SuperEdge: A CNN-Based Approach for Quadruped Traversability Analysis from Incomplete Observations

William Huey
Cornell University
wph52@cornell.edu

1 Introduction

Quadruped robots have the potential to revolutionize search and rescue, construction site and factory inspection, delivery services, entertainment, and the exploration of dangerous areas. Most of these applications depend on navigating challenging unknown environments with variable terrain. In recent years, large advances have been made in quadruped locomotion over rough terrain [4] [3]. Nevertheless, as with all locomotion policies, these are not infallible. In order to navigate an environment, it is important to be able to predict when the policy will fail, so that the robot can plan around these areas. A significant amount of research has been done in the area of terrain analysis, where robots are equipped with algorithms that can detect and classify terrain features such as rocks, gravel, and steep inclines. However, traditional methods for terrain analysis often require a significant amount of labeled data, which can be both expensive and time-consuming to collect. Furthermore, they often require user intuition with regards to the maximum allowable parameters of the robot and the features that correspond to difficult terrain.

Image based approaches can be used to detect difficult terrain [6], but it is challenging to map images of terrain to specific areas in 3d coordinates. Furthermore, in an image that contains many different types of terrain, it is necessary to manually label masks, which is time consuming and expensive. Lidar is a technology that uses laser light to measure distances to objects and create a 3D map of the environment. It has been traditionally used in applications such as surveying, autonomous vehicles, and mapping. Lidar can provide the precise locations of features in space, which is useful when determining traversable terrain at a high resolution. In the past, the high cost of lidar systems has been a barrier to wider adoption in many industries. Recent developments have reduced the cost of lidar technology, allowing it to be equipped on smaller and cheaper robotic systems. Currently, the three main quadruped robots for industry applications are Boston Dynamics' Spot, Anybotics' Anymal, and Unitree's Go, all of which have support for a lidar attachment.

In a review of common methods for traversability analysis, Sevastopoulos and Konstantopoulos found that learning-based methods usually necessitate either self-supervised learning or automatically generated data [5]. This is due to the wide variations in terrain for different applications and the cost associated with labeling data. Lee demonstrated a self-supervised learning approach, where a human initially defines thresholds for steps that the robot can take [2]. Traversability grids are then labeled based on this heuristic, and added to the set of training data for a supervised learner. Suger took an approach which does not use any heuristic for labeling, but does require a human to control the robot in the environment in which it will predominantly be operating [7]. Chavez-Garcia presented a method that classifies simulated height maps using a CNN [1].

To address the issues associated with traversability analysis, this paper introduces SuperEdge, a novel supervised learning method for quadruped traversability analysis in difficult terrain. SuperEdge uses a height map generated from a point cloud to classify simulated terrain as being traversable or untraversable. It builds on Chavez-Garcia by including partial observations of the environment, an ability to "look ahead" along its current path, and a lazy evaluator which does not require computing a full traversability grid. This approach is able to learn from large amounts of automatically labeled

data collected in simulation, making it both scalable and cost-effective. This paper describes the SuperEdge algorithm and evaluates it on procedurally generated terrain in Nvidia Isaac Sim using Anybotics' Anymal quadruped robot.

