Rough Terrain Navigation for a Quadruped Robot Using Deep Reinforcement Learning-based Blind Locomotion Control and a Stuck-Escape Strategy

Kiyoshi Irie and Tomoaki Yoshida and Takaaki Matsuzawa and Taro Suzuki and Yoshitaka Hara and Masahiro Tomono

Future Robotics Technology Center, Chiba Institute of Technology, 2-17-1, Tsudanuma, Narashino, Chiba, 275-0016, Japan

ARTICLE HISTORY
Received 01 Jun 2025, Accepted 18 Aug 2025, Published online: 13 Oct 2025.

This is an original manuscript of an article published by Taylor & Francis in Advanced Robotics. The final published version is available at `https://doi.org/10.1080/01691864.2025.2561643`

ABSTRACT
This paper presents our navigation system for rough terrains using a quadruped robot. Our system is based on the commercial quadruped robot Unitree Go2, equipped with our deep reinforcement learning-based locomotion control. While the locomotion control is capable of walking on rough terrains, in challenging situations, the robot may get stuck. To address this, we developed a randomized escape strategy that allows the robot to recover from these situations. We tested our system in the IEEE RAS Quadruped Robot Challenge at ICRA 2024, where it successfully navigated a complex course and achieved the highest overall score among six competing teams.

KEYWORDS
Quadruped robots; Terrain navigation; Deep reinforcement learning

1. Introduction

Quadruped robots are expected to be effective for rough terrain scenarios, and recently started being used in real applications [1]. While significant advancements in locomotion control using deep reinforcement learning have been made in a decade [2] [3], integrating these techniques into robust autonomous navigation systems remains challenging. Although some studies report experiments conducted in environments such as low mountains, industrial plants, and mining areas [4] [5] [6], the solutions employed often vary widely depending on the robot and the environment, and no universally applicable method for autonomous navigation has been established.

To help address these challenges, the IEEE RAS Quadruped Robot Challenge (QRC) was organized, aiming to offer a platform for development and evaluation of quadruped robots for rough terrain navigation [7]. Motivated by this competition,

CONTACT Kiyoshi Irie Email: irie@furo.org

we developed our own autonomous navigation system, using a commercially available quadruped robot enhanced with deep reinforcement learning-based locomotion control and high-level perception and planning capabilities. Although we have previously tested our preliminary system in real-world urban environments [8], the QRC competition provided a unique opportunity to evaluate our system in a controlled environment with a standardized course and evaluation criteria.

Our team, fuRo, participated in the QRC at ICRA 2024 in Yokohama, and achieved the highest overall score among the six participating teams. In this paper, we describe the system we developed, present the experimental results from the competition, and discuss the challenges and limitations we encountered.

2. Related Work

In recent years, there has been significant progress in quadruped robot technologies, both in terms of hardware and software. Quadruped platforms such as Boston Dynamics' Spot, ETH's ANYmal [9], and MIT Cheetah 3 [10] have demonstrated impressive performance. Additionally, more affordable commercial quadruped robots have been released from companies such as Unitree and Ghost Robotics, making quadruped robot research more accessible to academia and industry.

2.1. Quadruped Locomotion Control using Deep Reinforcement Learning

Deep reinforcement learning (DRL) has emerged as a powerful tool for developing robust locomotion controllers for these platforms. A fundamental challenge in applying deep reinforcement learning to quadruped robotics is the requirement for large amounts of training data, which makes direct learning on physical robots impractical. Researchers have developed several approaches to address this challenge.

Haarnoja et al. introduced a sample-efficient learning approach using Soft Actor-Critic (SAC), which enabled policy learning directly on physical robots without simulation [11]. Their method significantly reduced the training time required to achieve effective locomotion behaviors on real hardware, allowing robots to learn walking gaits with minimal intervention.

One approach focuses on simulation-to-reality transfer (sim-to-real). Tan et al. [2] pioneered this direction, developing control policies trained exclusively in simulation that could be successfully deployed on physical robots. They employed domain randomization on physical parameters such as friction, mass, and motor response during simulation training to effectively bridge the reality gap between simulated and physical environments, enabling robust transfer of learned behaviors.

Taking a different approach to the sim-to-real problem, Hwangbo et al. [12] focused on improving simulation fidelity for the ANYmal quadruped robot. Their methodology involved learning actuator characteristics from real-world data and incorporating these models into the simulation environment. This actuator-focused approach enabled more accurate simulation of the robot's dynamics, resulting in more successful policy transfer to the physical platform.

Kumar et al. [13] introduced RMA (Rapid Motor Adaptation), which addresses the challenge through a hybrid approach. Rather than relying solely on pre-trained policies, RMA combines a base policy trained in simulation across various terrains with an online adaptation module that continuously adjusts the policy based on proprioceptive feedback. This approach enables adaptation to unexpected environmental changes

without requiring exhaustive pre-training for every possible condition.

2.2. Quadruped Robot Navigation

Several research groups have applied these locomotion capabilities to practical navigation applications in challenging environments. Bouman et al. presented a system enabling long-range autonomous exploration of extreme environments with Boston Dynamics' Spot robot [6]. Their work demonstrated extended autonomous operation in underground tunnels and complex cave systems, integrating perception, planning, and control for navigation without prior maps or GPS. Miki et al. combined ANYmal's proprioceptive control with visual perception to navigate difficult natural terrains [4]. Their approach used an exteroceptive policy network to process elevation maps from onboard sensors, allowing the robot to adaptively navigate uneven surfaces, dense vegetation, and rocky landscapes.

Building upon these prior studies, we focus on extending autonomous navigation capabilities into more challenging and unpredictable environments. Rather than solely achieving blind locomotion via reinforcement learning, we additionally integrate a failure recovery mechanism that enables the robot to autonomously escape from stuck situations. This combination of deep reinforcement learning-based blind locomotion and a randomized escape strategy results in a resilient navigation system capable of traversing complex terrain without reliance on detailed environmental perception.

3. Quadruped Robot Challenge

3.1. Quadruped Robot Challenge at ICRA 2024

The IEEE RAS Quadruped Robot Challenge was held at ICRA 2024 in Yokohama. This is the second time the competition was held, following its debut at ICRA 2023 in London [7]. This competition aims to promote the development of quadruped robots for applications such as mobility and exploration in rough terrain, including disaster areas.

In the first competition in 2023, the challenge focused on navigating an uneven terrain course, and the teams were divided into autonomous and teleoperation categories. In this year ' s competition, most of the uneven terrain course was retained, but the distinction between the autonomous and teleoperation categories was removed, and a new exploration task was added.

3.2. Overview of the Competition Field

The competition took place on a course with a total length of approximately 30 m, which consisted of five different types of rough terrain modules connected together (Fig. 1). Robots were required to autonomously navigate or be teleoperated through this course within a time limit. In addition, the course included exploration targets, with alphabets placed inside pipes, which the robots had to read.

Crossing Ramps Uneven terrain composed of multiple blocks tilted in different directions.

Soft Form A section of low obstacles placed diagonally on top of soft foam-like material, which causes the robot to sink, making it difficult to maintain balance.

