Path Planning With Local Motion Estimations

Jérôme Guzzi, R. Omar Chavez-Garcia, Mirko Nava, Luca Maria Gambardella, and Alessandro Giusti

***Abstract*—We introduce a novel approach to long-range path planning that relies on a learned model to predict the outcome of local motions using possibly partial knowledge. The model is trained from a dataset of trajectories acquired in a self-supervised way. Sampling-based path planners use this component to evaluate edges to be added to the planning tree. We illustrate the application of this pipeline with two robots: a complex, simulated, quadruped robot (ANYmal) moving on rough terrains; and a simple, real, differential-drive robot (Mighty Thymio), whose geometry is assumed unknown, moving among obstacles. We quantitatively evaluate the model performance in predicting the outcome of short moves and long-range paths; finally, we show that planning results in reasonable paths.**

***Index Terms*—Motion and path planning, deep learning in robotics and automation, probability and statistical methods.**

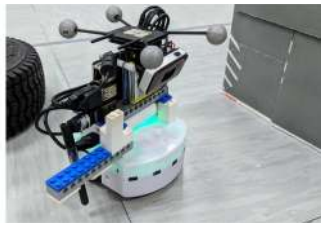
I. Introduction

SAMPLING-BASED planning [1] is a powerful and general solution to path planning that casts the problem as a search for a sequence of feasible local motions, each of which links two nearby states: the first local motion starts at the source state and the last ends at the target state. The feasibility of local motions is determined by a *local motion estimator*, which should account for the robot kinematics, the local planners and controllers at play, and all available knowledge about the environment.

The local motion estimator is typically a simple component that checks whether a direct move linking two states collides with known obstacles and complies with the robot's kinematic constraints. However, this approach is unsuitable for cases with complex, possibly stochastic interactions between the robot and its environment.

Consider, as an example, the case of a legged robot planning a long path on a known, rugged terrain. A possible alternative to evaluate whether a local motion is feasible is to accurately simulate the robot moving on the terrain patch between two locations. Such simulation must account, among other factors, for the terrain shape, the robot-terrain interaction, the characteristics of the robot's sensing ability and low-level controllers, and the probability that the robot could slip on the terrain. This is problematic for two reasons: computational expense (a local motion estimator is queried very often, and therefore needs to be fast); and partial knowledge about the environment (e.g., at planning time, we might know the terrain topography but not its softness/friction characteristics).

(a) ANYmal simulation on challenging terrain.



(b) Real Mighty Thymio with an extension on the right.

Fig. 1. Platforms used in our experiments.

Our **main contribution** is a novel approach in which:

- we *learn* an estimator to predict the outcome (e.g., success probability, duration, energy, ...) of motions linking two nearby states, given the available (possibly incomplete) knowledge about the local environment;
- we use the learned estimator in a sampling-based global planning framework to identify feasible local motions and assign them a cost to be minimized (e.g., finding the path with minimal failure probability),

under the assumptions that: (1) a local planner and a controller are available; (2) our knowledge of the environment is static (in particular, at planning time, the map covers source and target locations and the robot does not acquire new information during trajectory execution); and (3) the robot can not recover from failures and has thus no need for dynamic re-planning. We will briefly discuss in Section VI how some of these restrictions can be lifted.

In this context, training the estimator requires to collect instances in the form of input-output pairs: the input is a pair of nearby states and any available knowledge about the surrounding environment; the output is the outcome of the move. Such data can be acquired either in simulation or in the real world, with a robot attempting local motions and measuring their outcomes. If the robot can sense the outcome without external help (e.g., by visual odometry to measure advancement), the approach can be implemented in a self-supervised [2] fashion, and the robot can progressively adapt the estimator to the environment.

Manuscript received September 10, 2019; accepted January 8, 2020. Date of publication February 10, 2020; date of current version February 21, 2020. This letter was recommended for publication by Associate Editor Dr. C. Ekenna and Editor Prof. N. Amato upon evaluation of the reviewers' comments. This work has been conducted as part of ANYmal Research, a community to advance legged robotics. This work was supported by the Swiss National Centre of Competence in Research Robotics. *(Corresponding author: Jerome Guzzi.)*

The authors are with the Dalle Molle Institute for Artificial Intelligence (IDSIA), USI-SUPSI, Lugano 6928, Switzerland (e-mail: jerome@idsia.ch; omar@idsia.ch; mirko@idsia.ch; luca@idsia.ch; alessandrog@idsia.ch).

This article has supplementary downloadable material available at https://ieeexplore.ieee.org, provided by the authors.

Digital Object Identifier 10.1109/LRA.2020.2972849

The approach naturally handles planning in partially-known, stochastic environments, i.e., where robot's actions do not lead to deterministic outcomes. In particular, by adopting probabilistic machine learning [3] tools, one can expect that any uncertainty in the estimator inputs is properly mapped to uncertainty in the outputs, which is handled in a principled way when planning.

Note that the approach does not require that a model of the robot is available (unless one uses simulations to acquire training data) and resembles the way humans plan paths in open, rugged terrains: estimating from a distance whether a given passage is traversable relies on a quick, intuitive evaluation based on partial information about the local environment and on previous experience attempting to negotiate similar-looking passages. Crucially, this evaluation accounts for the operation and performance of lower-level controllers, i.e., in our case, the motor skills of the subject. An example of online learning in this context occurs when inexperienced hikers learn how to plan paths that avoid wet rocks after slipping and falling once on such surface.

