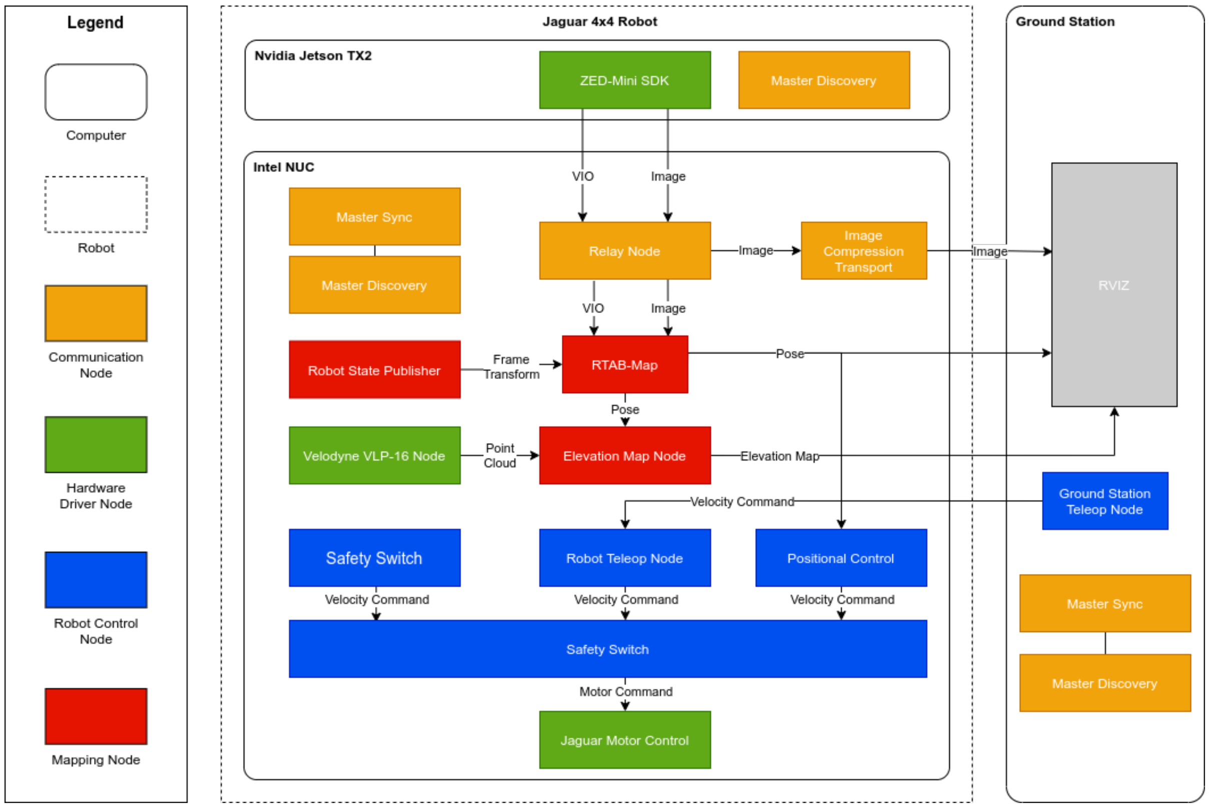


Figure 3.3: Software architecture diagram of Jaguar4×4.

The Jetson TX2 and Intel NUC computers are two different computing units that connect to a common network switch via ethernet. The ground system is used for interactions with the onboard computers such as launching nodes, sending navigation commands, visualization, and make code changes through a wireless network.

The interprocess communication between the Intel NUC, Jetson TX2, and the ground station is facilitated as a decentralized ROS Master system using the `multimaster_fkie` ROS package. The `multimaster_fkie` node allows each computer to bring up their dedicated node manager called ROS Master. Topics are synchronized from the ROS Master under the same network to transfer data between one another.

Compared to the traditional multi-machine setup [87], this decentralized computing design does not require a powerful central server as computation processes can run on different machines. This modularization allows the running processes to not depend on a stable connection to the central ROS Master that is running on another machine. For instance, in a multi-machine design, the SLAM algorithm that is running on the Jaguar4×4 can fail due to weak communication signals if the ROS Master is running on the ground station.

However, this design can be limited by the communication bandwidth. For instance, the quality of the sensor information may be limited when it is acquired on a separate computer than where SLAM is running. The ZED-Mini is capable of capturing a high definition image at a high frame rate. However, it can only be used at 640x480p at 30fps due to the bandwidth limit. Additionally, the wireless communication between the rover computers and the ground station does not allow large data to be transferred. A `relay_node` and a `image_compression_transport` node is used to alleviate this issue. It is important to use a high bandwidth network switch to avoid this bottleneck.

Master Discovery and Sync

Each of the Intel NUC, Jetson TX2, and the ground station has a `master_discovery` node. This node allows the ROS Master on other machines to be discovered. The `master_sync` node is running on the ground station and the Intel NUC, which is needed to subscribe to a topic published by nodes under another ROS core. This node synchronizes the topics specified in the parameter server to its ROS core.

Relay

The `relay` node [88] on the Intel NUC subscribes to all the messages from the Jetson TX2 and re-broadcast to other topics on Intel NUC to avoid redundant network traffic between the machines. This is needed because each subscription to a topic on Jetson TX2 from Intel NUC will result in a different data stream, thereby occupying extra bandwidth.

Image Compression Transport

The `image_compression_transport` node [89] compresses and transports the image seen by the sensors to the ground station for low latency visualization. The topics sent from Jetson TX2 are not compressed because compressing the image topic would result in a loss of image information, thereby causing performance degradation in SLAM algorithm.

Jaguar Motor Control

The `jaguar_motor_control` node is the Jaguar4×4 driver that interacts with the motor control board. This node converts velocity commands into motor control signals using a skid steer control model [90].

ZED SDK Wrapper

The `zed_sdk_wrapper` node [52] interacts with the driver software of the ZED-Mini sensor to publish information such as VIO, RGB images, depth images, etc to ROS topics.

Velodyne VLP-16

The `velodyne_vlp_16` node [91] interacts with the VLP-16 LiDAR to manage the hardware settings and publishes the 3D point cloud as a ROS topic. The 3D point cloud topic is subscribed by `rtabmap` node and the `elevation_map` node to build a height map of the surrounding environment.

RTAB-Map

The `rtabmap` node is the RTAB-Map algorithm [51] running on the Intel NUC. It takes the VIO calculated by ZED-SLAM as a prior and publishes pose corrections to the global pose of the rover based on the image stream sent over from the ZED-Mini on the Jetson TX2. This node must run on the Intel NUC because the Jetson TX2 lacks the CPU power to compute pose correction update at a high frequency. This node also can build a 3D reconstruction of the environment surrounding the robot for visualization.

Elevation Map

The `elevation_mapping` [92] [93] node is a ROS package designed for producing a robot centric elevation map that aggregates a series of measurements with consideration to motion uncertainty. In this software architecture design, this node takes in the pose estimated by RTAB-Map and the 3D point cloud scanned by VLP-16 sensor.

Robot State Publisher

The `robot_state_publisher` [94] node publishes the Jaguar4×4's transform (TF) frames, which is a series of frame transformation information that tells the rover where are the sensors and the body of the rover located relative to one another.

Twist Multiplexer

The `twist_multiplexer` [95] node is a prioirty-based multiplexer that multiplex multiple velocity command input into a sends a single velocity input to the `jaguar_motor_control`. The sources in decreasing priority order are `robot_teleop`, `ground_station_teleop`, `safety_switch` node, and `positional_pid_control` node.

Robot Teleop and Ground Station Teleop

The `robot_teleop` node running on the Intel NUC allows teleoperation control of the robot using a game controller connected to the Jaguar4×4 platfrom using a USB connection. Similarly, the `ground_station_teleop` node allows teleoperation of the Jaguar4×4 from the ground station. The `robot_teleop` node is given the highest priority and the `ground_station_teleop` is given the second highest priority such that the operator can always manually overide unsafe actions taken by the robot during autonomous navigation.

Safety Switch

The `safety_switch` node is designed to instantly stop any unsafe autonomous operations by requiring the operator to hold down a button on the joystick that is physically connected to the Jaguar4$\times$4 platform. This node sends a continuous signal to the `twist_multiplexer` to enable all controls other than manual teleoperation.

Positional Controller

The `positional_controller` node is a cascade controller shown in Figure 3.4 that controls the robot's position within the environment using the VIO computed by ZED-Mini. This cascade controller compensates for the wheel slippage that is present within the 3D rough terrain environment.

The inner controller loop uses a proportional [96] wheel-spin rate controller with encoders as feedback. Only the proportional term was used because no observable wheel oscillation was seen when the derivative and the integral term were set to 0 under the Ziegler-Nichols tuning method [97].

The outer loop uses a bang-bang controller [98] or a robot positional PID controller to output linear and angular robot velocity with a goal distance or yaw heading estimated by VIO as feedback. The linear and angular velocities are converted into individual wheel spin rates using the kinematic model of skid steer robot [90]. The bang-bang controller outputs a constant linear and angular robot velocity until the robot has passed the goal distance or yaw angle. The robot positional PID controller outputs a linear and angular robot velocity to track the set point goal distance and yaw heading. A low pass filter was applied to the noisy positional feedback signal estimated by ZED-Mini. A minimum effort is introduced to overcome the weight of the physical robot system, wheel slippage, and the motor's counter-electromotive force. The maximum effort is saturated to 45% of the maximum motor power for safety. The advantage of this controller is that it corrects the robot position errors caused by skid steer control on 2D terrain. However, its accuracy and reaction time is lower than the bang-bang controller on 3D terrain due to a combination of significant system lag, low-resolution encoder on the motors, and noisy position feedback. The system lag is between the positional controller node, motor control board, and the motors.

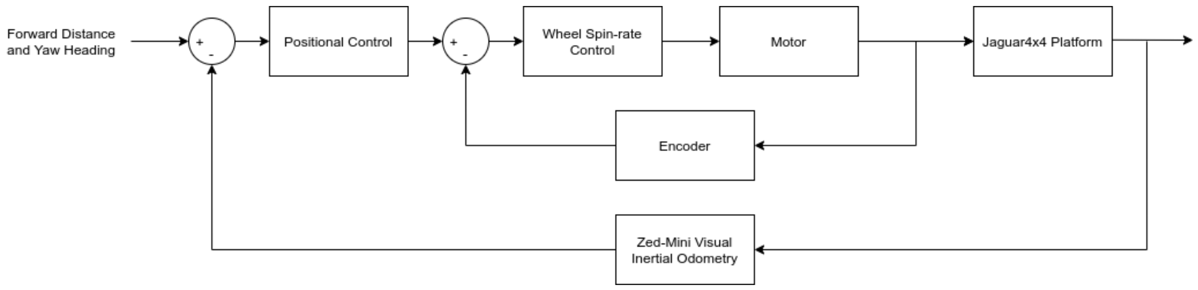


Figure 3.4: Jaguar4×4 cascasde positional controller block diagram.

RVIZ

The user interface in this software architecture for interacting with the Jaguar4×4 from the ground station is based on terminal and RVIZ [99]. Terminal commands are used to remotely launch ROS processes on the machines on Jaguar4×4 through Secure Shell Protocol (SSH). RVIZ is used for the visualization of data such as 3D point cloud, camera image, depth image, etc.

3.4 Experiments

The robustness of the SLAM module is critical given it provides the pose estimation and mapping information that the navigation and control modules require.

Existing literature [48, 54, 55, 58, 59, 100–102] evaluates the robustness of the ROS compatible SLAM algorithms by measuring the accuracy with public datasets [43–45] or in house datasets. These datasets have environmental conditions that are favorable for the SLAM algorithm, as they have rich visual features and good illumination conditions.

In SLAM, challenges such as poor illumination, textureless surfaces, and reflections can cause the algorithm to fail [103]. Both poor illumination conditions [104, 105] and textureless surfaces [106, 107] in the operating environment poses a challenge to SLAM algorithms due to a lack of visual keypoints. Surface reflection from glass surfaces can prevent RGB-D cameras and LiDAR from producing accurate depth estimation [108]. Additionally, empty environments such as a corridor or outdoor space can cause RGB-D cameras to fail due to range limitations [23]. Moreover, in cluttered rough terrain, robot-terrain interactions such as slippage, sudden drops, and uneven terrain can result in motion disturbance in the form of irregular vibration, sudden movements, or collisions to the robot chassis. The motion disturbances cause motion blur to the perception sensors that are rigidly linked to the robot chassis. It can severely degrade the accuracy of pose estimation [109–111]. The SLAM module and the sensor design must be robust against these challenges that are present in the environment and providing continuous pose estimation information.

There is a lack of literature on the robustness of the SLAM algorithm under challenging environmental conditions such as poor illumination conditions, sparse visual keypoints, long-range depth estimation, and motion disturbance. This experiment presents an analysis on the robustness of the SLAM algorithms under these challenging environmental conditions.