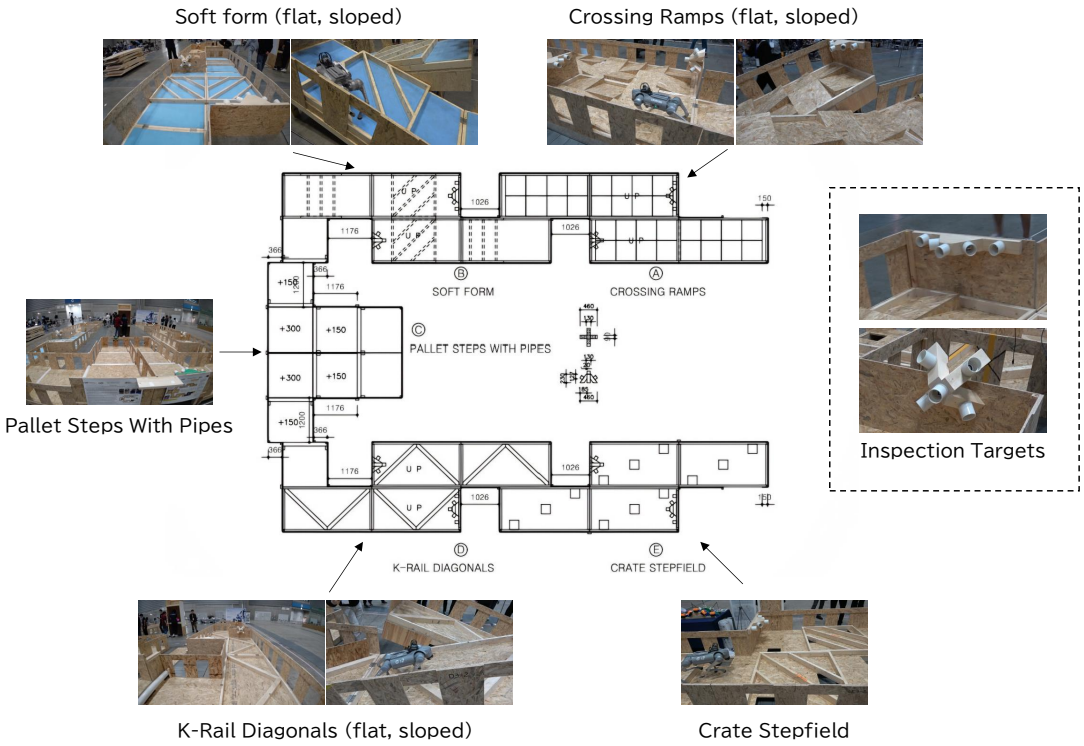


Figure 1. ICRA 2024 Quadruped Robot Challenge Course

Pallet Steps with Pipes A section with step-like obstacles and pipes placed at the edges. The pipes rotate when stepped on, causing the robot to slip.

K-Rail Diagonals A section with obstacles arranged in a K-shape along the course's perimeter and diagonals.

Crate Stepfield A section that was intended to simulate random heights using stacked crates, but for this competition, it was implemented as a field with holes.

Of these, the Crossing Ramps, Soft Form, and K-Rail Diagonals sections had both flat and inclined variants, allowing for adjustments in difficulty.

4. System Overview

This section provides an overview of the quadruped robot system we developed. Details on the locomotion control and behavior planning are discussed in Sections 5 and 6, respectively.

4.1. Hardware

Table 1 summarizes the specifications of the robot used. We used the Unitree Go2 robot (Edu edition), and additionally mounted a Livox Mid-360 3D Lidar on the top of the robot. The Lidar, along with the Go2's built-in forward-facing monocular camera, served as the external sensors we employed (Fig. 2). The Livox Lidar was mounted with a 18-degree downward angle, slightly modifying the mounting jig provided by the

Table 1. Hardware Specifications of Our Robot

Size (standing)	0.7 m x 0.4 m x 0.5 m
Weight	13 kg
Battery capacity	29.6 V, 15 Ah
Joints	12 DOF (3 per leg)
Camera resolution	1280×720
Main computer	NVIDIA Jetson Orin NX

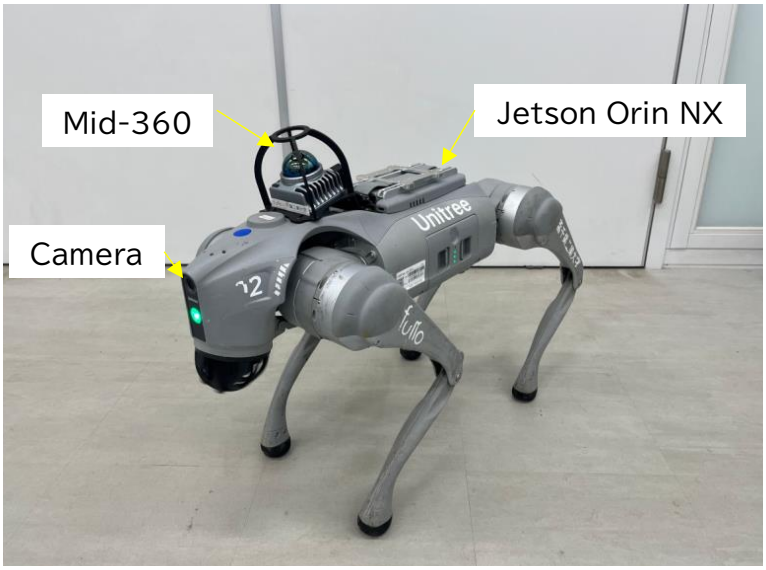


Figure 2. Outlook of our robot

manufacturer. While the Go2 robot comes equipped with a downward-facing rotating 3D Lidar (L1) on its head, we opted not to use it due to its low observation density and potential interference with obstacles. The primary computation resource was an NVIDIA Jetson Orin NX mounted on top of the robot, and we run all the software we developed on this computer.

4.2. Software

The software system architecture is shown in Fig. 3. We did not use the embedded locomotion control provided by the Go2, but instead implemented our own locomotion control policy obtained through deep reinforcement learning. The locomotion control policy takes the target velocity (v_x, v_y, ω) as input and outputs the target joint angles for each leg. These commands are sent to the Go2's embedded controller, which manages the position control of each servo motor.

4.2.1. Navigation Strategy

As for the autonomous navigation strategy, we created a 3D point cloud map using SLAM based on pre-collected data (using a hand-held setup), and manually provided waypoints on the map to direct the robot's path. The robot follows the given path and avoids collisions with walls while walking.

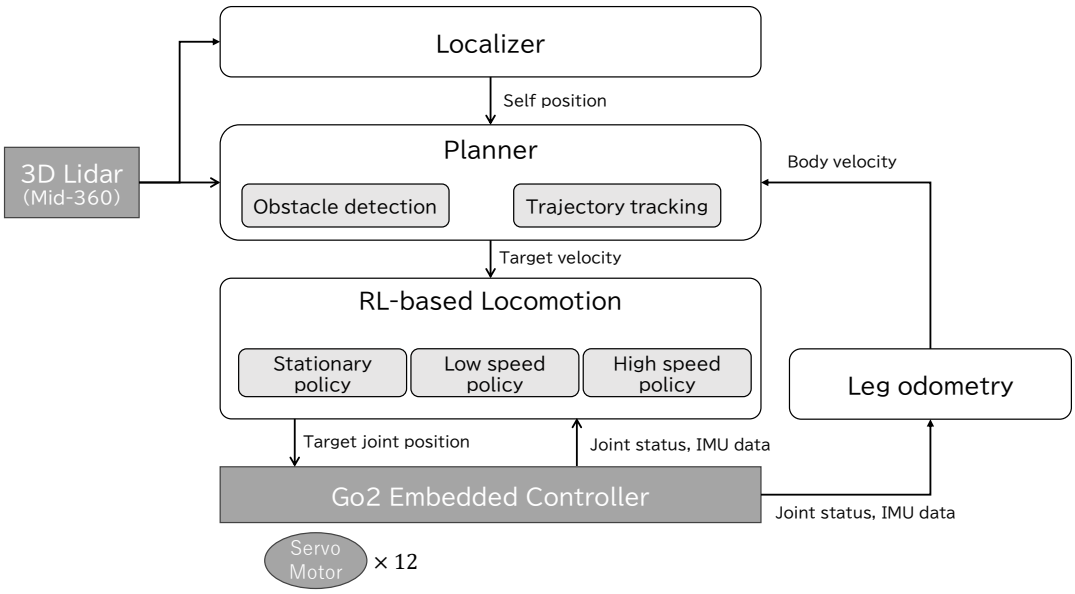


Figure 3. Software system architecture

4.2.2. Self-localization

For localization, we employed a combination of Lidar-Inertial Odometry (LIO) using FAST-LIO2 [14] and point cloud registration. When starting navigation, we employ Generalized-ICP (GICP) [15] to align the initial pose of the robot with the pre-built map. During navigation, we primarily rely on FAST-LIO2 for continuous localization, which fuses data from the Inertial Measurement Unit (IMU) and the 3D Lidar. This fusion is essential when navigating rough terrain, as the robot may experience rapid, unpredictable movements or even fall, making it difficult to maintain reliable localization using point cloud registration methods alone. Although our system is capable of periodically correcting accumulated odometry errors by matching local point cloud maps with the pre-built global map using GICP, we found that for the relatively short QRC course, the localization accuracy provided by FAST-LIO2 alone was sufficient without requiring additional correction steps during the competition.