2 The SuperEdge Algorithm

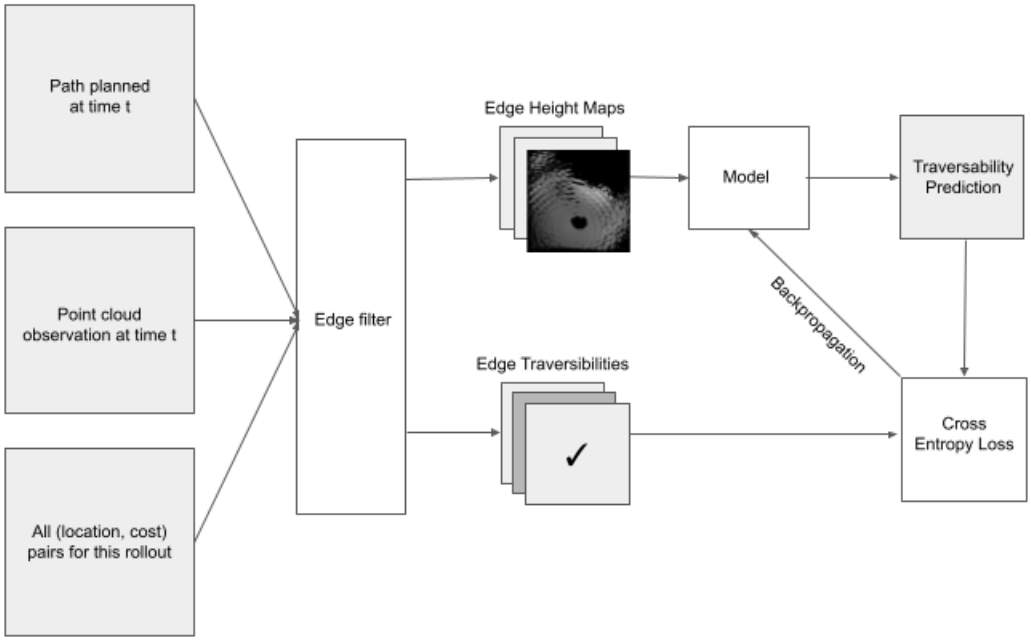

Figure 1: The SuperEdge algorithm. In order to predict edge costs instead of traversability classes, the loss function should be changed to Mean Squared Error.

In the SuperEdge algorithm, the robot is first initialized to a specific starting location in the environment and given a random goal location to reach. It is also given a replanning frequency and an environment sampling frequency. Each time the robot replans, it saves its current observation of the world and the areas of the world that it plans to traverse. The robot's observation of the world is represented as a point cloud, and the areas that it plans to traverse are represented as edges in a graph over the world. Each time the robot samples its environment, it saves its location in the environment and its current observed cost, and it accumulates points from the current scan of the world. Finally, a model is trained to predict the cost of each edge in the path, given the robot's incomplete observation of the environment when it planned that path. The cost function is defined by the user as a metric for how traversable the terrain is. If it is defined as the cost that the locomotion policy was trained on, this method can be interpreted as meta-learning on top of the locomotion policy .

After the data have been collected, a model can be trained using traditional supervised learning techniques. In this paper, the cost along each edge was converted to a binary classification corresponding to whether or not the robot fell. Edges that were only minimally traversed were removed from the dataset. There were two main reasons for this: either the robot replanned or it fell before starting to traverse the edge. Finally, a CNN was trained to classify the observed height fields as being traversable or not traversable. Figure 1 shows how the algorithm works, and Figure 2 shows the architecture of the CNN that was used for the purposes of this paper. The pseudocode for the SuperEdge data collection is included in Algorithm 1.

In SuperEdge, the points corresponding to an edge are defined as the points of distance at most R from the edge center, where R is a hyperparameter, and the distance is taken over the x and y axes only. The L1 distance naturally creates a square frame, which can easily be transformed into a square height field by averaging the heights of points that lie in the same cell of the height field. This method

Algorithm 1 SuperEdge

procedure COLLECT DATA(T, Planning frequency, Sample frequency)
  for T rollouts **do**:
    Reset the location, target, and knowledge of the robot
    if replanning on the current timestep **then**
      Let P be the path from the current location to t
      Let O be the current observation of the world (a global point cloud)
      Aggregate (P, O)
    end if
    if sampling on the current timestep **then**
      Let L be the current (x,y) location of the robot
      Let C be the current locomotion cost
      Aggregate (L, C)
    end if
    if reached target location, fell over, or stopped moving **then**
      Continue to the next rollout
    end if
  end for
end procedure
procedure TRANSFORM DATA(All rollout data)
  for each rollout in Rollout Data **do**:
    for each edge in the set of paths for this rollout **do**
      Get the points in the observation corresponding to the edge
      Transform these points into a height field
      Normalize the height field to get an image array
      Get the traversed locations corresponding to the edge
      Average the cost over all these locations
      Save (height field, average cost) for this edge
    end for
  end for
end procedure

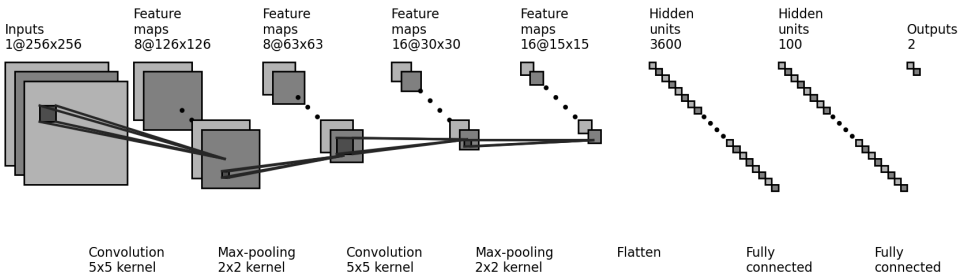

Figure 2: Architecture of the Convolutional Neural Network used to predict whether an edge is traversable. This figure is generated by adapting code from https://github.com/gwding/draw_convnet

is not without its faults. For example, if passing under a roof, the average height of points at a given location will not be a useful measure. However, cases like these can be avoided using heuristics to prune certain points (such as the direction of their normals, or their distance from other points at that location).

