Robotics: Science and Systems 2022
New York City, NY, USA, June 27-July 1, 2022

Learning Forward Dynamics Model and Informed Trajectory Sampler for Safe Quadruped Navigation

Yunho Kim, Chanyoung Kim, Jemin Hwangbo
Korea Advanced Institute of Science and Technology (KAIST), Republic of Korea
{awesomericky, slowturtle99, jhwangbo}@kaist.ac.kr

***Abstract*—For autonomous quadruped robot navigation in various complex environments, a typical SOTA system is composed of four main modules – mapper, global planner, local planner, and command-tracking controller – in a hierarchical manner. In this paper, we build a robust and safe local planner which is designed to generate a velocity plan to track a coarsely planned path from the global planner. Previous works used waypoint-based methods (e.g. Proportional-Differential control and pure pursuit) which simplify the path tracking problem to local point-goal navigation. However, they suffer from frequent collisions in geometrically complex and narrow environments because of two reasons; the global planner uses a coarse and inaccurate model and the local planner is unable to track the global plan sufficiently well. Currently, deep learning methods are an appealing alternative because they can learn safety and path feasibility from experience more accurately. However, existing deep learning methods are not capable of planning for a long horizon. In this work, we propose a learning-based fully autonomous navigation framework composed of three innovative elements: a learned forward dynamics model (FDM), an online sampling-based model-predictive controller, and an informed trajectory sampler (ITS). Using our framework, a quadruped robot can autonomously navigate in various complex environments without a collision and generate a smoother command plan compared to the baseline method. Furthermore, our method can reactively handle unexpected obstacles on the planned path and avoid them. (Video[1])**

I. Introduction

Thanks to recent advances in legged robot control research [23, 1, 13, 51, 15], legged robots have become more robust and more agile in diverse environments. Notably, learning-based control approaches have shown great performance in complex outdoor environments. Unfortunately, those are largely blind: they only use proprioceptive sensors [33] or very local information of the terrain [38]. Furthermore, they can follow the given velocity commands but it is not trivial how we can generate adequate velocity commands in various environments. For autonomous navigation, the robot should be able to plan a safe and efficient plan.

We are interested in a *point-goal navigation* task, where the legged robot should autonomously navigate to the given goal position using real-time exteroceptive and proprioceptive sensor data. For robust autonomous navigation, many previous works [30, 6, 4, 48, 18, 11, 35, 50] proposed a hierarchical framework composed of four main modules: mapper, global planner, local planner, and command-tracking controller. The roles of each modules are as following:

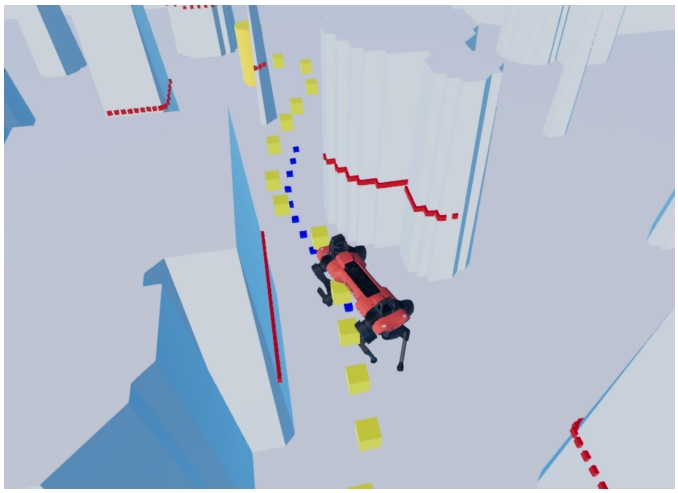


Fig. 1: Using our safe navigation framework, quadruped robot can safely navigate in complex environments. The yellow cylinder is the goal position. The yellow dotted line is the planned path by the global planner. The blue dotted line is the generated trajectory by the optimized command sequence.

- Mapper: Build a map of the environment using exteroceptive sensors such as a lidar and camera.
- Global planner: Find a rough path from the current location to the goal location using the map built by the mapper. A variety of planning algorithms such as search-based methods (e.g. A* [16]), sampling-based methods (e.g. RRT [32], PRM [31], RRT*, PRM* [28]), and heuristic methods (e.g. Potential Field [29]) can be used.
- Local planner: Generate a high-level command for the robot to track the planned path from the global planner. During the process, the roughly planned path is projected to a locally feasible trajectory that the robot can track. Various types of high-level commands can be planned by the local planner. In this work, we will focus on a local planner generating velocity commands.
- Command-tracking controller: Generate a joint-level command to track the given high-level command from the local planner. A learning-based [23] or model-based [1] controller can be used for this purpose.

All four modules should work in concert for safe and robust navigation. However, compared to recent advances

[1]Supplementary materials:
awesomericky.github.io/projects/FDM_ITS_navigation/

in mapping and global planning techniques [17, 7, 10, 9], there were relatively little developments in local planning. In geometrically complex environments, local planning becomes a challenging problem. In these environments, the local planner should be capable of generating long horizon plans that satisfies the kinematic and dynamics constraints of the robot and the performance characteristics of the command-tracking controller. Furthermore, it has to be reactive enough to handle unexpected obstacles that have not been accounted for during global planning.

In this light, we propose a learning-based, fully autonomous navigation framework composed of three innovative elements: a learned forward dynamics model (FDM), an online sampling-based model-predictive control module, and an informed trajectory sampler (ITS). We demonstrate how the dynamics of a quadruped robot and its surroundings can be captured accurately using deep neural networks (i.e. FDM), which is trained with a self-supervised learning framework, and used for predicting the future outcomes of a given command sequence. We then used FDM in conjunction with an online sampling-based model-predictive control module to track a roughly planned path from the global planner, as illustrated in Fig. 1. To handle the curse of dimensionality in sampling-based motion planning, we learned the implicit command sampling distribution (i.e. ITS) which generates samples close to optimal solutions and used it with the proposed online sampling-based model-predictive control module.

Because the learned FDM can be evaluated very fast in a GPU-enabled hardware, we can simulate 1,500 of 6-second trajectories in a 3 ms window, which is more than 20,000 times and 20 times faster than the simulation of the full and approximate robot model, respectively. An extensive evaluation in a physics simulator [22] shows that the proposed method for local planning enables safer autonomous navigation in diverse geometrically complex environments compared to widely used baseline methods. Furthermore, the proposed method can reactively handle unexpected obstacles on the planned path and be easily modified for both fully-autonomous and semi-autonomous tasks.

II. Related work

To generate a velocity command to track a path from the global planner, many previous works [6, 4, 48, 50] used waypoint-based methods which simplify the path tracking problem into local point-goal navigation. These methods repeat (1) determining a waypoint on the planned path that is within an adequate distance range and (2) generating a command to move toward the determined waypoint. Commands are generated using a PD (i.e. Proportional-Differential) controller based on the position and orientation error between the current and desired configuration in the waypoint. However, these methods suffer from frequent collisions in complex and narrow environments. Enlarging the approximated robot collision body can be a solution for the global planner to find a more conservative path, but it results in a compromising path or failure to find one. Post-processing the planned path heuristically [50] for smoothness still cannot be a complete solution to guarantee safety in local planning. Some works used reactive methods [37, 30, 18] with vector field representation for local obstacle avoidance, but they result in jerky and discrete command changes due to the lack of long horizon planning. Gilroy et al. [11] and Li et al. [35] used collocation-based trajectory optimization to handle long horizon planning, but the decoupled nature between the local planner and the command-tracking controller can still cause a collision in complex environments because the local planner cannot take into account the performance characteristics of the command-tracking controller. Furthermore, these methods cannot handle unexpected obstacles on the globally planned path.