4.2.3. Operator Control

In addition to the robot, we prepared an operator control PC, which receives the robot's status and sensor data and displays them to the operator. For teleoperation, a joystick connected to the PC is used to send the robot's target velocity.

5. Locomotion Control using Deep Reinforcement Learning

This section describes the method used to train the policies for leg control and their application to the real robot. We control the joints of the robot based on a policy obtained through deep reinforcement learning in a simulator. The higher-level behavior planner provides commands to the locomotion control system in the form of translational velocities v_x, v_y [m/s] and rotational velocity ω [rad/s], without engaging in the detailed movement of the legs. On the QRC field, various obstacles such as steps and

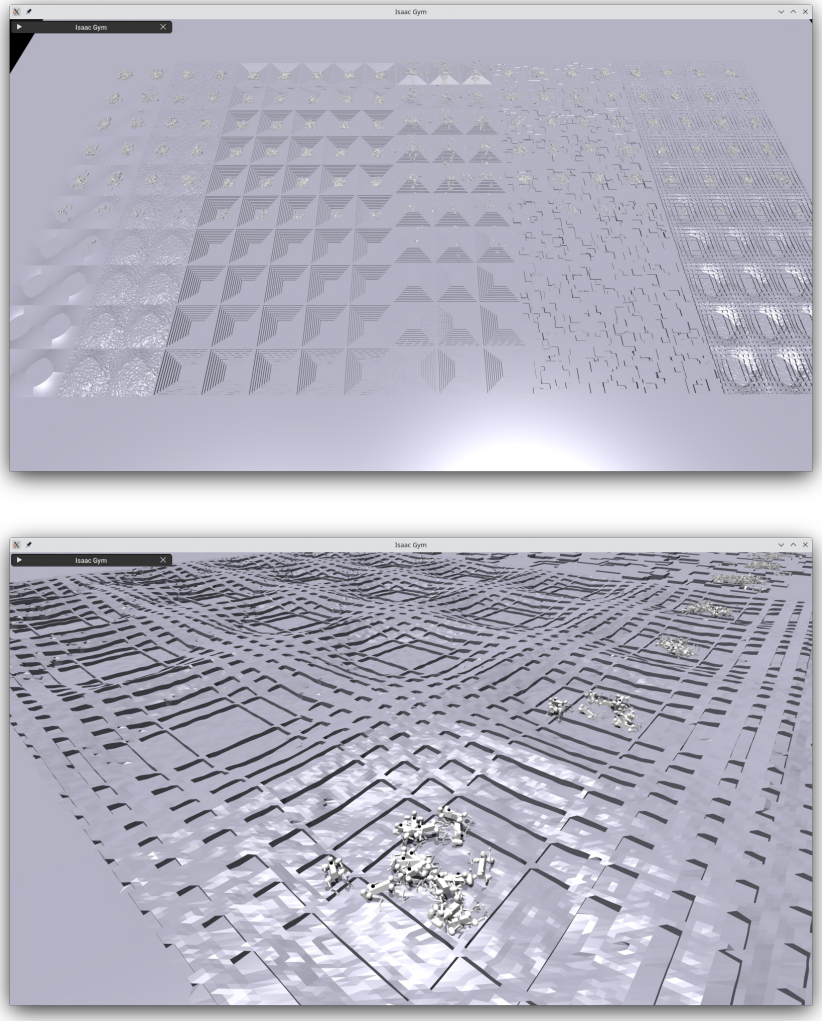


Figure 4. Terrain used for training a low-speed walking policy. Top: full view, Bottom: enlarged view of slopes with grid-shaped obstacles

slopes are set up, and the robot is required to continue functioning without significant failure, even if it loses balance or falls. Two types of walking policies are prepared: one is *low-speed policy* for slow walking, and the other is *high-speed policy* for fast walking. In addition, a *stationary policy*, which allows the robot to recover from a fallen state and return to a stable standing posture, was also trained. These three policies are switched according to the situation.

5.1. Training the Control Policy in the Simulator

5.1.1. Training Configurations

In this study, we implemented a *blind policy*, using only inertial sensor observations $s_t \in \mathbb{R}^{45}$ (gravity vector, angular velocity, joint positions and velocities, previous actions, commands) as inputs for the leg control policy. Previous actions, which are the joint angle targets executed in the previous timestep, are included to mitigate the

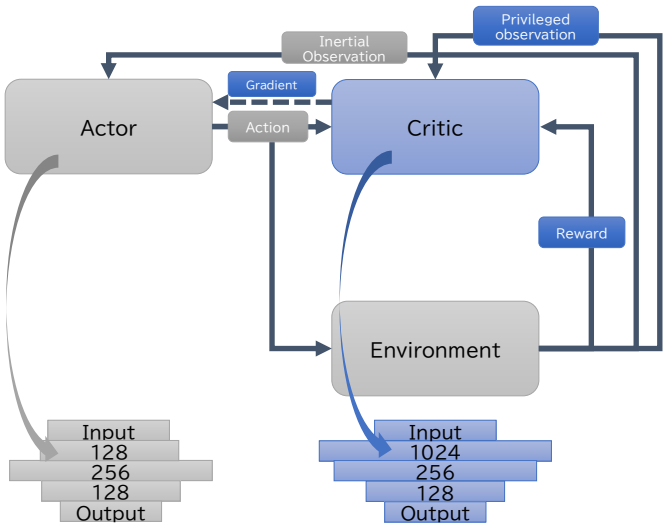


Figure 5. Policy training using Asymmetric Actor Critic

Table 2. Observations used in the control policy

Inertial observations	Privileged observations
Joint angles (12)	Joint angles (12)
Joint velocities (12)	Joint velocities (12)
Body angular velocity (3)	Body angular velocity (3)
Gravity direction (3)	Gravity direction (3)
Target velocity command (3)	Target velocity command (3)
Previous actions (12)	Previous actions (12)
	Terrain (187)
	Body linear velocity (3)
	Friction coefficient (1)
	Body weight (1)
	Payload weight (1)
	Payload position (3)
	Delay (1)
total 45	total 242

POMDP nature of the control problem. Although the Go2 robot is equipped with foot contact sensors, we did not use them for locomotion control. The policy was trained using the Legged Gym framework [16]. The policy consists of a fully connected neural network with 128, 256, and 128 nodes in each layer. To achieve stable walking using only internal sensors, we applied Asymmetric Actor Critic [17] during training, using privileged observations $s_t^{priv} \in \mathbb{R}^{242}$ that include information on nearby terrain, translational velocity, and domain randomization. The overview of the training using the Asymmetric Actor Critic is shown in Fig. 5, and the comparison of the inertial and privileged observations is summarized in Table 5.1.1.

To ensure the trained policy would transfer well to the real world, several adjustments were made during training. Since the real robot experiences a delay of approximately 30 ms longer than in the simulator, this delay was introduced into the training environment to better simulate real-world conditions. Additionally, to enhance temporal information, we employed an exponential moving average (EMA) of observations s_t. In addition to s_t, three sets of $s_t^{ema(\alpha)}$ with different smoothing coefficients α were combined to form an observation vector in $\mathbb{R}^{180}$. We observed that this extended observation approach improved performance on handling steps compared to policies using only single observation s_t.

Each joint is controlled by a position controller, using the target joint angles that the policy outputs. For the Go2 robot, the policy was trained with the position controller's PD gains set to $K_p = 28$ and $K_d = 0.7$. To improve the policy's robustness, domain randomization was applied. We varied the coefficient of friction between 0.175 and 0.9 and placing up to 1 kg of payload on the robot's back. The position of the payload was also randomized. The terrain used in training followed that of Legged Gym, including smooth slopes (uphill/downhill), rough slopes (uphill/downhill), stairs (uphill/downhill), and discrete obstacles. For the low-speed walking policy, we added slopes with grid-shaped obstacles (Fig. 4).

