2021 IEEE International Conference on Robotics and Automation (ICRA 2021)
May 31 - June 4, 2021, Xi'an, China

2021 IEEE International Conference on Robotics and Automation (ICRA) | 978-1-7281-9077-8/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICRA48506.2021.9561861

Real-time Optimal Navigation Planning Using Learned Motion Costs

Bowen Yang[1]*, Lorenz Wellhausen[2], Takahiro Miki[2], Ming Liu[1], Marco Hutter[2]

***Abstract*— Navigation on challenging terrain topographies requires the understanding of robots' locomotion capabilities to produce optimal solutions. We present an integrated framework for real-time autonomous navigation of mobile robots based on elevation maps. The framework performs rapid global path planning and optimization that is aware of the locomotion capabilities of the robot. A GPU-aided, sampling-based path planner combined with a gradient-based path optimizer provides optimal paths by using a neural network-based locomotion cost predictor which is trained in simulation. We show that our approach is capable of planning and optimizing paths three orders of magnitude faster than RRT* on GPU-enabled hardware, enabling real-time deployment on mobile platforms. We successfully evaluate the framework on the ANYmal C quadrupedal robot in both simulations and real-world environments for path planning tasks on multiple complex terrains.**

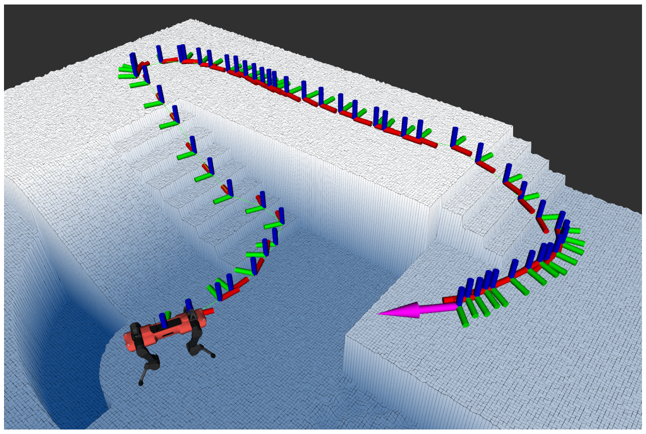


Fig. 1. Our navigation framework returns an optimized path based on elevation maps. Since the planner is aware of the locomotion capabilities of the robot, it plans a safe, yet efficient path over challenging terrain.

I. Introduction

With the intensive research on legged system control in recent years, quadrupedal robots are presenting increasingly strong capabilities when operating on complex terrains. The robot may have various motion outcomes and undertake different risks when interacting with diverse environments [1] and is expected to move along a smooth and reasonable path with minimum travel expense and failure probability provided the local information. Furthermore, their locomotion capabilities strongly depend on the applied locomotion controllers. Therefore, traditional navigation approaches which simply classify traversability and understand the planning cost as distances, are unsuitable for quadrupedal robots confronting complicated navigation tasks.

A popular research direction formulates motion costs from topographical information and acquires an optimized path with traditional planning approaches. Local terrain characteristics are directly used to evaluate footprint traversability and construct the cost function with a group of hand-tuned formulas in [2], which may require strong specialty and extensive experience on specific robots when the terrain complexity increases or more cost factors such as energy and time consumption are introduced. [3] adopts a deep neural network that learns local motion cost estimates from simulation, and [1] combines the estimator with variants of RRT for global path planning. However, the pure sampling-based planning frameworks require frequent queries of the network, which result in long planning times and may be impractical for real-time applications.

We present a navigation framework that supports rapid global path planning and optimization on complex terrains and test it on an ANYmal C [4], a quadrupedal robot with high mobility. We first train an accurate local motion cost predictor in simulation in a similar approach [1], which estimates several motion attributes based on a local height scan and locomotion commands. To search for a globally optimized path in a short time, we rapidly build a roadmap by batch-processing of predefined local motions on a GPU which we use to obtain a raw path using the A* algorithm. To further improve the path, a gradient-based path optimizer tunes the position and orientation of each intermediate state on the raw path and further optimizes the planning cost to find a final path for direct execution. We deploy the whole framework in simulation and on the real ANYmal C robot, demonstrating that the navigation loop is able to run in real-time on mobile computation hardware. The main contributions of this work include:

- Creating a sampling-based planner which exploits batch-processing for parallel roadmap construction on a GPU to obtain a path close to the global optimum in a much shorter time than traditional approaches.
- Developing a novel gradient-based path optimizer that leverages learned motion costs.
- Demonstrating the whole framework in reality and successfully conducting real-time autonomous navigation on the ANYmal C quadrupedal robot.