In order to get the best use out of a CNN, it was necessary to construct a square height map of constant dimension. However, edges are of variable length, so it is impossible to store the information corresponding to different edges in an array of the same dimension without some loss of information.

The method described above makes the assumption that the points lying in a square around the center of an edge are representative of the edge as a whole. In most robotic applications, it is necessary to create a relatively dense graph representing the environment. Otherwise, the planner may fail to find a path in an environment with many obstacles. Thus, edge lengths tend to be relatively short compared to the entire path, so it is reasonable to assume that the terrain will not vary too much along an edge.

For the sake of simplicity and performance, SuperEdge is described here in an offline, off-policy manner. It could easily be trained online by calculating the points and costs corresponding to edges and backpropagating through the predictor on each rollout. However, training on-policy would break the assumption that data are i.i.d., because the training data used to update the policy would become dependent on the policy itself. This could result in the robot failing to fully explore the environment, thus losing the ability to classify certain difficult terrain.

3 Results

In order to evaluate the SuperEdge algorithm in simulation, Nvidia Isaac Sim was used to simulate an Anybotics Anymal quadruped traveling over varying terrain. Each rollout, the Anymal was initialized to a starting location in the center of the terrain, and given a random goal in the outside border of the terrain. A 2D graph was constructed over the environment using Halton sampling. This was chosen instead of a grid-based approach in order to achieve smoother and more realistic paths. Paths were planned using A^* and tracked using the Pure Pursuit algorithm. Figure 3 displays the setup of the simulation.

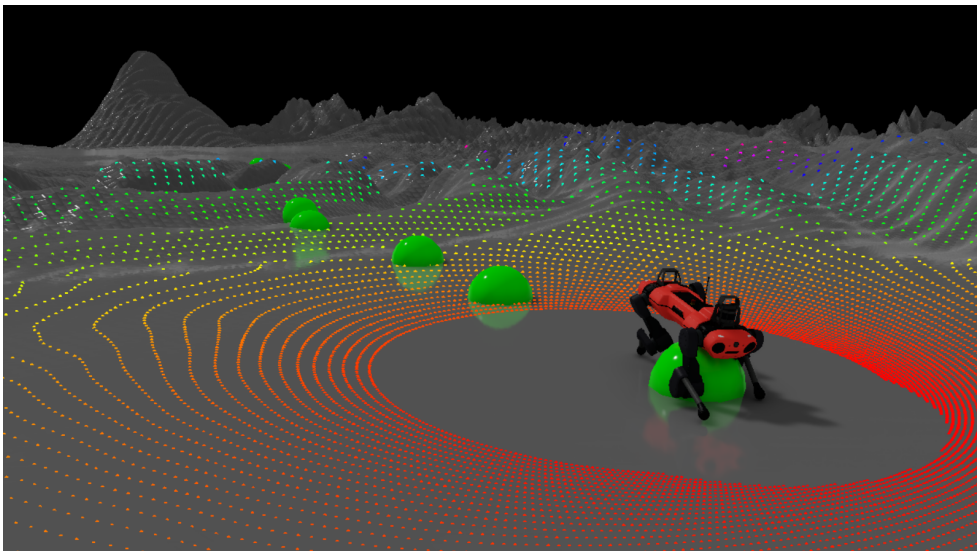

Figure 3: An Anymal traversing one of the training environments in Nvidia Isaac Sim. The large green spheres are the nodes in the path that the Anymal is tracking, and the smaller multicolored points are the lidar observation at the current time step.