5.1.2. Stationary Policy

The stationary policy, which allows the robot to transition to and maintain a stable standing position, was trained in an environment where rewards were given when all four legs were grounded, and the body remained level. It is important to expose the robot to various real-world scenarios during training. However, it is challenging to consistently configure the position and posture of the robot when it is rolling on the ground in various orientations.

Thus, the initial state is set at a sufficiently high position (0.35 m) so that collisions with the ground can be ignored. The posture is an intermediate one, obtained by normalizing a linear interpolation (Nlerp) between a random posture and a level posture using quaternions. The difficulty of the task can be controlled by adjusting the interpolation coefficient. In this case, the interpolation coefficient was set to 0.5. Joint angles were sampled from 90 % of the range of motion for each axis.

5.1.3. Walking Policies

The walking policy was first trained with translational commands up to 1 m/s to acquire basic walking abilities. At this stage, the initial posture was set to level without randomization. Then, the policy was further trained with commands up to 2 m/s to adapt to higher-speed commands. The policy trained through this procedure was designated as the high-speed walking policy, which handles translational commands of 0.5 m/s and above. However, at translational speeds below approximately 0.3 m/s, the high-speed walking policy does not raise its legs sufficiently, and the robot is unable to move at the commanded speed.

To address the issue of low-speed movement not adhering to commands, and to enhance mobility on uneven terrain, we trained a separate low-speed walking policy. To encourage the robot to raise its legs even at low speeds, we penalize a situation when more than three feet were grounded (feet contact count penalty). Additionally, a stance phase ratio penalty was introduced to encourage the even use of all four legs. The stance phase ratio penalty uses the standard deviation of the stance phase duration for each leg, estimated by time-averaging the contact states of the legs using EMA. These two penalties enabled the low-speed walking policy to exhibit periodic leg-lifting behavior, even when the translational command was 0 m/s. In order to train for the QRC field, such as K-Rails on slopes, we added slopes with grid-shaped obstacles (Fig. 4) in a training environment for low-speed walking policy. The low-speed walking policy was first trained in an easy environment without the stance phase ratio penalty or feet contact count penalty for acquiring basic walking abilities. Afterward, the penalties were added, and training continued in a more challenging environment.

5.1.4. Switching between Policies

In actual operation, the low-speed walking policy begins from a standing posture with all legs grounded, transitioning from the stationary policy. Since the training environment does not expose the robot to this initial state, we observed instability when the real robot started walking. To address this, just before collecting training experience in each training iteration, we advanced the physics simulation for a certain period under the control of a different policy to create a state that is difficult to replicate by simple means. During this period (idle period), the policy under training is not involved, and the robot is moved under the control of a previously trained stationary policy. Observations during the idle period are discarded and not used for training. The idle period is inserted with a certain probability, and its length is determined randomly. In this case, an idle period of up to 2 s was inserted with 5 % of probability. This method allowed the policy to experience the transition from walking to the stationary policy in the simulator and improve its behavior.

In this study, idle periods were introduced only during the training of the low-speed policy. However, applying similar techniques during the training of other policies, such as cutting torque or using different walking policies, could expose the robot to a broader range of conditions, thereby improving robustness.

5.2. Implementation on the Real Robot

The policy trained in the simulator is used directly on the robot without any modifications, and it runs on the CPU without utilizing the GPU. No additional modifications (such as low-pass filtering) were introduced to the policy, except for the joint ordering difference between the real robot and the training environment. The observation vector is constructed from the robot's internal state, obtained from the embedded controller, and fed into the policy to generate actions. The actions are scaled similarly to the training phase and sent to the motors as target joint angles. Each motor operates in position control mode, using the same PD gains in the training simulator.

The stationary policy, low-speed walking policy, and high-speed walking policy are switched based on the translational velocity commands. Action generation runs at 50 Hz, and the policy switching occurs at the same frequency.

When the robot tilts significantly during walking, it is considered to have fallen, and the normal policy is temporarily halted to initiate the recovery sequence. The fall detection is based on the orientation data obtained from the IMU integrated in the Go2 robot. Specifically, if the robot's tilt angle exceeds 60 deg from the upright pose and this condition persists for more than 1 second, the system determines that the robot has fallen. In the recovery sequence, the torque of the legs is significantly reduced, and the robot is position-controlled toward the initial posture while waiting for the external forces on the legs to cease. Next, the stationary policy is activated to attempt to stand up. If the robot fails to stand within a certain time, the recovery sequence is restarted from the beginning.

6. Behavior Planning

Although the control policy obtained through deep reinforcement learning is capable of traversing various types of rough terrain stably, it is not perfect. In preliminary experiments, we experienced cases where the robot became stuck and could not move

forward after being caught on obstacles. This issue arises because the simulation model and the real robot model are not perfectly aligned, and it is impossible to simulate all real-world environments. To address this issue, we implemented a resilient behavior planning method. In this system, since the walking control does not use external sensors, the foot placement cannot be adjusted based on the shape of the steps. Therefore, the robot selects random actions to change its state and attempts to overcome the obstacle again.

6.1. Path Following and Obstacle Avoidance

The basic behavior strategy is to walk along a predefined path while avoiding contact with obstacles. We integrate these two objectives using a artificial potential field method [18]. For path following, we use a two-dimensional path following policy that was pre-trained [19], which generates the target translational velocity and rotational velocity $\boldsymbol{v}^{\mathrm{tra}} = (v_x^{\mathrm{tra}}, 0, \omega^{\mathrm{tra}})$. To this, we add the lateral velocity $\boldsymbol{v}^{\mathrm{rep}} = (0, v_y^{\mathrm{rep}}, 0)$, which is based on a repulsive force from the distance to obstacles.

$$\boldsymbol{v} = \boldsymbol{v}^{\mathrm{tra}} + \boldsymbol{v}^{\mathrm{rep}} \tag{1}$$

This combined velocity is then provided to the walking controller as the target velocity and is updated at 10 Hz. In this method, path following and obstacle avoidance are computed in separate dimensions, preventing local minima where attractive and repulsive forces cancel each other out.

6.2. Stuck Detection

To determine whether the robot is stuck, we use leg odometry based on the following kinematic calculations [20]. Let the joint angles of the robot be $\boldsymbol{q}$ and the joint angular velocities be $\dot{\boldsymbol{q}}$. For each leg l, the velocity of the foot is used to calculate the velocity of the body by applying the Jacobian matrix $J^{(l)}(\boldsymbol{q})$ and forward kinematics $\mathrm{fk}^{(l)}(\boldsymbol{q})$,

$$\boldsymbol{v}^{(l)} = -J^{(l)}(\boldsymbol{q})\dot{\boldsymbol{q}} - \boldsymbol{\omega} \times \mathrm{fk}^{(l)}(\boldsymbol{q}), \tag{2}$$

then we calculate the average over the legs that are in contact with the ground,

$$\boldsymbol{v}_{odom} = \frac{1}{\sum_{l=1}^{4} \sigma^{(l)}} \sum_{l=1}^{4} \sigma^{(l)} \boldsymbol{v}^{(l)}, \tag{3}$$

to obtain the velocity from leg odometry. Here, $\sigma^{(l)}$ is a binary value that represents the contact state of leg l. If the absolute value of the velocity from leg odometry remains below a certain threshold for a set period of time, the robot is determined to be stuck, and the following escape behavior is initiated. For the QRC competition, we set this time threshold to 2 seconds, which was experimentally determined.

6.3. Randomized Escape Strategy

To escape from being stuck, a randomized strategy is used, where the target velocity is randomly selected and executed, repeating until the robot successfully escapes.

The rationale behind this approach is the difficulty of pre-programming an escape strategy for every possible stuck scenario. The robot used in this experiment does not have sensors to observe the area around its feet, making it difficult for the robot to determine why and in what situation it is stuck.

Furthermore, in our experience, even when a human operator observes the robot and controls it, escaping from a stuck position can be challenging. For example, when the rear leg is caught on a step, the robot may be able to escape by moving backward, but in some cases, it must change its body orientation before it can escape. This requires trial and error by the operator using the joystick. Based on this observation, we decided to implement a randomized selection of target velocity commands to allow the robot to attempt to escape from a stuck state.

