Path Planning for a Quadruped in Arbitrary 3-Dimensional Environments

Siddharth Saha, *RI,* Sayan Mondal, *RI,* Zhikai Zhang, *MECHE,* and Gregory Su, *ECE*

Abstract

Navigating challenging terrains demands a nuanced understanding of the robot's versatile capabilities. Highly agile mobile robots endowed with complex locomotion skills, including climbing, jumping, and walking can often better handle such environments than traditional wheels, but planning for such systems can often be much more difficult. This paper introduces a comprehensive framework for real-time autonomous navigation of such motions by determining path costs using a neural network-based cost predictor by leveraging elevation maps and A* path planning. We demonstrate in simulation that this system can plan paths for multiple different cost functions, such as minimum energy or minimum time, depending on the specified criteria.

I. Introduction

In recent years, the substantial advancements in the control of legged systems have significantly enhanced the capabilities of quadrupedal robots, particularly in navigating intricate terrains. Operating in diverse environments introduces various motion outcomes and associated risks, necessitating robots to traverse along optimal paths with minimal travel expense and failure probability based on local information. The efficacy of their locomotion is intricately tied to the specific locomotion controllers employed. Consequently, conventional navigation methods that merely categorize traversability and perceive planning costs as distances are inadequate for addressing the complexities of navigation tasks faced by quadrupedal robots.

A prevalent research avenue involves deriving motion costs from topographical information and obtaining an optimized path using conventional planning methods. In earlier work [1], local terrain features are directly utilized to assess footprint traversability, forming the basis of the cost function. This approach employs a set of manually tuned formulas, potentially demanding specialized expertise and substantial experience with specific robots, especially as terrain complexity escalates or additional cost factors, such as energy and time consumption, are introduced. Alternatively, [2] employs a deep neural network to learn local motion cost estimates from simulation data, integrating the estimator with variations of RRT for global path planning in [3].

We introduce a navigation framework designed to facilitate swift global path planning across intricate terrains, validated through testing on the highly mobile quadrupedal robot, Unitree Go1 in simulation. Employing a methodology akin to [3], we initially train a precise local motion cost predictor within a simulation environment. This predictor estimates various motion attributes based on a local height scan and locomotion commands. To expedite the search for a globally optimized path, we perform variants of A* search on costmaps inferred by the cost predictor for any given enironment.

II. Approach

A. Controller

We trained an end-to-end policy for the quadrupedal Unitree Go1 robot using Deep Reinforcent Learning [4] in IsaacGym simulator [5]. The inputs to the policy network are the following three things: Scandots which is the privileged ground-truth information obtained in simulator for capturing the local terrain information, Oracle heading by following the defined waypoints over various obstacle-based courses and Proprioception which helps in state estimation of the robot. The output of the policy is joint motor torques. Thus, we obtain a general policy capable of a variety of motion skills like climbing and jumping in addition to walking. This policy is capable of making decisions about which skill to use based on the surrounding terrain and heading direction and is capable of following wayponts that are reachable using any of the learned skills.

B. Cost Predictor

The cost predictor forms the backbone of our framework. Our path-planning strategy can only outperform the baseline (a naive strategy that avoids all obstacles like gaps and blocks) if the cost estimates are meaningful. As we shall see in the next stage, the planner has no direct information about the robot's controller and treats the cost predictor as the oracle regarding the robot's capabilities.

The objective of the cost predictor is to predict the energy and time costs and the success probability for any given local motion. Similar to B. Yang et al. [6], we define a local motion $e(q_a, q_b)$ to denote the robot's translation and rotation from the start pose q_a to the target pose q_b. The robot state q_i contains the robot's 2D center of mass position (coordinates denoted as x_i and y_i) and orientation (denoted as ψ_i), i.e., $q_i = (x_i, y_i, \psi_i)^T$.