After reviewing related work in Section II, we formalize our abstract model (Section III) and describe implementations (Section IV) in two contexts: a simulated legged robot navigating rugged terrain and a real wheeled robot with unknown geometry navigating among obstacles. Qualitative and quantitative experimental results are reported and discussed in Section V.

II. Related Work

A. Path Planning

A common strategy for integrating machine learning in robotic path planning [4] is imitation learning of expert trajectories. Along this line of research, Pfeiffer *et al.* [5] train a CNN to compute steering commands to reach the desired target pose given current readings from a laser scanner; Ollis *et al.* [6] analyze expert trajectories to learn features of traversed map areas and then compute cost maps in a Bayesian framework; Bagnell *et al.* [7] use inverse optimal planning to infer a cost function for which the expert trajectories would be optimal.

In contrast, our work relies on *sampling-based* graph search. Sampling-based motion planning algorithms [1] handle high-dimensional configuration spaces and rely on a local planner (or steering function) that determines the connectivity between nodes of a graph; in this context, one can search in the space of controls to take into account dynamical or differential constraints. The Rapidly Exploring Random Trees (RRT) [8] algorithm iteratively constructs a tree by sampling the configuration space to add edges that lead to feasible states; once the target state is connected to the tree, the path is returned. Alternatively, the Probabilistic Road Maps (PRM) [9] algorithm builds a graph of feasible edges in which the best path is found using, e.g., Dijkstra's algorithm. In general, sampling-based planners are probabilistically complete, i.e., they find a path from source to target with probability 1 if such a path exists and if they are given enough time to grow the graph. Optimal variants such as RRT* and PRM* asymptotically converge to the path of least cost [10]. Machine learning has been used to augment components of sampling-based planners, e.g., for automatically selecting the best sampler [11] or the best expansion strategy depending on the neighborhood of a node [12]. In this work, we apply machine learning on a different component, i.e., the one that determines the connectivity of two nearby states.

B. Search Spaces

Some planning strategies explore the space of states and determine feasible paths as sequences of feasible states to traverse, for example, focusing on collision-free constraints or searching for a stable gait state [13]. Alternatively, planners search in the action space: feasible paths are sequences of feasible actions to execute. In our previous work [14], we estimated the feasibility of motions on a regular horizontal grid. In this work, we instead randomly sample from a much larger space.

C. Local Motion Estimation

Estimating the feasibility of local motions is a key component of global path planning algorithms. For instance, for legged robots, traversability scores can be assigned to cells of 2D maps to account for the robot's geometry, and the terrain roughness and slope [15]; in a more complex approach, Fankhauser *et al.* [13] first compute stable footholds, and then body and leg trajectories to overcome steep sections and high steps. In this context, when the interaction between the robot and the environment is complex, one can *train* a model to output traversability scores: Wellhausen *et al.* [16] first use a weakly-supervised method to segment terrain classes from a front-facing camera image, then use a CNN to compute the confidence of keeping balance on potential footholds, which is mapped to a cost function. Similar approaches, which first classify terrain and then assign a cost to a grid map, have been developed, e.g., during the LAGR project [17].

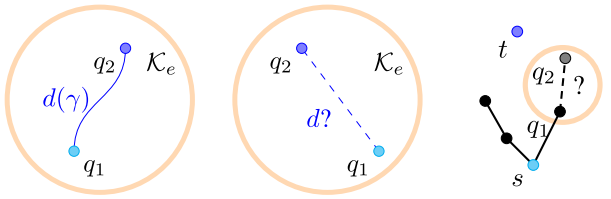
Our proposed approach, given a local planner, trains an estimator to predict which local motions are feasible, using many *random* trials on diverse terrains; then, instead of building a fixed resolution grid map, samples from the whole space of local motions to build a dense planning tree. Moreover, it does not rely on hand-modeled terrain features (like slope or roughness) but learns more general features from examples on a variety of terrains. In contemporary work, Chiang *et al.* [18] propose a similar approach for integrating a local planner trained using reinforcement learning in a global sampling-based planner for kinodynamic constrained robots. The general approach of using machine learning to approximate the outputs of a computationally expensive planner is also found in imitation learning, e.g., when the expert trajectories for a legged robot originate from a sophisticated high-dimensional footholds planner [7] instead of a human pilot.

III. Model

In this section, we describe the global path planning problem as a search in the space of sequences of local motions, whose feasibility and costs are estimated through machine learning (see Fig. 2).

A. Local Motions

The robot moves in an environment of which it has, at planning time, partial knowledge $\mathcal{K}$ modeled as a set of relevant pieces of



(a) A sample $(\mathcal{K}_e, e, d(\gamma))$ is an outcome of a local motion from q_1 to q_2.

(b) A trained model predicts $\mathbb{E}[d(e, \mathcal{K}_e)]$ for an unobserved local motion e.

(c) A planner queries the model for a possible new edge (q_1, q_2).

Fig. 2. Basic definitions, as introduced in Section III: a dataset (a) of local motion outcomes is used to train a model (b), which a sampling-based planner (c) queries to get the success probability and cost of candidate edges for a tree connecting source s with target t. The orange circle represents local knowledge $\mathcal{K}_e$ that the model uses to predict the outcome.

information. A local motion $e = (q_1, q_2) \in \mathcal{C}^2$ is defined by two *nearby* states in the robot's configuration space $\mathcal{C}$. For example, for a differential-drive wheeled robot moving on a plane among obstacles, we may represent a local motion as a pair of poses in the two-dimensional special Euclidean group $\mathrm{SE}(2)$, and knowledge as an obstacle occupancy grid.