A straightforward approach would be what we call the *Random Uniform Strategy*, which involves randomly selecting a target velocity from a uniform distribution within the maximum speed range. However, this strategy often results in the selection of small or moderate velocities, which are not always efficient for escaping from stuck situations that require more forceful movements. Therefore, we developed the *Random Choice Strategy*, where velocity commands are specifically chosen from discrete values that include the maximum possible values in each direction, as follows:

$$\boldsymbol{v}^{\text{escape}} \sim \mathcal{A}^{\text{escape}}, \tag{4}$$

$$\mathcal{A}^{\text{escape}} = \left\{ (v_x, v_y, \omega) \;\middle|\; \begin{array}{l} v_x \in \{v_{x\min}, 0, v_{x\max}\}, \\ v_y \in \{v_{y\min}, 0, v_{y\max}\}, \\ \omega \in \{\omega_{\min}, 0, \omega_{\max}\} \end{array} \right\}. \tag{5}$$

Using these discrete velocity choices in each direction, the robot can generate more distinct and powerful movements to escape from a stuck state more efficiently. We have included an experimental comparison between the Random Uniform and Random Choice strategies in Appendix B.

For the QRC competition, each randomly selected velocity command was maintained for 0.3 seconds. After this duration, the system re-evaluates whether the robot is still stuck. If the robot has successfully escaped (detected by sufficient movement from leg odometry), normal navigation resumes; otherwise, a new random command is selected and the escape process continues.

7. Experiments in QRC

7.1. Competition Format and Scoring at ICRA 2024 QRC

The ICRA 2024 QRC was held at the ICRA 2024 venue in Pacifico Yokohama from May 14 to 17, 2024. The competition was divided into three stages: the Preliminary, Semi-final, and Final rounds. In the Preliminary rounds, the competition took place on a flat course, while from the Semi-final round onward, the competition was held on a sloped setup. All teams participated in the Final round, as the organizers aimed to provide every team with the full, difficult competition experience, regardless of their performance in the earlier stages.

In the Preliminary round, the course was divided into several sections, and multiple teams competed in parallel, each on a limited part of the course. The Semi-final round was conducted on half of the course, and extra obstacles were added before this round

to further increase the difficulty. In the Final round, the competition utilized the entire course for the first time. From the Semi-final onward, no time was given to the teams to practice on the new layout, meaning all teams had only one chance to try the fully-setup course during the competition.

There were two types of tasks in the competition: the mobility task and the inspection task, each with a 5-minute time limit. In the mobility task, teams earned 2 points for each section completed. If the robot could navigate autonomously, the score was multiplied by 4. Teams could earn full marks if the robot successfully traversed the entire course, reaching the far end and returning to the start. In the inspection task, points were awarded if the operator could display the letters placed inside the pipes on the operator's PC.

Regardless of whether the robot performed the task autonomously or via teleoperation, the operator was not allowed to directly observe the robot. Instead, they had to rely solely on sensor data transmitted from the robot to issue commands and make decisions. Due to poor Wifi quality at the venue, wired connections were also allowed for operators to send commands and receive video feeds from the robot. While wired connections provided more stable communication quality, we found that handling the cables was challenging. As a compromise, a team member (non-operator) carried the wireless access point and followed the robot from outside of the course, keeping it close to the robot to shorten the wireless communication distance and address communication quality issues. Even with this setup, the communication was slow and delayed, the operator was only able to receive video feeds at around 3 fps.

7.2. Experiments in the Final Round

This section describes the experimental results in the final round. In the mobility task, our robot was able to autonomously navigate all five sections of the course and reached the far end of the course. Figure 6 presents a detailed analysis of our robot's performance during the final round. The visualization shows three key aspects: section completion times (top), stuck recovery events (middle), and locomotion policy transitions (bottom). Our robot faced challenges in the Soft Form, K-Rail Diagonals, and Pallet Steps with Pipes sections during the return trip, requiring time-consuming recovery actions. The time limit expired while navigating the Pallet Steps with Pipes section on the return journey, resulting in a final score of 56 out of 80 points.

In the inspection task, the operator remotely controlled the robot, according to the images from the front camera of Go2 robot to search for the letters inside the pipes. We were able to identify 9 targets, resulting in a score of 18 points out of 40.

Figure 7 shows scenes of our robot during the competition. Figure 7(a) shows the robot autonomously navigating the entire course, (b) shows the robot avoiding an obstacle after a collision (the obstacle was added just before the Final round), (c) shows the robot escaping from being stuck in section 8), and (d) shows the robot executing the inspection task.

7.3. Competition Results

Table 7.3 shows the total scores from the Semi-final and Final rounds for all participating teams. Our team, fuRo, achieved first place overall, primarily due to our system's ability to successfully complete many of the Mobility tasks autonomously. While we don't have detailed information about all teams' technical approaches, we

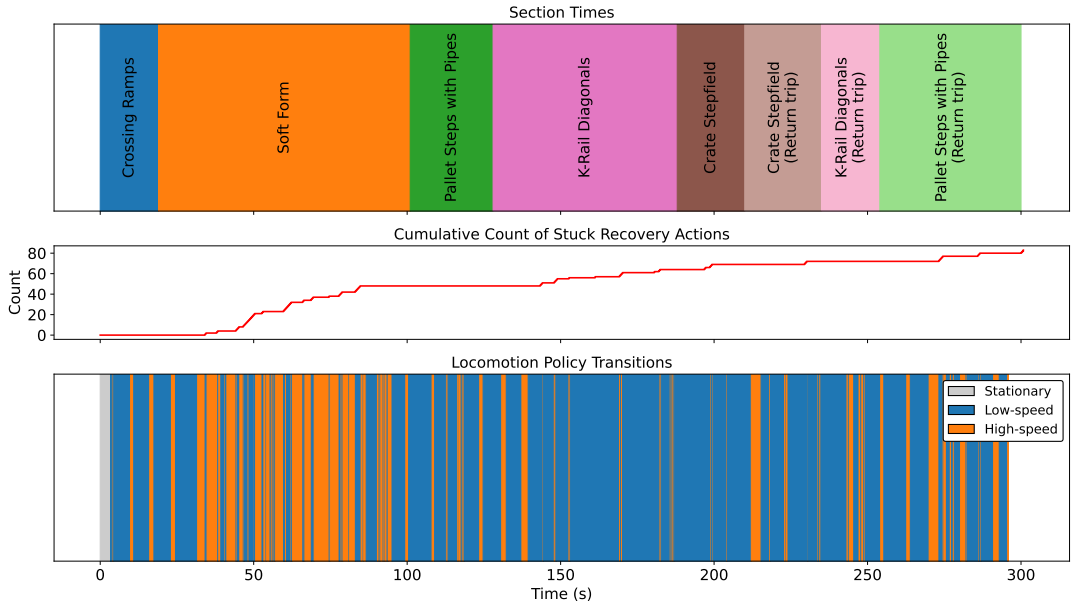


Figure 6. Analysis of robot performance during the ICRA 2024 QRC final round navigation task. Top: time spent in each course section. Middle: cumulative count of stuck recovery actions. Bottom: locomotion policy transitions between stationary, low-speed, and high-speed modes.

Table 3. Semi-final and Final scores

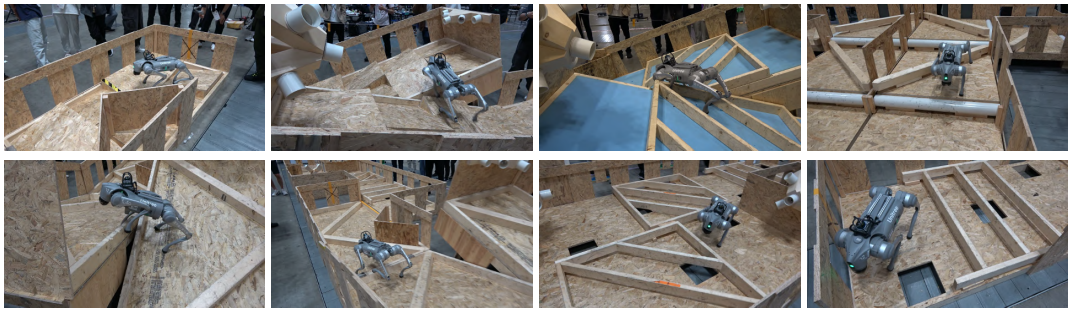
Team	Organization	Mobility	Inspection	Autonomy	Teleoperation	Total
fuRo	**Chiba Inst. Tech.**	264	72	264	72	**336**
Wongtsai	Zhejiang Univ.	100	122	0	222	222
Dream TRIP	KAIST	104	96	172	28	200
ARCLAB	Univ. of Hong Kong	90	92	0	182	182
reRo	Chiba Inst. Tech.	84	94	0	178	178
Tsinghua Univ.	Tsinghua Univ.	56	68	0	124	124

observed that only our team and Dream TRIP were able to score points in autonomous mode, with our autonomous score being significantly higher (264 vs 172). This scoring advantage was significant because autonomous navigation earned four times more points than teleoperation. We believe our success came from combining robust deep reinforcement learning-based locomotion control with an effective randomized escape strategy, creating a resilient system that could recover from challenging situations without human intervention.

