Deep Reinforcement Learning for Efficient Navigation of the Moose Robot

Constança Fernandes Isabel Sá Pedro Silva

June 10, 2025

Abstract

This paper presents a deep reinforcement learning (DRL) approach for efficient navigation of the Moose robot in complex environments with unknown rough terrain. We train an Advantage Actor-Critic (A2C) model to optimise the robot's trajectory by minimising energy consumption while maintaining performance. The proposed solution is validated in simulation using diverse terrain maps, and its effectiveness is compared with traditional planning algorithms such as Greedy and A*.

1. Introduction

Mobile robots face significant challenges when navigating rough terrain, particularly in terms of balancing energy efficiency with robustness and speed. The Moose robot, equipped with various sensors, provides a suitable platform for exploring intelligent navigation strategies. The aim of this work is to train a DRL model that allows Moose to autonomously reach target locations efficiently, using minimal energy.

2. Related Work

Several approaches have been proposed for mobile robot navigation, including traditional path planners like A* and greedy heuristics. These algorithms often rely on complete terrain information and do not explicitly optimize for energy efficiency. More recently, deep reinforcement learning has shown promise in solving complex control problems. Notably, Zhang et al. [1] introduced a DRL-based framework for rough terrain navigation, demonstrating the potential of neural policy learning approaches in this domain.

3. Methodological Approach

3.1. Terrain Generation Using Perlin Noise

To support the evaluation of navigation strategies under diverse topographic conditions, terrain surfaces are procedurally synthesised using Perlin noise — a gradient-based method known for generating smooth, naturalistic elevation profiles. This approach enables the creation of continuous, non-repetitive landscapes that emulate real-world geographical features such as slopes, hills, and valleys.

The simulation environment is constructed within *Webots* and incorporates a 200×200 elevation grid that defines a three-dimensional surface. Terrain generation is based on a custom implementation of 2D Perlin noise, configured with a single octave and no bias correction. The resulting noise values are scaled to enhance terrain features, yielding realistic elevation differences suitable for evaluating the robot's mobility and energy usage.

To integrate the generated elevation data, the world model is programmatically modified by replacing the default height map with the synthesised terrain. The region immediately surrounding the robot's starting position is flattened during a post-processing step to ensure stable episode initiation. The robot's initial coordinates are then adjusted to match this modified elevation, ensuring consistency between terrain and spawn location.

The procedural generation pipeline includes tools for analysing and visualising terrain characteristics. Statistical summaries — such as mean, standard deviation, and elevation range — are computed to evaluate the spatial variability of each generated map. Additionally, heatmaps of the elevation surface are produced using perceptually uniform colour gradients, allowing rapid visual inspection of terrain diversity.

This synthetic terrain generation framework supports domain randomisation during training, enabling the learning and planning algorithms to generalise to a wide variety of environmental conditions. By adjusting noise parameters such as scale and frequency, it is possible to control the complexity and steepness of the generated maps, facili-

tating systematic robustness testing of navigation policies.

3.2. Path Planning with A* Algorithm

Navigation commences with a global planning module based on the A* search algorithm, adapted to account for terrain irregularities. The operational environment is discretised into a two-dimensional grid, with each cell assigned a traversal cost influenced by local terrain features, such as slope and elevation.

The pathfinding cost function is defined as:

$$f(n) = g(n) + h(n)$$

where $g(n)$ represents the cumulative cost from the origin to node n, computed as the Euclidean distance scaled by a terrain roughness factor derived from local elevation gradients.

To ensure physically plausible trajectories, impassable areas and steep inclines are assigned prohibitively high traversal costs. The slope at each cell is estimated via numerical gradients over a local elevation window, and terrain flatness is evaluated by checking the alignment of the local surface normal with the vertical axis.

The heuristic term $h(n)$ estimates the remaining cost to the goal:

$$h(n) = \lambda \cdot \sqrt{(x_g - x_n)^2 + (y_g - y_n)^2}$$

where λ modulates the influence of distance based on estimated terrain roughness. This formulation guides the planner towards geometrically direct yet traversable paths that respect the robot's locomotion constraints.