[1]Authors are with the Robotics and Multi-Perception Laboratory, Robotics Institute, The Hong Kong University of Science and Technology, Hong Kong SAR, China. byangar@connect.ust.hk Ming Liu is the corresponding author. eelium@ust.hk
[2]Authors are with the Robotic Systems Lab, ETH Zürich, Switzerland.
*Substantial part of the work was carried out during his stay at 2

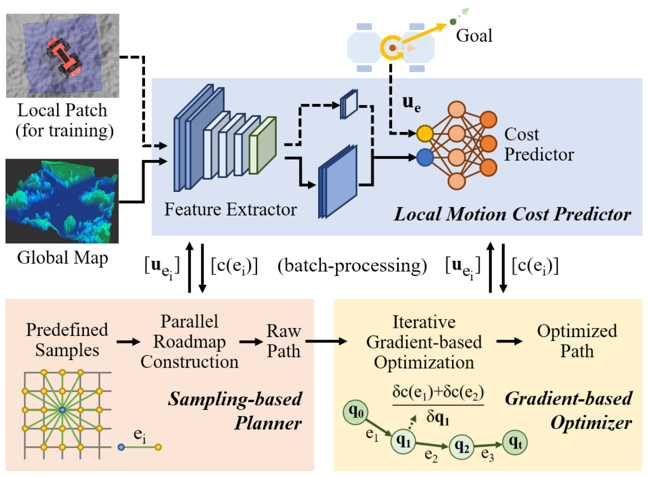


Fig. 2. A local motion cost predictor trained in simulation provides cost estimates to a sampling-based planner and a gradient-based optimizer. The path planner achieves fast computation time by sampling a fixed set of motions batched on GPU. The gradient-based optimizer refines the raw path by applying cost-gradients to path nodes.

II. Related Work

One possible way to solve the robotic navigation problem is to directly learn a planning policy. For instance, a Gaussian process model is applied to improve navigation on different terrains over time [5], which requires repeated attempts for a better solution. Silver *et al.* [6] train an agent from human demonstration for planning on unstructured terrains. Pfeiffer *et al.* [7] present an end-to-end framework for planning in a 2D view using a convolutional neural network (CNN) that learns from imitation. However, these approaches either require a large number of manual demonstrations or rely on another existing global motion planner. Reinforcement learning is also applied in commanding the robot to reach the target while avoiding obstacles on a flat ground [8], but this approach might be unsuitable for applications on complex terrains due to the extension of observation dimension, which brings difficulty in network training.

Another research direction focuses on predicting the motion properties from environmental information. Point cloud data is directly used to evaluate terrain traversability for optimal six-dimensional navigation [9] or online path planning through GPU accelerated tensor voting [10]. Elevation information obtained from raw perception is employed to construct a traversability map for individual cells [11] or quadruped footprints [2]. Learning-based methods help the prediction to generalize better in novel environments. Wellhausen *et al.* [12] train a CNN to predict terrain properties from image data and use them to compute locomotion costs. In addition, regressing the robot motion outcomes gathered from simulation helps to formulate the planning cost function in a more convenient and intuitive way [6]. A CNN trained with simulated data can outperform feature-based approaches in traversability classification [3] and can be used in a framework that estimates multiple local motion descriptors in continuous configuration space to construct the cost function for path planning using RRT* and Stable Sparse RRT [1]. Nevertheless, it requires a long planning time to successively predict a large number of sampled motions for a smooth and feasible path using a deep neural network. Based on the concept of estimating local motion costs, we present a novel approach that takes advantage of parallel computing on GPUs to accelerate the path planning and optimization process.

III. Approach

The navigation framework comprises three components (Fig. 2): The local motion cost predictor estimates several attributes of a given motion command over the local terrain. The sampling-based path planner computes a rough path which is close to the global optimum. Finally, the gradient-based optimizer improves the rough path to find the locally optimal path.

A. Local Motion Cost Predictor

We reference the model in [1] to develop the local motion cost predictor: Each robot state $\mathbf{q_i}$ in the configuration space Q contains the robot's center position x_i, y_i and the heading direction ψ_i in the world frame:

$$Q = \{\mathbf{q_i} | \mathbf{q_i} = (x_i,\ y_i,\ \psi_i)^T\}.$$

A valid local motion $e(\mathbf{q_a},\ \mathbf{q_b})$ denotes the translation and rotation from the start pose $\mathbf{q_a}$ to the target $\mathbf{q_b}$. The local height scan ρ_q is centered at $\mathbf{q_a}$ while its direction aligns with the world frame. We define the transformation vector $\mathbf{u_e} = (\Delta x,\ \Delta y,\ \Delta\psi,\ \psi_a)^T$, where

$$\Delta x,\ \Delta y = x_b - x_a,\ y_b - y_a,$$

$$\Delta\psi \in (-\pi,\ \pi] = \text{the rotation angle from } \psi_a \text{ to } \psi_b,$$