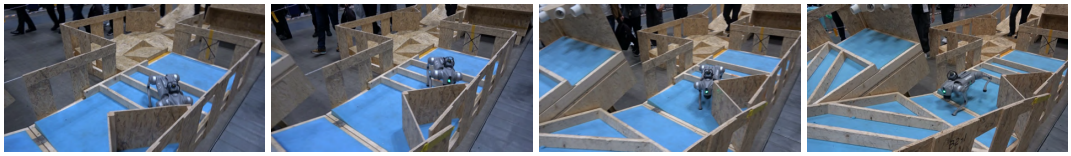
In contrast to our success in mobility, we ranked fifth in the inspection category. This lower performance reflects our primary research interest in autonomous navigation, with less attention devoted to inspection capabilities. The Go2's built-in camera presented significant limitations, particularly the inability for users to adjust shutter speed settings, which resulted in motion blur during movement. For future improvements, we would recommend implementing external cameras with user-adjustable parameters and developing specialized control systems to properly position the robot and orient the camera toward inspection targets.

8. Discussion

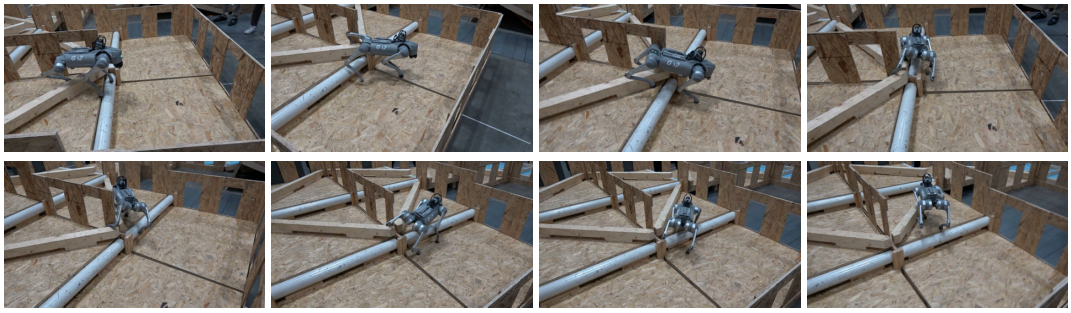
During the QRC competition and in preliminary experiments, we experienced three main types of failures in autonomous mobility.



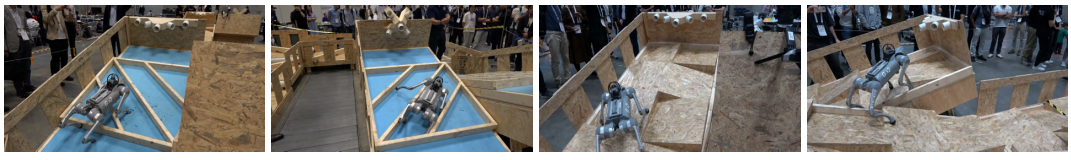
(a) Navigating the entire course



(b) Negotiating unseen obstacles



(c) Exiting from trapped situation



(d) Inspection (teleoperation)

Figure 7. Scenes from ICRA 2024 Quadruped Robot Challenge final

Figure 8. Failure because of abnormal state of the lower layer after being stuck

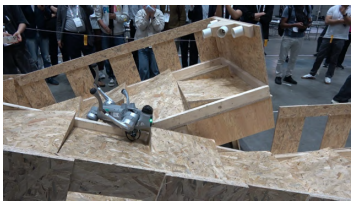


Figure 9. Failure to get up after falling

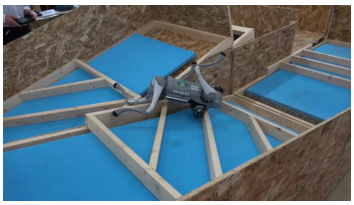


Figure 10. Failure of self-localization after falling

8.1. Analysis of Failures

Embedded Controller Failure Due to Load

Figure 8 shows an instance where the robot became stuck on an obstacle and, while attempting to escape, the Go2's embedded controller entered an abnormal state (indicated by the red LED flashing [21]) and became unresponsive. In our experience, this abnormal state occurs when a joint remains under load and is kept unmovable for an extended period. We consider it a sort of defect of the Go2's embedded controller, and we have not yet found a solution to this issue.

Failure to Recover from Falling

While our system can recover from falls with high reliability on flat terrain, failures have been observed on rough terrain. For example, in Figure 9, the robot was unable to get up because the obstacle and the slope created too much distance between the foot and the ground, preventing the robot from pushing off properly. In the Soft Form section, the robot was sometimes unable to recover because its legs sank into the soft ground, preventing it from lifting its body. We experienced this failure in the Semi-final round, and we had to manually reset the robot to continue the competition.

Failure of Self-Localization

The LIO used in the system is robust even under rapid and noisy movements, but we observed several failure cases when the number of points in the observable point cloud became very small. An example of this is shown in Figure 10. In this case, the Lidar's field of view was blocked by the ground and the very close obstacles, and the number of points observed was reduced, leading to a failure of the LIO. To overcome this issue, we may need to consider adding another Lidar, for example, downward-looking, to ensure a sufficient field of view even if the robot is upside down.

8.2. Future Work

In this research we employed a blind gait control policy, which does not use external sensors. However, it is important to note that our approach assumes the robot is operating in a relatively safe environment, where failures, such as falls or missteps, do not lead to catastrophic results. In real-world scenarios, where the terrain may be more hazardous, such an assumption might not hold.

For example, in terrain with a narrow bridge or stepping stones, a single misstep can result in fatal system malfunction. In such environments, we need to consider a

more perception-based approach, where the robot can observe the terrain from external sensors and plan its steps accordingly. A system that integrates these external observations could dynamically plan each step to ensure safe traversal.

In the future, we plan to integrate a perception-based approach, such as [22], with the current system to improve the robot's performance in challenging environments. This would involve developing a external sensing-based step planning system that can adjust foot placement based on the terrain, as well as planning a strategy when to switch between the blind locomotion control and the perception-based planning.

Our stuck-recovery strategy could also be improved, especially in terms of its efficiency. An alternative approach would be to implement leg-level control actions rather than just random velocity commands. This would allow the robot to execute specific leg movements to escape from challenging positions. However, this approach would require significant modifications to the planning module, as it would necessitate expanding beyond the current simple 2D velocity command structure.

Moreover, repeated failed escape attempts not only delay task completion but also increase physical load on the robot's joints and actuators. In extended deployments, this could lead to mechanical wear or thermal issues. Therefore, reducing the number of retries through smarter action selection or perception-guided planning is an important direction for future work. To prevent getting stuck in the first place, strategies utilizing external sensors for terrain perception would be promising.

9. Conclusions

In this paper, we introduced a rough terrain navigation system using the commercially available quadruped robot Go2. When using deep reinforcement learning-based blind locomotion control, the robot cannot always adjust its foot placement optimally to the terrain, which sometimes results in the robot getting stuck on terrain it would normally be able to traverse. To address this issue, we combined behavior planning methods to enable the robot to escape from stuck states and retry traversal. We demonstrated that the system could autonomously traverse the various rough terrains posed by the QRC challenges.

Appendix A: Learning Details Locomotion Policies

Here we provide additional details on the training of locomotion policies.