Learning-based methods are an appealing alternative because they can accurately learn the safety and path feasibility from experience. Previous works [46, 47, 40, 20] used imitation learning and model-free deep reinforcement learning to learn a collision-free control policy for a wheeled mobile robot and a quadruped robot. However, the results were a reactive single-step planning policy and also cannot be integrated with the existing hierarchical navigation framework. When deploying the resulting policy for a *point-goal navigation* task, the robot will easily get stuck in a local optimum due to the lack of a long horizon planning module.

There were recent advances in the global planner using learning-based methods to find the global path faster [2, 41, 42, 24, 53] or to find a controller-aware path [14, 52]. However, these methods still require a robust local planner to safely track the planned path.

Our work is closely related to the one presented by Kahn et al. [27]. They presented a forward dynamics model, composed of deep neural networks, that predicts the path that the robot will take, the corresponding collision probabilities, and the terrain properties. Our work differs from theirs in three ways. First, their focus was on identifying the traversability of the terrain from an RGB image in open field environments and use this information in finding a command trajectory that leads the robot toward the goal. Because the environments they experimented had less obstacles, the robot can safely navigate to the goal without a globally planned path. However, in geometrically complex environments that we are interested in, the robot will easily get stuck in a local optimum without using the global planner. Thus, we focused on finding a command trajectory to track the planned path from the global planner in environments with densely placed obstacles, requiring a more reactive controller and a different problem formulation. Second, we improved the sampling-based motion planner used by Kahn et al. [27] by learning the implicit command sampling distribution (i.e. ITS) using Conditional Variational Inference [43], which results in a safer motion plan. Lastly, we tested our controller with a legged robot, which manifests more complicated dynamics and kinematics compared to wheeled mobile robots, and a 360-degree lidar sensor readings for safe navigation in all directions, rather than front view camera images.

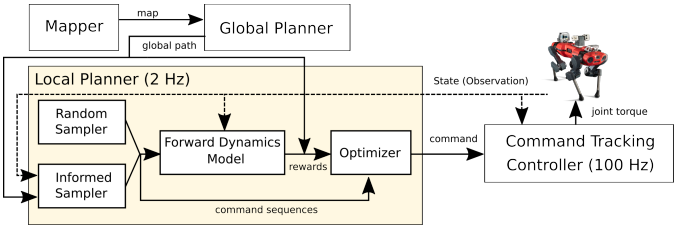


Fig. 2: Block diagram of the proposed safe navigation framework.

Environment type	Parameter	Sampling distribution
Open-fields	cylinder radius	$U(0.05, 1.0)$ [m]
	box side	$U(0.1, 2.0)$ [m]
	grid size	$U(2.3, 5.0)$ [m]
	center randomness	$U(0.1, 0.9)$ [m]
Cross-corridors	cylinder radius	$U(0.05, 1.0)$ [m]
	box side	$U(0.1, 2.0)$ [m]
	grid size	$U(2.3, 5.0)$ [m]
	center randomness	$U(0.1, 0.9)$ [m]
	corridor width	$U(2.0, 6.0)$ [m]
	corridor length	$U(8.0, 30.0)$ [m]

TABLE I: Parameters for the random environment generation

III. Method

Our goal is to improve the existing hierarchical navigation system by building a more robust and safe local planner that can generate a sequence of velocity commands to track the coarsely planned path from the global planner in a geometrically complex environment with flat terrain and many static obstacles. Thus, the proposed safe navigation framework is constructed similarly to the existing navigation system with four main modules: mapper, global planner, local planner, and command-tracking controller (Fig. 2). Although our framework can be easily combined with a mapper for navigation in unknown environments similar to previous works [35, 11], we assume that the map is given ahead of time, because it is not the focus of this paper. For global planning, we used an open-source implementation [44] of BIT* [10] because of its speed and robustness. However, many other path planning algorithms [16, 28, 9] can be used with our framework as well. The global planner finds a rough path from the start location to the goal location and returns a list of x, y coordinates, represented in the world frame, to the local planner. The global planner does not consider the orientation of the robot due to the computation complexity in real-time usage and uses a minimal bounding sphere to simplify the robot's collision bodies [48].

The local planner is composed of three core elements: a learned forward dynamics model (FDM), an online sampling-based model-predictive control module, and an informed trajectory sampler (ITS) (Fig. 2). In the rest of the subsections, we will introduce each element and summarize the overall working pipeline. For the command-tracking controller, we use a controller proposed in our previous work, which is acquired using model-free deep reinforcement learning [23].

We use a simulated ANYmal C robot [21] for both training and testing of our proposed framework in the RaiSim simulator [22]. For perception, we use a 2D line-scan lidar sensor attached to the back of the robot. Lidar readings are modeled with a Gaussian noise $N(0, 0.2)$ [m] and normalized with the maximum available sensing range of 10 m.

A. Forward dynamics model

To plan for a long horizon, we learn the dynamics of the environment rather than a task-specific control policy, inspired by numerous previous works [8, 39, 27, 26]. Our learned FDM works as a fast virtual simulator for a quadruped robot and predicts the future base coordinates and the probability of collision on the trajectory. To be specific, it takes as input the current lidar sensor data, history of selected generalized coordinates and velocities, and a sequence of future intended commands, and predicts the future x, y coordinates of the robot with respect to the robot's current body frame and the probability of collision. The history of selected generalized coordinates and velocities correspond to the history of base orientation, base linear velocity, and base angular velocity (10 steps of history, each step corresponding to 0.05 second). The history term was included due to the dynamic movements of the origin of the lidar frame. We denote this model as $f_\theta(\mathbf{o}_t, \mathbf{c}_{t:t+H}) = \{\hat{\mathbf{x}}_{t+1:t+H+1}, \hat{\mathbf{y}}_{t+1:t+H+1}, \hat{\mathbf{p}}_{t+1:t+H+1}\}$, which is parameterized by the vector θ, that takes as input the current observation $\mathbf{o}_t$ and a sequence of H future commands $\mathbf{c}_{t:t+H} = \{\mathbf{c}_t, \mathbf{c}_{t+1}, ..., \mathbf{c}_{t+H-1}\}$. The model predicts the x, y coordinates $\hat{\mathbf{x}}$, $\hat{\mathbf{y}}$ and probability of collision $\hat{\mathbf{p}}$ for H future time steps.

FDM is a deep neural network composed of fully connected layers and subsequent recurrent layers to predict the future positions and collision probabilities conditioned on the current observations and a sequence of commands. The current observations, which include both proprioceptive and exteroceptive sensor information, are encoded to latent features using fully connected layers. The final output of these layers serves as a hidden state initialization for a recurrent neural network, which sequentially processes each of the H future commands $\mathbf{c}_{t:t+H}$ and outputs the future navigational outcomes. We use the LSTM (i.e. Long Short-Term Memory) cells for the recurrent layers [19].