$$\psi_a = \text{robot's initial yaw angle on } \rho_q.$$

From ρ_q and $\mathbf{u_e}$, the predictor returns normalized motion costs $c_E(e)$, $c_T(e)$, $c_R(e)$, which represent the extent of the general energy consumption E, time duration T, and the failure probability (risk) R of the robot in finishing the local motion e. The prediction network consists of a convolutional *feature extractor* and a fully connected *cost predictor* (Fig. 3), which are jointly optimized. The *feature extractor* generates terrain features from a height scan around the current robot position. Then, the features combined with the transformation vector $\mathbf{u_e}$ are passed to the *cost predictor*, which outputs motion cost estimates. Because the *feature extractor* is convolutional, we can efficiently precompute features for an entire map, which accelerates the path planning and optimization steps. This, combined with batch-processing queries in the *cost predictor*, is how we leverage the massive parallel processing power of modern GPUs to achieve real-time performance.

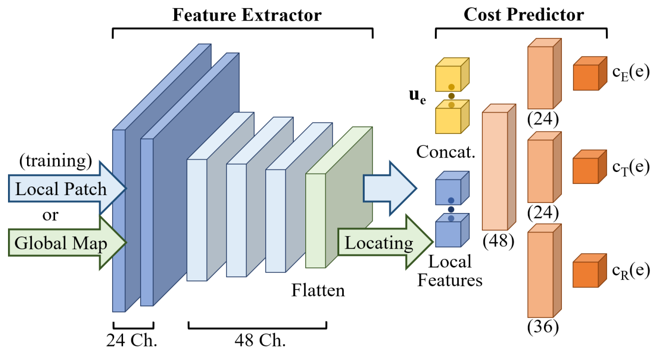


Fig. 3. The prediction network contains a *feature extractor* that is constructed using convolutional layers, which supports height map inputs of varying size, and a *cost predictor* using fully connected layers, which uses the localized map features at the robot location and the motion commands to predict motion costs.

B. Path Planner and Optimizer

The navigation problem is modeled as globally minimizing the cost to move from the initial state $\mathbf{q_0}$ to the target $\mathbf{q_t}$ in a given search space Q. We interpret the path P as sequential robot configurations: $P = \{\mathbf{q_0},\ \mathbf{q_1},\ \dots,\ \mathbf{q_t}\}$ with $\mathbf{q_i} \in Q$. Sampling-based planners are widely adopted to solve high-dimensional planning problems in robotics [13]. For instance, the probabilistic road map (PRM) algorithm [14] randomly samples free states inside Q and checks the connectivity among their neighbours to build a roadmap, then returns a solution through a graph traversal approach. In our case, for a motion e between two adjacent states, the linking state $l(e)$ can be obtained by taking a threshold on R, and the cost $c(e)$ can be formulated from E, T, and R, which are represented by predicted motion costs $c_E(e),\ c_T(e),\ c_R(e)$ in Section III-A.

For path optimization, PRM* [15] achieves asymptotic optimality [13] by continuously adding samples to the search graph. However, in our case, it may result in an unacceptable planning time if we frequently query the prediction network on new samples to obtain an adequately smooth path for deployment. Directly applying batch-processing on multiple samples may accelerate the prediction process, but the locations and the lengths of randomly sampled motions may vary greatly, which requests massive individual preprocessing for valid network inputs. Therefore, we define a specific robot state group along with its corresponding edge set from a grid map to simplify the tensor preparations of parallel prediction. After obtaining a rough path through graph traversal, we use a gradient-based method for further optimization.

1) Path Planner: We modify the PRM algorithm to develop the path planner for parallel roadmap construction. We initialize the graph by sampling a group of uniformly distributed grid states $\mathbf{q_{gi}}$ from a grid map which covers the planning space Q and guarantees appropriate distances between neighbouring states. After each update of the global map, the predictor on the GPU is provided with predefined coordinates to locate local features and corresponding transformation vectors $\mathbf{u_{e_i}}$. It then returns motion costs $c_E(e_i),\ c_T(e_i),\ c_R(e_i)$ for each e in the edge set. We obtain connectivity $l(e_i)$ by applying a threshold R_{max} as the maximum allowed risk for traversable edges and obtain the motion cost $c(e_i)$ through a weighted sum of the individual cost terms:

$$l(e_i) = true \text{ where } c_R(e_i) < R_{max},$$

$$c(e_i) = w_E c_E(e_i) + w_T c_T(e_i) + w_R c_R(e_i).$$

Because the number and distribution of samples depend on a fixed grid, this approach is probabilistically incomplete and can fail in narrow environments due to aliasing effects. To overcome this, we introduce *vague sampling*: For each e_i, we apply multiple random tiny location and orientation perturbations, as shown in Fig. 4, to obtain a group of vague samples $V = \{v_1,\ \dots,\ v_{n_v}\}$, where n_v denotes the group size. We choose the minimum risk value among the grid sample $c_R(e_i)$ and vague samples $c_R(v),\ v \in V$ as the final risk $c'_R(e_i)$ of the motion, and adapt connectivity $l(e_i)$ based on this:

$$c'_R(e_i) = \min(c_R(e_i),\ c_R(v_1),\ \dots,\ c_R(v_{n_v})),$$

$$l(e_i) = true \text{ where } c'_R(e_i) < R_{thresh}.$$

Vague sampling enables the planner to consider a large number of possible motions in continuous space, which increases the probability of finding a solution. V is only used to refine the connectivity of e_i and is excluded from the roadmap, which avoids introducing additional nodes and therefore computational complexity to the graph search. The roadmap still uses the original cost $c(e_i)$, such that the original risk value $c_R(e_i)$ is used for cost computation rather than the possibly lower $c'_R(e_i)$. This way, the graph traversal algorithm typically prefers safer paths and does not rely on *vague sampling* to find connectivity.

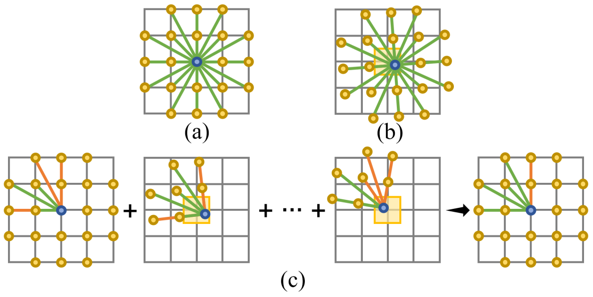


Fig. 4. All sampled local motions (green lines are traversable, orange lines untraversable) between a grid state (blue) and its neighbours (yellow) shown in (a) are shifted and rotated within a small range to create vague samples (b). We then combine all traversable edges obtained from grid and vague samples to determine the connectivity of the planner graph (c), which has nodes placed on the grid.

2) Path Optimizer: We design a gradient-based path optimizer to iteratively adjust path nodes based on their motion costs. Provided a raw path $P_{raw} = \{\mathbf{q_0},\ \mathbf{q_1},\ \dots,\ \mathbf{q_t}\}$ with t local motions $e_i(\mathbf{q_{i-1}},\ \mathbf{q_i}),\ i \in [1,\ t]$, the local motion cost predictor returns cost terms $c_E(e_i),\ c_T(e_i),\ c_R(e_i)$. To keep the distance d_i between nodes in the range for a valid cost prediction, the optimizer ignores micro motions which are smaller than prediction resolution and introduces an additional penalty p_{d_i} if d_i is overlong: $p_{d_i} = \omega_p {d_i}^2$ if $d_i > d_{max}$, where ω_p is the penalty factor and d_{max} is the maximum prediction distance. The local costs $c(e_i)$ and the global planning cost $f(P)$ for path P are formulated as

$$c(e_i) = \omega_E c_E(e_i) + \omega_T c_T(e_i) + p_{d_i},$$

$$f(P) = t \cdot \omega_R \cdot \max(c_R(e_i)) + \sum_{i=1}^{t} c(e_i),$$

where ω_E, ω_T, ω_R are the weights for different costs and the optimizer regards the highest risk value among all motions on the path as the overall risk of the path. The initial state $\mathbf{q_0}$ is determined by the robot's current pose and the target $\mathbf{q_t}$ is manually selected. Therefore, the objective of the optimizer is to take all intermediate states $\mathbf{q_i} = (x_i,\ y_i,\ \psi_i)^T$, $i \in [1,\ t-1]$ as parameters and minimize the overall planning cost, which is performed through iterative gradient-based optimization:

$$P_{optim} = \arg\min_{P}\ f(P).$$

IV. Implementation Details

A. Local Motion Cost Predictor

1) Dataset: We simulate the ANYmal C in Raisim [16] with a state-of-the-art learning-based locomotion controller [17], which enables the robot to travel on challenging terrains. The data collection scheme is adapted from [1] using more samples and 3-DoF motion commands. Considering the robot's collision space, the local height scan ρ_q is 2m×2m around the robot's geometric center with a resolution of 4cm. In each attempt, we randomly sample a local motion e with a translation distance within $(0.0,\ 0.5]$m and a rotation $\Delta\psi \in (-\pi,\ \pi]$, while ensuring a minimum $|\Delta\psi| = 10$deg if the motion distance is close to zero. The general energy and time consumption values E, T are approximated by averaging over 12 repeated attempts while applying trivial changes to the terrain and randomizing friction coefficients $\mu \in [0.75,\ 0.80]$ between the robot's feet and the ground. The motion risk $R \in [0.0,\ 1.0]$ is computed as the empirical failure probability over all repeats.