3.4.1 Experiment Setup

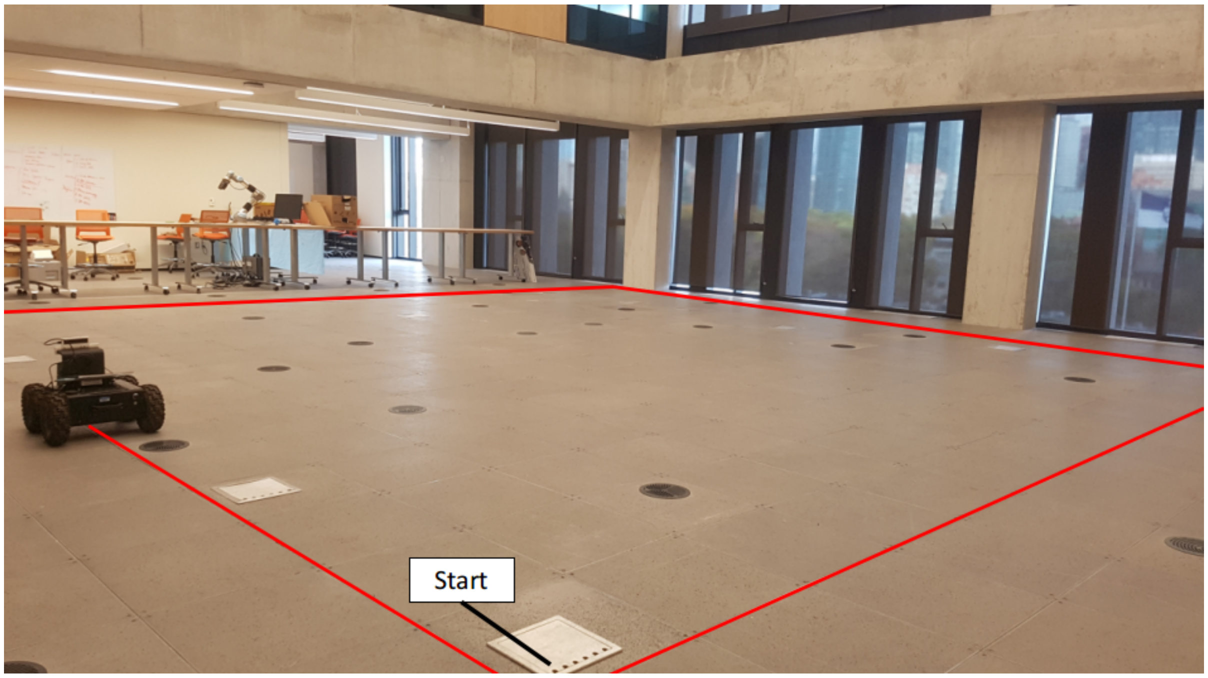
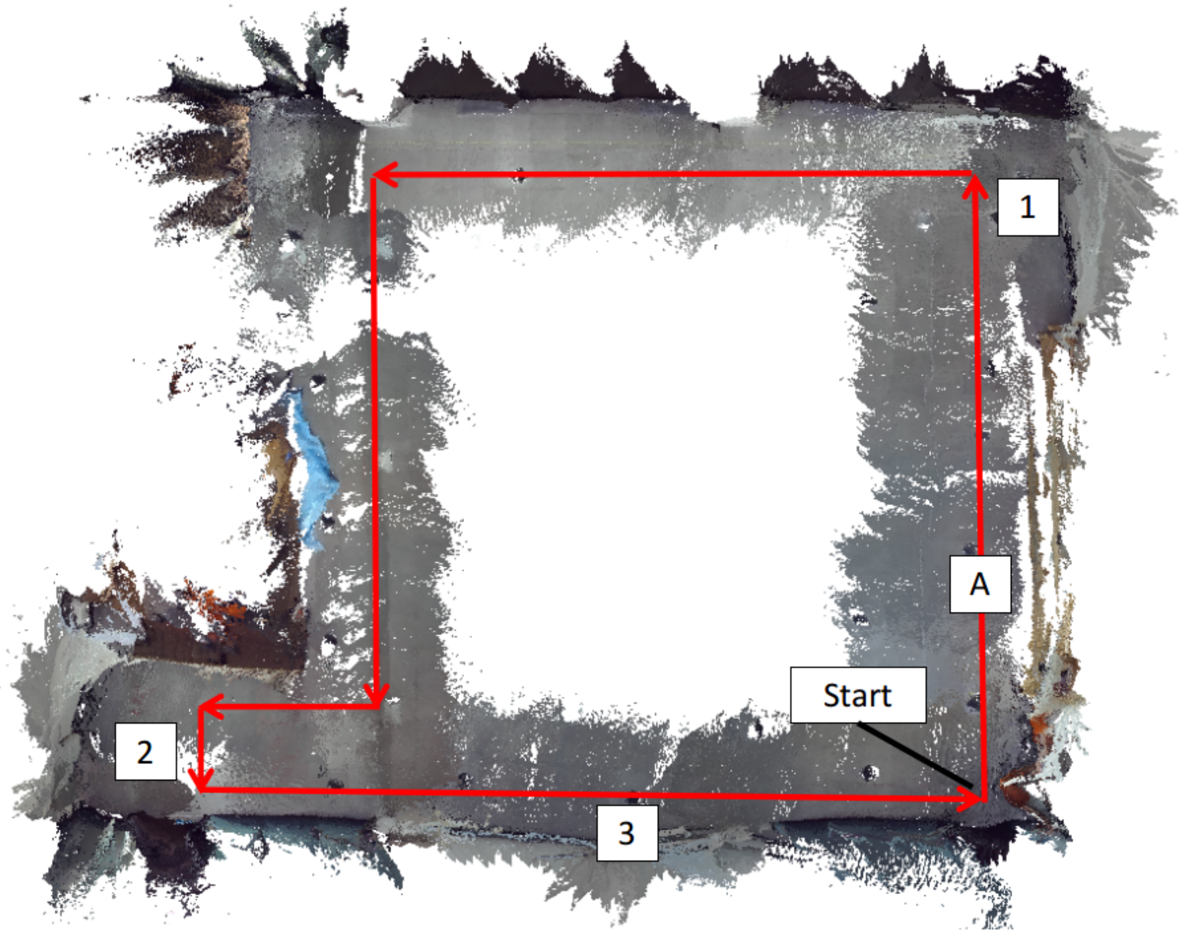


Figure 3.5: SLAM algorithm robustness comparison experiment environment. Red lines marks the pre-determined path followed by the robot during the experiment.

This experiment tests the robustness of SLAM algorithms under challenges of motion disturbance, poor illumination conditions, sparse visual key points, and long-range depth estimation. The experiment environment is the open space shown in Figure 3.5, where a robot followed a 10m $\times$ 10m square path shown by the red lines in the figure. The robot was manually moved along the path shown in Figure 3.6a and recorded the data collected by the sensors. The collected data was then passed to the SLAM algorithms to evaluate the quality of the map.

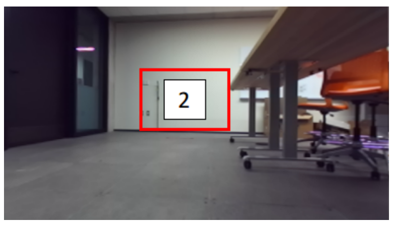
Motion disturbance was introduced in the form of consistent vibrations to the platform. This was done by adjusting the wheels of the platform such that it was slightly off-axis.

The open environment provided the sparse keypoints challenge as there were little to no objects along most of the experiment path. This also challenged the depth estimation ability of the sensors as it requires long-range depth sensing. There were three locations

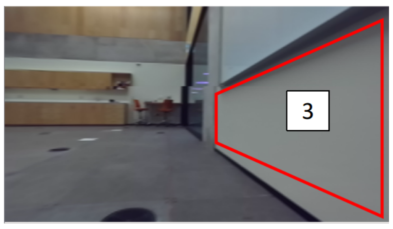


(a) Experiment environment top view. Red lines marks the pre-determined path followed by the robot during the experiment.

(b) Featureless surface at 1.



(c) Featureless surface at 2.



(d) Featureless surface at 3.

Figure 3.6: Featureless surface present in the experiment environment labeled in the order that the robot visits. See Table 3.3 for reference to location A.

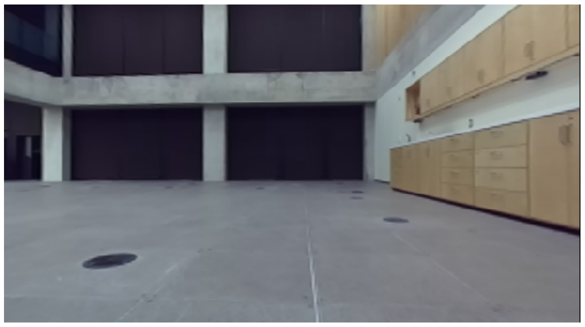
along the track shown in figs. 3.6b to 3.6d. At each location, the robot was rotated into the wall such that its entire field of view was covered by a blank wall.

The illumination condition of the experiment was tested under dim and dark illumination conditions. The dim condition has an average light intensity of 75 lux. It

Table 3.1: Specifications of sensors tested in SLAM algorithm robustness experiment

Sensor	Type	Effective Range	Resolution	FPS	IMU
Orbbec Astra [113]	RGB-D	0.6-8m	852 × 480	30	no
ZED Mini [114]	stereo	0.15-12m	3840 × 2160	60	yes
Intel-D435 [115]	active stereo	0.2-10m	1280 × 720	60	no

is comparable to a regular office space [112]. The dark condition has an average light intensity of 5 lux. Figure 3.7 shows the light intensity comparison.


(a) Dim illumination with lux=75.

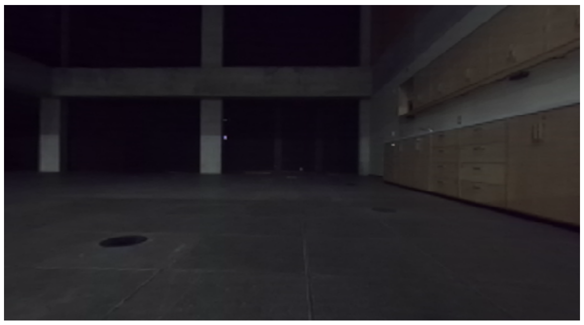

(b) Dark illumination with lux=5.

Figure 3.7: Experiment environment illumination conditions. Image taken by ZED-Mini camera.

3.4.2 Sensors

Three representative vision sensors used within mobile robotics were tested; the Orbbec Astra RGB-D sensor, Zed Mini stereo sensor with built-in IMU, and Intel-D435 active stereo sensor.

The Orbbec Astra sensor is an RGB-D sensor. It is a monocular camera combined with active depth sensing. RGB-D sensors use a laser array to project a series of dots in front and match the dots to its monocular camera image to attain depth information [116]. It has the advantage of measuring accurate depth information without requiring visual texture. However, it has the drawback that it cannot operate outdoors due to interference from natural lighting and range limit due to the limited laser power.

The ZED-Mini from Stereo Labs is a stereo sensor with a built-in 6-Degrees of Freedom

(DOF) IMU and low-light sensitivity camera sensor. A stereo vision camera uses two cameras with a fixed distance between one another. Instead of using a laser to measure the depth, it calculates the depth from the pixel position difference seen from the left and right camera [117]. It has the advantage that it is not limited to indoors and not limited by the laser power and density of the laser array. However, the depth estimation relies on visual texture, and it is less accurate than an RBG-D camera due to errors when calculating depth information.

The Intel-D435 camera is an active stereo camera, which uses a laser array projector to assist in stereo-based depth estimation [115]. The laser array projector is activated where lighting condition permits, such as in indoor environments. Stereo-only depth estimation is used in environments where the laser array cannot operate due to lighting interference. This camera produces a more accurate depth estimation than a stereo camera in indoor environments and can work outdoors. However, its depth estimation is less accurate than an RGB-D camera.

An additional difference between the ZED-Mini and the other two sensors is its 6-DOF IMU. This IMU provides linear and angular acceleration information of the sensor, which can calculate linear and angular velocity information through integration. Having an IMU as one of the inputs to the SLAM means that the SLAM would be using VIO. Without IMU, the SLAM would be using visual odometry (VO). The difference between VIO and VO is that VIO fuses the inertial information with the visual information when calculating odometry, whereas VO only uses the visual information. Incorporating VIO into pose estimation can improve the robustness by estimating gravity, velocity, and rotational rates [118].

Each of the sensors is different in terms of its recording capability. The ZED-Mini camera has the highest recording capabilities, equip with a low-light sensitive sensor, and can take up to 4K resolution videos [114]. Intel-D435 can output 720p with depth [115] and Orrbec Astra are limited limited to 480p at 30fps [113]. A summary of sensor specification is shown in Table 3.1.

3.4.3 SLAM Algorithms

Table 3.2: Characteristics of ROS compatible SLAM algorithms in robustness experiment.

SLAM	**Supported Sensor**				**Supported Odom**	
	Stereo	RGB-D	LiDAR	IMU	VIO	VO
RTAB-Map [51]	✓	✓	✓	✓	✓	✓
ORB-SLAM2 [49]	✓	✓				✓
ZED-SLAM [52]	✓				✓	✓

The three different SLAM algorithms compared in this experiment are shown in Table 3.2; RTAB-Map [51], ORB-SLAM2 [49], and [52] (ZED-SLAM). These algorithms are chosen for their low error compared to other ROS based SLAM algorithms [48, 54, 55, 58, 59]. RTAB-Map and ORB-SLAM2 are two open source SLAM algorithms that were highly ranked on the Kitti Vision Benchmark Suite [119]. ZED-SLAM is the close-sourced, built-in SLAM algorithm of the ZED-Mini camera. Experiments conducted by [48, 55] have shown that the trajectory estimated by these three SLAM packages have the least error to the ground truth.

RTAB-Map is a loop-closure based SLAM algorithm, therefore it can accept external odometry. In this experiment, it is tested with the VIO odometry provided by the ZED-Mini and VO computed from RGB-D SLAM [60]. ORB-SLAM2 computes its visual odometry using the image inputs. The ZED-SLAM is a close-sourced algorithm that is ran using the onboard computing from the ZED-Mini camera. The most similar image resolution available from each camera were used to minimize the performance difference due to resolution.