To let FDM learn the dynamics of the environment, we focus on generating diverse environments to capture both a broad lidar data distribution and a future outcome distribution. We train FDM on two types of parameterized environments: open-fields with densely placed cylinders/boxes, and cross-shaped corridors with densely placed cylinders/boxes. Each of these environments is randomly generated by sampling the corresponding parameters from the predefined range given in Table I. The map is divided into equal sizes of grids, and each grid contains one obstacle. The position of the obstacle inside the grid is sampled from $U(\textit{center randomness},\ \textit{grid size} - \textit{center randomness})$, where *center randomness* determines the overall randomness of the obstacle distribution. Using our parameterization, we can generate various environments with both convex and concave obstacles, due to the occlusion between cylinders/boxes, and let FDM learn the general ability to detect collisions.

To take advantage of the fast parallel data generation process

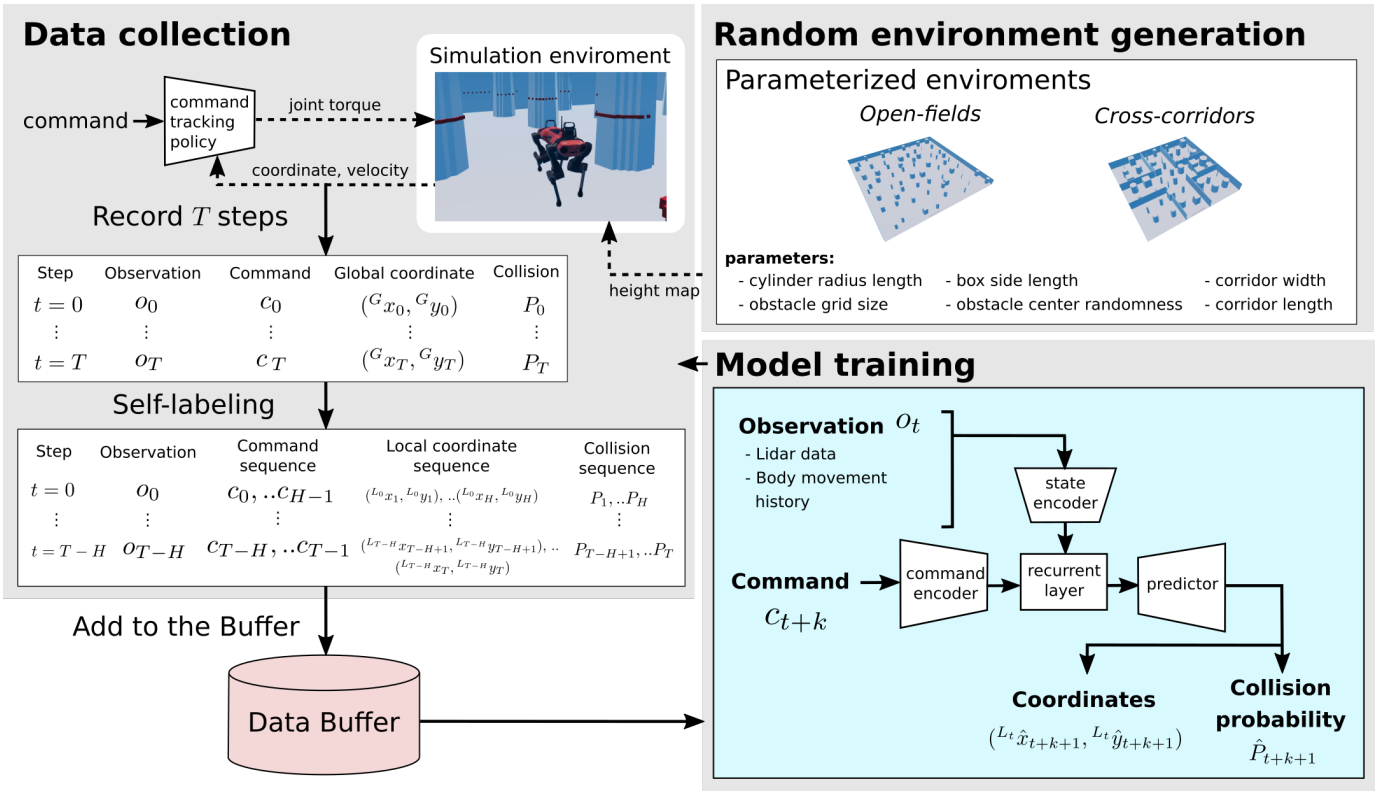


Fig. 3: Overall framework to train FDM. The neural network architecture of FDM was inspired by the ones used by Kahn et al. [27].

of the simulator, we alternate between a data collection and model training step. In the data collection step, we randomly generate N_{env} environments with different types of obstacles in the same proportion. During the data collection period, a quadruped robot is randomly placed in each environments and given velocity command sequences sampled from {U(-1.0, 1.0), U(-0.4, 0.4), U(-1.2, 1.2)}, each elements corresponds to forward velocity [m/s], lateral velocity [m/s], and turning rate [rad/s]. The specific command sequence sampling method used for training FDM is described in APPENDIX-A.

We use a self-supervised data labeling technique, similar to [27], to process the data for training without human supervision. Simple coordinate transformation is used to compute x, y coordinates in the robot's current body frame. For the probability of collision, binary collision state, computed by solving the contact dynamic in the physics simulation, is used [22]. Data tuples $\{\mathbf{o}_t, \mathbf{c}_{t:t+H}, \mathbf{x}_{t+1:t+H+1}, \mathbf{y}_{t+1:t+H+1}, \mathbf{p}_{t+1:t+H+1}\}$ are recorded to the data buffer and used later for training.

The model is trained to minimize a loss function that penalizes the distance between the predicted and actual outcomes. The mean squared error and cross-entropy loss are used for the x, y coordinates and probability of collision, respectively. The overall training framework is summarized in Fig. 3. Hyperparameters used for the data collection and training are shown in the APPENDIX-B.

We train FDM to predict 12 steps of future navigational outcomes, each step corresponding to 0.5 s period which results in 6 s of the maximum prediction horizon, and use it for the experiments done in section IV.

B. Online sampling-based model-predictive control

In this section, we present an online sampling-based model predictive control algorithm using learned FDM, which works as a fast virtual simulator with collision checking, for safe quadruped navigation. We define a reward function $R(f_\theta(\mathbf{o}_t, \mathbf{c}_{t:t+H}))$ in terms of the future navigational outcomes predicted by FDM, conditioned on the given task of the robot. Using this reward function and FDM, we repeatedly solve the following optimization problem,

$$\mathbf{c}^*_{t:t+H} = \underset{\mathbf{c}_{t:t+H} \in \mathscr{C}}{\arg\max}\, R(f_\theta(\mathbf{o}_t, \mathbf{c}_{t:t+H})), \tag{1}$$

and execute the first step of the resulting command sequence for a single command period, where $\mathscr{C}$ is the available command range.

To solve Eqn. 1, we use a gradient-free optimizer similar to the ones used by Nagabandi et al. [39] and Kahn et al. [27]. At every time step, N commands $c^{0:N}_{t:t+H}$ are generated by computing the weighted average of time-correlated random command sequences $\tilde{c}^{0:N}_{t:t+H}$ and previously optimized results $\hat{\mathbf{c}}_{t:t+H}$ as

$$\begin{aligned} c^n_{t:t+H} &= (1-\beta)\cdot\tilde{c}^n_{t:t+H} + \beta\cdot\hat{\mathbf{c}}_{t:t+H} \\ \text{s.t.}\quad \tilde{c}^n_{t+k+1} &\sim N(\tilde{c}^n_{t+k}, \sigma) \\ \tilde{c}^n_t &\sim Bin(c_{min}, c_{max}) \\ \forall\, k \in \{0..,H-2\},\ &\forall\, n \in \{0..,N-1\} \end{aligned} \tag{2}$$

where *Bin* is a bin sampling method, which divides the command range into N_b equally spaced bins and samples uniformly from each of them.