When the robot attempts to travel from q_1 to q_2 using a *local* planner and a controller, it follows a time-parametrized trajectory $\gamma : [0, T] \to \mathcal{C}$, with $\gamma(0) = q_1$. In general, multiple executions of the same local motion may result in different trajectories; this randomness has diverse sources, such as: partial knowledge of the environment (possibly based on the robot's sensors); stochastic interaction with the environment while executing the motion (e.g., when a robot slips); local planners that use random algorithms. Therefore, we assume that γ is randomly drawn from a distribution, which depends on the environment, e, and the *local* knowledge $\mathcal{K}_e \subset \mathcal{K}$, which we define as the subset that contains any information about the surroundings of e; for the previous example, local knowledge would be a portion of the occupancy grid map containing the source and target poses.

Let $d(\gamma) = (d_1(\gamma), \ldots, d_n(\gamma)) \in \mathcal{D}$ be an n-dimensional descriptor that should contain any information relevant to evaluate trajectories. For instance, $d_1 \equiv S \in \{0, 1\}$ denotes whenever the robot has reached the target, i.e., if $\gamma(T) = q_2$. Other interesting descriptors may be duration T, energy consumed, number of collisions occurred, and maximal motor torque. Finally, we use the notation $d(e, \mathcal{K}_e)$ to denote the *distribution of descriptors* for trajectories resulting from a local motion e when the robot has local knowledge $\mathcal{K}_e$; the dependency on the environment is always implicit.

B. Learning to Predict Local Trajectories

We use supervised machine learning to predict the outcome of a given local motion (i.e., the expected value of the trajectory descriptors). In particular, we learn the mapping $(e, \mathcal{K}_e) \mapsto y = \mathbb{E}[d(e, \mathcal{K}_e)]$ (see Fig. 2(b)). As a first step, we collect many instances of local trajectories γ by sampling local motions in different environments and recording the values of their descriptors $d(\gamma)$; then, we train a machine learning model using this data.

1) Collecting the Dataset: The robot can collect the dataset (see Fig. 2(a)) in a self-supervised way if the robot directly extracts the trajectory descriptors from on-board sensors and its internal state. As relevant descriptors should only depend on the relative position of q_2 with respect to q_1, and the local knowledge around q_1, we use a local coordinate frame, centered at q_1, to represent q_2 and $\mathcal{K}_e$. The resulting dataset is composed of many samples of the form $(\mathcal{K}_{q_2}, q_2, d_1, d_2, \ldots, d_n)$.

2) Learning: Because some descriptors are categorical (e.g., *S*) while others are continuous (e.g., *duration* and *energy*), the model has multiple outputs: some for classification and others for regression. Fig. 5 shows the deep neural network architecture that we use as a local motion estimator for two specific planning problems.

Algorithm 1: RRT-based planner that builds a random tree $\mathcal{T}$ with resolution δ to compute a path from $s \in \mathcal{C}$ to $t \in \mathcal{C}$, using a model to predict the expected descriptors y of local motions trajectories and knowledge $\mathcal{K}$ about the environment. Only edges with success probability larger than τ are added to the tree and carry a cost function $\mathscr{C}$.

```
Initialize T with a node in s
while t ∉ T do
    q ← random sample from C with a small bias toward t
    q1 ← vertex in T nearest to q
    e ← local motion from q1 toward q of at most length δ
    y ← E[d(e, K_e)]
    if y_S > τ then
        add edge e with cost C(y) to T
    end if
end while
```

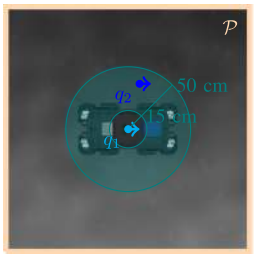
C. Global Path Planning

The global path planner searches for the best trajectory for the robot to travel from $s \in \mathcal{C}$ to $t \in \mathcal{C}$, encoded as a sequence $\pi = (e_1, e_2, \ldots e_m)$ of local motions to perform. We discriminate among trajectories using the descriptors computed by the trained model. In particular, we define an additive cost for the path as $\mathscr{C}(\pi; \mathcal{K}_\pi) = \sum_{e \in \pi} \mathscr{C}(e; \mathcal{K}_e) = \sum_{e \in \pi} \mathscr{C}(\mathbb{E}[d(e, \mathcal{K}_e)])$, where the cost of the local motions depends on the predicted descriptors.

