Elijah Tabachnik, Adam Gans, Leonardo Gutierrez

RL-Based Navigation Final Report

Introduction and Problem Statement

Any legged robot should be able to go through any form of the terrain that it encounters. Those kinds of conditions may be uneven ground, obstacles, and other changing terrain. It needs a control system that is capable of interpreting sensor data, maintaining balance and determining its movement in real time. In this project, we were able to train a Unitree Go2 quadruped robot in IssacSim simulation and walk, stabilize, avoid obstacles, and move towards a goal with Proximal Policy Optimization (PPO) in NVIDIA Isaac Lab to simulate physics, the environment, and the quadruped model.

The fundamental issue that we are solving in our project is: **How can a quadruped be trained using reinforcement learning to move efficiently and navigate a difficult environment to reach an object?** To achieve this, we adopted the two-layered hierarchy of controllers; a low-level locomotion controller (also referred to as LLC) and a high-level navigation controller (also referred to as HLC). The lower level locomotion controller would be in charge of learning to be walking steadily, stabilize, position foot placement, and simple gait behavior. The higher level controller will be trained to steer to a goal direction of command whilst using LiDAR to avoid obstacles. Finally, we added global path planning (SLAM-style) observations to the HLC to help it use the optimal path if it's available.

The end result of the system that we have created is able to display emergent locomotion, be stable on flat and rough surfaces, navigate dense obstacles, utilize optimal path hints, and be able to efficiently navigate to its goals.

Related Work

Quadrupedal locomotion reinforcement learning has been extensively developed with respect to quadrupedal locomotion, most notably on the robot in Hwangbo et al. project entitled ANYmal. Their work showed that RL policies which are trained in simulation can transfer to a physical robot to execute strong locomotion. However, previous methods tended to either use complicated model based priors or restricted to flat ground. As soon as a GPU-accelerated simulation, in particular the NVIDIA-based Isaac Gym was introduced, the ability to train agents in thousands of concurrent environments emerged, enabling a significant cut in training time and learning a more intricate curriculum on rough terrain.

The project is unique in its implementation where a hierarchical control stack is applied in this massively parallelized structure. Although most of the literature only trains the locomotion stability of the blind or high-level navigation with a perfect low-level controller, our system trains both. We combine a low level policy of proprioceptive stability and a high level policy of exteroceptive navigation, comprising the transition between robust walking and goal directed behavior on rough ground.

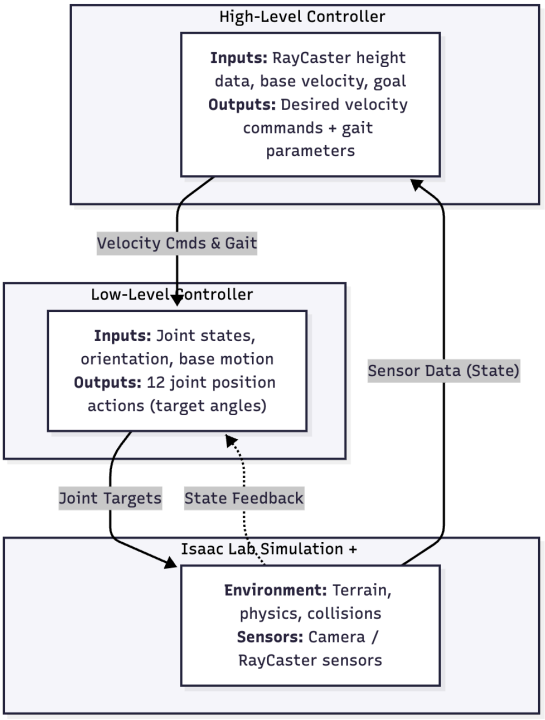
Dataset

Our project itself doesn't use an external dataset. Instead, we use data that is generated continuously throughout the simulation using Issac Lab. It uses its ability to have parallel simulation with 4096 parallel simulated Go2 robots during training. The physics engine itself is provided by NVIDIA PhysX 5 and the terrain generated is either a completely flat surface or randomized rough heightfields, boxes, and uneven terrain.

The sensor data that our robots use, which would effectively be our "dataset", includes: base linear and angular velocities, joint positions and velocities, orientation, RayCaster height, Foot contact states, and command inputs. These readings are all recorded at every simulation step; reward terms, foot heights, and other metrics are also logged per episode and later exported for analysis.