3.4.4 Results

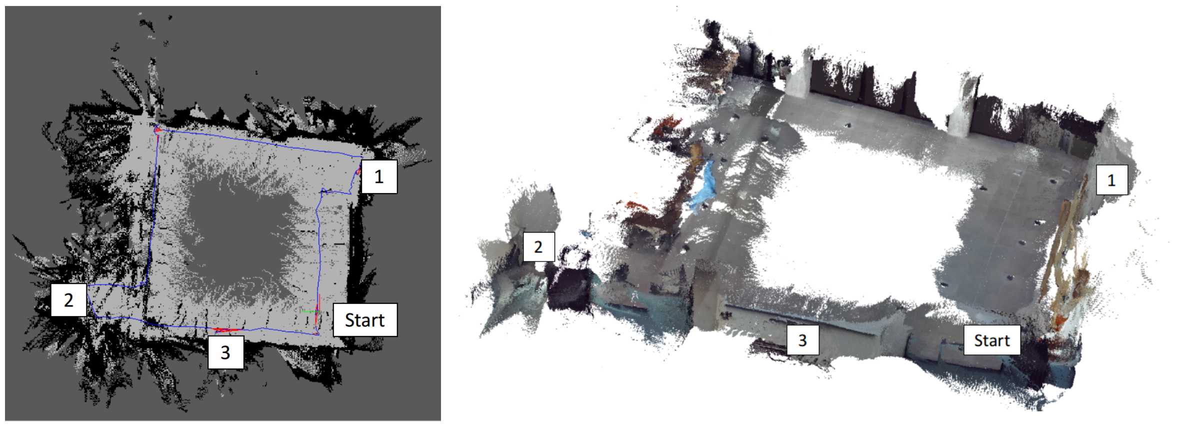
Table 3.3 shows the experiment results. Only RTAB-Map in combination with ZED-Mini provided VIO odometry completed the entire track under both dim and dark illumination levels. For all the VO experiments, the camera sensors and the SLAM algorithms were limited by a lack of visual features and limited range.

This experiment showed that SLAM algorithms with robust loop-closure (i.e. RTAB-Map) using VIO and stereo sensor (i.e. ZED-Mini) is the most robust against challenging environmental conditions; motion disturbance, poor illumiantion, sparse visual features, and long-range depth estimation. Such environmental challenges are common for applications that require 3D rough terrain navigation; USAR, hazardous material clean up, construction, and mining. Stereo vision sensor is more robust than RGB-D sensors in these environments because its depth estimation is not limited by the power of the IR projector, therefore can operate in an open field. VIO can maintain odometry estimation in conditions where vision odometry cannot be estimated. As such, vision sensors with an IMU are less prone to losing odometry under poor lighting conditions.

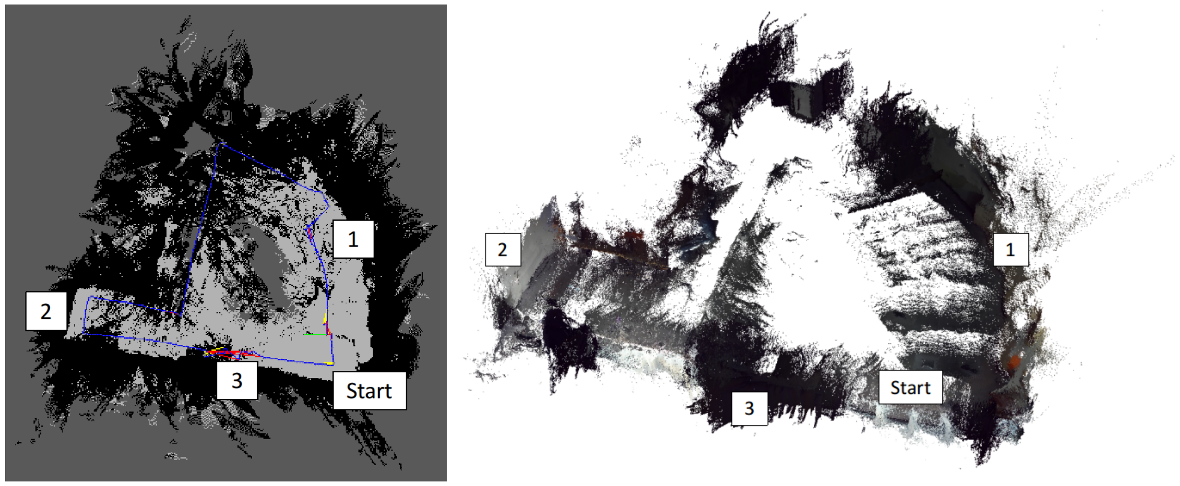
The RTAB-Map SLAM is more robust compared to the ORB-SLAM2 and ZED-SLAM algorithm. RTAB-Map can use VIO as input, where as ORB-SLAM2 cannot. Additionally, RTAB-Map is less prone to losing odometry trajectory compared to ORB-SLAM2 when both are using VO. RTAB-Map can also produce a more accurate map than ZED-SLAM because it implemented Versatile extensions for robust inference using graph optimization (Vertigo) to prevent loopclosure errors caused by objects with similar visual features at different locations. Furthermore, RTAB-Map is more versatile than both ORB-SLAM2 and ZED-SLAM because it can uniquely use externally computed odometry as well as be used with a LiDAR, where the other tested SLAM algorithms cannot.

Table 3.3: SLAM algorithm robustness experiments result. See Figure 3.6a for location.

Algorithm	Light	Sensor	Odometry	Resolution	FPS	Result
RTAB-Map	dim/75 lux	Intel-D435	VO	640 × 480	30	failed at 1
RTAB-Map	dark/5 lux	Intel-D435	VO	640 × 480	30	failed at A
RTAB-Map	dim/75 lux	Orbbec Astra	VO	852 × 480	30	failed at start
RTAB-Map	dark/5 lux	Orbbec Astra	VO	852 × 480	30	failed at start
RTAB-Map	dim/75 lux	ZED-Mini	VO	672 × 376	30	failed at 1
RTAB-Map	dark/5 lux	ZED-Mini	VO	672 × 376	30	failed at 1
RTAB-Map	dim/75 lux	ZED-Mini	VIO	672 × 376	30	completed
RTAB-Map	dark/5 lux	ZED-Mini	VIO	672 × 376	30	completed
ORB-SLAM2	dim/75 lux	Intel-D435	VO	640 × 480	30	failed at A
ORB-SLAM2	dark/5 lux	Intel-D435	VO	640 × 480	30	failed at start
ORB-SLAM2	dim/75 lux	Orbbec Astra	VO	852 × 480	30	failed at start
ORB-SLAM2	dark/5 lux	Orbbec Astra	VO	852 × 480	30	failed at start
ORB-SLAM2	dim/75 lux	ZED-Mini	VO	672 × 376	30	failed at 1
ORB-SLAM2	dark/5 lux	ZED-Mini	VO	672 × 376	30	failed at start
ZED-SLAM	dim/75 lux	ZED-Mini	VIO	672 × 376	30	failed loopclosure
ZED-SLAM	dark/5 lux	ZED-Mini	VIO	672 × 376	30	failed loopclosure



(a) Dim illumiantion condition result. Lux = 75. Left: 2D graph-view map. Right: denoised 3D point cloud.



(b) Dark illumiantion condition result, lux = 5. Left: 2D graph-view map. Right: denoised 3D point cloud.

Figure 3.8: Completed experiment results, RTAB-Map with external VIO from ZED-Mini.

The mapping result of RTAB-Map in combination with VIO from ZED-Mini is shown in Figure 3.8. The dim illumiantion condition produced the map shown in Figure 3.8a, which is similar to the actual layout of the experiment environment. The dark illumination experiment resulted in a distorted map shown in Figure 3.8b. This distortion was due to the dark illumination condition resulting in a lack of visual odometry estimation from the ZED-Mini camera. However, RTAB-Map did not lose tracking because ZED-Mini provided IMU-based odometry.

ORB-SLAM2 failed to initialize in the dark illumination experiment settings for all three sensors. It successfully initialized for the dim illumination settings when using Intel-D435 and ZED-Mini. However, it quickly lost odometry tracking when the input image had sparse features as seen in Figure 3.6. The dim illumination experiment with the Orbbec Astra camera failed on initialization because the camera's depth measurement range is too short for the experiment environment.

The RTAB-Map experiments that used Orbbec Astra camera and Intel-D435 to compute VO using RGBD-SLAM failed to compute odometry due to a lack of features for both dim and dark environments. It was only able to complete the full experiment path when using the external VIO provided by the ZED-Mini.

The ZED-SLAM maintained continuous odometry tracking during the experiment. However, its loop closure method was not robust to similar objects located at different locations, therefore resulted in faulty loop closure detections and added significant errors to the generated map [120]. RTAB-Map implemented Vertigo [121] in consideration of this error to increase robustness to graph optimization.

The initialization failure and losing odometry tracking for ORB-SLAM2 and RTAB-Map with VO was caused by the poor illumination and lack of visual features. Both were required for computing the visual odometry.

Experiment with a dark illumination level of 5 lux resulted in images that were too dark or over-saturated as seen in Figure 3.9. This degraded the performance of the

SLAM algorithm [104, 105]. The camera drivers of the Intel-D435 and the ZED-Mini camera have a built-in driver that auto-compensates the exposure and saturation level to account for the low light environment. This feature resulted in noisy images for the Intel-D435 as seen in Figure 3.9b and Figure 3.9c. SLAM with noisy image input degrades in performance due to increased error in feature matching between frames [122]. The ZED-Mini camera's image was less noisy as seen in Figure 3.9c as it was equipt with a high sensitivity low-light camera sensor. This allowed the VO based experiments with ZED-Mini to have better performance than that with Intel-D435 as shown in Table 3.3 and Figure 3.6a.

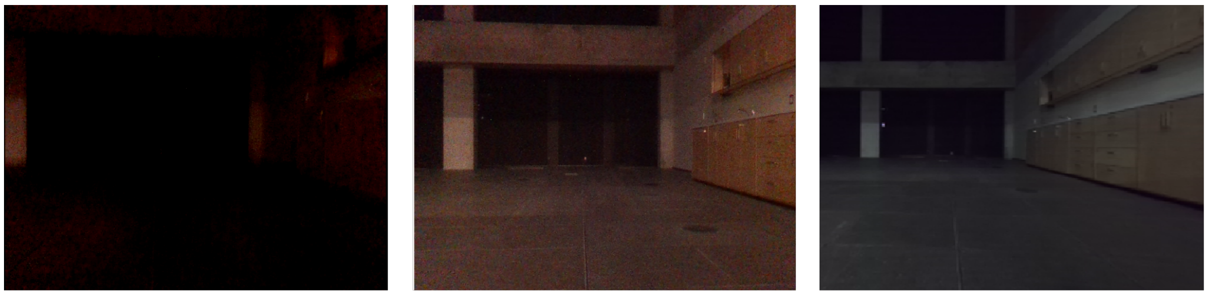

(a) Orbbec Astra (b) Intel-D435 (c) ZED-Mini

Figure 3.9: Camera sensor image from dark illumination experiment. Lux=5.

The monochrome colored surfaces in the experiment environment, such as uniform colored walls and floor, caused the SLAM algorithm to fail to compute odometry due to their lack of visual features [106, 107].

The effective range of depth estimation for each sensor shown in Table 3.1 is one of the causes of loss of odometry. The range of the Orbbec Astra RGB-D camera is limited by the power of the laser array. Objects outside of the effective range are unregistered. For the ZED-Mini stereo camera, the range is limited by the distance between the left and right camera. The estimated depth value of objects is less accurate for objects further away. Both limitations apply to Intel-D435 as it is an active stereo camera. These causes inaccurate odometry computation for the SLAM algorithms as they rely on depth information to compute VO. This is further shown by Table 3.3 and Figure 3.6a, where experiments with Intel-D435 failed earlier than ZED-Mini due to Intel-D435 has a shorter effective depth range. During the experiment, the Orbbec Astra camera always failed on initialization because it faces an open field that does not have any object within its effective depth range.

3.5 Chapter Summary

This chapter presented the hardware and software design of a decentralized ROS based architecture for a wheeled mobile platform. The architecture was integrated into the Jaguar4×4 mobile wheeled platform to enable autonomous navigation capabilities for 3D rough terrain navigation.

The detailed design of the decentralized architecture is presented in Section 3.1. The architecture facilitated communication between multiple computing units within the Jaguar4×4 and a ground station computer. The Jetson TX2 is used for Graphic Processing Unit (GPU) intensive opeartions; acquiring RGB-D data using a ZED-Mini stereo camera with integrated IMU. The Intel NUC onboard computer is used for CPU intensive operations; acquiring LiDAR data using a VLP-16 3D LiDAR, 3D mapping and localization using RTAB-Map, elevation map generation, motor control, and broadcasting data to the ground station for visualization.

Section 3.4 presented the 3D mapping experiment conducted to determine the combination of sensor and SLAM algorithm that is robust against environmental challenges. The experiment environment had features that are challenging to vision sensors and SLAM algorithms; poor illumination conditions, sparse visual key points, motion blur, and long-range depth estimation. The experiment demonstrated that the RTAB-Map using VIO provided by the ZED-Mini was able to complete the mapping of the experiment environment. The other tested combinations lost odometry tracking during the experiment due to environmental challenges.