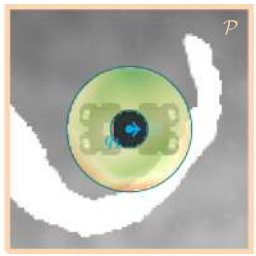
For instance, if we assume that the probabilities of failing different local motions are independent, one interesting cost to minimize is the path *survival* risk $\mathscr{R}$, given by the negative log-probability that the robot completes all local motions

$$\mathscr{R}(\pi; \mathcal{K}_\pi) = -\sum_{e \in \pi} \log \mathrm{P}(S(e, \mathcal{K}_e) = 1) = -\sum_{e \in \pi} \log y_S(e, \mathcal{K}_e). \quad (1)$$

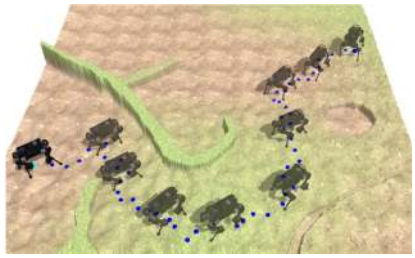
1) Sampling-Based Planner: We solve the global planning problem using a sampling-based planner that iteratively builds a graph of states connected by edges that represent feasible local motions (e.g., local motions with a sufficiently high probability of success). In our implementation, we use randomized variants of Rapid Exploring (Dense) Tree (RDT) [19] algorithms to answer one-shot path planning queries. In general, these algorithms iteratively construct a tree $\mathcal{T} \subset \mathcal{C}$ that covers the configuration space by sampling random points (with a bias towards t) and



(a) One local motion sample: the target point q_2 is uniformly drawn in an annulus (cyan) from 15 cm to 50 cm around q_1.



(b) Evaluation of the trained model: colors indicate predicted success probability y_S (red: low, green: high) for motions to a target point in the annulus: trying to step over the (white) wall is correctly assigned a low chance of success (orange).



(c) In a challenging terrain, ANYmal successfully executes the planned path of lowest risk (blue dots) between start (left) and target poses (right), computed using the trained model.

Fig. 3. ANYmal pipeline: (a) we collect about 90 K random trajectories by picking q_1 uniformly on a map and q_2 in an annulus around it. We use these trajectories to train a model (b) to finally compute plans (c) using RRT*. Local knowledge $\mathcal{K}_{(q_1,q_2)}$ is encoded as a $2\,\mathrm{m} \times 2\,\mathrm{m}$ grid patch $\mathcal{P}$ around the robot bounded by the orange line; heightmaps are depicted as gray-scale images. All drawing are in scale and use experimental data.

connecting them to the tree through local motions, as illustrated in Fig. 2(c) and Algorithm 1, which makes explicit the role of the predictions during planning. The main difference with the original RDT algorithms, like RRT, is that we cannot evaluate infinitesimally short local motions (otherwise their descriptors would be meaningless), and therefore the asymptotic properties are not preserved.

Optimal variants, like RRT* [10] and SST [20] (Stable Sparse RRT), use a cost $\mathscr{C}$ (in our case computed from the predicted descriptors) to grow a tree of optimal paths; adding an edge to the tree may involve locally reorganizing the tree (RRT*) or competing with neighbor branches (SST).

IV. Experimental Setup

In this section, we present two concrete applications of the model presented in Section III for two different robots depicted in Fig. 1: a simulated ANYmal robot and a real Mighty Thymio robot.

Both robots move on 2D terrain whose knowledge $\mathcal{K}$ is encoded as heightmaps (i.e., grey-scale images with an altitude value associated with each pixel); we ignore any property that is not geometrical (like the surface material). Knowledge used by the robot for local motions is given by local heightmaps, i.e., square heightmap patches centered at the robot location, and aligned with its orientation. We implement sampling-based planners using the Open Motion Planning Library (OMPL) [21].

A. Simulated ANYmal

1) Robot and Environment: ANYmal [22] is a state-of-the-art quadruped robot developed at ETHZ for autonomous operation in challenging environments, with the ability to walk on rough terrain. We simulate ANYmal in Gazebo (see Fig. 1(a)) to learn how well the robot copes with different obstacles (e.g., steps, holes, slopes, bumps) and later plan safe paths; ANYmal's footprint is $80\,\mathrm{cm} \times 60\,\mathrm{cm} \times 70\,\mathrm{cm}$. For simplicity, we limit our analysis to local motions that maintain orientation; therefore, the configuration space $\mathcal{C} = \mathbb{R}^2$ is given by the horizontal position of ANYmal's center. Heightmaps have a resolution of 2 cm per pixel; local knowledge is represented as a $100\,\mathrm{px} \times 100\,\mathrm{px} (2\,\mathrm{m} \times 2\,\mathrm{m})$ patch $\mathcal{P}$ of such heightmap (see Fig. 3(a)).

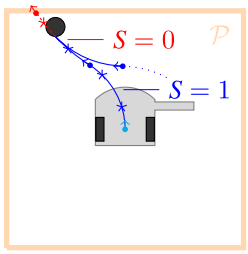
2) Local Motion: The simulated ANYmal uses a simple,[1] locomotion planner and a complex closed-loop feedback controller to follow a local trajectory while compensating for terrain irregularities [23]. The local planner takes as its only input the relative target pose $q_2 = (x, y, \theta)$. For any sampled trajectory γ, $d(\gamma) \in \{0, 1\} \times \mathbb{R}^+$ is composed of *success* S ("has the robot arrived near enough to q_2?") and *duration* T.

3) Dataset Collection: We collect a dataset of about 90 K samples (76% with $S = 1$), where each sample is a tuple $(\mathcal{P}, x, y, S, T)$, by randomly spawning the robot on stable poses and randomly sampling a relative target point at a distance between 15 cm to 50 cm along a random direction; on average the robot needs 8 steps (about 10 seconds) to complete a local motion (Fig. 3(a)). We gather data on 12 different terrains (with total area $1200\,\mathrm{m}^2$), which contain challenging obstacles such as slopes, bumps, holes, rails, and steps. As in our previous work [14], terrain heightmaps are procedurally generated as superpositions of simplex noise at different scales.