Each command sequence is then fed to FDM to compute the corresponding reward $R^n = R(f_\theta(\mathbf{o}_t, c^n_{t:t+H}))$. Using the sampled command sequences and computed rewards, we update the optimized command sequence via reward-weighted average as

$$\hat{\mathbf{c}}_{t:t+H} = \frac{\sum_{n=0}^{N-1} \exp(\gamma \cdot R^n) \cdot c^n_{t:t+H}}{\sum_{n'=0}^{N-1} \exp(\gamma \cdot R^{n'})}, \tag{3}$$

similar to the equation by recent model-predictive path integral work [49, 36].

We then execute the first step of the planned command sequence for a single command period and repeat the process. Parameters $\beta \in [0,1]$ and $\gamma \in \mathrm{R}^+$ of the optimizer work as a time correlation factor and a high-reward weighting factor, respectively. The optimizer we used shows stable performance in complex environments due to the ability to sample a dense set of trajectories. The performance of our algorithm will be further analyzed in section IV. Our overall algorithm is summarized as follows (Alg. 1).

Algorithm 1 Online sampling-based model-predictive control with learned FDM

1: **input**: learned FDM f_θ, reward function R
2: **while** task is not complete **do**
3: get current observation $\mathbf{o}_t$ from sensors
4: solve Eqn. 1 using Eqn. 2 and Eqn. 3
5: execute $\hat{\mathbf{c}}_t$ in optimized command sequence $\hat{\mathbf{c}}_{t:t+H}$

C. Implementation for path tracking

For the robot to safely track the planned path, we use the sum of two rewards, R_{track} and R_{safety}, defined as

$$\begin{aligned} R_{total} &= R_{track} + R_{safety} \\ s.t.\ R_{track} &= exp(\frac{-DTW(\mathbf{g}_t, \mathbf{q}_{t+1:t+H+1})}{\tau}) \\ R_{safety} &= \frac{\sum_{k=1}^{H}(1-\hat{\mathbf{p}}_{t+k})}{H} \end{aligned} \tag{4}$$

where $\mathbf{g}_t = {}^{L_t}\mathbf{g}$ ($\mathbf{g} \subset \mathbf{G}$, $\mathbf{G}$: Global path), $\mathbf{q}_{t+k} = \{{}^{L_t}\hat{\mathbf{x}}_{t+k}, {}^{L_t}\hat{\mathbf{y}}_{t+k}\}$, and τ is a temperature constant for normalization.

R_{track} is to incentivize command sequences that result in a similar path with the currently considered global path. Thus, we used the output of normalized Dynamic Time Warping (DTW) for R_{track}. DTW is a similarity function between two series of data by finding their optimal alignment which minimizes the cumulative distance between aligned elements. Because DTW does not require two series of data to have the same length or constant difference in successive data, it has recently been used as an evaluation metric for navigation tasks [25]. In this work, we applied it in continuous planning for path tracking.

To compute R_{track}, we truncate the globally planned path that is 4.8m ahead from the current location, resulting in 0.8 m/s of desired average speed, and transform the coordinates of the considered path into the robot's local frame. DTW between the considered path in the robot's local frame and the predicted path from FDM is computed and normalized. Open-source implementation [12] is used for computing DTW. The truncated global path considered in the current step will be referred to as the waypoint trajectory hereinafter.

To generate safer navigation plan, the collision probability outputs from FDM is used for both hard and soft constraints. For hard constraint, we filter out the sampled command trajectories that collide with obstacles within a fixed period (3s). For soft constraint, we compute R_{safety} and include it in the total reward formulation to give additional weights to the trajectories that show a low probability of collision.

The planning module runs at 2Hz. In each planning period, we plan 12 steps of command trajectory to track the global path, each step corresponding to 0.5s period which results in 6s of the maximum planning horizon.

D. Informed trajectory sampler

Although the robot showed overall high performance by just sampling random time-correlated command sequences as Eqn. 2, it is still not free from the curse of dimensionality in sampling-based motion planning and thus often fails in complex environments. Considering that the command dimension is three, the number of planning steps is H, and the number of bins used for bin sampling is N_b, we need at least ${N_b}^{3H}$ number of samples to guarantee a near-optimal solution in any cases. However, as we are building methods that are capable of long-distance planning (i.e. large H) it is impossible to consider ${N_b}^{3H}$ samples online.

Therefore, we additionally use the informed trajectory sampler (ITS) that can generate command sequences close to the optimal solution.

ITS is a deep neural network modeled with Conditional Variational AutoEncoder (CVAE) [43]. CVAE is a conditional generative model that uses conditional variational inference to handle the intractable posterior distribution. The model is trained by optimizing the Evidence Lower BOund (ELBO) as shown in

$$\begin{aligned} & logp_\theta(x|y) \\ &= \mathbb{E}_{z\sim q_\phi(z|x,y)}[logp_\theta(x|z,y)] - D_{KL}(q_\phi(z|x,y)||p_\theta(z|y)) \\ &\quad + D_{KL}(q_\phi(z|x,y)||p_\theta(z|x,y)) \\ &\geq \mathbb{E}_{z\sim q_\phi(z|x,y)}[logp_\theta(x|z,y)] - D_{KL}(q_\phi(z|x,y)||p_\theta(z|y)) \\ &= \underbrace{\mathbb{E}_{z\sim q_\phi(z|x,y)}[logp_\theta(x|z,y)] - D_{KL}(q_\phi(z|x,y)||p_\theta(z))}_{ELBO}. \\ &\quad (\because \textit{Assume } y \textit{ and } z \textit{ independent}) \\ &\quad (D_{KL} \textit{ corresponds to Kullback–Leibler divergence}) \end{aligned} \tag{5}$$

In our case, x corresponds to a command trajectory and y corresponds to an observation and a global path considered in the current step (i.e. waypoint trajectory). Observation includes the current lidar sensor data and history of selected generalized

coordinates and velocities, same as FDM. Waypoint trajectory is represented as a list of x, y coordinates included in the globally planned path and transformed into the robot's local frame.

Because sampling-based path planning algorithms, such as BIT*, output list of nodes that show irregular distances between them, waypoint trajectory sometimes include very few nodes. In this case, we interpolate it to guarantee a minimum number of nodes.

ITS is composed of fully connected layers and recurrent layers. We use the GRU (i.e. Gated Recurrent Unit) cells for the recurrent layers [5] and sequence-to-sequence model to encode and decode command trajectory and waypoint trajectory [45]. 160K number of training data is collected by rolling out the robot, controlled with Alg. 1, in randomly generated environments (Fig. 3) and goal positions. For the learned sampler to generate locally diverse samples, we use a variant of the CVAE objective function proposed by Bhattacharyya et al. [3]. Hyperparameters used for training ITS are shown in the APPENDIX-B.

Learned ITS is then used with the time-correlated random sampler to generate command sequences for the online sampling-based model-predictive control module. The detailed model architecture of ITS is shown in Fig. 4.

E. Summary

We now provide a summary of our safe navigation framework (Fig. 2). *N* command sequences are first sampled from both a time-correlated random sampler and ITS. For each sample, the future navigation outcomes and rewards are then predicted with FDM and Eqn. 4. Based on Eqn. 3, the optimum command sequence is predicted and the first step command of it is executed by passing it to the command tracking controller, which outputs the target joint torque. This process repeats until the task is completed.