Chapter 4

Sim-to-Real Transfer of Deep Reinforcement Learning Policy for Rough Terrain Navigation

This section presents the pipeline shown in Figure 4.1 for transferring rough terrain navigation policy from the simulation into the real world. The pipeline was developed in collaboration with Kaicheng Zhang. My specific contributions are detailed in section 4.1

The proposed pipeline is built using ROS as the framework. It allows DRL based navigation policy learned from the simulation to be deployed into the real world such that no additional training is required in the real world.

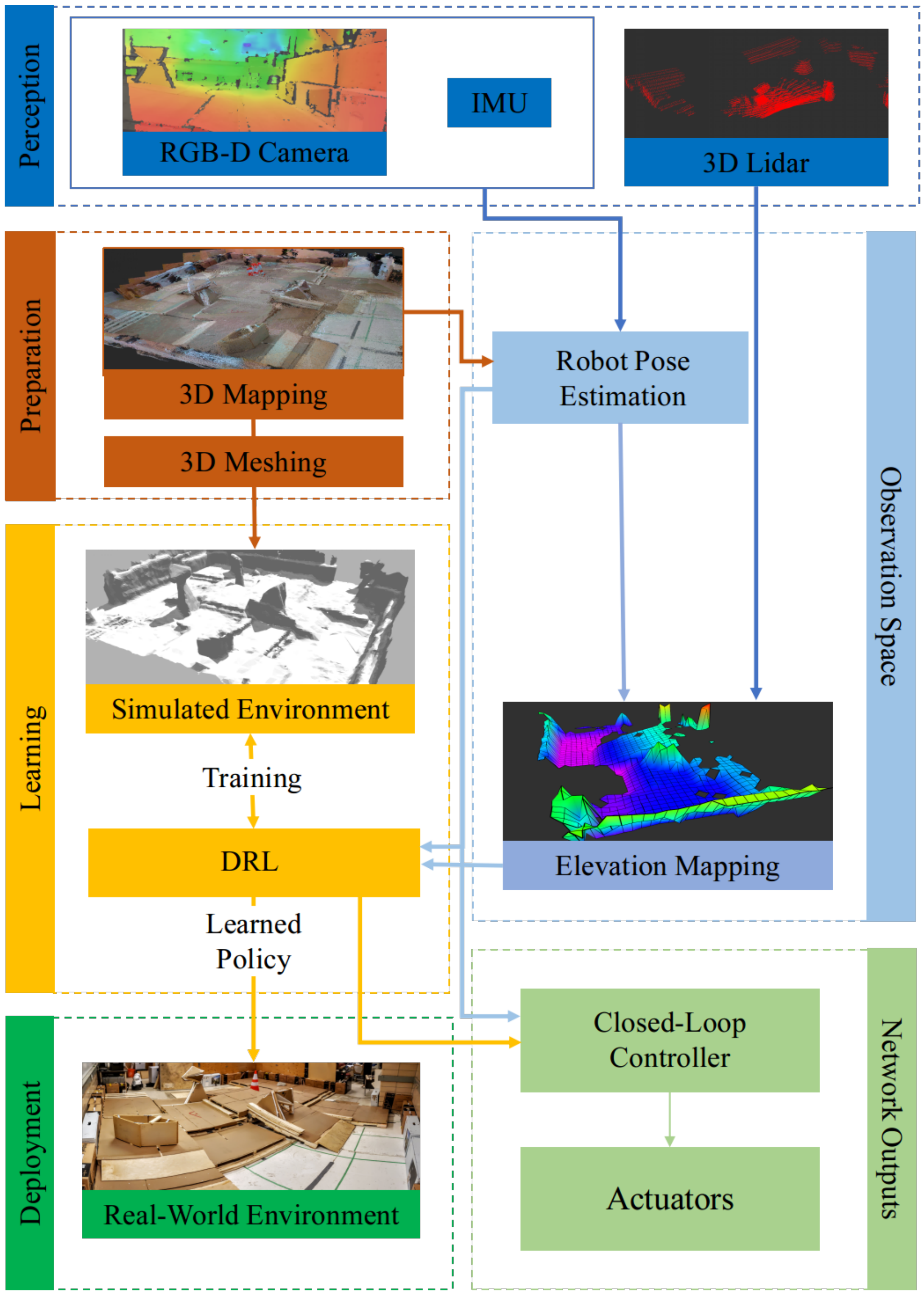


Figure 4.1: Pipeline for sim-to-real transfer of rough terrain navigation policy. This pipeline is developed in conjunction with Kaicheng Zhang.

This pipeline utilized three sim-to-real strategies to address the reality gap [32]: 1) improving simulation fidelity, 2) using domain invariant observation space, and 3) domain randomization of training parameters. Improving simulation fidelity is to tune the simulation learning environment such that simulation parameters are closer to the real world values [32]. Domain invariant features applied by [39, 40] show that using inputs that are invariant between the simulation environment and the real world avoids overfitting to features that are environment-specific, such as texture, color, and lighting condition. Domain randomization is a popular sim-to-real strategy used by [32, 35–37]. It exposes the policy to a wide range of parameter variations during training such that the policy is less prone to overfit to simulation-specific parameter values, and the real-world may appear to the policy as one of the variations [32, 35].

To improve the simulation fidelity, an accurate 3D point cloud model of the navigation environment with high geometric fidelity is first scanned using RGB-D cameras, 3D LiDAR, and IMU as the inputs to the RTAB-Map algorithm. The 3D point cloud is then cleaned and reconstructed into a continuous mesh surface at the *Preparation* phase. The *Simulated Environment* within the Gazebo simulator [46] is built using the reconstructed mesh. The *DRL* module interacts with the *Simulated Environment* using the robot's position information provided by *Robot Pose Estimation* and a robot centric 2D height representation provided by the *Elevation Mapping* module. This module learns a navigation policy represented by a CNN using A3C developed in Pytorch [123]. Both the robot position information and the elevation map are domain invariant. During *Learning*, domain randomization is applied to 3 simulation parameters to further narrow the reality gap for challenging 3D terrain. At *Deployment*, the robot's *Perception* is used to generate a real-time elevation map and robot pose, similar to the inputs to the *DRL*. The desired robot actions are produced by the *DRL* using the trained policy. These actions are passed to the *Closed-Loop Controller*, which is then transformed into motion commands for the robot's *Actuators*. The main modules of the presented pipeline are discussed in detail in the sections below.

4.1 Contributions

The sim-to-real pipeline presents a large undertaking that was developed in collaboration with Kaicheng Zhang in our lab. My specific contributions to this pipeline are:

- Design, development, and implementation of the perception module.
- Design, development, and implementation of the preparation module.
- Design, development of the simulation learning environment terrain.
- Design of using domain randomization technique for sim-to-real.
- Design, development, and implementation of the terrain steepness parameter randomization.
- Design and formulation of the pose estimation parameter randomization.
- Design of motion disturbance parameter randomization.
- Integration and deployment of DRL navigation policy to the physical robot.
- Design of real-world experiment environment.

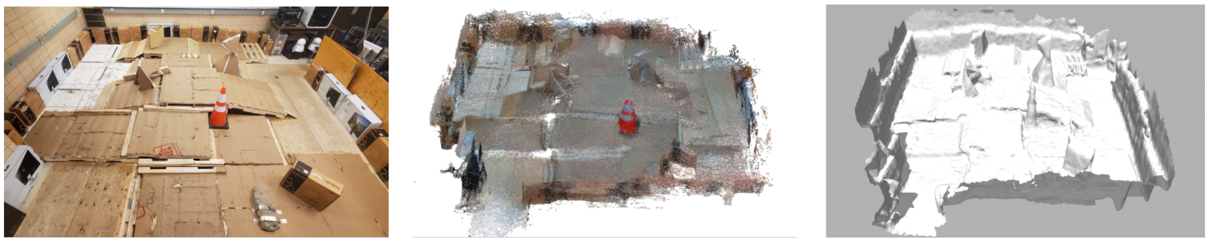
4.2 Perception

The *Perception* of the pipeline required an RGB-D Camera to provide RGB images and registered depth information, a IMU for linear and angular accelerations, and a 3D LiDAR for the 3D point cloud. In this thesis, he Intel-D435 with a built-in IMU was used to provide the RGB images, depth, and acceleration information to the *Preparation* during training and to the robot during *Deployment*. The Velodyne VLP-16 provided the 3D point cloud to the *Elevation Mapping* module.

4.3 Preparation

The preparation module bridges the reality gap due to the terrain geometric differences between the real world and the simulation. It constructs a geometrically accurate terrain model of the real-world environment. This strategy was shown to be successful in [32,41]

The model construction is shown in Figure 4.2a, where an RGB-D camera with a built-in IMU was used to build a 3D point cloud model as seen in Figure 4.2b with a *3D mapping* algorithm such as RTAB-Map [51]. Point cloud viewing software such as Meshlab [124] can be used to clean the point cloud model using denoise operations, hole filling, and filtering floating surfaces. *3D meshing* methods such as Poisson surface reconstruction [51] is then used to reconstruct the cleaned point cloud into a continuous 3D mesh surface. The reconstructed mesh seen in Figure 4.2c is imported into the Gazebo simulator as the simulation terrain.



(a) Real World Terrain (b) Point Cloud Model (c) Simulation Terrain

Figure 4.2: Preparation process of creating the simulation environment terrain for learning navigation policy.

4.4 Learning

4.4.1 Simulated Environment

The 3D simulation training environment was developed by me using the *Preparation* module and a simulation model of the Jaguar4×4 developed in [29]. Similar to the high fidelity simulators used by [32,41], the navigation policy was learned in a simulator with high geometric fidelity. The real-world robot is represented in the simulation environment with a physics model of the Jaguar4×4 platform developed by [29].

4.4.2 Deep Reinforcement Learning

The DRL learning method was developed by Kaicheng Zhang in [29]. It is discussed here for completeness. The DRL module is used to learn the optimal robot navigation actions within the environment. This module follows the A3C [47] algorithm to learn a policy $\pi(a_t|s_t, \theta)$ represented by a CNN network parameterized by θ. At each time step, $\pi(a_t|s_t, \theta)$ determines a discrete robot action a_t output that maximizes the expected future reward given the robot state s_t. The set of a_t are linear forward or backward motion for a distance L, or a positive or negative yaw rotation of θ_y. The s_t is the *Observation Space*; J_t and elevation map.

Network Architecture The network architecture used is shown in Figure 4.3. A LSTM layer is used to capture previous states in the decision-making process. The actor layer output is the navigation action a_t, and the critic layer output is the estimated value of a given state $V(s_t, \theta_v www)$.

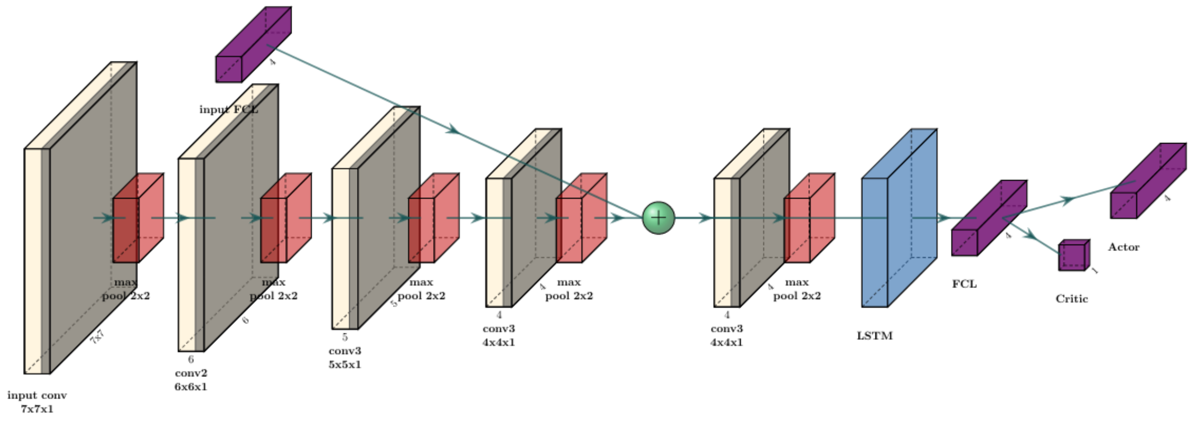


Figure 4.3: Network architecture with configuration details. Each convolution layer has a stride of 1 and padding of 0 with RELU activation.

4.4.3 Domain Randomization of Training Parameters

Domain randomization of training parameters' value increases the likelihood of the real-world parameter value being present during training, such that the real-world may appear to the model as a training variation [32, 35]. This strategy has been successfully applied to real robots in several applications [32, 35–37].

Three parameters were randomized during training to address the reality gap. Two parameters address challenges that only exist in 3D terrain navigation. The third parameter addresses the robot pose estimation error in the real-world environment due to factors such as sensor noise, lighting conditions, and texture.

Varying Terrain Steepness The cascading measurement errors from the *Perception* module and mesh reconstruction errors from the *Preparation* can cause the simulation terrain to deviate from the real-world terrain. The real-world terrain's steepness and traversability may not be accurately reflected in the simulated environment. These discrepancies can cause the robot to take undesirable navigation actions such as climbing steep slopes or taking non-traversable paths. Randomizing the terrain steepness can minimize the effect of these discrepancies. The steepness is randomized during training by uniformly scaling the mesh height h according to Equation 4.1. h_o is the original steepness of the mesh. Where $U_h(e_{lh}, e_{uh})$ is the distribution with lower bound e_{lh} and upper bound e_{uh}.

$$h \cong h_o U_h(e_{lh}, e_{uh}) \tag{4.1}$$

Given the terrain steepness measurement cannot be accurately modeled, uniform distribution is used to expose $\pi(a_t|s_t, \theta)$ during learning to a broad distribution of steepness value. At each training episode, the simulation terrain is scaled by $\pm 5\%$, i.e. $e_l h = 95\%, e_u h = 105\%$. This value is determined from the translational error of RTAB-Map [51] in the environment's z axis and the robot's geometry.