We collect 370k samples by simulating the robot on multiple randomly generated terrains, including structured terrains, like simple stairs, slopes, and steps with multiple sizes and steepness (Fig. 5 (a)(b)), as well as irregular terrains through compositions of features that are generated through the additive combination of Perlin noise at different scales (Fig. 5 (c)(d)). To improve the prediction accuracy in narrow environments, 36% of the samples are collected in narrow paths (Fig. 5 (e)(f)) with varying path width within $[0.5,\ 2.0]$m. Instead of sampling e randomly, one-third of the motions in the narrow environment command the robot to walk along the path without turning, so that the dataset will contain more meaningful samples of how to travel through narrow environments. We add noise with an

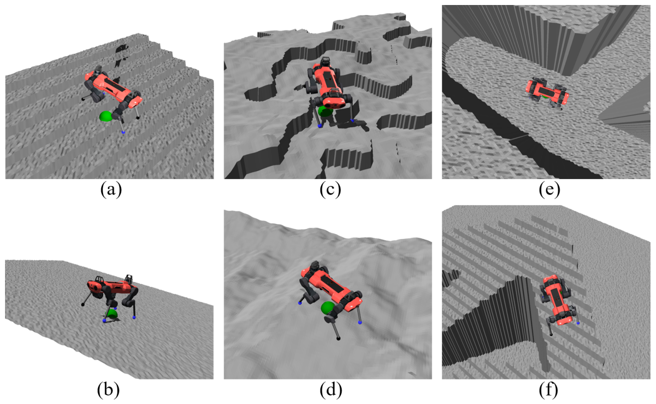


Fig. 5. We train our locomotion cost predictor in simulated environments which contains structured terrains (a)(b), irregular terrains (c)(d), and narrow paths (e)(f).

amplitude of a few centimeters to all terrains to approximate terrain roughness and mapping noise of the real system.

2) Training: We average the values of successful attempts and rescale them to $(0,\ 1]$ to obtain the training ground truth. We regard the prediction as a multi-output regression problem and train the network with the mean squared error loss. The prediction network in Fig. 3 is trained with 80% of samples in the dataset and validated with the remaining 20%.

B. Path Planner and Optimizer

1) Path Planner: In the initialization phase, we sample states from a grid mesh which is 50×50 points in size and covers Q with a resolution of 0.2m. For each state, we evaluate the motions to travel to its 20 neighbours from 16 different orientations as shown in Fig. 4 (a). For computational simplicity, the planner does not sample different target orientations but choose the direction of a motion e_i as the value of ψ_i while $\Delta\psi = 0.0$. This greatly increases planning speed and was found to yield suitable paths to initialize the path optimizer.

We take $n_v = 10$ vague samples for each local motion e_i with location perturbations in the range of $[-0.1, 0.1]$m and orientation perturbations within $[-0.4, 0.4]$rad, which generates totally 500k vague samples and magnifies the sample group by 11 times. Then, the planner creates the roadmap based on $l(e_i)$ and $c(e_i)$ introduced in Section III-B-(1). Here, we set the risk threshold $R_{max} = 0.5$, and the weights in $c(e)$ are tunable around the default values: $w_E = 5.0$, $w_T = 5.0$, $w_R = 100.0$, where w_R is significantly larger to ensure it has the highest priority in reducing path risk. We adopt A* for graph traversal with the heuristic cost function $h(\mathbf{q_{gi}})$ set to be one-tenth the euclidean distance between state $\mathbf{q_{gi}}$ and the goal $\mathbf{q_{gt}}$ to make it comparable to the cost of moving on flat ground with zero risk.

2) Path Optimizer: The path optimizer explained in Section III-B(2) locally adjusts the raw path in continuous space. We set cost weights ω_E, ω_T, ω_R to be the same values as the path planner w_E, w_T, w_R, and set the distance penalty factor to $\omega_p = 10.0$. While our cost network is fully differentiable and we could use back-propagation to obtain gradients, we found that finite-difference is faster in practice, e.g., applying a δx on the position of state $\mathbf{q_i}$ will affect local motions e_i and e_{i+1} and result in differences in local costs $\delta c(e_i)$ and $\delta c(e_{i+1})$. The gradient of parameter x_i w.r.t the path cost is calculated as

$$\nabla x_i = \frac{\delta c(e_i) + \delta c(e_{i+1})}{\delta x}.$$

∇y_i and $\nabla \psi_i$ are computed in the same way. For our experiments, we set $\delta x = \delta y = \pm 0.08$m and $\delta \psi = \pm 0.05$rad. Because the raw path may have a varying number of nodes depending on path length, we choose the Adam [18] optimizer to update P, for it can be interpreted as applying a vector of adaptive learning rates on different parameters [19] and has high performance and fast convergence speed with minimum tuning for hyper-parameters [20]. The learning rate is initially set to 0.16 and is adjusted by an exponential scheduler with decay factor $\gamma_{exp} = 0.96$ to ensure stable convergence in the late optimization stage. Normally, it requires 50 iterations for a feasible path.