Reward functions

Legged Gym provides a set of reward functions that encourage the robot to walk, which we used as the base reward structure for our locomotion policies. To enhance performance, we added several custom reward functions tailored to our specific requirements. The complete list of reward functions and their weights for each policy type (low-speed, high-speed, and stationary) are summarized in Table 4. Each policy was trained with different reward weightings to achieve its specialized behavior. The details of our additional custom reward functions are as follows.

body_pose

The body pose reward encourages the robot to maintain its body in a desired orientation relative to gravity. We compute this reward by measuring the negative gravity vector and a target pose vector:

$$R_{\text{body\_pose}} = -\mathbf{g}_{\text{proj}} \cdot \mathbf{o}_{\text{target}},$$

where $\mathbf{g}_{\text{proj}}$ is the gravity vector projected onto the robot's body frame, and $\mathbf{o}_{\text{target}}$ is the desired orientation vector. This function is maximized when the robot's body is aligned with the target upright orientation.

joint_pose

The joint pose reward encourages each joint angles to be close to their neutral position. This is calculated as:

$$R_{\text{joint\_pose}} = \sum_{j \in J} \exp\left(-\frac{\|\mathbf{q}_j - \mathbf{q}_{j,\text{neutral}}\|^2}{\sigma_{joint\_pose}} \right),$$

where J represents the set of all joints, and $q_{j,\text{neutral}}$ is the neutral position of joint j. We use $\sigma_{joint\_pose} = 0.03$ for training policies use this reward factor.

feet_contact

The feet contact reward encourages the robot to maintain all feet in contact with the ground at all times. This is calculated as:

$$R_{\text{feet\_contact}} = \prod_{i=1}^{4} \left(exp(-\frac{\|\mathbf{p}_i - \mathbf{p}_{i,\text{desired}}\|^2}{\sigma_{feet\_contact}}) \cdot \text{clamp}(F_i, 0, 1) \right)$$

where $\mathbf{F}_i$ is the filtered contact force on foot i, $\mathbf{p}_i$ is the 2D vector which is the foot position projected onto the ground, and $\mathbf{p}_{i,desired}$ is the desired position of the foot projected onto the ground. We use the hip link position as the desired position, to keep the foot link positioned directly beneath it. Also, we use fixed $\sigma_{feet\_contact} = 0.3$.

feet_contact_count_gt

The feet_contact_count_gt reward penalizes situations where too many feet remain in contact with the ground for extended periods. This is calculated as:

$$R_{\text{feet\_contact\_count\_gt}} = \log(T_s + 1),$$

where T_s represents the duration for which more than three feet have been continuously in contact with the ground. When the number of contacting feet drops below the threshold, T_s is reset to zero. This logarithmic penalty encourages the robot to regularly lift its legs even at low speeds, resulting in a more dynamic gait that is better suited for traversing obstacles.

slip

The slip reward penalizes foot slippage during ground contact, which can destabilize the robot and waste energy. This is calculated as:

$$R_{\text{slip}} = \sum_{i=1}^{4} |\mathbf{v}_i| \cdot \mathbb{I}\left(|F_i| > F_{\text{threshold}}\right),$$

where $\mathbf{v}_i$ is the velocity of foot i in the world frame, F_i is the filtered contact force on foot i, and $F_{\text{threshold}}$ is a force threshold (10N in our implementation) that determines whether the foot is in firm contact with the ground. The indicator function $\mathbb{I}$ returns 1 when the force exceeds the threshold and 0 otherwise.

support_phase_ratio_std

This is a penalty given when the ratio between stance and swing phases across the four legs becomes uneven. This penalty encourages balanced utilization of all legs.

$$\begin{aligned} \mathbf{r}_t &= (1-\alpha)\mathbb{I}\left(|\mathbf{F}| > F_{\text{threshold}}\right) + \alpha \mathbf{r}_{t-1}, \\ R_{\text{support\_phase\_ratio\_std}} &= \text{std}(\mathbf{r_t}) \end{aligned}$$

where $\mathbf{F}$ represents the contact force of the leg, $\mathbb{I}$ is the same indicator function as used for slip reward function, and $\mathbf{r_t}$ is the filtered ground contact state of the leg with smoothing coefficient α, representing the ratio between stance and swing phases. The standard deviation of this ratio across all four legs is calculated. The value of α was experimentally determined to be 0.9.

tracking_lin_vel_var

The tracking_lin_vel_var reward encourages the robot to follow linear velocity commands with an adaptive error tolerance based on command magnitude. This is calculated as:

$$\begin{aligned} R_{\text{tracking\_lin\_vel\_var}} &= \exp\left(-\frac{\|\mathbf{v}_{\text{cmd}} - \mathbf{v}_{\text{actual}}\|^2}{\sigma}\right), \\ \sigma &= \text{clamp}(\|\mathbf{v}_{\text{cmd}}\|^2, 0.01, 0.25), \end{aligned}$$

where $\mathbf{v}_{\text{cmd}}$ is the commanded linear velocity vector (x and y components), $\mathbf{v}_{\text{actual}}$ is the actual base linear velocity vector. This adaptive scaling plays a crucial role in preventing a common problem: when the commanded velocity is very small, a naive error scaling would produce negligible penalties, potentially causing the robot to stop moving entirely.

Training Parameters

The detailed training parameters are summarized in Table 5. The locomotion policies and the stationary policy were trained under different configurations.

The two locomotion policies – low-speed and high-speed policies – were each trained over 40,000 iterations using a two-stage curriculum. Each stage consisted of 20,000

Table 4. Reward functions and weights used in the locomotion policy training. The asterisk * indicates reward functions that were added to the original Legged Gym's reward functions.

Reward term	low speed first half	low speed second half	high speed first half	high speed second half	stationary
action_rate	-0.01	-0.01	-0.01	-0.01	-0.06
ang_vel_xy	-0.05	-0.05	-0.05	-0.05	-0.5
collision	-1.0	-1.0	-1.0	-1.0	-5.0
dof_acc	-2.5e-07	-2.5e-07	-2.5e-07	-2.5e-07	-2.5e-06
dof_pos_limits	-10.0	-10.0	-10.0	-10.0	-10.0
lin_vel_z	-2.0	-2.0	-2.0	-2.0	0
orientation	-1.0	0	-1.0	-1.0	0
tracking_ang_vel	0.5	0.5	0.5	0.5	0
torque_limits	0	-0.01	0	0	-0.1
torques	-0.0001	0	-0.0001	-0.0001	-0.002
termination	0	0	0	0	1.0
body_pose*	0	1.5	0	0	0.3
joint_pose*	0	0	0	0	0.6
feet_contact*	0	0	0	0	10.0
feet_contact_count_gt*	0	-5.0	0	0	0
slip*	-0.1	-1.0	-0.1	-0.1	0
support_phase_ratio_std*	-0.1	-1.0	0	0	0
tracking_lin_vel_var*	1.0	2.0	1.0	1.0	0

Table 5. Training parameters used for training each policies.

Parameter	high speed first half	high speed second half	low speed first half	low speed second half	stationary
number of iterations	20,000	20,000	20,000	20,000	20,000
episode length [s]	20	20	40	40	10
x velocity range [m/s]	±1	±2	±1	±1	0
y velocity range [m/s]	±1	±0.8	±1	±1	0
mass range [kg]	[-1, 5]	[-1, 5]	[-1, 5]	[-1, 5]	[-1, 5]
friction range	[0.05, 1.5]	[0.05, 1.5]	[0.05, 1.5]	[0.05, 1.5]	[0.05, 1.5]
payload range [kg]	[0, 1]	[0, 1]	[0, 1]	[0, 1]	[0, 1]
payload position [m]	±0.1	±0.1	±0.1	±0.1	±0.1
torque multiplier	1	[0.95, 1.05]	1	[0.95, 1.05]	[0.95, 1.05]
latency range [ms]	[15, 35]	[15, 35]	[15, 35]	[15, 35]	[15, 35]
soft dof pos limit	0.9	0.9	0.9	0.9	0.9

iterations. The stationary policy, on the other hand, was trained separately for 20,000 iterations. It used a shorter episode length of 10 seconds.