Robot Motion Disturbance The robot-terrain interaction during navigation can cause deviations to the robot's intended actions. Examples are wheel slippage and unintended rotations due to insufficient traction or wheel placement at different elevations. In [32], motion distribution was applied to a quadruped robot during training to improve stability. This idea can be applied to rough terrain navigation to increase the robustness of the navigation policy against robot-terrain interactions. However, because of the unpredictable nature of robot-terrain interactions, it is difficult to train a policy that is robust against all real-world robot-terrain interactions. Therefore, the applied motion disturbance aims to improve the robot's ability to recover from deviations to the intended navigation route instead of increasing its resistance to external disturbances.

To improve the robustness of the learned policy against these robot-terrain interactions, motion disturbances with a uniform distribution $U_L(e_{lL}, e_{uL})$ and $U_{\theta_y}(e_{l\theta_y} e_{u\theta_y})$ are applied to the robot's action output L and θ_y during training with a probability I to model the periodically occuring robot-terrain interactions. The e_{lL} and e_{uL} are the lower and upper bounds of L randomization values, and $e_{l\theta_y}$ and $e_{u\theta_y}$ are the lower and upper bounds of θ_y randomization values.

In additional to deviations caused by robot-terrain interactions, the robot motion is also affected by latency when executing control commands, which has been found to be a critical factor in DRL based controllers [32]. This latency is from the robot pose computation time taken by the *Robot Pose Estimation* module and network latency. These are modeled as a uniform distribution during training at each time step as $U'_L(e'_{lL}, e'_{uL})$ and $U'_{\theta_y}(e'_{l\theta_y} e'_{u\theta_y})$. The e'_{lL} and e'_{uL} are the lower and upper bounds of L randomization values, and $e'_{l\theta_y}$ and $e'_{u\theta_y}$

The robot motion during training with disturbances is shown in Equation 4.2 and Equation 4.3. I follows a Bernoulli distribution, $I \sim \text{Bern}(P)$ to model that the interaction disturbance occurs at random instances where P is the hyperparameter representing the probability that the disturbances occur.

$$L = L_o + IU_L(e_{lL}, e_{uL}) + U_L^{'}(e_{lL}^{'}, e_{uL}^{'}) \tag{4.2}$$

$$\theta_y = \theta_{yo} + IU_{\theta_y}(e_{l\theta_y} e_{u\theta_y}) + U_{\theta_y}^{'}(e_{l\theta_y}^{'} e_{u\theta_y}^{'}) \tag{4.3}$$

The interaction disturbance randomization parameter values are $e_{lL} = 20\text{cm}, e_{uL} = 40\text{cm}, e_{l\theta_y} = -45°, e_{u\theta_y} = 45°$. The values are chosen to be large enough to cause the robot to visibly deviate from its intended trajectory at the expected interval of 60s, i.e. $P = 1/60$.

The latency disturbance randomization parameter values are $e_{lL}^{'} = 0\text{cm}$, $e_{uL}^{'} = 3\text{cm}, e_{l\theta_y}^{'} = 0°$, and $e_{u\theta_y}^{'} = 3°$. These values were determined by measuring the physical robot's average deviation from the desired L and θ_y using visual odometry in an arbitrary environment.

Robot Pose Estimation Error The robot pose estimation error can occur due to sensor measurement errors, feature association errors, and scan alignment errors. Sensor measurement errors can be caused by factors such as lens distortion effects and inaccurate depth estimation errors [125]. Feature association errors are caused by the inclusion of ambiguous and sparse features in the computation [125]. Scan alignment errors may occur due to outliers when matching each subsequent scan with the previous scan using matching algorithms such as Iterative Closest Point(ICP). These errors can cause the estimated pose to deviate from the true values [125]. These pose estimation errors affects the input to the DRL network; the robot pose $J_t = \{d_t, \alpha_t, \beta_t, \gamma_t\}$ derived from the pose estimation and the elevation map which uses J_t to aggregate point cloud scans. To ensure the learned policy $\pi(a_t|s_t, \theta)$ is robust against the pose estimation errors in the real world, a zero-mean Gaussian distribution $N_p(0, \sigma_p)$ with standard deviation σ_p is added to the robot pose during the learning process as seen in Equation 4.4. Where p_o is the 6 DOF pose estimated by the *Robot Pose Estimation* module.

$$p_t = p_o + N_p\left(0, \sigma_p\right) \tag{4.4}$$

The robot pose randomization values are $\sigma_{pl} = 6\text{cm}$ for all positional values and $\sigma_{pr} = 6°$ for all euclidean angle pose values. These values were chosen based on the expected translational and orientation errors when mapping a similar sized environment using visual slam [51, 60].

4.4.4 Training

The robot was trained in simulation by moving between a pair of randomly generated starting and goal locations. Nine parallel threads were used to collect experiences from independent simulations in parallel. Navigation actions of the robot during training were computed using the most recent policy $\pi(a_t|s_t, \theta)$ updated by the A3C algorithm. The reward received during training was used to update the weights of the network. The action output value are $L = 10\text{cm}$, $\theta_y = 15°$. The details of the training conducted are presented in [126].

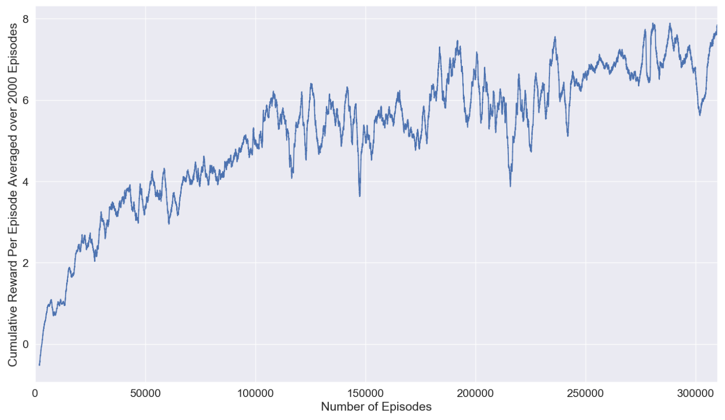


Figure 4.4: Cumulative reward per episode per 2000 episodes.

4.5 Network Output and Deployment

The policy network outputs L and θ_y as the action command. A two-layer closed-loop controller shown in Figure 3.4 executes the action command using visual odometry as feedback. The outer layer bang-bang controller outputs a constant velocity commands to the inner proportional controller [96] until L or θ_y has been reached. The velocity commands are converted into individual wheel spin rates using a skid steer kinematic model [90]. The proportional controller tracks the wheel spin rate using wheel encoders as feedback.

4.6 Experiments

Three experiments were conducted to analyze the effectiveness of the sim-to-real transfer pipeline proposed in 4; a real-world experiment to determine the success rate of autonomous navigation in a cluttered rough terrain, a comparison study of the DRL method against a classical method, and an ablation study on the effectiveness of the the domain randomization parameters.

4.6.1 Real-World Experiments

Mobile Robot

The physical robot used in the experiment was the Jaguar4×4 developed in 3. The perception sensors mounted on the robot is the Velodyne VLP-16 3D LiDAR and the Stereo Labs ZED-Mini stereo camera with an integrated IMU. The robot has a Intel NUC with a Core i5 CPU and a Jetson TX2 mobile GPU. The Jetson TX2 collects RGB images with registered depth information and IMU data from the ZED-Mini sensor. The Intel NUC uses the collected information, 3D LiDAR point cloud, and encoders to compute the robot pose, elevation map, and executes the robot navigation actions. A ground station server with an Intel i7-7700K CPU uses the pose and the elevation map to generate the navigation action commands.

Experiment Environment

A $36m^2$ cluttered rough terrain experiment environment shown in Figure 4.5 was built using wooden pallets and scattered objects. The environment is divided into 22 regions of different elevations and transition features shown in Figure 4.5b. The full steps have a height of 0.13mm, and the half steps have a height between 0.5 to $0.08m$ shown in Figure 4.5g. The slope and the length of the ramps range between $20° - 40°$ and $0.3 - 0.9m$, Figure 4.5h. The traversability of these transition regions depends on the robot's position, angle of approach, and intended action. The Jaguar4×4 cannot climb full steps when moving directly toward it, and it can climb half steps.

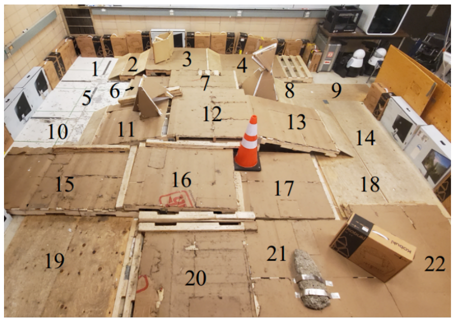
Procedure

In each experiment trial, the robot navigated between a pair of randomly selected start and goal locations. A successful trial is when the robot autonomously arrive at the goal within a tolerance distance $d_g = 0.3m$ and a time $t_{max} = 10\text{min}$. 20 combinations of start and goal location pairs were randomly chosen across the environment, where each pair is repeated 3 times for a total of 60 trials.

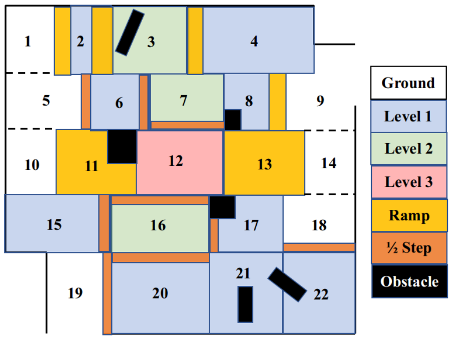
Results

The experiments had shown that the robot traversed the real-world environment using a deep reinforcement learning-based navigation policy learned in simulation with a success rate of 86.67% in the real world. The success rate for this experiment in simulation is 90.0%. The success rate across the three trials of each location pair is presented in Table 4.1.

Navigation strategies have been learned during training to climb challenging terrain.



(a) Terrain with regions (1 to 22).



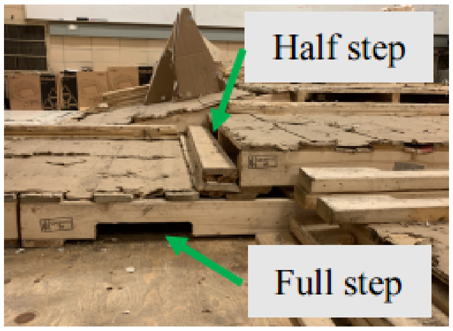
(b) Labeled terrain features.

(c) Terrain image from region 1.

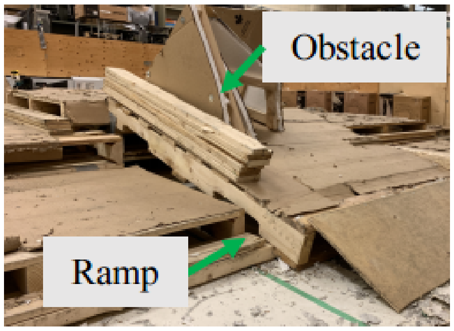
(d) Terrain image from region 19.

(e) Terrain image from region 22.

(f) Terrain image from region 9.



(g) Full and half steps.



(h) Obstacles and ramps.

Figure 4.5: Real world experiment environment shown in (a). Terrain images taken from different angles shown at (c) to (f). Zoomed in view of terrain features at (g) and (h).

For instance, the robot learned to leverage adjacent ramps and half steps to climb full steps. It also learned to keep a minimum distance from walls and ledges to avoid a collision or falling when traversing through the terrain. The robot would also try different heading directions when it becomes stuck. The test 10 trial shown as the yellow trajectory in

Table 4.1: Success rate of all 60 trials in the real environment.