System Overview (High-Level Architecture)

Figure 1: High-Level diagram of our system



Our system uses a hierarchical reinforcement learning structure consisting of two coordinated PPO controllers:

High-Level Controller (Navigation Policy)

The high-level policy is the strategic brain of the system, which decides movement direction and adjustment of the gait of the robot to the environment. It takes in exteroceptive and proprioceptive information, such as a Raycaster (LiDAR) scan, the base linear and angular velocities, the relative goal position, and the lookahead hints. This controller does not achieve the necessary motion by manipulation

of the joints, but rather by feeding the high-level motion instructions (linear velocities and turning rates) to the LLC.

Low-Level Controller (Locomotion Policy)

The low-level policy takes the motion instructions sent by high-level controllers and works on body stabilization, leg control as well as mitigation of slippage. Its input state consists of joint angles and velocities, base orientation, Raycaster height scan, and root linear/angular velocities. Using these inputs, it produces the particular instruction on 12 joint positions to be transmitted to the command manager of the robot. It is important to note that the termination conditions and stability checks are checked in this layer as well.

Isaac Lab Simulator (PhysX Engine)

The Isaac Lab Simulator is the location of the physics that is used to implement the joint commands of the low-level controller to achieve realistic locomotion. It models the physical dynamics such as interaction with the terrain, collisions and sensor data such as the RayCaster data. This state information of the ground truth is then inputted back to the high-level controller to complete the loop of future episodes.

Software

Custom Code:

File	Purpose
custom_rewards.py	Responsible for implementing custom reward functions to help shape agent behaviors. Some functions included styles for gait style, motion stability, and obstacle avoidance.
custom_obs.py	Helps to compute custom observation inputs for the network. Functions would be things like goal_relative_target (a vector going to the goal) and lidar_scan (simulates analytic LiDAR for obstacle perception).
commands.py	Defines custom commands to generate. It more or less implements the logic for sampling high-level user commands.
flat_env_cfg.py	Specializing configuration for flat-ground locomotion as it inherits from the rough configuration but simplifies it to the flat plane terrain. From a different terrain, it adjusts rewards to train as a baseline gait for the robot.
rough_env_cfg.py	Configures the low-level locomotion environment on rough terrain. It defines the scene assets (robot, terrain), sensor setups, and the reinforcement learning parameters (observations, actions, rewards) for the base policy.

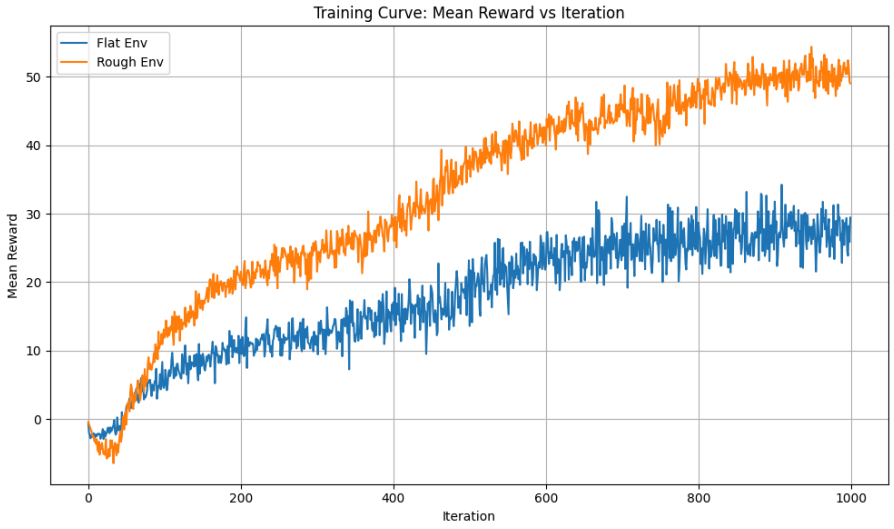
navigation_env_cfg.py	Configures the hierarchical navigation environment. It sets up a high-level policy that outputs commands to a pre-trained low-level controller.
custom_events.py	Handles simulation events and domain randomization. Handles static obstacle placements.
custom_terminations.py	Defines termination conditions for the training episodes. It includes logic to reset the environment if specific failure states occur, such as the robot colliding with an obstacle detected via LiDAR.

External Libraries and Software:

NVIDIA Isaac Lab: This is the basic model that is embedded on how to define the simulation tasks, control the environments and sensor inputs. NVIDIA Isaac Sim, PhysX 5: The physics engine used to perform collision detection, rigid body dynamics and terrain generation in a real world way.RSL-RL: We implemented the PPO (Proximal Policy Optimization) algorithm of the Robotic Systems Lab (RSL) to process the on-policy learning algorithm. Unitree Go2 Asset: URDF and mesh data of the robot model were supplied by Unitree and modified to Isaac Lab.