We take as input the local elevation map ρ_q and the desired transformation u_e to effect the local motion. The local elevation map ρ_q is centered at the robot's start pose q_a and aligned with the world frame. We define the transformation vector $u_e =$

$(\Delta x, \Delta y, \Delta\psi, \psi_a)^T$, where $\Delta x = x_b - x_a$, $\Delta y = y_b - y_a$, $\Delta\psi = \psi_b - \psi_a$. As represented in the architecture diagram (Fig. 1), we output three different costs called the energy, time, and risk costs. We shall discuss more details about the architecture in the *Implementation Details* section.

C. Planner

With the cost predictor defining the path costs now defined, the next question is how it is applied to the planner. In our planner, each state is a 2-dimensional coordinate in space that defines the robot's current position. Each (x, y) coordinate is defined by an input environment, where each coordinate has a height in meters relative to the ground plane that could result in an obstacle to the robot's motion. During planning, we don't necessarily consider waypoint connections to be adjacent pixels in our heightmaps, because pixels are very small distances that may be infeasible for our robot to precisely navigate. Instead, each waypoint connects to waypoints that are some distance away in pixels; in general, we assume that a connection is half the width of the robot. This has the benefit of reducing the realistic size of the state space to be searched, allowing us to improve the runtime of the planner. The independent variables of the state space are thus the coordinate positions of the robot and the environmental heightmap. The dependent variables necessary for the cost predictor alongside the start and end positions are the height positions at defined positions around the robot, taken according to pixel distances away from the robot position that match the real-life sensors of the robot.

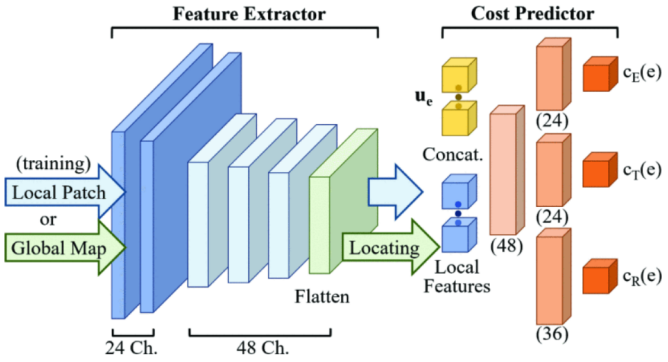

Fig. 1: Architecture of the cost predictor. Source: [6]

For our planner, we use an implementation of A* [7], where the heuristic is the minimum expected energy and/or time cost for traveling a given distance over flat ground, which in our problem is assumed to be the cheapest possible case for a given straight distance. The minimum expected cost for flat ground was determined by our cost predictor training set for the experimental costs over distances in any situation by taking the cost/distance value at one standard deviation below the mean. In general, we expect this to be a consistent heuristic, since it is simply euclidean distance multiplied by a constant factor that should be less than the minimum cost to actually traverse the given distance. As such, we expect generally optimal planning outputs based on motion costs from our learned cost predictor.

III. Results

A. Implementation Details

1) Environmental Setup: In order to measure the performence of our planner after execution on the robot, we set up the Issac gym simulator such that it reads the generated waypoints from the planner and directs the robot to follow these waypoints, replacing the oracle heading, with its RL-based controller.. We then collect the total time (s) and energy (J) taken for the Robot to take. We have prepared two types of terrains, hard and easy, where the gap size and depth remains the same but the block height is 0.4 m and 0.1 m respectively.

2) Controller: Instead of training individual skills and planning over them, we chose to have the trained policy perform the task of a local planner as well. Thus, we have trained one generalist policy [4] instead, capable of executing all such complex maneuvers based on the local terrain information and the heading direction that is dictated by our global planner. The main reward terms for this Deep RL policy are shaped such that they make the quadruped follow the consecutive nearest waypoint and at the same time performing the local navigation elegently by minimizing the energy usage. It took roughly 20,000 epochs to train this policy. There was a good amount of effort inovlved in figururing out the Go1 robot's URDF file as well as the PD gains for converting the policy's joint angle outputs into joint motor torques. This was crucial for getting an agile trained policy.

3) Dataset Collection: Dataset collection is essential for our cost predictor to learn the capabilities of the robot. We execute the learned controller policy in Isaac Gym and run multiple experiments with different terrain configurations and start and end positions. The terrain information is stored in our dataset in the form of scandots, a privileged information that we query Isaac Gym at the beginning of the run. Scandots is another term for the elevation map.