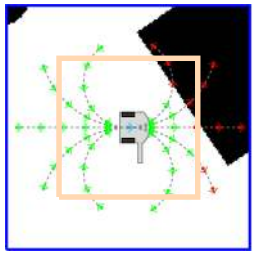
4) Training: The success and duration of local motions are modeled by a deep neural network, outlined in Fig. 5. The estimator has two stages: the first is composed of convolutional layers that operate on the heightmap patch (which has an image-like 2D structure); the second stage processes the resulting features and accepts, as an additional input, the relative target point (x, y). The output of the model consists of the success probability y_S from a dense layer with softmax activation and the estimation of duration y_T from a dense layer. The loss function is given by the sum of categorical cross-entropy (success) and mean squared error (duration). The training dataset contains 70 K samples from 7 terrains; the validation dataset contains 10 K samples from 2 terrains; the evaluation dataset contains 8.7 K samples from 3 terrains; all 12 terrains are distinct.

5) Path Planning: We use the trained local motion model to compute global paths through a planner, which is derived from RRT* in the same way Algorithm 1 has been derived from

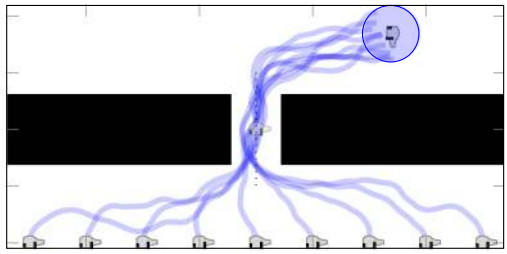
[1] ANYmal also features a more complex local planner [13] which we do not use in this work, that selects the best foothold locations on a terrain.



(a) When the robot collides with an obstacle, it inverts the linear speed (here moving backward) before having reached the target position (red); from the trajectory (blue line), we extract and label many 1 s segments (delimited by star markers).



(b) We use the trained model to predict a sequence of five 1 s local motions with constant linear and angular speeds: target poses are colored by success probability y_S (red: low, green high): collisions are correctly identified (red).



(c) Planning a path through a narrow opening of 44 cm from various starting poses (robots at the bottom) to the same target pose (blue circle) shows that the trained model makes the robot pass on the left side of the corridor to avoid touching with its arm poking out to the right.

Fig. 4. Mighty Thymio pipeline: (a) the robot randomly changes angular speed every 2 s to collect 1 s training samples labeled according to the occurrence (or not, $S \in \{0, 1\}$) of a collision; (b) this model predicts the probability of a collision in the next 1 s for given angular and linear speeds, which (c) we use to plan safe trajectories ($\tau = 0.98$) that minimizes duration using SST. Local knowledge $\mathcal{K}_{(q_1,q_2)}$ is a grid patch $\mathcal{P}$ (orange stroke) of 50 cm around the robot. All drawing are in scale and use experimental data.

RRT in Section III-C1. Namely, the planner iteratively samples a random point q and attempts to grow the tree with a motion from the nearest neighbor of q, towards q, on a segment of length $\delta = 45\,\mathrm{cm}$. If q is closer than δ, the candidate is retained as long as the distance is above 15 cm; otherwise, it is discarded. Based on the success score y_S returned by the estimator, we assign a risk $-\log y_S$ to each edge while we consider any edge with a score lower than a threshold $\tau = 0.5$ as not feasible. We run RRT* to find the path with minimal survival risk, as defined in Eq. (1); as RRT* is an anytime algorithm that converges towards the optimal solution, we let the tree grow for a given computational time budget.

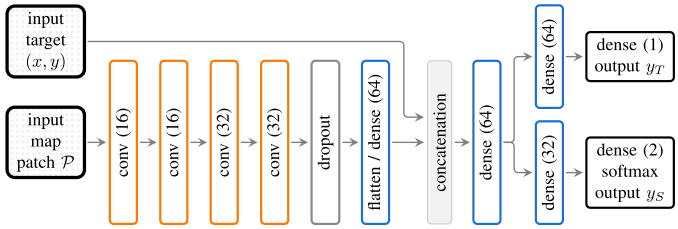


Fig. 5. Neural Network architecture for ANYmal discussed in Section IV-A. Given the robot-centered heightmap patch $\mathcal{P}$ and the relative target goal (x, y), the model outputs a probability of reaching the target pose and an estimation of the required time. The architecture used for Mighty Thymio is very similar (see Section IV-B).

B. Mighty Thymio

1) Robot and Environment: The Mighty Thymio [24] is an extension of the small educational robot Thymio [25] for research use. It features two differentially driven wheels and its configuration space $\mathcal{C} = \mathrm{SE}(2)$ is composed of the robot's position and orientation. The small mobile robot moves on the flat floor of a room that has several obstacles on the ground (see Fig. 1(b)); to make the planning problem more interesting, we have extended the body of the robot by adding a rigid arm on its right. The Mighty Thymio size is $11\,\mathrm{cm} \times 11\,\mathrm{cm} \times 20\,\mathrm{cm}$, and the arm size is $10\,\mathrm{cm} \times 2\,\mathrm{cm} \times 0.5\,\mathrm{cm}$. As another application of the model introduced in Section III, the goal is to learn to predict if the robot (whose geometry is assumed unknown) would collide with obstacles during a short motion and use this information to plan collision-free paths. We represent environments as heightmaps with a resolution of 0.625 cm per pixel; local knowledge is represented by robot-centered $80\,\mathrm{px} \times 80\,\mathrm{px}$ ($50\,\mathrm{cm} \times 50\,\mathrm{cm}$) patches $\mathcal{P}$.