V. Experiments

A. Setup

We deploy the navigation framework in both simulation and reality on the ANYmal C. The global maps used for planning are all 12m×12m and 4cm in resolution and for each map, the valid search space Q is 10m×10m and 8cm in resolution after the convolutional layers. In the simulation experiments, we compare the performance of our planning framework to the RRT* planner, as used in [1]. All the tested approaches run on a modern desktop with an AMD Ryzen 9 3950X and an Nvidia GeForce GTX 970 graphics card, and work with the same motion cost predictor that directly receives a static map without perception noise. The RRT* planner is set to finish within 150s, which results in a tree of 10k edges, and ignores the rotation optimization for simplicity. We first randomly sample start-goal pose pairs that are 4.0m in distance on three maps of different topographies including rough terrain, irregular steps, and the combination of stairs and slopes (Fig. 6 (a) to (c)). In an individual attempt, all planners are provided with the same start-goal pair and we analyze the planning times and the costs of returned paths. Next, we visualize the paths of multiple navigation tasks from RRT* and our framework in two environments: a *mountain* with unstructured terrain and a *garden* with various urban features (Fig. 6 (d)(e)).

For real-world experiments on ANYmal C, we run our navigation framework in a loop on an onboard Nvidia Jetson AGX Xavier which is also used to continuously construct and update the surrounding height scan in an unknown environment with a GPU version of elevation mapping [21], [22] with two RoboSense RS-BPearl lidars. In each experiment, we provide a target pose which is far away from the starting point and outside the initial search space. As the robot moves, our framework runs in a loop on the updated map. In each loop, the framework sets a temporary goal in the direction of

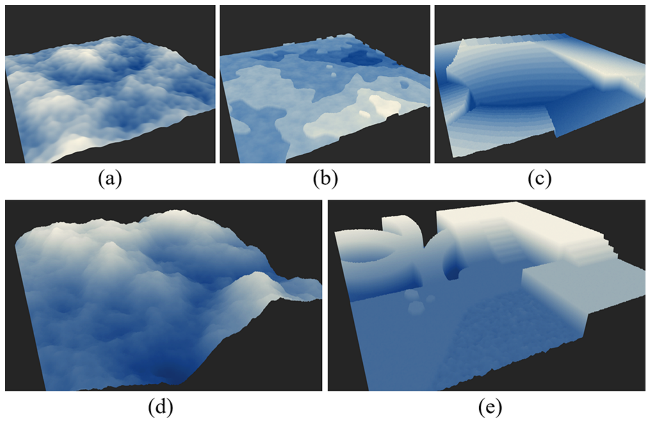


Fig. 6. The first simulation experiment is conducted on rough terrain (a), irregular steps (b), and the combination of stairs and slopes (c). Next, we prepare *mountain* (d) and *garden* (e) for the visualization experiments.

the final target, which is reachable and close to the boundary of the current search space, and provides an optimized path that encourages the robot to explore towards the final target with a simple path follower.

B. Results in Simulation

First, we compare the path costs from three planners: the RRT* planner, our sampling-based planner without optimization (*raw*), and our whole navigation framework with the gradient-based optimizer (*optimized*). On each of the three maps we record 60 successful planning attempts. Fig. 7 presents the path costs of equally distant start-goal pairs from different planners. On all terrains our path planner outperforms RRT*, even without feeding the results to the path optimizer. This indicates that our approach of combining fixed-grid and *vague sampling* is competitive with standard sampling-based approaches in producing high-quality solutions within reasonable computation times. The path optimizer can significantly reduce path cost mean and variance even further, without introducing significant computational burden.

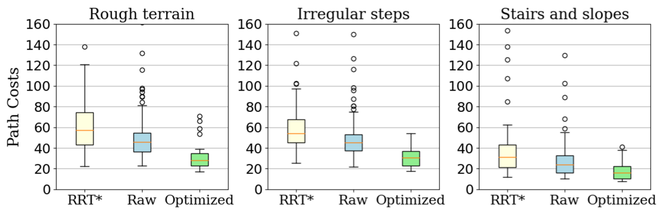


Fig. 7. Path costs from three tested planners on different maps. In all testing environments, the performance of our raw path planner is competitive with RRT* and our *optimized* approach can further reduce the path cost means and variances.

