Deep Reinforcement Learning for Efficient Navigation of the Moose Robot

Constança Fernandes Isabel Sá Pedro Silva

June 11, 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 while maintaining navigation 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 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, optimising its trajectory for reliable navigation.

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 optimise trajectory quality in uncertain environments. 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 in diverse topographical conditions, terrain surfaces are procedurally synthesised using Perlin noise — a gradient-based method known for generating smooth and naturalistic elevation profiles. This approach allows for the creation of continuous, non-repetitive landscapes that emulate real-world geographical features such as slopes, hills, and valleys.

The simulation environment is built within Webots and incorporates a 200×200 at 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 improve terrain characteristics, producing realistic elevation differences suitable for evaluating the robot's mobility and navigation performance.

To integrate the generated elevation data, the world model is modified programmatically, replacing the standard height map with the synthesised terrain. The region immediately surrounding the robot's initial position is flattened during a post-processing step to ensure a stable start to the episode. The robot's initial coordinates are then adjusted to match this modified elevation, ensuring consistency between the terrain and the 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 calculated to assess the spatial variability of each generated map. In addition, elevation surface heatmaps are produced using perceptually uniform colour gradients, allowing for quick visual inspection of terrain diversity.

This synthetic terrain generation framework supports the randomisation of domains important for training, allowing 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 slope of the generated maps, facilitating systematic testing of the robustness of navigation policies.

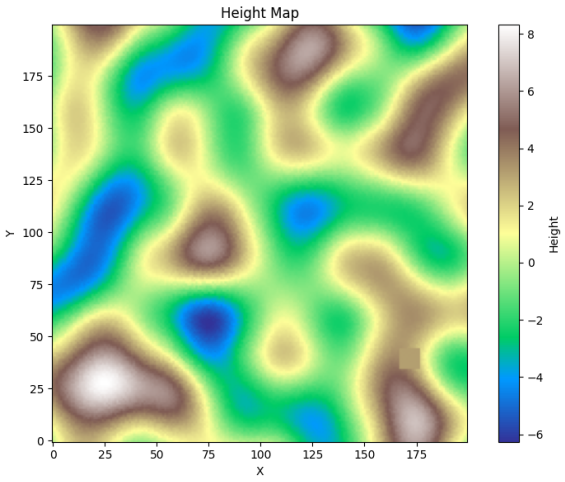

Figure 1: Heatmap visualization of the generated terrain elevation using Perlin noise. The colour gradient reflects elevation levels, with warmer colours indicating higher elevations.

3.2. Greedy Navigation Controller

As a baseline control strategy, a rule-based greedy navigation controller was implemented to guide the robot towards the target using only local sensor data.

At each simulation step, the robot estimates its current heading and position relative to the goal using data from its GPS and IMU sensors. The heading angle is extracted from the yaw component of the IMU, while the displacement vector to the goal is derived from GPS coordinates.

The controller prioritises actions that reduce the Euclidean distance to the target:

$$d_t = \sqrt{(x_g - x_t)^2 + (y_g - y_t)^2}$$

It then calculates the angular deviation $\Delta\theta$ between the robot's orientation and the vector to the goal. If $|\Delta\theta| > 0.01745$ radians (1 degree), the robot rotates in place to align with the goal direction. Once alignment is achieved, it proceeds forward with a constant velocity.

In addition to heading control, the robot monitors its pitch and roll values continuously. If these exceed a predefined inclination threshold (indicating unsafe slopes), the robot prioritises stability by entering a recovery manoeuvre: it induces rotation in the direction opposite to the tilt. This behaviour prevents traversal of steep regions and reduces the risk of falling or flipping, thereby improving navigation safety.

Recovery Behaviour To detect stagnation or local minima, the controller compares successive GPS positions. If minimal displacement is observed over a defined window, the robot performs a rotational escape routine. This heuristic aims to dislodge the robot from potentially blocked or unstable terrain.

Termination Criteria An episode terminates under the following conditions:

- The robot reaches within a specified proximity to the goal.
- A timeout of 120 seconds is exceeded.

Despite its simplicity, this greedy strategy establishes a valuable performance baseline. It enables comparisons against planning-based and learning-based approaches under diverse terrain conditions.

3.3. 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.

Each cell considers an **8-connected neighborhood**, allowing transitions in both axial and diagonal directions. Diagonal movements incur higher traversal costs, reflecting their increased Euclidean distance ($\sqrt{2}$) relative to axial steps. This preserves geometric consistency and penalises sharp turns on rugged terrain.

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 evaluatedby measuring the terrain gradient derived from local elevation differences.

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.

3.4. 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 normalized 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.

This reward function follows the design proposed by Zhang et al. [1].

3.5. 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.

3.6. 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.

3.7. 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

This section presents the evaluation results of the navigation algorithms Greedy and A* over 100 runs (10 runs on each of 10 different terrain maps). The performance metrics considered include planned distance, actual distance traveled, completion time, average speed, and navigation outcome (success or failure).

4.1. Greedy Algorithm

Table 1 shows the aggregated statistics for the Greedy algorithm across the 100 runs.

Table 1: Summary statistics for the Greedy algorithm (100 runs).

Metric	Mean	Std Dev	Min	Median	Max
Distance (m)	109.48	40.18	29.89	100.50	190.34
Distance Traveled (m)	110.44	40.10	22.09	103.63	190.55
Time (s)	117.41	43.64	20.03	105.15	211.72
Average Speed (m/s)	0.98	0.21	0.57	0.97	1.53
Result (Success=1)	0.80	0.40	0	1	1

The Greedy algorithm achieved a success rate of 80%, with an average completion time of 117.41 seconds. The average speed was 0.98 m/s. The small difference between planned and traveled distance indicates generally efficient path following, although some variability is present.

4.2. A* Algorithm

Table 2 presents the summary statistics for the A* algorithm over the same set of runs.

Table 2: Summary statistics for the A* algorithm (100 runs).

Metric	Mean	Std Dev	Min	Median	Max
Distance (m)	112.02	37.11	52.23	103.33	179.67
Distance Traveled (m)	113.67	37.56	52.23	103.34	183.99
Time (s)	109.03	39.89	33.32	94.00	214.97
Average Speed (m/s)	1.04	0.19	0.58	1.05	1.65
Result (Success=1)	0.85	0.36	0	1	1

The A* algorithm demonstrated a higher success rate of 85% and slightly better average speed (1.04 m/s) compared to Greedy. The average completion time was lower at 109.03 seconds, reflecting the efficiency of the path planning.

4.3. Comparison and Discussion

The comparative analysis indicates that the A* algorithm outperforms the Greedy method in terms of success rate, speed, and completion time. Although distances planned and traveled are comparable, the lower time and higher speed achieved by A* suggest more efficient navigation paths and better adaptation to terrain.

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.