IV. Experiment

In our experiments, we study how the proposed learning-based safe navigation framework enables a quadruped robot to autonomously navigate in various complex environments. Hyperparameters of the sampling-based model-predictive controller were set constant for all the experiments (APPENDIX-B).

In the experiments, we used AMD Ryzen9 5950X and NVIDIA GeForce RTX 3070 for computation. FDM and ITS, which are composed of neural networks, inference were done on GPU (i.e. Graphics Processing Unit) and other computations such as sampling-based optimization were done on CPU (i.e. Central Processing Unit).

A. FDM evaluation

We first evaluated the performance of FDM in terms of prediction accuracy and computation speed. As FDM works as a fast virtual simulator to predict the future navigation outcomes, the performance was compared with two baseline models that use a real-time collision checking algorithm in the physics simulator [22].

- **Complete**: A method that runs the forward simulation of a quadruped robot with complete kinematic and dynamic models. Velocity tracking controller [23] was used to map the given velocity command to a joint-level command.
- **Approximate**: A method that runs the forward simulation of a quadruped robot with approximated kinematic models. For fast collision checking, the robot was approximated to a box with a size that covers the entire body of it in the nominal configuration. Future coordinates were computed analytically by assuming that the robot follows the velocity command perfectly.

For the baseline methods, we assumed the terrain model of the environment is known apriori for the physics simulator to run a collision checking algorithm.

The accuracy of FDM was evaluated on 3.5M samples collected in randomly generated environments, which were not seen during training. The generated environments include both open-fields with densely placed cylinders/boxes, and cross-shaped corridors with densely placed cylinders/boxes (Fig. 3, Table I). A probability threshold of 0.3 was used for deciding collision.

FDM showed high collision checking accuracy (94.6%) and low coordinate prediction error per step (0.1m) (Fig 5.A). Furthermore, we could simulate 1,500 of 6-second trajectories in about 3 ms window with only using real-time exteroceptive sensor data, which is more than 20,000 times and 20 times faster than *Complete* and *Approximate* that use accurately constructed environment models (Fig 5.C), respectively. *Complete* especially entailed very high computation costs. Thus, it was not suitable for applications that require lots of command samples to be simulated in real-time, such as our proposed algorithm. *Approximate*, which simplifies the kinematic and dynamic model, could be a solution to alleviate the computation cost. However, such a method showed a large coordinate prediction error as the horizon became longer due to error accumulation (Fig 5.B). Our ablation study between FDM and *Approximate* will further explain the importance of prediction accuracy of forward simulation for the overall navigation performance.

In the following experiments, we will show how the effect of slight errors in FDM can be mitigated by its low computational cost and other elements in the proposed framework for safe navigation. When using FDM for safe navigation, we added a padding step. If FDM predicted the first collision in h step, the subsequent predicted outputs from h step were set constant as $\mathbf{x}_{t+h} = \mathbf{x}_{t+h+1} .. = \mathbf{x}_{t+H}$, $\mathbf{y}_{t+h} = \mathbf{y}_{t+h+1} .. = \mathbf{y}_{t+H}$, and $\mathbf{p}_{t+h} = \mathbf{p}_{t+h+1} .. = \mathbf{p}_{t+H}$ $(1 \leq h \leq H)$.

B. Point-Goal Navigation

In *point-goal navigation* task, the robot should safely track the path, planned from the global planner, to navigate from the start location to the goal location. We compared the proposed method with a waypoint-based method that generates command using a PD controller, which will be referred to as

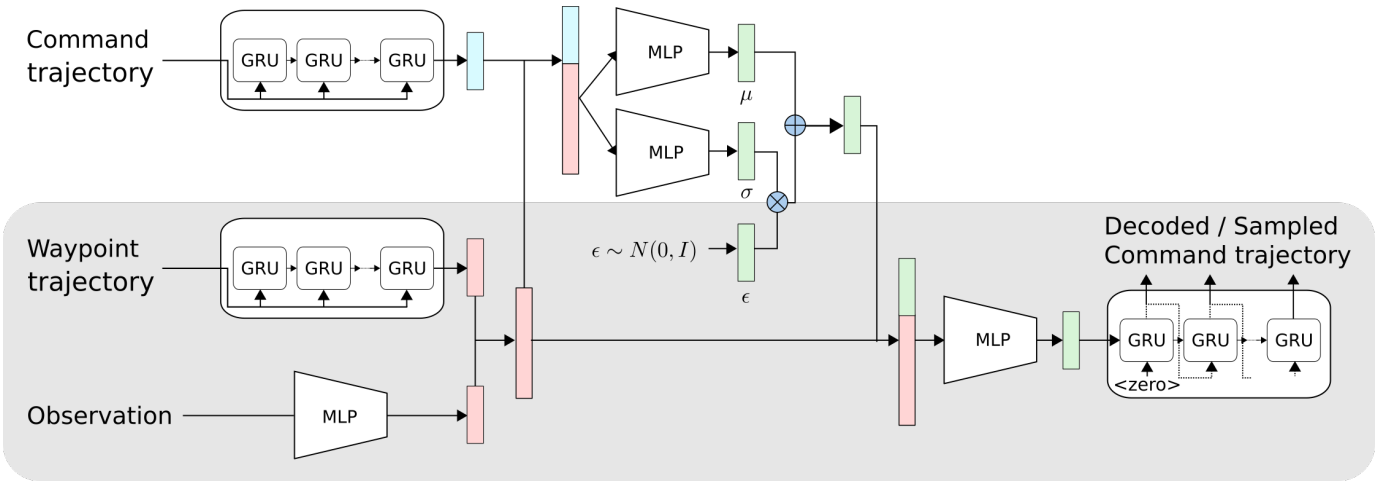
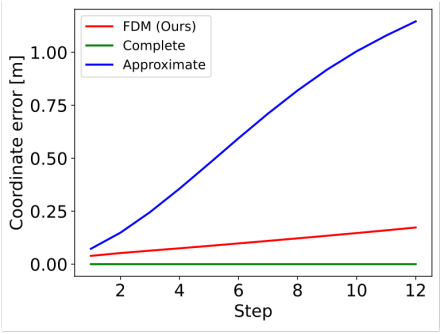


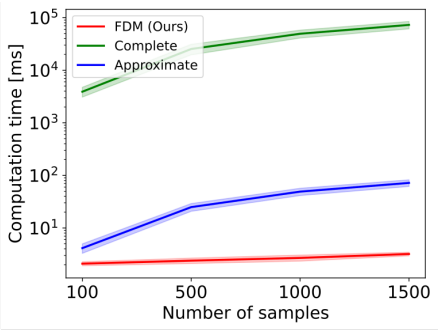
Fig. 4: Model architecture to train ITS. Gray scaled part is ITS used for generating command trajectory samples in real-time navigation. The neural network architecture of ITS was inspired by the ones used by Ichter et al. [24] and Lee et al. [34]. MLP in the figure indicates fully connected layers.

Total collision accuracy [%]	**94.6**
Collision accuracy [%]	94.1
No collision accuracy [%]	94.8
Coordinate error per step [m]	**0.1**

(A)



(B)



(C)

Fig. 5: FDM evaluation results. (A) shows the overall prediction accuracy of FDM. (B) and (C) show the comparison between the baseline models in terms of coordinate prediction error and computation time.

PD hereinafter. The detailed mechanism is explained in section II. The desired orientation, used in PD, was computed for the robot to be aligned with the planned path.