2) Local Motion: Like ANYmal, the Mighty Thymio could use a relatively complex planner/controller combination to perform arbitrary local motions in free space. Instead, we present a simpler model that highlights the generality of our approach; namely, we limit local motions to have constant linear speed $v \in \{-8\,\mathrm{cms}^{-1}, -8\,\mathrm{cms}^{-1}\}$, angular speed $\omega \in [-0.5\,\mathrm{rad\,s}^{-1}, -0.5\,\mathrm{rad\,s}^{-1}]$, and duration $T = 1\,\mathrm{s}$; in this case, the local planner is trivial and just executes the corresponding control. For every local motion in the training set, we record $S \in \{0, 1\}$, which discriminates whenever a collision occurred during the local motion.

3) Dataset Collection: We acquire data using a controller that randomly selects a new angular speed every 2 s. Each time a collision occurs, the robot records it and inverts its motion direction (i.e., starts moving backward if it was moving forward, and vice versa). A motion capture system records the position of all obstacles and the robot. The robot runs for a total time of 120 min (in 6 different maps), during which about 460 collisions were recorded. After the acquisition, we process the data to extract 1 s trajectories with constant controls: we assign $S = 1$ to those without collisions, while we assign $S = 0$ to those where a collision occurs (see Fig. 4(a)). In total, we collect a dataset of 104 K samples of the form $(\mathcal{P}, v, \omega, S)$.

4) Training: We adopt the CNN architecture depicted in Fig. 5, with the only difference that the output is only the success probability (no-collision) of the motion. We generate the dataset from three disjoint sets of maps: 52 K samples were used for training (3 maps), 20 K for validation (1 map), and 30 K for evaluation (2 maps).

5) Path Planning: To take into account the differential constraints of the robot's motion, we use a variant of SST, which samples control inputs to reach a local target point. More precisely, SST iteratively samples a random pose $q \in \mathrm{SE}(2)$ and several control inputs, among which it selects the one that would lead the robot nearest to q from its nearest neighbor; the planner then adds an edge to the tree only if it decreases the local cost-to-come. We discard any edge with a success probability lower than $\tau = 0.98$ and assign a fixed cost of 1 (i.e., proportional to their duration) to feasible edges. SST is an anytime algorithm that converges towards a nearly optimal solution; therefore, we let the tree grow for a fixed amount of computational time.

V. Experimental Results

How well are we able to predict the outcome of local motions? Are the local models useful for path planning? Are the assumptions we made in Section III justified? In particular, are we justified to interpret y_S as the *success* probability for local motions and use it to compute the *success* probability for an entire path? In this section, we present experimental results that partially answer these questions for the two setups introduced in Section IV.

Qualitative tests, like those illustrated in Figs. 3(b) and 4(b), show that the learned model's predictions are coherent with the map: the ANYmal model correctly estimates that motions that step over the short wall are risky; the Mighty Thymio model correctly predicts collisions. Similarly, computing paths between hand-picked locations on interesting maps, like in Figs. 3(c) and 4(c), shows that the planner computes sensible paths: ANYmal avoids high steps and slopes, and Mighty Thymio (with an arm on the right) stays on the left side of a corridor to avoid collisions. The video in the supplementary material contains several examples of planned paths and their executions.

In the rest of this section, we report quantitative, statistically significant results. First, we present metrics on the evaluation datasets for the prediction of local motions. Then, we report results for the planned paths, focusing on the ANYmal case, for which we gathered a large number of executed samples in simulation.

A. Local Motion Prediction

1) ANYmal: Fig. 6:left illustrates how well the ANYmal model outputs correspond to the true values on the *evaluation* dataset; the corresponding performance scores are listed in the evaluation row of grey columns (related to testing single local motions) in Table I (AUC $= 0.889$). The model is very well calibrated for both outputs.

2) Mighty Thymio: Similarly, we report in Table II and Fig. 6:right the results for the Mighty Thymio model predicting success probability (AUC $= 0.972$). During data collection, the robot had a relatively small number of collisions, which lead to training and evaluation datasets that are very unbalanced (one failed sample every 25 successful samples), which could explain the inferior calibration (compared to the ANYmal model) of a model that tends to overestimate the success probability.

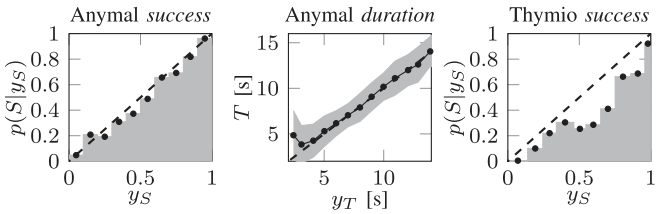


Fig. 6. Model evaluation (ground truth vs prediction) on local motions for ANYmal (8700 samples, left and center) and Mighty Thymio (30 K samples, right). In the *success* plots, we group y_S in 0.1 width bins. In the *duration* plot, dots mark the mean value of T for outputs y_T grouped in 1 s width bins, while the grey area is delimited by $\pm$ one standard deviation. Perfect models are represented by dashed black lines.