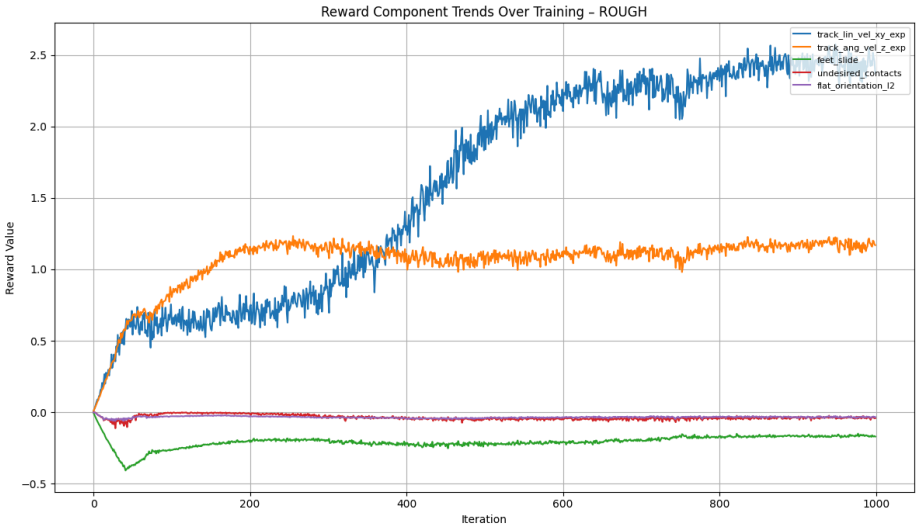
Experiments and Results

Graph: LLC Training Curve (Reward vs Iteration)

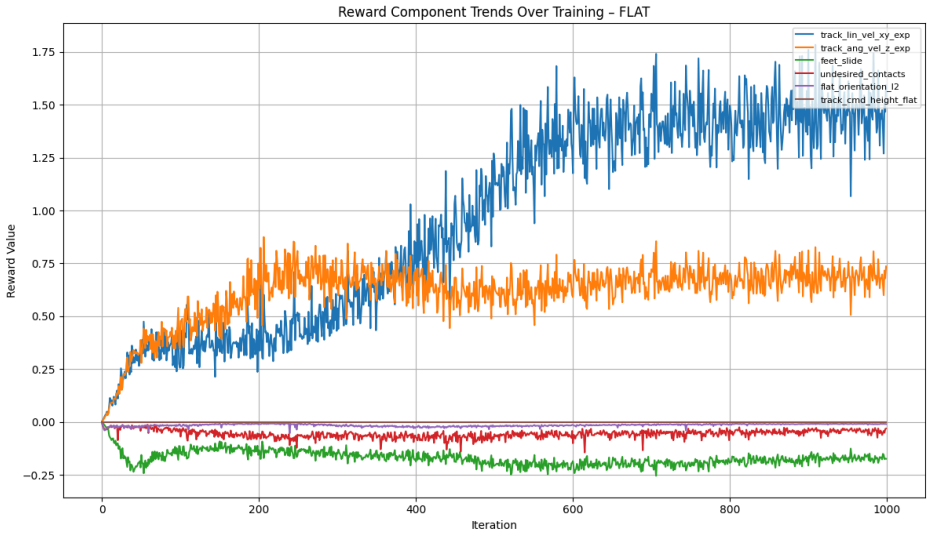


Analysis: The training curve shows a trend upwards for the mean rewards for both Flat (blue) and Rough (orange) environments indicating the agent effectively learns the task at hand. A good note shows that our agent actually performs better on rough environments than flat ones.