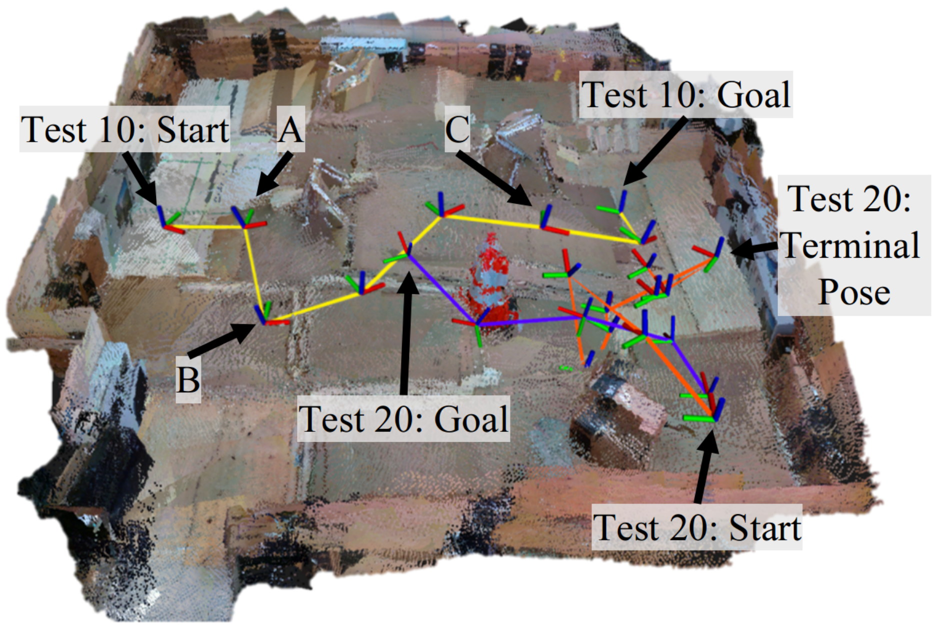
Test	**Start x, y, yaw (m,m,rad), Region**	**Goal x, y (m,m), Region**	**Success %**
1	[-0.18, 0.27, -3.14], 12	[-0.33, -1.86], 10	100
2	[-0.45, -1.9, 1.57], 10	[-0.16, 0.22], 12	100
3	[-2.56, -1.65, 0], 1	[-0.17, 2.47], 14	66.67
4	[-0.34, 2.62, 3.12], 14	[-2.28, -1.85], 1	100
5	[-2.22, 1.39, -1.46], 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%**

Figure 4.6a showed successful navigation to the goal from region 10 to region 9 where the robot used the learned skills. In this test, the robot climbed a ramp and stopped before an unclimbable obstacle as shown in Figure 4.6b. Following that, the robot used the adjacent ramp and climbed over a full step to avoid the unclimbable obstacle in Figure 4.6c. Finally, the robot descended the ramp shown in Figure 4.6d to the goal location.

Tests 3, 5, and 7 had a few failure trials. These were caused by repeated vibrations due to drops and bumps during traversal that caused large drifts in the sensor measurements and led to variability in the robot's decisions. Additionally, the limited resolution of the pipeline's *3D mapping* module with RTAB-Map and 3D meshing caused deviations of the simulation terrain from the real world. Some gaps and edges in the terrain appeared smaller in simulation, which allowed the robot to cross some gaps in simulation that were not traversable in the real environment.

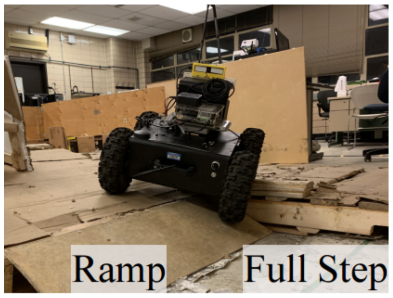
In Test 20, the robot navigated to the goal in simulation (orange in Figure 4.6a), but failed to do so in the real environment (purple in Figure 4.6a). It was because the angle of approach to the ramp during the trial as shown in Figure 4.6d caused the ramp to be unclimbable. This issue resulted in the robot failing the experiment trials at the terminal pose shown in Figure 4.6a due to reaching t_{max}.

This experiment repeated in a simulated environment had a total average success rate of 90.9%. It is close to the 86.67% success rate of the real world, which showed that the proposed pipeline is successful in the sim-to-real transfer of a DRL navigation policy.

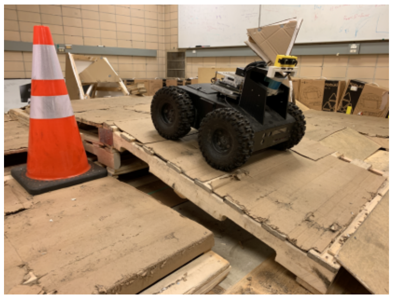


(a) 3D point cloud model of the environment with 3 diferent trajectories shown. Test 10 real trajectory (yellow), Test 20 real trajectory (orange), Test 20 simulation trajectory (purple).

(b) Zoomed in view at A in (a).



(c) Zoomed in view at B in (a).



(d) Zoomed in view at C in (a).

Figure 4.6: Robot trajectories taken during experiment trials 10 and 20.

4.6.2 Comparison Study

A study was conducted in the simulated environment to compare the performance of the DRL policy trained using the sim-to-real pipeline and a classical binary traversability method [26].

The method presented by [26] is chosen as the classical method. This method compared the linear combination of the terrain's height, slope, and roughness against a threshold to determine a binary traversability value. It is representative of the existing classical methods as it uses hand-picked feature inputs to its algorithm and relies on handcrafted heuristic rules to determine the terrain traversability.

This comparison study measures the success rate, cumulative travel distance to account for terrain and path variability, cumulative travel time for successful trials, and path replanning rate. The same location pairs as the ones listed in Table 4.1 were used in this experiment.

Results

Figure 4.7 shows the performance metrics of all the methods. The success rates of the compared methods are 18% and 90% for the classical and the DRL based method respectively. The DRL method had a shorter cumulative distance and faster time of 14.51m and 227s. The classical method cumulatively traveled 15.47m in 451s. The overall replanning rate of the DRL method was a faster 3.2Hz, where the classical was 1.8Hz.

This comparison showed that the DRL method is more successful and faster at reaching the goal location compared to the classical method.

The robot trajectory for Test 3 is shown Figure 4.8; a representative example of where the classical method failed and the DRL succeeded. In Figure 4.8b, the robot's path using the classical method prematurely terminated at location A. The classical method misidentified some grid cells as obstacles, therefore caused the robot to be stuck near the

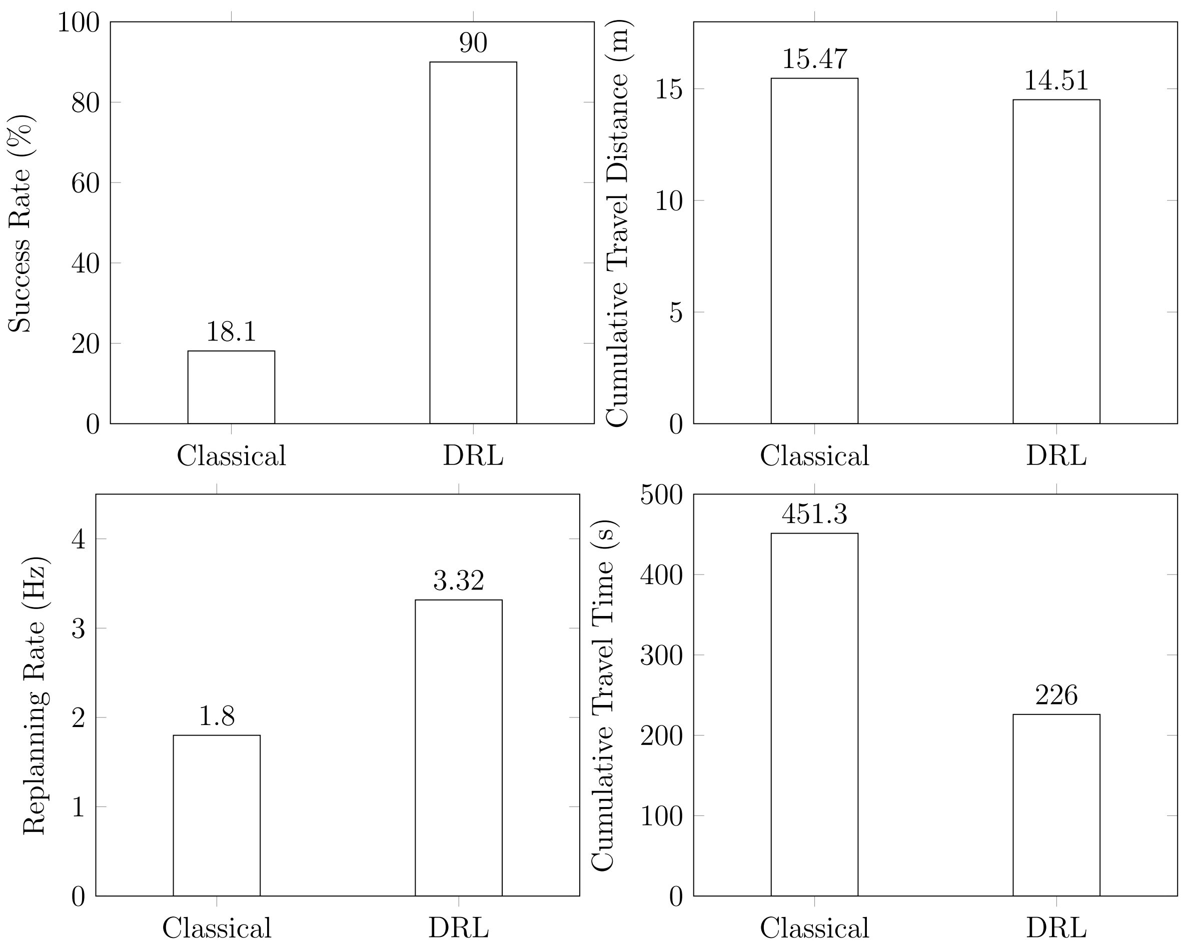
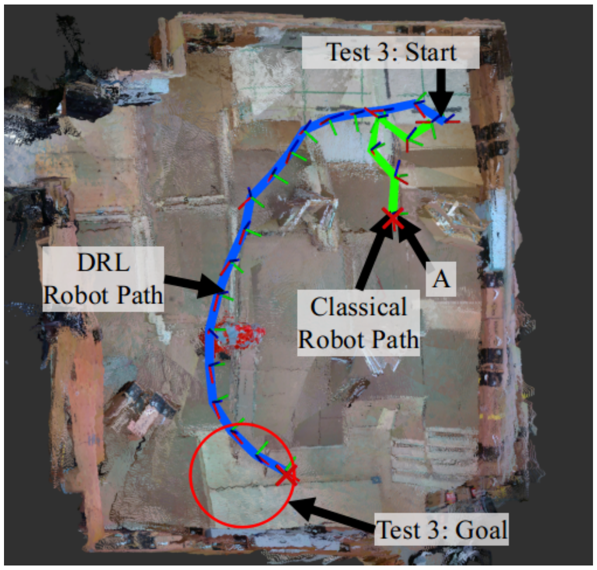
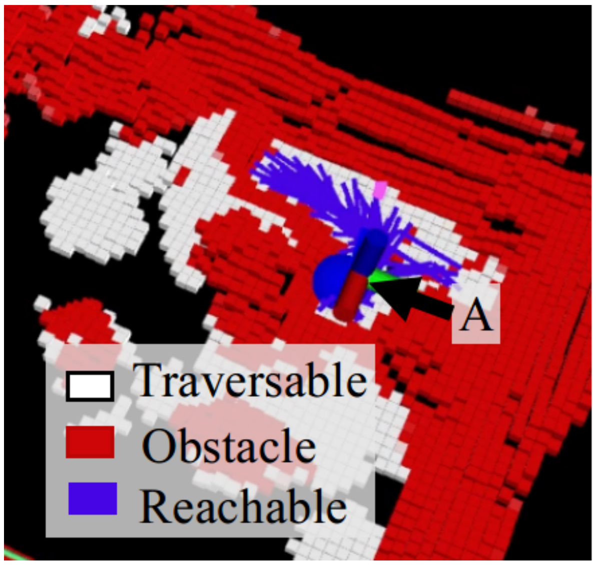


Figure 4.7: Comparison study result. Performance metric: (a) success rate, (b) cumulative travel distance, (c) cumulative travel time, and (d) path replanning rate.

starting location. This misidentification was due to the reliance on manually designed heuristics for traversability estimation, which was shown to be unreliable in complex terrain. These errors cascaded into the planning stage, which caused the planner to fail to find valid paths to the goal. Furthermore, a classical method is a multi-stage approach that overly simplifies the terrain traversability representation. This simplification does not provide the planning stage with any information regarding the robot-terrain interaction that is needed when navigating 3D rough terrain [126]. The DRL reached the goal as it learned the traversability of the path using a CNN and avoided simplifying the traversability representation by directly mapping the observation space to an action output.



(a) Robot path in Test 3.



(b) Classical traversability map at A.

Figure 4.8: Compairson study of Test 3 between clasical and DRL based navigation method.

4.6.3 Ablation Study

An ablation study was conducted in simulation on the domain randomization parameters presented in subsection 4.4.3 to evaluate the importance of each parameter in sim-to-real transfer. The performance of a model trained with all three parameters was compared against other models trained in the absence of each parameter. To have controlled evaluation with large terrain variations, this is conducted in the simulated environment by doubling the randomized parameters during evaluation until they were 8 times the values used during training. The evaluation environment parameters are shown in Table 4.2.

Results

Table 4.2: Ablation study evaluation environment hyperparameters.

	Baseline	**x2**	**x4**	**x8**
Steepness	5%	10%	20%	40%
Latency	0~0.03m, $0° \sim 3°$	$0 \sim 0.06m$, $0° \sim 6°$	$0 \sim 0.12m$, $0° \sim 12°$	$0 \sim 0.24m$, $0° \sim 24°$
Pose Error	$0.06m, 5°$	$0.12m, 10°$	$0.24m, 20°$	$0.48m, 40°$

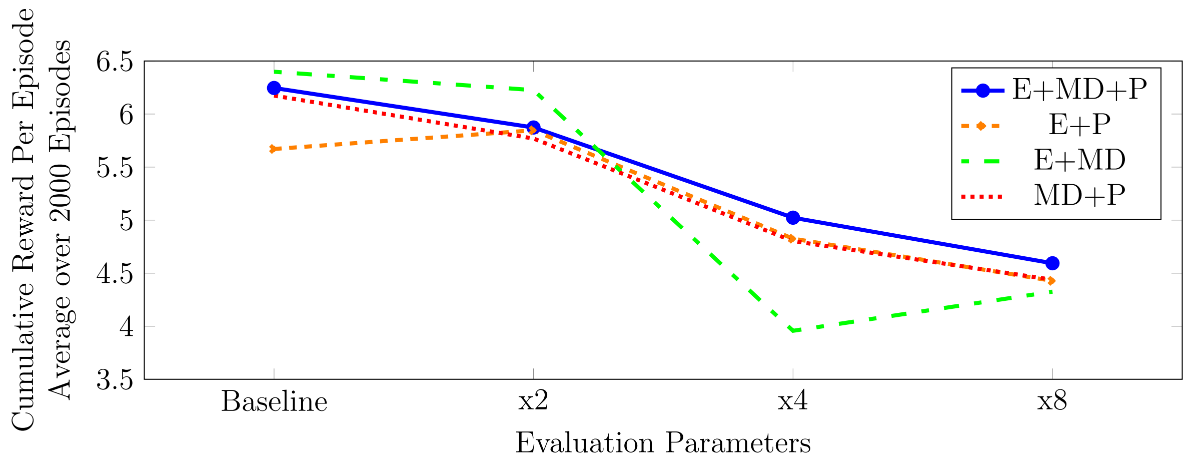


Figure 4.9: Ablation study of model performance. E: trained with elevation steepness, MD: trained with motion disturbance, and P: trained with pose estimation errors.