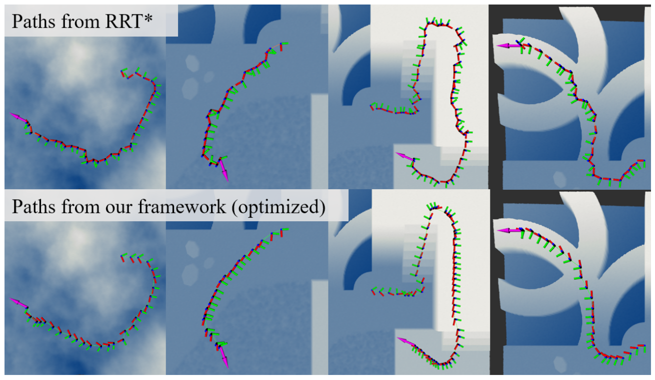
Table I shows the averaged path costs and planning times for the three planners on all terrains. It shows that not only does our approach produce much lower cost paths than traditional RRT*, it is also faster by multiple orders of magnitude, thereby enabling deployment in real-time on mobile robots.

TABLE I

AVERAGED PATH COSTS AND COMPUTATION TIMES.

Planner types	Rough terrain		Irregular steps		Stairs and slopes	
	Cost	Time[s]	Cost	Time[s]	Cost	Time[s]
RRT*	62.75	141.20	58.65	143.20	38.82	148.01
Raw	52.65	0.40	51.93	0.42	29.93	0.36
Optimized	**30.32**	0.52	**31.04**	0.55	**17.32**	0.49

We further visualize the planning results of RRT* and our *optimized* framework in *garden* and *mountain* environment in Fig. 6 (d)(e). The sampling-based planner in our framework ensures the same globally optimized results as the RRT*, which navigates between steep peaks and valleys on irregular topographies in *mountain* (Fig. 8 (a)), avoids rough terrain and steps, plans up and down stairs to the target pose, and walks along a narrow path in *garden* (Fig. 8 (b) to (d)). For task (b), although the learned locomotion policy can usually overcome the steps and the slightly rough region, the planners prefer walking on flat land instead of planning straight. Comparing the visualized results, our framework provides smoother paths than RRT*, which are more feasible for real-world deployment.



(a) (b) (c) (d)

Fig. 8. The RRT* planner and our framework perform navigation tasks in simulation based on the same local motion cost prediction. Both approaches provide globally optimized solutions when planning on unstructured terrain in *mountain* (a), avoiding rough terrain (b), navigating through stairs (c), and planning on a narrow path (d) in *garden*. Our approach with the gradient-based optimization provides smoother solutions than the pure sampling-based RRT* planner.

C. Results on ANYmal C

In the real-world experiments, we deploy the navigation framework on ANYmal C with an Nvidia Jetson AGX Xavier. The *feature extractor* generates global features from the elevation map within 0.05s. Next, the *cost predictor* takes 0.20s for checking 550k grid and vague samples to generate the roadmap, and A* takes 0.30s to plan a raw path. Finally, the raw path is optimized for 50 iterations in 1.00s. Altogether, the framework requires around 1.50s for each planning loop to find a solution.

The first experiment (Fig. 9) navigates ANYmal in a crowded corridor with a goal set 10m in front of the robot,

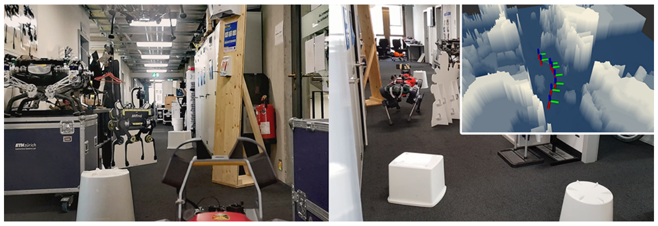


Fig. 9. Our framework navigates the robot to explore along a crowded corridor and move through obstacles based on the elevation map that is continuously constructed by the robot.

demonstrating the ability to plan in narrow environments. The robot was able to successfully reach the goal without collision or human intervention in multiple repeats.

In the second experiment, the robot is presented with a low-lying obstacle to demonstrate the locomotion capabilities awareness of our approach. A wooden block with a step height of 12cm is placed in front of the robot. The locomotion policy can overcome this obstacle but might be briefly tripped up, since it does not use exteroceptive sensing. When it can avoid the obstacle, the robot takes a detour while it walks across the obstacle if the detour is obstructed (Fig. 10).

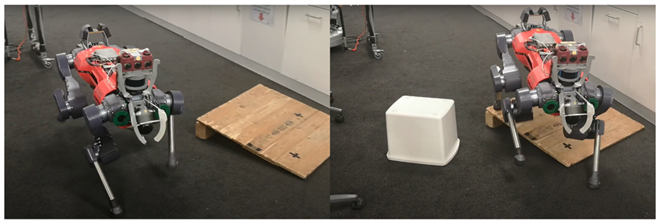


Fig. 10. The navigation framework prefers flat ground than the low-lying obstacle, while if other ways are blocked, it can also navigate the robot to walk across the obstacle.