Graph: LLC Reward Decomposition — Flat & Rough

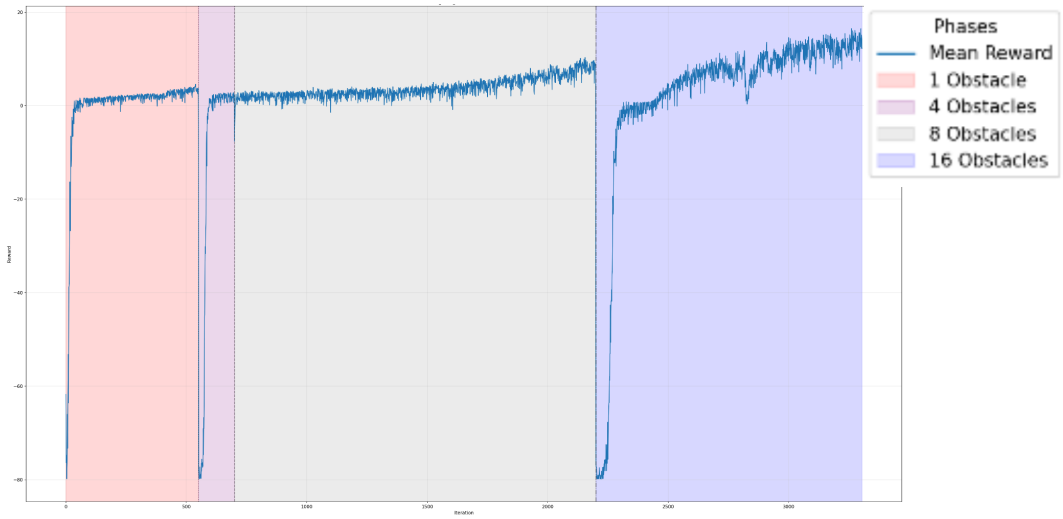


Analysis: The agent presents a more dynamic process of learning on rough terrain. The linear velocity tracking (blue) increases considerably, and that is the cause of the large total reward of Figure 1. Importantly, the feet slide penalty (green) slopes (worsens) at the start of the movement of the robot and then becomes better over the time. This recovery shows that the policy is in the process of learning to place its feet so that the policy minimizes the risk of slipping on rough surfaces, a major feature of robust rough-terrain locomotion.



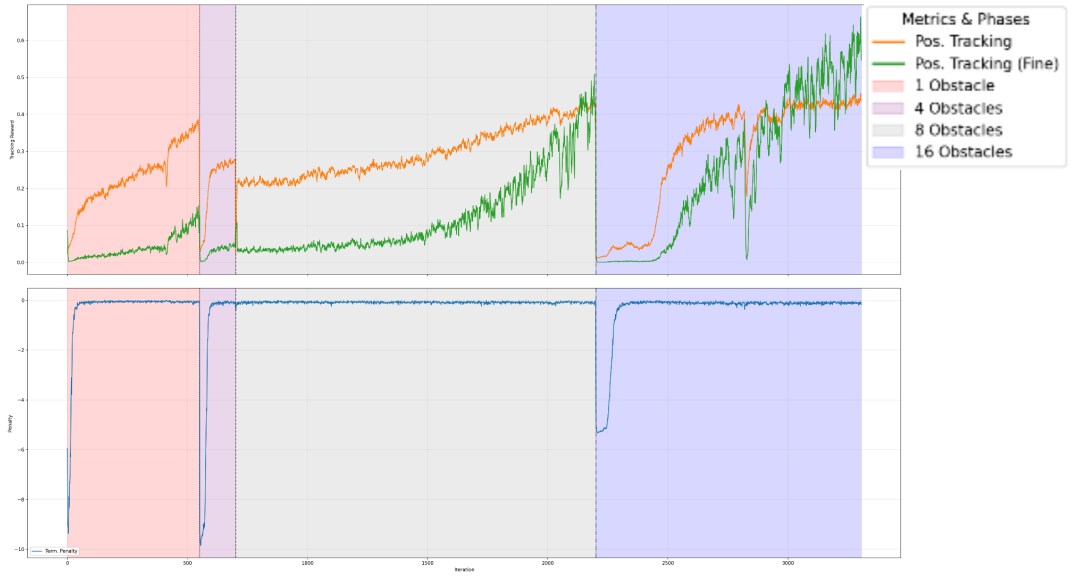
Analysis: In flat land the reward signal can be seen to be controlled by the linear velocity tracking term (blue), which tends to increase consistently to about 1.5, which confirms that the robot is learning to walk along with the commanded speed. Angular velocity reward (orange) levels off early on which means that orientation control is quickly learned. Notably, the feet slide penalty (green) also does not increase significantly, which indicates that at high-friction flat terrain, the slip is not a significant enemy of locomotion.

Graph: HLC Training Curve (Reward vs Iteration)



Analysis: The blue line tracks the mean reward of the RL at each iteration throughout the training. The different shades represent the number of obstacles (randomly placed cubes) that we used at each stage. So we started with 1 obstacle (red), and the reward started to plateau pretty quickly. Once we bumped it up to 4 obstacles (pink), the reward plummeted, but eventually rose once it understood how to navigate the obstacles. Interestingly, when we went to 8 obstacles (green), it hardly took a hit, and eventually rose even higher than before. Finally, at 16 obstacles (blue), it took an immediate hit and took a while to recover, but eventually rose to its highest peak by the end. It almost seems like challenging the RL ended up making it even smarter than it was originally.