Three evaluation metrics were used to compare the performance: Success Rate (SR), Traversal Time, and Dynamic Time Warping (DTW). Success here means that the robot reached the location within 0.6 m from the goal without any collision with obstacles. DTW was computed between the globally planned path and the robot COM's traversal path. We use DTW per step to make the metric invariant to the path length.

We evaluated the performance in 60 different randomly generated open field environments with densely placed obstacles and 8 goal positions. As the proposed method relies on a random command sampler, we repeated the experiments with three different random seeds and the mean values of the evaluation metrics are reported in Table II. The standard deviations were small enough to be ignored.

Our method showed a higher success rate compared to PD with similar traversal time and DTW. PD suffered from collisions in complex and narrow environments (i.e. environments with high obstacle density) due to unaccounted safety during the local planning. Performance of ITS will be further analyzed in section IV-D.

Because the proposed model-predictive control module iteratively replans a long horizon command trajectory to improve both path similarity and safety, our local planner resulted in a smoother command trajectory on the robot than PD (Fig. 6.A-C). Furthermore, it could handle unexpected obstacles on the globally planned path and avoid them without a new plan from the global planner (Fig. 6.D).

To check the generalizability and robustness of our method, we generated two different environments with various geometric complexity as shown in Fig. 7. Visualized traversal paths of the robot in each environment indicate that our method could safely navigate over a long distance.

C. *Semi-autonomous task*

FDM and model-predictive control module with a random sampler are not limited to fully-autonomous tasks like *point-goal navigation* and can also be applied to semi-autonomous tasks. We evaluated our proposed method on a semi-autonomous task named *safety remote control*. In *safety*

Obstacle density [1/m]	0.43			0.33			0.25			0.2		
	SR	Time	DTW	SR	Time	DTW	SR	Time	DTW	SR	Time	DTW
Ours	**83.2**	48.0	0.43	**95.9**	33.5	0.34	**96.9**	31.6	0.34	98.2	30.9	0.34
PD	45.2	32.9	0.30	84.5	32.7	0.28	91.7	32.5	0.27	94.6	32.4	0.26
Approximate *(ABL)*	76.5	43.8	0.44	93.4	33.2	0.36	95.9	31.9	0.35	98.3	31.4	0.35
only Random *(ABL)*	73.6	51.7	0.45	89.5	34.2	0.35	96.6	31.8	0.34	**98.6**	31.0	0.34
only ITS *(ABL)*	63.2	39.3	0.38	90.3	33.0	0.35	91.5	31.6	0.36	91.9	31.1	0.37

TABLE II: *Point-Goal Navigation* results in open fields with densely placed obstacles. Success rate (SR) [%], traversal time [s], and DTW [m] are reported. The results are the mean values after running the evaluation with three different random seeds. Obstacle density represents the number of obstacles per meter and is computed as $\frac{1}{obstacle\ grid\ size}$. Candidates with a *ABL* flag were evaluated for the ablation study.

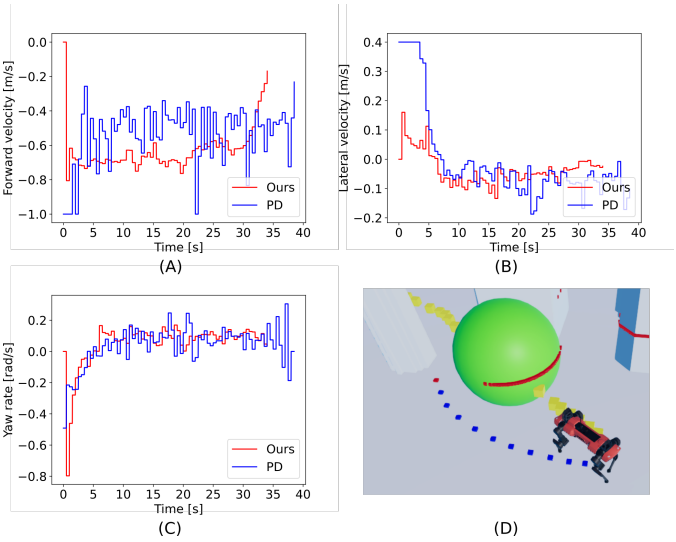


Fig. 6: (A-C) shows the velocity command trajectory executed on the robot to reach the goal. (A), (B), (C) each corresponds to forward velocity, lateral velocity, yaw rate command. (D) shows the reactiveness of our method to avoid unexpected obstacles (green) located on the globally planned path.

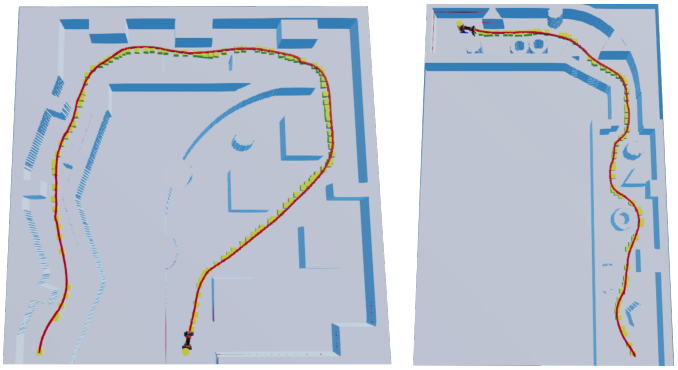


Fig. 7: Map and the robot's traversal path in novel environments

remote control task, the robot should decide whether the given remote command from the user is safe and project the command onto the safe command set if it is not. (Fig. 8).

We used R_{safety} and the proposed model-predictive control algorithm for the task with an additional simple logic. If FDM predicted the given user command to collide with obstacles within a fixed period (3s), it executes the optimized command.

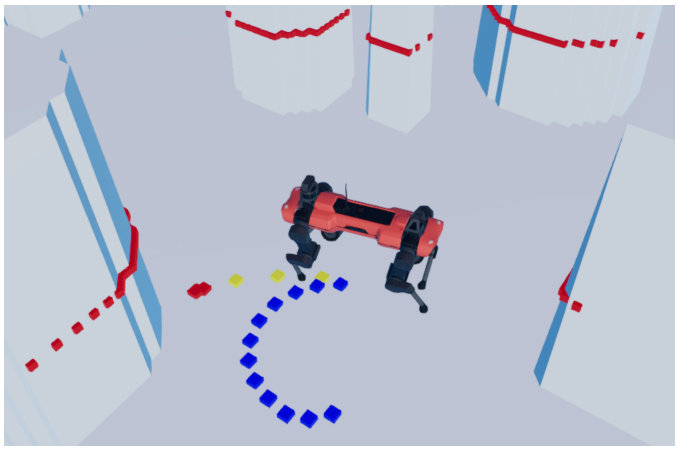


Fig. 8: Visualization of *Safety Remote Control* task. The blue dotted line is the generated trajectory by the optimized command. The yellow and red dotted line is a generated trajectory by the given command. The red dots indicate that FDM predicts collision along the trajectory.