Figure 4.9 presents the result of the ablation study. The E+MD+P (elevation steepness + motion disturbance + pose estimation errors) trained with all domain randomization parameters was the most consistent in maximizing the cumulative rewards. The E+P and the MD+P models performed similarly. It is because uncertainties in the training domain can be seen as a perturbation on the system [32]. The MD+P showed the most drastic change in its performance. This change further supports the idea that training with pose estimation error and motion disturbance has a similar effect on the system while also emphasizing the importance of training with elevation steepness randomization to generalize the model to larger terrain variations.

4.7 Chapter Summary

This chapter presented a novel sim-to-real pipeline for transferring 3D rough terrain navigation policy learned in simulation to the real world. This is achieved through using sim-to-real strategies; improving the simulation fidelity, using domain-invariant inputs, and applying domain randomization during training.

Simulation fidelity is improved through the procedure presented in section 4.3 to create a geometrically accurate 3D model of the real-world terrain for the training environment.

Robot position information and elevation map are the domain invariant inputs to the learning algorithm. These do not use raw data, which can overfit to domain-specific details such as color, texture, lighting conditions, and sensor noise patterns.

Section 4.4 described the DRL navigation algorithm used to train the navigation algorithm. During training, domain randomization of terrain steepness, motion disturbance, and robot pose randomization was applied to exposes the policy to a wide range of parameter variations such that it is less prone to overfit to simulation-specific parameter values.

Experiments were conducted for 1) validating the effectiveness of the presented sim-to-real pipeline in section 4.6.1, 2) comparison of DRL based navigation policy against existing literature in section 4.6.2, and 3) an ablation study on the importance of the domain randomization parameters in section 4.6.3.

Section 4.6.1 showed that the sim-to-real pipeline proposed in chapter 4 is effective in transferring navigation policy learned in simulation to the real world. The pipeline enabled the robot to successfully navigate the real-world target environment with a success rate of 86.67% given a simulation navigation success rate of 90.9%.

The comparison study in subsection 4.6.2 demonstrated that a DRL end-to-end based method for rough terrain is more successful and efficient in reaching the target location when compared to multi-stage methods that separate terrain traversability classification and path planning. Multi-stage methods suffer from an overly simplified terrain traversability representation. This simplification causes loss of information important to understanding 3D rough terrain navigation, such as robot-terrain interaction. The end-to-end DRL based method avoided this simplification and achieved a higher success rate.

The ablation study subsection 4.6.3 has shown that randomizing terrain steepness, motion disturbance, and robot pose estimation noise is important to learning a method that can consistently maximize reward in terrains with greater variability in the training environment.

Chapter 5

Conclusion

5.1 Final Concluding Statement

This thesis presented the design of a decentralized architecture of a mobile robot and a sim-to-real transfer pipeline for DRL navigation policy in a 3D cluttered rough terrain with abrupt changes in elevation and surface normals. The ROS based distributed computing software architecture allows the robot to have multiple computing units that operate independently of a remote ground station. The mobile platform architecture is designed to use the SLAM and sensor combination that is robust to the challenges of rough terrain environments.

A novel sim-to-real pipeline was presented for transfer reinforcement learning-based rough terrain navigation policy learned in simulation to the real world. This pipeline addressed the discrepancies between the real-world and the simulation by improving the simulation fidelity by 1) utilizing a geometrically accurate terrain model in simulation, 2) use domain invariant inputs to the DRL learning algorithm, and 3) apply domain randomization to three simulation parameters that were designed to address the challenges in 3D rough terrain by considering the robot-terrain interaction.

The feasibility of the proposed sim-to-real pipeline was shown by a set of successful real-world navigation experiments in a real-world 3D rough terrain. This experiment showed that the proposed pipeline had a real-world navigation success rate of 86.67% given a simulation navigation success rate of 90.9%. Comparison studies have shown that the DRL based navigation approach is more successful in performing rough terrain navigation compared to existing classical methods. An ablation study showed that using all the domain randomization parameters is needed to maximize the cumulative reward given terrain with higher variability.

5.2 Future Work

A limitation of the proposed pipeline is that DRL algorithms suffer from overfitting to the training environments. Therefore the proposed pipeline is difficult to apply to unknown terrain. The future work would be to overcome this limitation. It would require representing the vast variability of the real-world environment in the training terrain. Additional work would include improving the robustness of the pipeline to address the failure cases seen.

5.3 Summary of Contributions

The contributions of this thesis are

1. The design of a ROS based decentralized architecture for a wheeled mobile platform
2. The first to empirically compared the robustness of ROS compatible SLAM algorithms and sensor combination under a multitude of challenging environmental conditions.
3. Development of a sim-to-real pipeline that is the first to address the cluttered 3D rough terrain's reality gap for autonomous navigation.

4. Proposed three new domain randomization parameters to specifically address the challenges in a 3D rough terrain navigation environment.

5. The first to demonstrate successful implementation of DRL for real-world rough terrain navigation.

6. Comparison experiments between existing rough terrain navigation methods and the DRL based method.

7. Conducted an ablation study that analyzed the effectiveness of the new domain randomization methods.

Bibliography

[1] Yugang Liu and Goldie Nejat. Robotic urban search and rescue: A survey from the control perspective. *Journal of Intelligent & Robotic Systems*, 72(2):147–165, 2013.

[2] Yugang Liu and Goldie Nejat. Multirobot cooperative learning for semiautonomous control in urban search and rescue applications. *Journal of Field Robotics*, 33(4):512–536, 2016.

[3] Barzin Doroodgar, Yugang Liu, and Goldie Nejat. A learning-based semi-autonomous controller for robotic exploration of unknown disaster scenes while searching for victims. *IEEE Transactions on Cybernetics*, 44(12):2719–2732, 2014.

[4] Y Baudoin and Maki K Habib. *Using robots in hazardous environments: Landmine detection, de-mining and other applications*. Elsevier, 2010.

[5] Mohamed J Bakari, Khaled M Zied, and Derek W Seward. Development of a multi-arm mobile robot for nuclear decommissioning tasks. *International Journal of Advanced Robotic Systems*, 4(4):51, 2007.

[6] Keiji Nagatani, Seiga Kiribayashi, Yoshito Okada, Satoshi Tadokoro, Takeshi Nishimura, Tomoaki Yoshida, Eiji Koyanagi, and Yasushi Hada. Redesign of rescue mobile robot quince. In *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*, pages 13–18. IEEE, 2011.

[7] R. R. Murphy, J. Kravitz, S. L. Stover, and R. Shoureshi. Mobile robots in mine rescue and recovery. *IEEE Robotics Automation Magazine*, 16(2):91–103, 2009.

[8] Steve Grehl, Helmut Mischo, and Bernhard Jung. Research perspective-mobile robots in underground mining. *AusIMM Bulletin*, (Feb 2017):44–47, 2017.

[9] H. Ardiny, S. Witwicki, and F. Mondada. Construction automation with autonomous mobile robots: A review. In *2015 3rd RSI International Conference on Robotics and Mechatronics (ICROM)*, pages 418–424, 2015.

[10] Nathan Melenbrink, Justin Werfel, and Achim Menges. On-site autonomous construction robots: Towards unsupervised building. *Automation in Construction*, 119:103312, 2020.

[11] Morgan Quigley, Brian Gerkey, and William D Smart. *Programming Robots with ROS: a practical introduction to the Robot Operating System.* " O'Reilly Media, Inc.", 2015.

[12] Peter Hart, Nils Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[13] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[14] M. Kazemi, K. K. Gupta, and M. Mehrandezh. Randomized kinodynamic planning for robust visual servoing. *IEEE Transactions on Robotics*, 29(5):1197–1211, 2013.

[15] F. Colas, S. Mahesh, F. Pomerleau, M. Liu, and R. Siegwart. 3d path planning and execution for search and rescue ground robots. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 722–727, 2013.

[16] B. Morisset, R. B. Rusu, A. Sundaresan, K. Hauser, M. Agrawal, J. Latombe, and M. Beetz. Leaving flatland: Toward real-time 3d navigation. In *2009 IEEE International Conference on Robotics and Automation*, pages 3786–3793, 2009.

[17] Philipp Krüsi, Paul Furgale, Michael Bosse, and Roland Siegwart. Driving on point clouds: Motion planning, trajectory optimization, and terrain assessment in generic nonplanar environments. *J. Field Robot.*, 34(5):940984, August 2017.

[18] R. Fedorenko, A. Gabdullin, and A. Fedorenko. Global ugv path planning on point cloud maps created by uav. In *2018 3rd IEEE International Conference on Intelligent Transportation Engineering (ICITE)*, pages 253–258, 2018.

[19] Fabio Ruetz, Emili Hernndez, Mark Pfeiffer, Helen Oleynikova, Mark Cox, Thomas Lowe, and Paulo Borges. Ovpc mesh: 3d free-space representation for local ground vehicle navigation, 2018.

[20] P. Papadakis. Terrain traversability analysis methods for unmanned ground vehicles: A survey. *Eng. Appl. Artif. Intell.*, 26:1373–1385, 2013.

[21] J. Sock, J. Kim, J. Min, and K. Kwak. Probabilistic traversability map generation using 3d-lidar and camera. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5631–5637, 2016.

[22] Shengyan Zhou, Junqiang Xi, Matthew W. McDaniel, Takayuki Nishihata, Phil Salesses, and Karl Iagnemma. Self-supervised learning to visually detect terrain surfaces for autonomous robots operating in forested terrain. *J. Field Robot.*, 29(2):277297, March 2012.

[23] Michael Happold, Mark Ollis, and Nikolas Johnson. Enhancing supervised terrain classification with predictive unsupervised learning. In *Robotics: science and systems*. Citeseer, 2006.

[24] B. Wang, J. Ren, and M. Cai. Car-like mobile robot path planning in rough terrain with danger sources. In *2019 Chinese Control Conference (CCC)*, pages 4467–4472, 2019.

[25] D. Silver, J. Bagnell, and A. Stentz. Learning from demonstration for autonomous navigation in complex unstructured terrain. *The International Journal of Robotics Research*, 29:1565 – 1592, 2010.

[26] Tixiao Shan, Jinkun Wang, Brendan Englot, and Kevin Doherty. Bayesian generalized kernel inference for terrain traversability mapping. In *In Proceedings of the 2nd Annual Conference on Robot Learning*, 2018.

[27] R. O. Chavez-Garcia, J. Guzzi, L. M. Gambardella, and A. Giusti. Learning ground traversability from simulations. *IEEE Robotics and Automation Letters*, 3(3):1695–1702, 2018.

[28] F. Schilling, X. Chen, J. Folkesson, and P. Jensfelt. Geometric and visual terrain classification for autonomous mobile navigation. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2678–2684, 2017.

[29] K. Zhang, F. Niroui, M. Ficocelli, and G. Nejat. Robot navigation of environments with unknown rough terrain using deep reinforcement learning. In *2018 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pages 1–7, 2018.

[30] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[31] S. Josef and A. Degani. Deep reinforcement learning for safe local planning of a ground vehicle in unknown rough terrain. *IEEE Robotics and Automation Letters*, 5(4):6748–6755, 2020.

[32] Jie Tan, Tingnan Zhang, Erwin Coumans, Atil Iscen, Yunfei Bai, Danijar Hafner, Steven Bohez, and Vincent Vanhoucke. Sim-to-real: Learning agile locomotion for quadruped robots, 2018.

[33] Sylvain Koos, Jean-Baptiste Mouret, and Stéphane Doncieux. Crossing the reality gap in evolutionary robotics by promoting transferable controllers. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 119–126, 2010.

[34] Adrian Boeing and Thomas Bräunl. Leveraging multiple simulators for crossing the reality gap. In *2012 12th International Conference on Control Automation Robotics & Vision (ICARCV)*, pages 1113–1119. IEEE, 2012.

[35] Fereshteh Sadeghi and Sergey Levine. Cad2rl: Real single-image flight without a single real image. 07 2017.

[36] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, May 2018.

[37] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30, 2017.

[38] Andrei A. Rusu, Mel Vecerik, Thomas Rothrl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. Sim-to-real robot learning from pixels with progressive nets, 2018.

[39] L. Tai, G. Paolo, and M. Liu. Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation, 2017.

[40] A. Francis, A. Faust, H. T. L. Chiang, J. Hsu, J. C. Kew, M. Fiser, and T. W. E. Lee. Long-range indoor navigation with prm-rl. *IEEE Transactions on Robotics*, 36(4):1115–1134, 2020.