We applied the anytime idea from the field of planning and came up with an approach called **Anytime Dataset Collection**. Our dataset collector runs multiple passes over our parameters with finer and finer granularities. For example, if we want to vary the obstacle height H between $[0m, 1m]$, in the first pass we run the coarsest granularity and only run experiments for $H = \{0, 1\}$. After iterating over all the other parameters, we exhaust this granularity, so we divide the granularity by 2, and run another pass over all parameters with $H = \{0, 0.5, 1\}$. The critical observation here is that a finer granularity will repeat all the experiments that were run with the coarser granularities. This will bias the dataset heavily, so, to combat this, we

maintain a hashset of the combination of parameters that we have run so far. As a result, we can guarantee zero repetition of experiments. Thus, we call this approach *Anytime Dataset Collection*, as we can use the collected dataset at any time and be assured of its diversity.

4) Cost Predictor: As shown in Fig. 1, we first run a feature extractor on the input scandots. The feature extractor comprises a sequence of 4 convolutional networks and ReLU activation functions so that our predictor can reason about the spatial features. The output contains the local terrain features. We flatten this information, combine it with the desired local motion, and pass it through a Linear layer with a ReLU activation. This whole structure forms the common backbone of the cost predictor. Our network splits into three heads from here, corresponding to energy, time, and risk costs. Each head comprises a sequence of two Linear layers with a ReLU activation in between. Thus, each head is capable of modeling a non-linear function.

5) Planner: To implement our planner, as described in Section II-C, we built it in C++, with the cost predictor called using the embedded Python interface [8]. We built many variables to be configurable, including the number of pixels that equate to half the width of the UniTree Go1 [9]. In practice, we defined the side of a pixel to have a length of 0.05 meters, allowing us to convert pixels to real distances based on this value.

The termination condition of A* is usually the robot reaching its goal [7]. However, because we have discretized our environment, the exact coordinates of the goal are not guaranteed to be within the search space, so we add a secondary termination condition that checks if we are closer than one step from the goal, in which case we immediately add a connection to the goal and exit. One step is considered to be half the width of the robot in pixels, or 4 pixels (0.2 m). Once the goal is found, the parent tree is then used to backtrack to the beginning to find the final optimal path.

As mentioned in Section II-C, the heuristic we use is expected to be consistent. We found that using simple euclidean distance generally resulted in the heuristic being too weak to direct search, because the numerical scales of euclidean distance and energy cost were too disparate, which required us to use a higher weight. Unfortunately, because our solution is based on data post-processing, there may be some cases where the solution may be suboptimal because of the approximate nature of the heuristic. On the other hand, it is not expected to be extremely suboptimal because we use a low value for energy relative to the dataset.

B. Metrics

In analyzing the performance of the cost predictor, we use the same metric used during training. Because the two costs, energy, time, are on significantly different scales, the outputs and label costs are normalized between 0 and 1 by dividing by the maximum label values. These normalized values are then used to compute the Mean Square Error between the outputs and the targets, which are then added together to compare how well the cost predictor performs overall for all three costs. The probability of success will be used as cut off value that determines whether a state will be expanded.

To verify the performence of the Path Planner, we compare our cost predictor-based planner with a baseline ground-plane planner, which runs the planner without a cost predictor by completely avoiding obstacles. The edge cost is then simply the euclidian distance between the neghbouring states. The risk will be inflated in the face of obstacles, thus preventing the states from being expanded by the A* search. The resulting time and energy costs after running the two planners are then tracked by the robot in Issac gym and compared against each other.

C. Cost Predictor Performance

Because of the experimental and specialized nature of our environment, no strong data regarding state of the art performance for cost predictors in our domain exists. As such, we compare how well the cost predictor performs relative to purely random outputs from our untrained neural net architecture. These results can be seen in Table I.