Obstacle density [1/m]	0.4	0.33	0.25	0.2
Collision safe [%]	81.0	85.8	88.4	88.4
No collision safe [%]	98.2	98.8	99.8	99.8

TABLE III: *Safety Remote Control* result in open fields with densely placed obstacles. Success rate (SR) [%] is reported. Obstacle density represents the number of obstacles per meter and is computed as $\frac{1}{obstacle\ grid\ size}$. "Collision safe" and "No collision safe" each represent the success case when directly executing the given command results in a collision or not. To be specific, "Collision safe" corresponds to "(number of collision-free trajectories among the trajectories when directly executing the given command results in a collision) / (number of trajectories when directly executing the given command results in a collision)". "No collision safe" corresponds to "(number of collision-free trajectories among the trajectories when directly executing the given command does not result in a collision) / (number of trajectories when directly executing the given command does not result in a collision)"

On the other hand, if the given command was predicted as safe, it executes the given command. We sampled the commands for optimization in a multivariate normal distribution with given user command as mean values to search solutions close to the user's intended command.

We evaluated the performance in open field environments

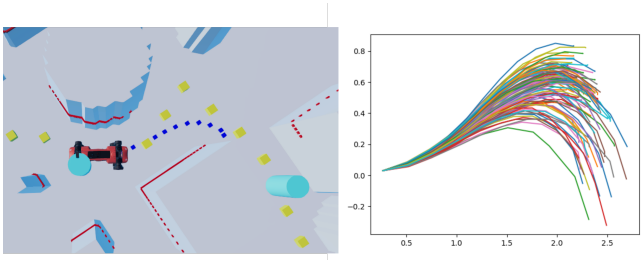


Fig. 9: Visualization of trajectories sampled using ITS (Right) and the output trajectory from the optimizer using these samples (Left). The robot is placed at the origin. The yellow dotted line is the global path to track. The yellow dotted line between two light-blue cylinders is the waypoint trajectory considered when generating the visualized samples. As ITS samples command sequences, outputs of learned FDM is used to visualize the trajectories.

with densely placed obstacles. In each environment, 300 randomly sampled commands were given to the robot and recorded as a success if the robot did not make a contact with the environment other than on the feet, which is the same rule applied for *point-goal navigation*. Results shown in Table III indicate that our method with FDM can predict collision and project the command onto the safe command set, if needed, with a high accuracy. Furthermore, it shows that our proposed method can be used for both fully-autonomous and semi-autonomous tasks by just changing the command sampling distribution and reward functions.

D. Ablation study

In the ablation study, we aim at showing the importance of the following contributions for the overall performance in safe navigation:

- Learned FDM that outputs the future coordinates and the collision probabilities
- Usage of both random sampler and learned ITS for command sampling

For evaluation, we used the same evaluation metrics and procedure explained in section IV-B.

First, we used *Approximate* described in section IV-A, instead of FDM, for the proposed navigation framework and evaluated the performance (Table II). *Complete* was not used because of the heavy computation for real-time usage. Ours using FDM showed higher success rate than ones with *Approximate*. This was due to the large coordinate prediction error shown in *Approximate* (Fig 5.B). It caused the predicted trajectory to deviate a lot from the actual trajectory and thus returned an inaccurate reward signal for the optimization. Furthermore, compared to the physics simulator that outputs discrete collision state, FDM outputs continuous collision probability which enabled distinct safety reward signal for each sample.

To check the importance of the command sampler, ours were compared with two methods, each using only the random sampler or ITS. Results in Table II indicate the complementary relationship between random sampler and ITS for safe navigation. The random sampler generates diverse command sequences and enables the optimizer to find an approximate solution. However, it fails to find the precise solution which is critical in complex and narrow environments. On the other hand, ITS can guide the optimizer to find a more precise solution by generating biased command sequences (Fig. 9). However, it fails to generate diverse samples, compared to the random sampler, to handle a local optimum and thus showed relatively low performance when used alone.

V. Conclusion

We proposed a learning-based fully autonomous navigation framework composed of three innovative elements: a learned forward dynamics model (FDM), an online sampling-based model-predictive control module, and an informed trajectory sampler (ITS). We demonstrated how the dynamics of a quadruped robot and its surroundings can be learned accurately using deep neural networks (FDM) and be used for various downstream tasks, including safe *point-goal navigation*, with the sampling-based model-predictive control module. To handle the curse of dimensionality in sampling-based motion planning, we further suggested the informed sampler, composed of deep neural networks, and its training method. Extensive evaluation in the physics simulator [22] showed the superiority of the performance of the proposed method for autonomous navigation in complex environments compared to the widely used baseline method. The ablation studies further demonstrated how the elements of our framework worked in a complementary manner resulting in a high-performance navigation system.

Promising directions for future work include expanding our method to different robot platforms (e.g. drone, wheeled robot, biped robot) using different exteroceptive sensors (e.g. RGB/D camera, 3D lidar sensor) and transferring it to the real world.

Acknowledgment

This work was supported by Samsung Research Funding & Incubation Center of Samsung Electronics under Project Number SRFC-IT2002-02.

References

[1] C Dario Bellicoso, Fabian Jenelten, Christian Gehring, and Marco Hutter. Dynamic locomotion through online nonlinear motion optimization for quadrupedal robots. *IEEE Robotics and Automation Letters*, 3(3):2261–2268, 2018.

[2] Mohak Bhardwaj, Sanjiban Choudhury, and Sebastian Scherer. Learning heuristic search via imitation. In *Conference on Robot Learning*, pages 271–280. PMLR, 2017.

[3] Apratim Bhattacharyya, Bernt Schiele, and Mario Fritz. Accurate and diverse sampling of sequences based on a "best of many" sample objective. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8485–8493, 2018.

[4] Annett Chilian and Heiko Hirschmüller. Stereo camera based navigation of mobile robots on rough terrain. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4571–4576. IEEE, 2009.

[5] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, oct 2014. Association for Computational Linguistics. doi: 10.3115/v1/D14-1179. URL https://aclanthology.org/D14-1179.

[6] Thomas Dudzik, Matthew Chignoli, Gerardo Bledt, Bryan Lim, Adam Miller, Donghyun Kim, and Sangbae Kim. Robust autonomous navigation of a small-scale quadruped robot in real-world environments. In *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3664–3671. IEEE, 2020.

[7] Péter Fankhauser, Michael Bloesch, and Marco Hutter. Probabilistic terrain mapping for mobile robots with uncertain localization. *IEEE Robotics and Automation Letters (RA-L)*, 3(4):3019–3026, 2018. doi: 10.1109/LRA.2018.2849506.

[8] Chelsea Finn and Sergey Levine. Deep visual foresight for planning robot motion. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2786–2793. IEEE, 2017.

[9] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2997–3004. IEEE, 2014.

[10] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. Batch informed trees (bit*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 3067–3074. IEEE, 2015.

[11] Scott Gilroy, Derek Lau, Lizhi Yang, Ed Izaguirre, Kristen Biermayer, Anxing Xiao, Mengti Sun, Ayush Agrawal, Jun Zeng, Zhongyu Li, et al. Autonomous navigation for quadrupedal robots with optimized jumping through constrained obstacles. In *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, pages 2132–2139. IEEE, 2021.

[12] Toni Giorgino. Computing and visualizing dynamic time warping alignments in r: the dtw package. *Journal of statistical Software*, 31:1–24, 2009.

[13] Yukai Gong, Ross Hartley, Xingye Da, Ayonga Hereid, Omar Harib, Jiunn-Kai Huang, and Jessy Grizzle. Feedback control of a cassie bipedal robot: Walking, standing, and riding a segway. In *2019 American Control Conference (ACC)*, pages 4559–4566. IEEE, 2019.

[14] Jérôme Guzzi, R Omar Chavez-Garcia, Mirko Nava, Luca Maria Gambardella, and Alessandro Giusti. Path planning with local motion estimations. *IEEE Robotics and Automation Letters*, 5(2):2586–2593, 2020.