[41] Xinlei Pan, Yurong You, Ziyan Wang, and Cewu Lu. Virtual to real reinforcement learning for autonomous driving, 2017.

[42] Zhang-Wei Hong, Chen Yu-Ming, Shih-Yang Su, Tzu-Yun Shann, Yi-Hsiang Chang, Hsuan-Kung Yang, Brian Hsi-Lin Ho, Chih-Chieh Tu, Yueh-Chuan Chang, Tsu-Ching Hsiao, Hsin-Wei Hsiao, Sih-Pin Lai, and Chun-Yi Lee. Virtual-to-real: Learning to control in visual semantic segmentation, 2018.

[43] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[44] Michael Burri, Janosch Nikolic, Pascal Gohl, Thomas Schneider, Joern Rehder, Sammy Omari, Markus W Achtelik, and Roland Siegwart. The euroc micro aerial vehicle datasets. *The International Journal of Robotics Research*, 35(10):1157–1163, 2016.

[45] Jakob Engel, Vladyslav Usenko, and Daniel Cremers. A photometrically calibrated benchmark for monocular visual odometry. *arXiv preprint arXiv:1607.02555*, 2016.

[46] Gazebo. `http://gazebosim.org/`, 2019. Accessed: 2021-04-22.

[47] Volodymyr Mnih, Adri Puigdomnech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016.

[48] I. Z. Ibragimov and Ilya M. Afanasyev. Comparison of ros-based visual slam methods in homogeneous indoor environment. *2017 14th Workshop on Positioning, Navigation and Communications (WPNC)*, pages 1–6, 2017.

[49] R. Mur-Artal and J. D. Tards. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.

[50] A. Concha and J. Civera. Dpptam: Dense piecewise planar tracking and mapping from a monocular sequence. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5686–5693, 2015.

[51] Mathieu Labbé and François Michaud. Rtab-map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation. *Journal of Field Robotics*, 36(2):416–446, 2019.

[52] Stereolabs Inc. zed-ros-wrapper. `https://github.com/charlespwd/project-title`, 2013.

[53] S. Kohlbrecher, O. von Stryk, J. Meyer, and U. Klingauf. A flexible and scalable slam system with full 3d motion estimation. In *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*, pages 155–160, 2011.

[54] Nicolas Ragot, Redouane Khemmar, Adithya Pokala, Romain Rossi, and Jean-Yves Ertaud. Benchmark of Visual SLAM Algorithms: ORB-SLAM2 vs RTAB-Map. In *2019 Eighth International Conference on Emerging Security Technologies (EST)*, Colchester, United Kingdom, July 2019.

[55] R. Giubilato, S. Chiodini, M. Pertile, and S. Debei. An experimental comparison of ros-compatible stereo visual slam methods for planetary rovers. In *2018 5th IEEE International Workshop on Metrology for AeroSpace (MetroAeroSpace)*, pages 386–391, 2018.

[56] T. Pire, T. Fischer, J. Civera, P. De Cristforis, and J. J. Berlles. Stereo parallel tracking and mapping for robot localization. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1373–1378, 2015.

[57] A. Geiger, J. Ziegler, and C. Stiller. Stereoscan: Dense 3d reconstruction in real-time. In *2011 IEEE Intelligent Vehicles Symposium (IV)*, pages 963–968, 2011.

[58] Maksim Filipenko and Ilya Afanasyev. Comparison of various slam systems for mobile robot in an indoor environment. In *2018 International Conference on Intelligent Systems (IS)*, pages 400–407. IEEE, 2018.

[59] Bruno MF da Silva, Rodrigo S Xavier, Tiago P do Nascimento, and Luiz MG Gonçalves. Experimental evaluation of ros compatible slam algorithms for rgb-d sensors. In *2017 Latin American Robotics Symposium (LARS) and 2017 Brazilian Symposium on Robotics (SBR)*, pages 1–6. IEEE, 2017.

[60] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard. 3-d mapping with an rgb-d camera. *IEEE Transactions on Robotics*, 30(1):177–187, 2014.

[61] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *2011 International Conference on Computer Vision*, pages 2564–2571, 2011.

[62] Sven Koenig and Maxim Likhachev. D*lite. pages 476–483, 01 2002.

[63] S. Katz and A. Tal. On the visibility of point clouds. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1350–1358, 2015.

[64] J. J. Kuffner and S. M. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, volume 2, pages 995–1001 vol.2, 2000.

[65] P. Soueres and J. . Laumond. Shortest paths synthesis for a car-like robot. *IEEE Transactions on Automatic Control*, 41(5):672–688, 1996.

[66] Bruno Siciliano and Oussama Khatib. *Springer handbook of robotics*. springer, 2016.

[67] Annett Chilian and Heiko Hirschmüller. Stereo camera based navigation of mobile robots on rough terrain. In *Proceedings of the 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IROS'09, page 45714576. IEEE Press, 2009.

[68] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2015.

[69] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge, 2015.

[70] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding, 2016.

[71] Anh Nguyen, Ngoc Nguyen, Kim Tran, Erman Tjiputra, and Quang D. Tran. Autonomous navigation in complex environments with deep multimodal fusion network, 2020.

[72] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[73] Jean Kossaifi, Adrian Bulat, Georgios Tzimiropoulos, and Maja Pantic. T-net: Parametrizing fully convolutional nets with a single high-order tensor, 2019.

[74] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, 3:2149–2154 vol.3, 2004.

[75] Blender Online Community. Blender - a 3d modelling and rendering package, 2002.

[76] Nicolas Heess, Jonathan J Hunt, Timothy P Lillicrap, and David Silver. Memory-based control with recurrent neural networks, 2015.

[77] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay, 2018.

[78] ROS. navigation. `https://github.com/ros-planning/navigation`, 2013.

[79] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996.

[80] Hao-Tien Lewis Chiang, Aleksandra Faust, Marek Fiser, and Anthony Francis. Learning navigation behaviors end-to-end with autorl, 2019.

[81] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2019.

[82] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation, 2016.

[83] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs, 2017.

[84] Bolei Zhou, Hang Zhao, Xavier Puig, Tete Xiao, Sanja Fidler, Adela Barriuso, and Antonio Torralba. Semantic understanding of scenes through the ade20k dataset. *International Journal of Computer Vision*, 127(3):302–321, 2019.

[85] Hengshuang Zhao, Xiaojuan Qi, Xiaoyong Shen, Jianping Shi, and Jiaya Jia. Icnet for real-time semantic segmentation on high-resolution images. In *Proceedings of the European conference on computer vision (ECCV)*, pages 405–420, 2018.

[86] Andrei A. Rusu, Neil C. Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks, 2016.

[87] Running ros across multiple machines. `http://wiki.ros.org/ROS/Tutorials/MultipleMachines`. Accessed: 2021-04-22.

[88] relay. `http://wiki.ros.org/topic_tools/relay`. Accessed: 2021-04-22.

[89] Julius Kammerl Patrick Mihelich. compressed_image_transport. `http://wiki.ros.org/compressed_image_transport`, 2015. Accessed: 2021-04-22.

[90] Frantisek Solc and Jaroslav Sembera. Kinetic model of a skid steered robot. In *Proceedings of the 7th WSEAS International Conference on Signal Processing, Robotics Automation*, pages 61–65, 2008.

[91] Michael Quinlan Jack O'Quin, Patrick Beeson and Yaxin Liu. velodyne. `https://github.com/ros-drivers/velodyne`, 2021.

[92] Péter Fankhauser, Michael Bloesch, Christian Gehring, Marco Hutter, and Roland Siegwart. Robot-centric elevation mapping with uncertainty estimates. In *International Conference on Climbing and Walking Robots (CLAWAR)*, 2014.

[93] Péter Fankhauser, Michael Bloesch, and Marco Hutter. Probabilistic terrain mapping for mobile robots with uncertain localization. *IEEE Robotics and Automation Letters (RA-L)*, 3(4):3019–3026, 2018.

[94] Wim Meeussen Ioan Sucan, Jackie Kay. robot_state_publisher. `http://wiki.ros.org/robot_state_publisher`, 2014. Accessed: 2021-04-22.

[95] Enrique Fernandez. twist_mux. `http://wiki.ros.org/twist_mux`, 2014. Accessed: 2021-04-22.

[96] KJ Astrom and RM Murray. Feedback systems-an introduction for scientists and engineers, version v 2.10 c, 2010.

[97] John G Ziegler, Nathaniel B Nichols, et al. Optimum settings for automatic controllers. *trans. ASME*, 64(11), 1942.

[98] Richard Bellman. I. flügge-lotz, discontinuous automatic control. *Bulletin of the American Mathematical Society*, 60(4):401–402, 1954.

[99] Ulisse Perusin et al. rviz. `https://github.com/ros-visualization/rviz`, 2014.

[100] Riccardo Giubilato, Sebastiano Chiodini, Marco Pertile, and Stefano Debei. An evaluation of ros-compatible stereo visual slam methods on a nvidia jetson tx2. *Measurement*, 140:161–170, 2019.

[101] Joonas Lomps, Artjom Lind, and Amnir Hadachi. Evaluation of the robustness of visual slam methods in different environments. *arXiv preprint arXiv:2009.05427*, 2020.

[102] David Prokhorov, Dmitry Zhukov, Olga Barinova, Konushin Anton, and Anna Vorontsova. Measuring robustness of visual slam. In *2019 16th International Conference on Machine Vision Applications (MVA)*, pages 1–6. IEEE, 2019.

[103] Jabulani K. Makhubela, Tranos Zuva, and Olusanya Yinka Agunbiade. A review on vision simultaneous localization and mapping (vslam). In *2018 International Conference on Intelligent and Innovative Computing Applications (ICONIC)*, pages 1–5, 2018.

[104] S. Park, T. Schps, and M. Pollefeys. Illumination change robustness in direct visual slam. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4523–4530, 2017.

[105] P. Kim, B. Coltin, O. Alexandrov, and H. J. Kim. Robust visual localization in changing lighting conditions. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5447–5452, 2017.

[106] H. Zhou, D. Zou, L. Pei, R. Ying, P. Liu, and W. Yu. Structslam: Visual slam with building structure lines. *IEEE Transactions on Vehicular Technology*, 64(4):1364–1375, 2015.

[107] Khalid Yousif, Alireza Bab-Hadiashar, and Reza Hoseinnezhad. 3d slam in texture-less environments using rank order statistics. *Robotica*, 35(4):809, 2017.

[108] Xun Wang and JianGuo Wang. Detecting glass in simultaneous localisation and mapping. *Robotics and Autonomous Systems*, 88:97–103, 2017.

[109] H. S. Lee, J. Kwon, and K. M. Lee. Simultaneous localization, mapping and deblurring. In *2011 International Conference on Computer Vision*, pages 1203–1210, 2011.

[110] Ludovico Orlando Russo, Giuseppe Airo Farulla, Marco Indaco, Stefano Rosa, Daniele Rolfo, and Basilio Bona. Blurring prediction in monocular slam. In *2013 8th IEEE Design and Test Symposium*, pages 1–6. IEEE, 2013.

[111] Alberto Pretto, Emanuele Menegatti, Maren Bennewitz, Wolfram Burgard, and Enrico Pagello. A visual odometry framework robust to motion blur. In *2009 IEEE International Conference on Robotics and Automation*, pages 2250–2257. IEEE, 2009.

[112] Australian Government Department of the Environment and Water Resources. Resource and training kit - lighting - 5 assessing lighting savings. `https://web.archive.org/web/20070415151053/http://www.greenhouse.gov.au/lgmodules/wep/lights/training/training9.html`, May 2005.

[113] Orbbec3d. Astra technical specs. `https://orbbec3d.com/product-astra/`.

[114] Stereolabs. Zed docs-stereolabs. `https://www.stereolabs.com/docs/`, 2018.

[115] Intel. *Intel RealSenseTM Product Family D400 Series*, 6 2020. Revision 009.

[116] Krystof Litomisky. Consumer rgb-d cameras and their applications. *Rapport technique, University of California*, 20:28, 2012.

[117] Rahul Kala. 2 - basics of autonomous vehicles. In Rahul Kala, editor, *On-Road Intelligent Vehicles*, pages 11–35. Butterworth-Heinemann, 2016.

[118] M. Bloesch, S. Omari, M. Hutter, and R. Siegwart. Robust visual inertial odometry using a direct ekf-based approach. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 298–304, 2015.

[119] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[120] Mathieu Labb. Rtab-map: Robust graph optimization with gtsam and vertigo sep. 4, 2015. [YouTube video].

[121] N Sünderhauf and P Protzel. Vertigo: Versatile extensions for robust inference using graph optimization. *URL: http://openslam. org/vertigo. html*.

[122] Like Cao, Jie Ling, and Xiaohui Xiao. Study on the influence of image noise on monocular feature-based visual slam based on ffdnet. *Sensors*, 20(17):4922, 2020.

[123] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[124] Paolo Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia. Meshlab: an open-source mesh processing tool. In *Eurographics Italian Chapter Conference*, 2008.

[125] HD Herath, S Kodagoda, and G Dissanayake. *Stereo vision based SLAM: Issues and solutions*. ITECH, 2007.

[126] Han Hu, Kaicheng Zhang, Aaron Hao Tan, Michael Ruan, Christopher Agia, and Goldie Nejat. A sim-to-real pipeline for deep reinforcement learning for autonomous robot navigation in cluttered rough terrain. *IEEE Robotics and Automation Letters*, 6(4):6569–6576, 2021.