TABLE I: Cost Predictor Normalized MSE Loss (x10^4)

Model	Overall	Energy	Time	Success Probability
Random	48.6152	5.0111	4.8503	30.3333
Trained	0.7852	0.2923	0.0600	0.3988

As can be seen, our trained cost predictor consistently performs better by an order of magnitude or more when compared to a random neural network. The success probability, in particular, improves significantly, although it is also the greatest source of normalized loss in both random and trained cases.

Qualitatively, the performance of the cost predictor can also be seen in the cost heatmap shown in Figure 2. As expected, costs near the starting position are generally lower than those that are farther away, and costs that involve passing an obstle, or especially going over it, are more expensive than staying locally.

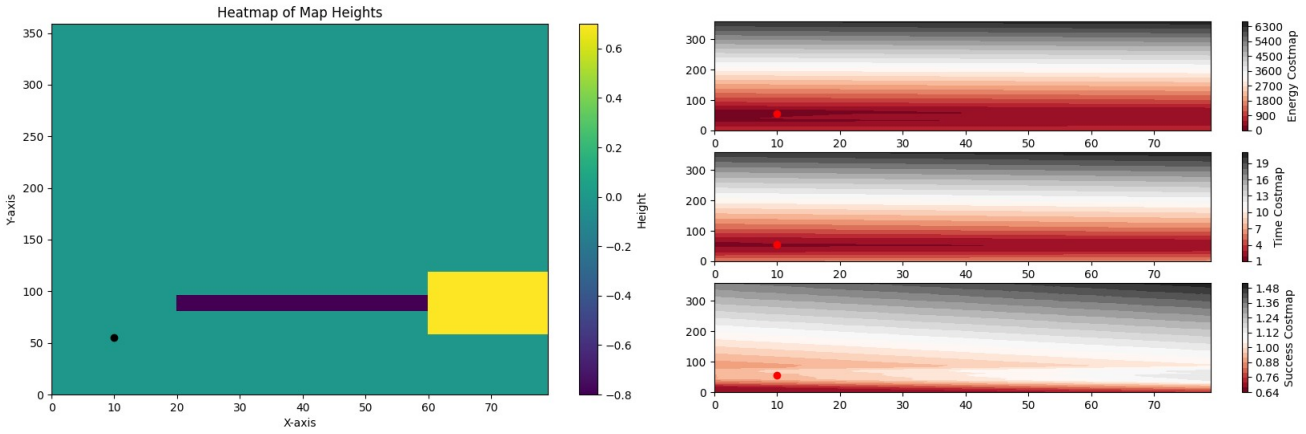

Fig. 2: Heatmap of cost predictor outputs for a given environment. The environment is seen on the left, the costs on the right, with energy, time, and success probability from top to bottom. The colored dots represent the position of the robot when analyzing costs.

TABLE II: Path Planner Output Path Costs

Cost Predictor	Easy Map Energy (J)	Easy Map Time (s)	Hard Map Energy (J)	Hard Map Time (s)
No	850.17	9.76	771.45	8.32
Yes	742.55	6.899	699.18	6.98

D. Planner Performance

As can be seen, our cost predictor based planner consistently performs better when compared to the baseline, in both easy and hard environments, in both energy and time cost. This is also seen qualitatively in 3, where the path generated by our planner on the right takes a shorter pathjumping over the gap while the baseline planner goes around the entire gap, incurring more energy and time cost. There are, however, cases when execution wasn't successful because the planner didn't take the heading of the robot into account. When the waypoints are too close to the obscale, robot will attempt to jump onto the block due to its scandots and also try to move back to its waypoint position. This causes instability where the robot will fall and thus considered a failure case.

IV. Conclusion

In this paper, we introduced a controller-aware navigation approach for robots operating in challenging terrains. The locomotion costs crucial for planning are acquired through the deployment of the locomotion controller in simulation, which are then estimated by a neural network during A* planning. We also showed that this method is effective at improving upon more naive approaches even in environments with varying obstacle heights.