[15] Tuomas Haarnoja, Sehoon Ha, Aurick Zhou, Jie Tan, George Tucker, and Sergey Levine. Learning to walk via deep reinforcement learning. In *Proceedings of Robotics: Science and Systems*, FreiburgimBreisgau, Germany, June 2019. doi: 10.15607/RSS.2019.XV.011.

[16] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[17] Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. Real-time loop closure in 2d lidar slam. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 1271–1278. IEEE, 2016.

[18] Arne-Christoph Hildebrandt, Moritz Klischat, Daniel Wahrmann, Robert Wittmann, Felix Sygulla, Philipp Seiwald, Daniel Rixen, and Thomas Buschmann. Real-time path planning in unknown environments for bipedal robots. *IEEE Robotics and Automation Letters*, 2(4):1856–1863, 2017.

[19] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[20] David Hoeller, Lorenz Wellhausen, Farbod Farshidian, and Marco Hutter. Learning a state representation and navigation in cluttered and dynamic environments. *IEEE Robotics and Automation Letters*, 6(3):5081–5088, 2021.

[21] Marco Hutter, Christian Gehring, Dominic Jud, Andreas Lauber, C Dario Bellicoso, Vassilios Tsounis, Jemin Hwangbo, Karen Bodie, Peter Fankhauser, Michael Bloesch, et al. Anymal-a highly mobile and dynamic quadrupedal robot. In *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 38–44. IEEE, 2016.

[22] Jemin Hwangbo, Joonho Lee, and Marco Hutter. Per-contact iteration method for solving contact dynamics. *IEEE Robotics and Automation Letters*, 3(2):895–902, 2018.

[23] Jemin Hwangbo, Joonho Lee, Alexey Dosovitskiy, Dario Bellicoso, Vassilios Tsounis, Vladlen Koltun, and Marco Hutter. Learning agile and dynamic motor skills for legged robots. *Science Robotics*, 4(26), 2019.

[24] Brian Ichter, James Harrison, and Marco Pavone. Learning sampling distributions for robot motion planning. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7087–7094. IEEE, 2018.

[25] Gabriel Ilharco, Vihan Jain, Alexander Ku, Eugene Ie, and Jason Baldridge. General evaluation for instruction conditioned navigation using dynamic time warping. *NeurIPS Visually Grounded Interaction and Language Workshop*, 2019.

[26] Gregory Kahn, Adam Villaflor, Bosen Ding, Pieter Abbeel, and Sergey Levine. Self-supervised deep reinforcement learning with generalized computation graphs

for robot navigation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5129–5136. IEEE, 2018.

[27] Gregory Kahn, Pieter Abbeel, and Sergey Levine. Badgr: An autonomous self-supervised learning-based navigation system. *IEEE Robotics and Automation Letters*, 6 (2):1312–1319, 2021.

[28] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

[29] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Autonomous robot vehicles*, pages 396–404. Springer, 1986.

[30] Donghyun Kim, D Carballo, Jared Di Carlo, Benjamin Katz, Gerardo Bledt, Bryan Lim, and Sangbae Kim. Vision aided dynamic exploration of unstructured terrain with a small-scale quadruped robot. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2464–2470. IEEE, 2020.

[31] Lydia E Kavraki Jean-Claude Latombe. Probabilistic roadmaps for robot path planning. *Pratical motion planning in robotics: current aproaches and future challenges*, pages 33–53, 1998.

[32] Steven M LaValle et al. Rapidly-exploring random trees: A new tool for path planning. 1998.

[33] Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning quadrupedal locomotion over challenging terrain. *Science robotics*, 5 (47), 2020.

[34] Namhoon Lee, Wongun Choi, Paul Vernaza, Christopher B Choy, Philip HS Torr, and Manmohan Chandraker. Desire: Distant future prediction in dynamic scenes with interacting agents. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 336–345, 2017.

[35] Zhongyu Li, Jun Zeng, Shuxiao Chen, and Koushil Sreenath. Vision-aided autonomous navigation of underactuated bipedal robots in height-constrained environments. *arXiv preprint arXiv:2109.05714*, 2021.

[36] Kendall Lowrey, Aravind Rajeswaran, Sham Kakade, Emanuel Todorov, and Igor Mordatch. Plan Online, Learn Offline: Efficient Learning and Exploration via Model-Based Control. In *International Conference on Learning Representations (ICLR)*, 2019.

[37] Matias Mattamala, Nived Chebrolu, and Maurice Fallon. An efficient locally reactive controller for safe navigation in visual teach and repeat missions. *IEEE Robotics and Automation Letters*, 2022.

[38] Takahiro Miki, Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning robust perceptive locomotion for quadrupedal robots in the wild. *Science Robotics*, 7(62):eabk2822, 2022.

[39] Anusha Nagabandi, Kurt Konolige, Sergey Levine, and Vikash Kumar. Deep dynamics models for learning dexterous manipulation. In *Conference on Robot Learning*, pages 1101–1112. PMLR, 2020.

[40] Mark Pfeiffer, Michael Schaeuble, Juan Nieto, Roland Siegwart, and Cesar 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)*, pages 1527–1533. IEEE, 2017.

[41] Ahmed H Qureshi and Michael C Yip. Deeply informed neural sampling for robot motion planning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 6582–6588. IEEE, 2018.

[42] Ahmed H Qureshi, Anthony Simeonov, Mayur J Bency, and Michael C Yip. Motion planning networks. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 2118–2124. IEEE, 2019.

[43] Kihyuk Sohn, Honglak Lee, and Xinchen Yan. Learning structured output representation using deep conditional generative models. *Advances in neural information processing systems*, 28:3483–3491, 2015.

[44] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. doi: 10.1109/MRA.2012.2205651. https://ompl.kavrakilab.org.

[45] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[46] Lei Tai, Shaohua Li, and Ming Liu. A deep-network solution towards model-less obstacle avoidance. In *2016 IEEE/RSJ international conference on intelligent robots and systems (IROS)*, pages 2759–2764. IEEE, 2016.

[47] Lei Tai, Giuseppe Paolo, and Ming 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)*, pages 31–36. IEEE, 2017.

[48] Martin Wermelinger, Péter Fankhauser, Remo Diethelm, Philipp Krüsi, Roland Siegwart, and Marco Hutter. Navigation planning for legged robots in challenging terrain. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1184–1189. IEEE, 2016.

[49] Grady Williams, Andrew Aldrich, and Evangelos Theodorou. Model predictive path integral control using covariance variable importance sampling. *arXiv preprint arXiv:1509.01149*, 2015.

[50] David Wooden, Matthew Malchano, Kevin Blankespoor, Andrew Howardy, Alfred A Rizzi, and Marc Raibert. Autonomous navigation for bigdog. In *2010 IEEE international conference on robotics and automation*, pages 4736–4741. Ieee, 2010.

[51] Zhaoming Xie, Glen Berseth, Patrick Clary, Jonathan Hurst, and Michiel van de Panne. Feedback control for cassie with deep reinforcement learning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1241–1246. IEEE, 2018.

[52] Bowen Yang, Lorenz Wellhausen, Takahiro Miki, Ming Liu, and Marco Hutter. Real-time optimal navigation planning using learned motion costs. In *IEEE International Conference on Robotics and Automation (ICRA 2021)*, page 699, 2021.

[53] Clark Zhang, Jinwook Huh, and Daniel D Lee. Learning implicit sampling distributions for motion planning. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3654–3661. IEEE, 2018.