VI. CONCLUSION

We present a controller-aware robot navigation approach for complex terrains which benefits from parallel computing on a GPU to achieve short planning times, making it applicable to real-time applications. The locomotion costs used for planning are learned by deploying the locomotion controller in simulation and then estimated by a neural network. We use predefined grid samples for fast roadmap construction on a GPU and introduce the *vague sampling* method to improve the probabilistic completion. By employing a gradient-based path optimizer we can rapidly obtain an optimized solution, which outperforms RRT* planning times by three orders of magnitude. The framework provides global path planning when provided a whole map, but also supports real-time autonomous exploration tasks in an unknown environment with continuously updated map information. Since the learning-based motion cost estimation is applicable to multiple types of robots [1] [3] and the training data are obtained from simulation, our planning framework can also be deployed on other robotic platforms with low manual effort.

References

[1] J. Guzzi, R. O. Chavez-Garcia, M. Nava, L. M. Gambardella, and A. Giusti, “Path planning with local motion estimations,” *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2586–2593, 2020.

[2] M. Wermelinger, P. Fankhauser, R. Diethelm, P. Krüsi, R. Siegwart, and M. Hutter, “Navigation planning for legged robots in challenging terrain,” in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1184–1189, 2016.

[3] R. O. Chavez-Garcia, J. Guzzi, L. M. Gambardella, and A. Giusti, “Learning ground traversability from simulations,” *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1695–1702, 2018.

[4] M. Hutter, C. Gehring, D. Jud, A. Lauber, C. D. Bellicoso, V. Tsounis, J. Hwangbo, K. Bodie, P. Fankhauser, M. Bloesch, R. Diethelm, S. Bachmann, A. Melzer, and M. Hoepflinger, “ANYmal - a highly mobile and dynamic quadrupedal robot,” in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 38–44, 2016.

[5] L. Nardi and C. Stachniss, “Actively improving robot navigation on different terrains using Gaussian process mixture models,” in *2019 International Conference on Robotics and Automation (ICRA)*, pp. 4104–4110, 2019.

[6] D. Silver, J. A. Bagnell, and A. Stentz, “Learning from demonstration for autonomous navigation in complex unstructured terrain,” *The International Journal of Robotics Research*, vol. 29, no. 12, pp. 1565–1592, 2010.

[7] M. Pfeiffer, M. Schaeuble, J. Nieto, R. Siegwart, and C. Cadena, “From perception to decision: A data-driven approach to end-to-end motion planning for autonomous ground robots,” in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1527–1533, 2017.

[8] L. Tai, G. Paolo, and M. Liu, “Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation,” in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 31–36, 2017.

[9] P. Krüsi, P. Furgale, M. Bosse, and R. Siegwart, “Driving on point clouds: Motion planning, trajectory optimization, and terrain assessment in generic nonplanar environments,” *Journal of Field Robotics*, vol. 34, no. 5, pp. 940–984, 2017.

[10] M. Liu, “Robotic online path planning on point cloud,” *IEEE Transactions on Cybernetics*, vol. 46, no. 5, pp. 1217–1228, 2016.

[11] A. Stelzer, H. Hirschmüller, and M. Görner, “Stereo-vision-based navigation of a six-legged walking robot in unknown rough terrain,” *The International Journal of Robotics Research*, vol. 31, no. 4, pp. 381–402, 2012.

[12] L. Wellhausen, A. Dosovitskiy, R. Ranftl, K. Walas, C. Cadena, and M. Hutter, “Where should I walk? predicting terrain properties from images via self-supervised learning,” *IEEE Robotics and Automation Letters*, vol. 4, no. 2, pp. 1509–1516, 2019.

[13] K. E. Bekris and R. Shome, “Asymptotically optimal sampling-based planners,” *arXiv preprint arXiv:1911.04044*, 2019.

[14] L. E. Kavraki, P. Svestka, J. . Latombe, and M. H. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[15] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[16] J. Hwangbo, J. Lee, and M. Hutter, “Per-contact iteration method for solving contact dynamics,” *IEEE Robotics and Automation Letters*, vol. 3, no. 2, pp. 895–902, 2018.

[17] J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter, “Learning quadrupedal locomotion over challenging terrain,” *Science Robotics*, vol. 5, no. 47, 2020.

[18] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.

[19] N. S. Keskar and R. Socher, “Improving generalization performance by switching from Adam to SGD,” *arXiv preprint arXiv:1712.07628*, 2017.

[20] R. Sun, “Optimization for deep learning: theory and algorithms,” *arXiv preprint arXiv:1912.08957*, 2019.

[21] P. Fankhauser, M. Bloesch, C. Gehring, M. Hutter, and R. Siegwart, “Robot-centric elevation mapping with uncertainty estimates,” in *International Conference on Climbing and Walking Robots (CLAWAR)*, 2014.

[22] P. Fankhauser, M. Bloesch, and M. Hutter, “Probabilistic terrain mapping for mobile robots with uncertain localization,” *IEEE Robotics and Automation Letters (RA-L)*, vol. 3, no. 4, pp. 3019–3026, 2018.