Graph: HLC Reward Decomposition



Analysis: This graph is the breakdown of the previous, where instead of looking at the mean reward, we look at the components that combine to become that mean. The mean reward is made up of two position tracking rewards (normal and 'fine-grain'), minus the termination penalty (falls and collisions). The normal position tracking reward just rewards moving closer to the goal, while the fine-grained reward gives a higher reward when the bot is very close to the goal. From this graph, you can see similar trends to the mean reward graph, like the plummets dealing with 4 and 16 obstacles. The key thing to look at is how the fine-grained arrival bonus doesn't really do much until around 1500 iterations, then it starts growing exponentially. When dealing with 16 obstacles, you can clearly see how the RL responded; it first tried to get rid of the termination penalty, and didn't try moving to the goal, then once it stopped terminating, it finally started to try and increase the position tracking rewards. The final interesting thing is the massive drop-off around 2800 iterations, and we aren't too sure what caused this.

Discussion & Conclusion

Link to Final Showcase Video:
https://drive.google.com/file/d/1fSjx_qlbS441zV9r0YLOX6ebqNFsbNmp/view?usp=sharing

Discussion: This project has been able to show that hierarchical reinforcement learning can be used to train a quadruped to navigate complex environmental conditions. One important lesson of our effort is the significance of exteroception. It was challenging to move the agent on rough ground without vision, but adding the height scans of RayCasters greatly aided in allowing the agent to predict and respond to alterations in the terrain. Also, we discovered that reward shaping is essential. In the absence of dedicated re-incentives against foot sliding and orientation constraints, the policy would make use of the physics engine (e.g. sliding on knees) instead of acquiring an actual gait.

Future Work: To improve our current project, we would start by improving the LLC. One reward that we realized we should have added was one to encourage the bot to stand still when not actively moving (currently, the bot shakes around a lot). We also started to add functionality to input commands for not just how the LLC should walk, but also how it should walk. This would allow the HLC to command the robot to crouch, high-step, speed up its pace, etc. As interesting as this would be, we decided to scrap it due to the extra complications and training that would be required. The biggest change we would need to make before going sim-to-real would be to use the proper Ray-Tracing-based LiDAR cameras offered by IsaacLab, rather than the Raycaster rays we currently use. Raycaster is the fast-running way to get depth rays in Isaac Lab, but using the real RTX LiDAR would properly simulate a real LiDAR camera. Once we use the RTX LiDAR rays, we can use it to create a more realistic SLAM algorithm. Finally, it would be necessary to add plenty of domain randomization policies to deal with the gap between sim and real worlds.

Conclusion: Using Isaac Lab simulations, we have managed to train a hierarchical controller, which decouples locomotion dynamics and navigation logic, and uses global path planning to efficiently navigate. We confirmed that a hierarchical RL, fed information about the optimal global path, can be an effective method of goal navigation, combining the efficiency of SLAM with the robust intelligence of Reinforcement Learning.

Individual Contributions (Per teammate)

Leonardo Gutierrez:

Due to my lack of NVIDIA GPU availability, I used Openlab for all my contributions; which in turn had me basically use experimental runs and statistical analysis through logs to contribute reward values for PPO. The GitHub version of my run was through IsaacLab 4.5 instead of the latest due to hardware constraints but was still able to run code just fine. Through this I also developed training/evaluations scripts to both run and manage the log files. Through these scripts I created the graphs seen above for both the report and presentation and managed the presentation for the proposal, final presentation, README, and authored the final report.

Elijah Tabachnik :

I, along with Adam, installed and configured Ubuntu to establish a stable robotics/RL development environment, then set up and debugged Isaac Lab with a focus on low-level gait and locomotion training. I worked through environment and pipeline issues (dependency/configuration problems, runtime errors, and training stability bugs) to get training runs executing reliably, and I oversaw most of the training overnight—launching runs, monitoring performance and failures, adjusting parameters when needed, and ensuring outputs/logs were captured for later analysis. In addition, I designed and ran a comparative experiment to evaluate an “emergent/single-task” setup that provides only the goal signal and observes what locomotion behavior develops without staged scaffolding. We ended up going with a curriculum+high-level brain approach. I compared these approaches in terms of learning stability, convergence speed, and the qualitative structure of learned movement, using the results to inform subsequent training choices and iteration on the locomotion setup.

Adam Gans:

My main contributions were towards planning and designing the algorithm and environment of the RL. I did research on previous creations of both locomotion and navigation robots, specifically quadrupeds. I planned out different ways that we could incorporate global path planning (from SLAM) into a fully RL robot, in the end designing our final LLC+HLC design, with a HLC that used “lookahead hints” from global path planning to assist it. I helped Elijah install Ubuntu and Isaac Lab on his PC. I helped design the observation and reward spaces. I helped create code to run Isaac Lab simulations and training, then tweak the rewards based on the results.