The data collection algorithm generally resulted in clear and interpretable height fields, but there were a few edge cases. Examples of height fields and corresponding ground truth terrain are shown in Figure 4. The two nodes corresponding to the edge being evaluated are represented as white spheres, and the corresponding height maps are shown on the right. It is clear that the top height field represents bumpy terrain, and the center height field represents smooth terrain. However, it is difficult to tell where the features in the bottom height field are coming from. In fact, when this height field was generated, the robot was far away from the edge, and only had a limited observation of it. It is extremely important to be able to classify observations with limited data in order to maximize the "look-ahead" distance of the edge evaluator. If the algorithm is only able to predict the costs of edges for which it has a complete observation (and is thus extremely close to), then it will not be very useful when integrated into a long horizon path planner.

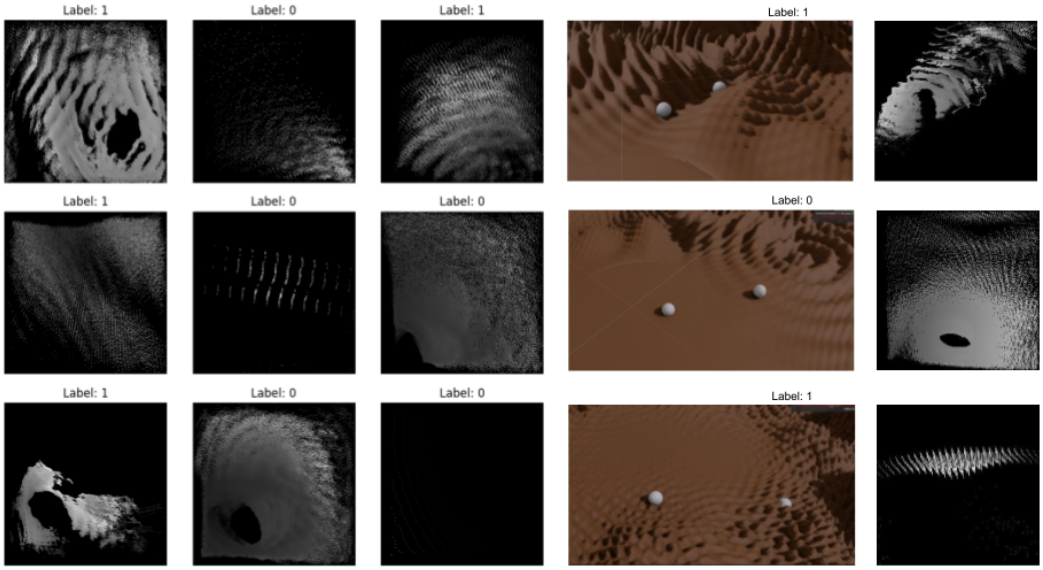

Figure 4: Shown on the left are 9 samples of height maps and labels from the training set. A label of 1 corresponds to untraversable terrain. On the right there is ground truth terrain for three different edges and the height map in the training set saved for each edge. The edge is a straight line between the two white spheres. There is an example of bumpy and untraversable terrain (top), smooth and traversable terrain (center), and a poor observation of bumpy, untraversable terrain (bottom).

The initial classifier was based on the ubiquitous CNN architecture: 2 convolutional layers separated by max pooling and flattening, and 3 fully connected layers with ReLU activation in between. In order to better understand the performance of the model, ablation studies were performed. The last convolutional layer was removed, then all convolutional layers were removed, and finally all layers were removed except for one linear layer (resulting in a single layer perceptron). The classifiers were trained on a set of 796 examples collected through simulation, of which 34% were non-traversable. They were then tested on a set of 400 examples collected through simulation on height fields generated using various initial parameters, to evaluate the generalization ability.

Figure 5 shows the performance of different model architectures. The ROC curves show the true positive vs. false positive rates of the model under different thresholds for a positive classification (that is, a classification of non-traversable). As the threshold increases, the total number of positive classifications decreases, reducing both the true positive rate and the false positive rate. Similarly, the recall will increase as more positives are chosen from the total set of positives, but the precision will decrease as there are more false positives. The AUC is defined as the area under the ROC curve, and the AP (Average Precision) is the area under the Precision-Recall curve. In this case, a naive classifier would have an expected AUC of .5 and an expected AP of .34. The full model and model with the last convolution removed have a similar AP score, but the full model has a slightly better AUC.

Surprisingly, both the MLP with no convolutions and the linear classifier are only slightly worse than the CNNs. Since these models just take the flattened image data as input, this implies that the predictions are being made primarily based on the heights of the pixels, rather than spatial features associated with bumps, edges, or slope. One potential reason for this is that traversable terrain generally consists of lower heights relative to the robot. Another reason could be that the robot takes longer to traverse rougher terrain (before eventually falling), meaning it gets a more complete set of points for the area. Thus, the point clouds for traversable terrain are generally more sparse, which is a pattern that can be learned by a linear classifier.