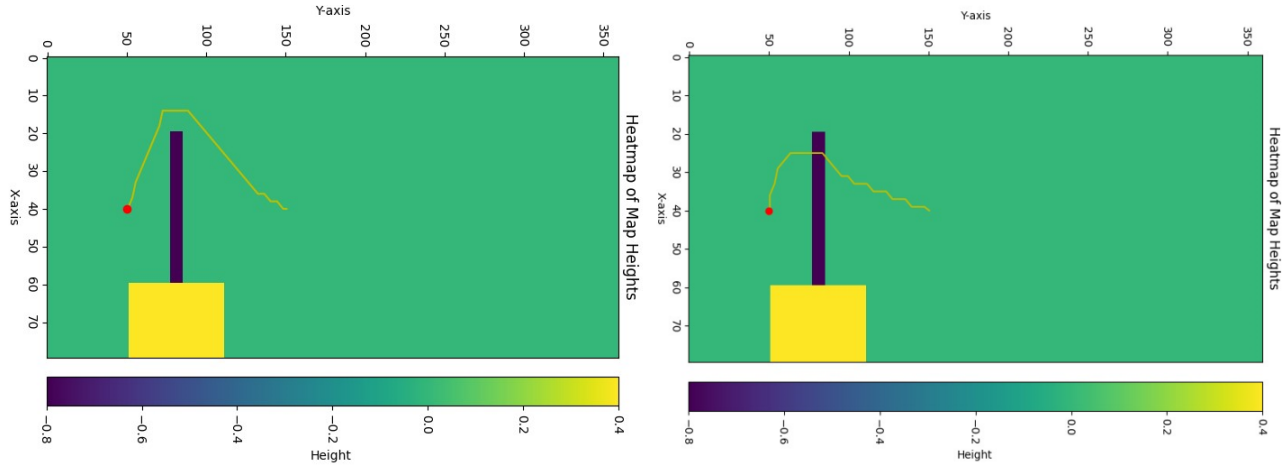

Fig. 3: Baseline Planner (left) vs Our Planner (right) on Hard Environment

The versatility of our approach is underscored by its applicability to multiple types of robots, facilitated by learning-based motion cost estimation and training data obtained from simulation. This feature streamlines the deployment of our planning framework on diverse robotic platforms with minimal manual effort. Future work includes adding heading for a 3D planning problem to avoid conflict in controller behaviour and waypoints and improving on the cost predictor. The training dataset comprised of 5000 data points which is not enough for a deep network with convolution and linear layers. We should add skip connections to prevent vanishing gradient problems from occurring. Also, we would like to incorporate a gradient-based optimizer to obtain an optimised path from the rough paths that are obtained from our algorithm due to the strict step sizes. We also would like to improve our waypoint sampling strategy to allow the policy to overcome obstacles by following smoother paths instead of each and every closely and precisely spaced waypoint.

REFERENCES

[1] Martin Wermelinger et al. "Navigation planning for legged robots in challenging terrain". In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2016, pp. 1184–1189. DOI: 10.1109/IROS.2016.7759199.

[2] R. Omar Chavez-Garcia et al. "Learning Ground Traversability From Simulations". In: *IEEE Robotics and Automation Letters* 3.3 (2018), pp. 1695–1702. DOI: 10.1109/LRA.2018.2801794.

[3] Jérôme Guzzi et al. "Path Planning With Local Motion Estimations". In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 2586–2593. DOI: 10.1109/LRA.2020.2972849.

[4] Xuxin Cheng et al. "Extreme Parkour with Legged Robots". In: *arXiv preprint arXiv:2309.14341* (2023).

[5] NVIDIA Corporation. *NVIDIA Isaac Gym*. URL: https://developer.nvidia.com/isaac-gym.

[6] Bowen Yang et al. "Real-time Optimal Navigation Planning Using Learned Motion Costs". In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. 2021, pp. 9283–9289. DOI: 10.1109/ICRA48506.2021.9561861.

[7] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107. DOI: 10.1109/TSSC.1968.300136.

[8] *Embedding Python in Another Application*. URL: https://docs.python.org/3/extending/embedding.html.

[9] *Unitree Go1*. URL: https://shop.unitree.com/products/unitreeyushutechnologydog-artificial-intelligence-companion-bionic-companion-intelligent-robot-go1-quadruped-robot-dog.