TABLE I
ANYmal Model Evaluation for *Success* Prediction: Number of Samples for the Two Classes and Area Under the ROC (AUC). The Columns Contain the Results for Single Local Motions e (gray), Paths π Between Random Source and Target Locations on Selected Maps (White), and all Subpaths $sub\pi$ Extracted From Such Paths (Blue)

dataset	# successes			# failures			AUC		
	e	π	$sub\pi$	e	π	$sub\pi$	e	π	$sub\pi$
evaluation	7.30K	-	-	1.40K	-	-	0.889	-	-
rough map	18.4K	562	297K	185	185	49.7K	0.878	0.904	0.912
surf map	22.4K	564	431K	237	237	63.5K	0.962	0.962	0.952

TABLE II
Mighty Thymio Model Evaluation: Number of Samples for the Two Classes, Area Under the ROC (AUC), and *Success* Probability for Scores Above $\tau = 0.98$, Which is the Threshold We Use to Compute the Plans According to Section IV-B

successes	failures	AUC	$\mathrm{P}(S = 1 \mid y_S > 0.98)$
28913	1087	0.972	0.997

Nonetheless, the model, with a threshold $\tau = 0.98$, yields a precision of 99.7% and a recall of 96.1%, which prove to be good enough for planning on our maps.

B. Path Planning for ANYmal

In this section, we report results that investigate the core of our contribution, i.e., using the learned local motion model to plan a path: we compute and then execute many paths between target and source locations for the simulated ANYmal. In particular, we randomly sample 1000 pairs of source and target states in two different maps (*rough* and *surf*) of size $10\,\mathrm{m} \times 10\,\mathrm{m}$ illustrated at the top of Fig. 8: we reject any pair outside of the green areas (where we expect a successful spawning of the simulated robot) and any pair with a distance shorter than 3 m. The two maps have not been used to train or evaluate the model and have been hand-designed as realistic (*rough*) and easy legible (*surf*, with a smooth, gradual slope) outdoor terrains.

One evaluation of the model requires about 1 ms on a single, modern desktop, CPU. We fix the RRT* planning duration to 120 s, which is sufficient to cover the maps with a granularity of $\delta = 0.45\,\mathrm{m}$, resulting in trees with about 10 K edges and that require about 100 K model evaluations.

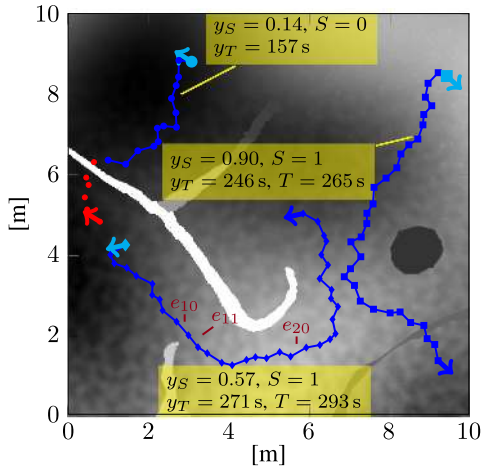


Fig. 7. Three samples of paths from source (cyan arrow) to target (blue arrow) and their execution on the *rough* map: one failure (partly red) and two successes (all blue). For any execution, we record ground truth and model prediction (yellow). Because of stochastic interactions with the terrain, multiple runs may have different outcomes; for examples, 30 trials of the bottom path result in 14 failures and 16 successes, for which $\mathrm{P}(S = 1) \approx 0.53$ is remarkably near to the predicted value $y_S = 0.57$, with failures distributed in 3 out of 34 segments ($e_{10} : 7, e_{11} : 3, e_{20} : 4$).

As illustrated[2] in Fig. 7, for any planned path, we spawn the robot at the source location and pass the sequence of intermediate target poses to the controller. We record the success and duration of each segment of the path: in about 80% of the cases the robot arrives safely at the target, while in the remaining cases we observe a failure to complete one segment. To increase the number of (path) samples (from 1 K to around 400 K), from any path we extract all subpaths that start before a failure (if any): each subpath has well-defined ground truth labels and predictions for *success* and *duration*.

Table I compares the prediction performance of *success* for single segments, source-target paths, and all their subpaths. For both maps, the AUC is similar across the different datasets.[3] Performance on the *rough* map is comparable to the evaluation dataset of single segments discussed before, while predictions on the *surf* map are easier.

Fig. 8 illustrates the quality of the prediction when applied to subpaths. In general, the success score is well calibrated although it tends to overestimate the success chance of risky paths (i.e., those with low score); the model is better calibrated on the *rough* map than on *surf*, which is smoother but has steeper slopes. Instead, the duration of subpaths is systematically underestimated; we believe that this is an artifact: in training, we record the time until a single pose is reached, which may happen while the robot has not fully stopped yet; instead, during path execution, the controller waits until the robot is still to start moving towards the next pose. Nonetheless, the error is relatively small at around 5%.

[2]The supplementary video captures ANYmal planning queries on the *surf* map too and planning queries for Mighty Thymio.

[3]We note that the vast majority of the segments along a path are successful, which leads to local motions datasets on *rough* and *surf* that are significantly unbalanced.