4 Discussion

One major benefit of the SuperEdge algorithm is that it can be trained out of the box on top of any locomotion framework. Certain locomotion methods may be better for steeper slopes, cluttered

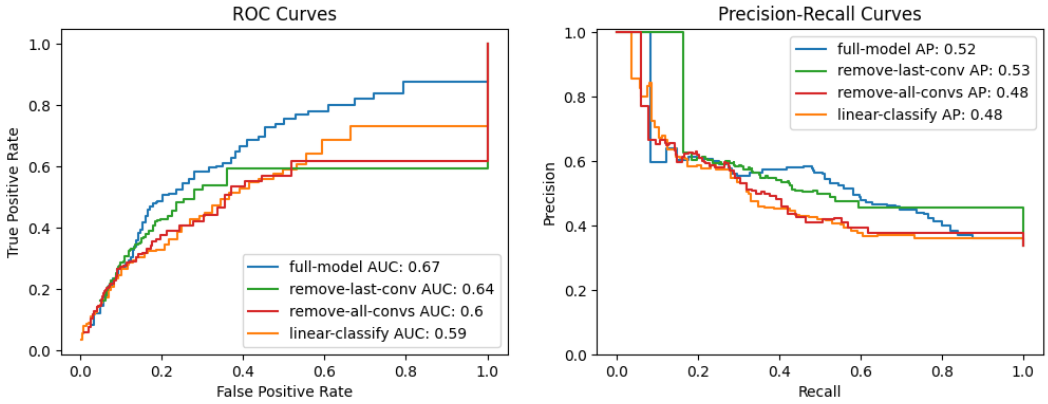

Figure 5: ROC and Precision-Recall curves for various classifier architectures. The test set was composed of 34% positive samples.

ground, or frictionless surfaces. A heuristic edge evaluator (for example, one that extracts features corresponding to slope and roughness) would require a significant amount of tuning to find the best way to weight each feature for a given locomotion policy. To train SuperEdge, any locomotion policy can just be rolled out in simulation, with no need to tune any hyperparameters.

An initial idea for traversability analysis was a model that predicts the cost of an edge using only the variance of the points along that edge. Intuitively, it seems like rougher terrain should by definition have higher variance. However, as is shown in Figure 6, the variances of traversable and non-traversable terrain are almost identically distributed, so this is unlikely to provide a useful classifier. There are two potential reasons for this: a) large amounts of noise in the data resulted in high variance regardless, or b) traversability depends on slope, frequency, and location of rough features, not just the variance. Because the terrain could be easily represented as a height field, it seemed natural to use a CNN for a learning based approach.

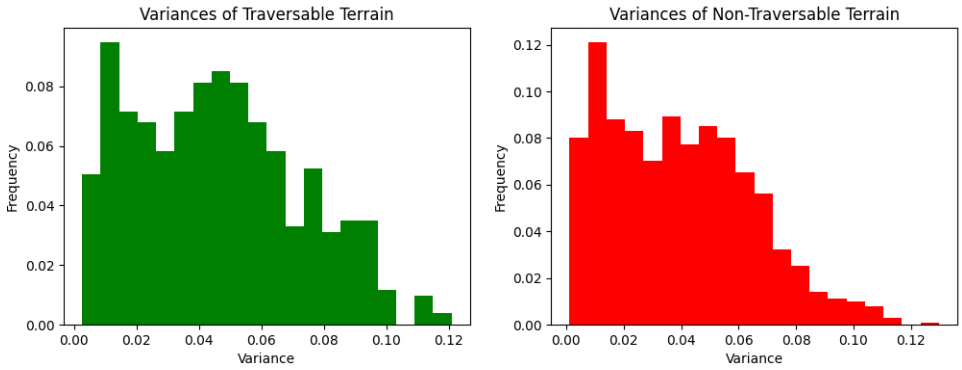

Figure 6: Distribution of variances for traversable and non-traversable terrain

In these experiments, the cost was defined as 10,000 if the robot fell while traversing an edge, and 0 otherwise (the cost was later converted to a binary classification). An interesting follow up would be to use the cost that the locomotion policy was trained on. This would allow the robot to not only predict whether it is going to fall, but also predict how difficult it will be to traverse the edge according to its current locomotion policy. This would potentially allow SuperEdge to be trained in the real world. In simulation, the robot can be rolled out in extremely challenging environments that force it to fall. In real life, any fall can be dangerous and expensive. However, difficult terrain requires more energy, time, and is more risky to cross, even if it usually doesn't make the robot fall over. Furthermore, it is impossible to construct a perfect simulation, especially with high resolution

features like pebbles, gravel, or sand. Thus, it is important that SuperEdge can be trained online and in deployment.