This enhanced cost model enables the planner to favour routes that avoid steep or unstable regions, promoting both efficiency and safety in terrain-constrained environments.

3.3. Robot Control via A2C

An alternative control architecture employs the Advantage Actor-Critic (A2C) algorithm. In this setup, a reinforcement learning (RL) agent interacts with the robot via a TCP/IP socket interface, enabling external control of motor velocities based on continuous feedback from the environment.

Action Space The robot operates in a continuous action space:

$$a = [v_{\text{left}}, v_{\text{right}}], \quad v \in [-1, 1]$$

where v_{left} and v_{right} represent the normalised velocities of the left and right wheels, respectively.

Observation Space The agent receives a six-dimensional observation vector consisting of:

- 3D position (x, y, z) obtained via GPS.
- Orientation (ϕ, θ, ψ) (roll, pitch, yaw) obtained from the IMU.

This state representation provides sufficient spatial awareness for navigating uneven terrain.

Reward Function The reward signal R_t at time step t is defined as:

$$R_t = \begin{cases} 0.5 + \frac{d_{\text{start}}}{T_{\text{total}}} & \text{if goal reached} \\ \mu \cdot \frac{d_{\min} - d_t}{\Delta t} & \text{if progress is made} \\ -0.5 & \text{if stuck too long} \\ 0 & \text{otherwise} \end{cases}$$

where d_t is the current distance to the goal, $d_{\min}$ is the minimal distance reached so far, T_{total} is the elapsed time, and μ is a scaling factor.

3.4. Reward Structure and Termination Conditions

The reward function is designed to promote efficient and goal-directed navigation:

- **Success:** Bonus proportional to initial distance and time efficiency.
- **Progress:** Reward for reducing distance to the goal.
- **Stagnation:** Penalty for extended inactivity.
- **Timeout:** Episode terminates with penalty after 120 seconds without reaching the goal.

Stuck conditions are detected by monitoring positional changes over time, preventing indefinite non-progress states.

3.5. Reinforcement Learning Environment

The control architecture is implemented as a Gymnasium-compatible environment, `MooseEnv`, which abstracts the Webots simulation and facilitates interaction with standard RL algorithms. It defines:

- A continuous action space for wheel velocities in $[-1, 1]$.
- A continuous observation space with 3D position and orientation.

- `reset` and `step` methods that manage simulation episodes via socket-based communication.

This modular interface enables seamless experimentation with various RL algorithms, decoupling control policies from simulation logic.

3.6. Reinforcement Learning Training Setup

Training is conducted using the `Stable Baselines3` implementation of A2C. The script `train.py` automates the training workflow:

- Initializes the `MooseEnv` environment wrapping the Webots simulation.
- Creates the directory `models/a2c_moose` for storing model checkpoints.
- If a pretrained model (`a2c_moose_model.zip`) is found, it resumes training from that checkpoint.
- Otherwise, it initializes a new A2C agent with an MLP policy, learning rate of 1×10^{-4}, and TensorBoard logging enabled.
- The agent is trained for 100,000 timesteps with verbose logging.
- Upon completion, the model is saved to disk.

4. Experimental Evaluation

Experiments were conducted over ***** independent runs within a simulated terrain environment. Key performance metrics included total distance travelled, execution time, average speed, battery usage, and altitude variation.

- The A* algorithm consistently found viable paths avoiding excessively steep slopes.
- Battery consumption correlated strongly with terrain elevation changes, validating the proposed energy model.
- Dynamic re-routing improved success rates by recovering from navigation failures.

5. Conclusions and Future Work

This study demonstrates the feasibility of using deep reinforcement learning to improve efficiency in robot navigation. While DRL outperforms heuristic-based approaches in varied environments, future work should focus on transfer learning for real-world deployment and adapting the model for dynamic obstacles.

References

[1] Kaichena Zhang, Farzad Niroui, Maurizio Ficocelli, and Goldie 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.