Termination conditions also differed across the policies. The locomotion policies did not terminate episodes early, even in the case of falls; instead, they continued until the fixed episode duration elapsed. In contrast, the stationary policy included an explicit early termination condition: episodes were terminated if the robot maintained both a `feet_contact` reward above 0.13 and a `body_pose` reward above 6.5 continuously for 1.5 seconds. This mechanism encouraged rapid convergence to a stable pose and prevented unnecessary motion after recovery was achieved.

Parameters not listed in Table 5, including those for PPO, were adopted from the Legged Gym implementation [1].

[1]We employed `Legged Gym` at commit `9ddda29` (`https://github.com/leggedrobotics/legged_gym/tree/9ddda29fbff7039848053f2104ec9ab4e8c3105a`) and `rsl_rl` at commit `2ad79cf` (`https://github.com/leggedrobotics/rsl_rl/tree/2ad79cf0caa85b91721abfe358105f869a784121`).

Figure 11. Experimental setup for escape strategy comparison

Appendix B: Comparison of Random Escape Strategies

We conducted additional experiments using a shorter test course constructed on our campus (Fig. 11). This course incorporated approximately half the elements of the competition course, including Crossing Ramps, Soft Foam, and a portion of the Pallet Steps with Pipes. To evaluate the effectiveness of our navigation strategies, we measured the time required for the robot to autonomously navigate from the starting point to the goal. We compared two approaches: Random Uniform strategy and Random Choice strategy discussed in Section 6.3. The results are summarized in Fig. 12. The Random Choice strategy completed the course in an average time of 157.6 seconds, while the Random Uniform strategy required an average of 209 seconds. We observed that the Choice strategy was able to escape from stuck situations more efficiently than the Uniform strategy, resulting in a significant reduction in overall navigation time.

Acknowledgments

Parts of the manuscript were translated from a Japanese draft using OpenAI ChatGPT-4o. Additionally, Chat GPT-4o and Claude 3.7 Sonnet were used to improve the language quality of the manuscript. The Python script used to generate Figure 6 was also developed with the assistance of ChatGPT-4o.

References

[1] Halder S, Afsari K, Chiou E, et al. Construction inspection & monitoring with quadruped robots in future human-robot teaming: A preliminary study. Journal of Building En-

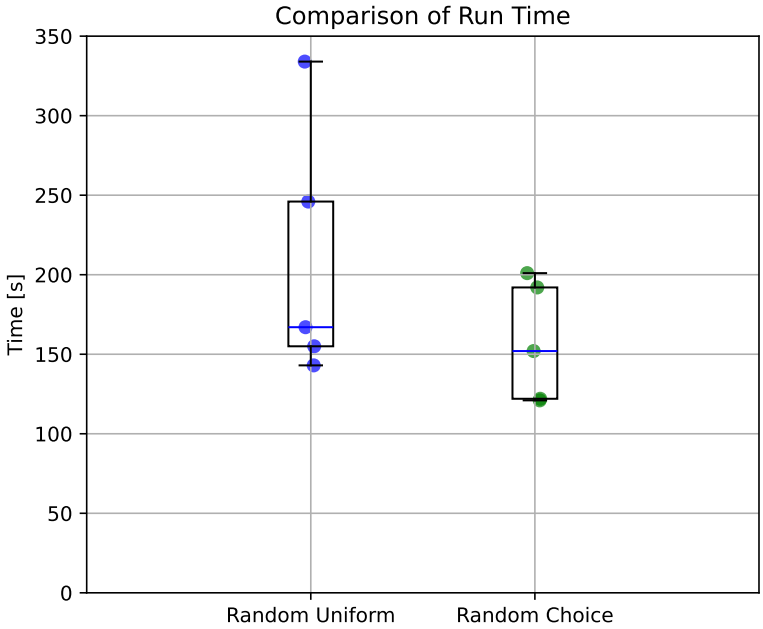


Figure 12. Comparison of escape strategies

gineering. 2023;65:105814. Available from: https://www.sciencedirect.com/science/article/pii/S2352710222018204.

[2] Tan J, Zhang T, Coumans E, et al. Sim-to-real: Learning agile locomotion for quadruped robots. In: Proceedings of Robotics: Science and Systems; June; Pittsburgh, Pennsylvania; 2018.

[3] Lee J, Hwangbo J, Wellhausen L, et al. Learning quadrupedal locomotion over challenging terrain. Science Robotics. 2020;5(47):eabc5986.

[4] Miki T, et al. Learning robust perceptive locomotion for quadrupedal robots in the wild. Sci Robot. 2022;7(62):eabk2822. Available from: https://www.science.org/doi/abs/10.1126/scirobotics.abk2822.

[5] Miller ID, et al. Mine tunnel exploration using multiple quadrupedal robots. IEEE RA-L. 2020;5(2):2840–2847.

[6] Bouman A, et al. Autonomous spot: Long-range autonomous exploration of extreme environments with legged locomotion. In: IEEE/RSJ IROS; 2020. p. 2518–2525.

[7] Jacoff A, Jeon J, Huke O, et al. Taking the first step toward autonomous quadruped robots: The quadruped robot challenge at ICRA 2023 in london [competitions]. IEEE Robotics and Automation Magazine. 2023;30(3):154–158.

[8] Irie K, Suzuki T, Hara Y, et al. Development of an outdoor navigation system using quadruped robot go1 and experiments in the tsukuba challenge 2023 (in japanese). In: Robomech 1A1-J04; 2024.

[9] Hutter M, Gehring C, Jud D, et al. Anymal - a highly mobile and dynamic quadrupedal robot. In: 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS); 2016. p. 38–44.

[10] Bledt G, Powell MJ, Katz B, et al. Mit cheetah 3: Design and control of a robust, dynamic quadruped robot. In: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS); 2018. p. 2245–2252.

[11] Haarnoja T, Zhou A, Ha S, et al. Learning to walk via deep reinforcement learning. In:

Proceedings of Robotics: Science and Systems; 2018.
[12] Hwangbo J, Lee J, Dosovitskiy A, et al. Learning agile and dynamic motor skills for legged robots. Science Robotics. 2019;4(26):eaau5872. Available from: https://www.science.org/doi/abs/10.1126/scirobotics.aau5872.
[13] Kumar A, Fu Z, Pathak D, et al. RMA: Rapid Motor Adaptation for Legged Robots. In: Proceedings of Robotics: Science and Systems; July; Virtual; 2021.
[14] Xu W, Cai Y, He D, et al. FAST-LIO2: Fast direct lidar-inertial odometry. IEEE Transactions on Robotics. 2022;38(4):2053–2073.
[15] Koide K. small_gicp: Efficient and parallel algorithms for point cloud registration. Journal of Open Source Software. 2024 Aug;9(100):6948.
[16] Rudin N, Hoeller D, Reist P, et al. Learning to walk in minutes using massively parallel deep reinforcement learning. In: Faust A, Hsu D, Neumann G, editors. Proceedings of the 5th Conference on Robot Learning (CoRL2021); (Proceedings of Machine Learning Research; Vol. 164); 08–11 Nov. PMLR; 2022. p. 91–100. Available from: https://proceedings.mlr.press/v164/rudin22a.html.
[17] Pinto L, Andrychowicz M, Welinder P, et al. Asymmetric actor critic for image-based robot learning. In: Proceedings of Robotics: Science and Systems; June; Pittsburgh, Pennsylvania; 2018.
[18] Khatib O. Real-time obstacle avoidance for manipulators and mobile robots. In: Proceedings. 1985 IEEE International Conference on Robotics and Automation; Vol. 2; 1985. p. 500–505.
[19] Irie K, Suzuki T, Hara Y, et al. Real world autonomous navigation experiments using a quadruped robot (in japanese). In: Robomech 2P1-H01; 2022.
[20] Camurri M, Ramezani M, Nobili S, et al. Pronto: A multi-sensor state estimator for legged robots in real-world scenarios. Frontiers in robotics and AI. 2020 Jun;7:68.
[21] Unitree. Unitree documentation center: About go2 [https://support.unitree.com/home/en/developer/about_Go2]; 2024.
[22] Agarwal A, Kumar A, Malik J, et al. Legged locomotion in challenging terrains using egocentric vision. In: Conference on Robot Learning; 2022. Available from: https://api.semanticscholar.org/CorpusID:252733339.