There are two natural next steps to the results of this experiment. First, it is important to improve the quality of the training data. In this implementation, the accumulated points are stored in a buffer, so the robot "forgets" points from long ago. This is meant to reduce the total number of points in memory and increase performance, but it means that the robot does not remember the entire environment. An improvement would be pruning the points as the robot explores (i.e., removing points that are too close together), and increasing the overall size of the memory, so the robot can obtain a better observation. Hopefully, this would help avoid the problems with the CNN architecture described in the results section. Furthermore, it is important to improve the quality of the generated terrain. Currently, the terrain is relatively periodic, and not exactly realistic. Even though past methods have shown decent sim to real transfer using similar terrain, it is still possible that our algorithm is learning to exploit the function that generated the terrain. One potential approach is to construct meshes from point cloud scans of real rough terrain environments, such as construction sites or hiking trails. It would also be interesting to add large obstacles on top of the rough terrain. Obstacles would block part of the observation of the world and add noise to the point cloud, which could limit the performance of SuperEdge. Finally, increasing the amount of training data and adding a mix of simulated and real training data could help improve the model's performance.

Second, it would be interesting to integrate the trained SuperEge model into the motion planning framework and evaluate its effectiveness at choosing paths against a naive Anymal. One potential issue is that the look ahead horizon for the robot is too short. In general, the dataset only contained the next 3-4 edges in the path. Looking any further ahead, the Anymal did not have sufficient point cloud data to make inferences about those edges, so these edges were removed from the training set. This would be necessary in order to integrate SuperEdge into a real time path planner. Increasing the resolution of the planning graph could also improve the robot's ability to plan into the future, because it would have more edges with a sufficient observation to make inference on.

5 Contributions

This work was done under Dr. Don Greenberg (dpg5@cornell.edu) as part of a larger research project towards generating digital twins of construction sites. The idea for this specific project came from a conversation with Dr. Choudhury during office hours. There are two other students working on robotics problems in the Greenberg lab: Leul Testefaye (graduate student, lst26@cornell.edu) and Sean Brynjolffson (undergraduate, smb459@cornell.edu). Leul provided help setting up and debugging the simulation. The terrain generation algorithm was developed and implemented entirely by Sean, which is why the method is not described in this paper. I had invaluable conversations with both of them about the idea behind the algorithm and potential implementations. The architecture of the CNN was heavily inspired by A4 in this class. Otherwise, the SuperEdge algorithm, the implementation of the motion planning framework/algorithms, the simulation, and this paper are entirely my own work. If you would like to see the code, please reach out to me at wph52@cornell.edu.

References

[1] R. Omar Chavez-Garcia, Jérôme Guzzi, Luca Maria Gambardella, and Alessandro Giusti. Learning ground traversability from simulations. *IEEE Robotics and Automation Letters*, 3:1695–1702, 2017.

[2] Hyunsuk Lee and Woojin Chung. A self-training approach-based traversability analysis for mobile robots in urban environments. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3389–3394, 2021.

[3] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science Robotics*, 5(47), October 2020.

[4] Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. In *5th Annual Conference on Robot Learning*, 2021.

[5] Christos Sevastopoulos and Stasinos Konstantopoulos. A survey of traversability estimation for mobile robots. *IEEE Access*, 10:96331–96347, 2022.

[6] G. Somua-Gyimah, S. Frimpong, W. Nyaaba, and E. Gbadam. A computer vision system for terrain recognition and object detection tasks in mining and construction environments. In *Proceedings of the 2019 SME Annual Conference and Expo and CMA 121st National Western Mining Conference (2019, Denver, CO), Society for Mining, Metallurgy and Exploration (SME)*, February 2019.

[7] Benjamin Suger, Bastian Steder, and Wolfram Burgard. Traversability analysis for mobile robots in outdoor environments: A semi-supervised learning approach based on 3d-lidar data. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3941–3946, 2015.