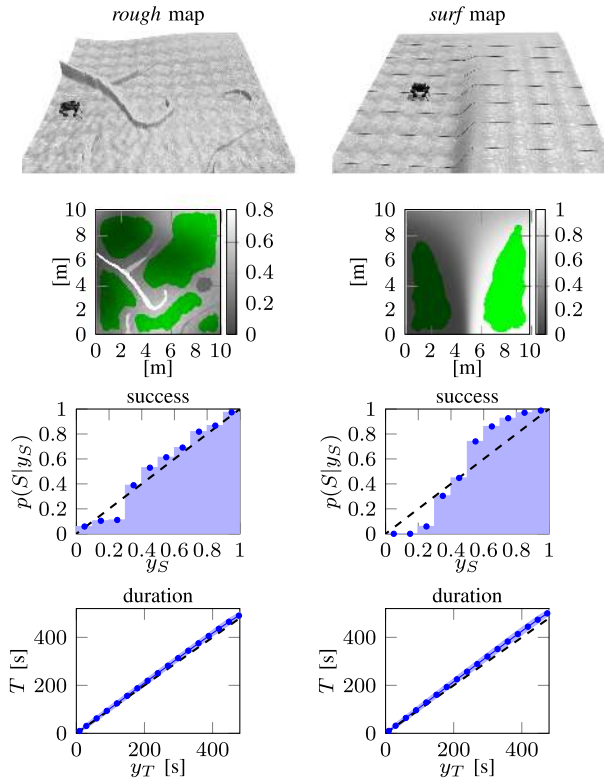


Fig. 8. ANYmal model evaluation on about 400 K subpaths for two maps (*rough* left and *surf* right). The first row shows the 3D view of the terrain with a correctly sized robot. The second row depicts the heightmap (in meters) with spawning areas in green. In the plots, dots marks mean values for binned prediction outputs, while the area on the bottom plots covers ± one standard deviation.

VI. Conclusions

We introduced a novel, data-driven approach to robotic path planning: first, we learn to predict the outcome of short trajectories from local information; then, we use this model to discriminate between potential edges while building a dense random tree to connect source and target states. We applied the model in two scenarios: computing geometrical paths on rough terrain for a simulated legged ANYmal robot with a sophisticated controller whose performance would be difficult to model by hand; and computing control trajectories for a real, non-holonomic, Mighty Thymio robot moving between obstacles with a simple controller. Note that, in the second case, it would be straightforward to analytically model the collision risk of the robot with known obstacles; yet, it is interesting that our data-driven approach yields acceptable results, using a small training dataset and requiring no explicit knowledge of the robot geometry.

To assign a risk to the path (as a function of the estimated risk of individual edges), we assumed that the model output was calibrated and that edge failure probabilities were independent (or at least Markovian). We verified that this is the case for ANYmal, for which we acquired a large and balanced dataset: our approach yields accurate predictions of success probability for entire paths. In general, we observe that the size and class balance of training datasets is a crucial factor; ongoing

work focuses on assessing requirements for path planning tasks. Moreover, the machine learning models that have been trained do not take into account the sequential nature of following a path: we can improve this by defining loss functions on sequences of segments rather than on individual segments.

We are presently working on a richer model for ANYmal, with local motions that change orientation and additional labels, such as consumed energy and multiple failure modes (getting stuck, capsizing, low-level control errors); we are going to compare the plans produced by our pipeline with those obtained by modeling a cost function by hand [26].

To keep the model simple, we assumed that knowledge is static, that the map is known, and that failures are fatal. To drop these assumptions, one can integrate our data-driven approach with online learning, receding horizons, and fast re-planning strategies for sampling-based planners [27]; this would address more realistic scenarios where the robot is able to re-plan once it discovers new information or experiences a non-fatal failure.

Finally, we illustrated the model in the context of path planning, but the application scope of the main idea is broad enough that it can be applied to any planning problem using actions whose outcome can be predicted by a machine learning estimator.

REFERENCES

[1] M. Elbanhawi and M. Simic, "Sampling-based robot motion planning: A review," *IEEE Access*, vol. 2, pp. 56–77, 2014.

[2] C. Doersch, A. Gupta, and A. A. Efros, "Unsupervised visual representation learning by context prediction," in *Proc. IEEE Int. Conf. Comput. Vision*, 2015, pp. 1422–1430.

[3] Z. Ghahramani, "Probabilistic machine learning and artificial intelligence," *Nature*, vol. 521, pp. 452–459, 2015.

[4] D. González, J. Pérez, V. Milanés, and F. Nashashibi, "A review of motion planning techniques for automated vehicles," *IEEE Trans. Intell. Transp. Syst.*, vol. 17, no. 4, pp. 1135–1145, Apr. 2016.

[5] 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 *Proc. IEEE Int. Conf. Robot. Autom.*, 2017, pp. 1527–1533.

[6] M. Ollis, W. H. Huang, and M. Happold, "A bayesian approach to imitation learning for robot navigation," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2007, pp. 709–714.

[7] J. Bagnell, J. Chestnutt, D. M. Bradley, and N. D. Ratliff, "Boosting structured prediction for imitation learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2007, pp. 1153–1160.

[8] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," Iowa State University, Tech. Rep., 1998.

[9] L. E. Kavraki, P. Švestka, J.-C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. Robot. Autom.*, vol. 12, no. 4, pp. 566–580, Aug. 1996.

[10] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *Int. J. Robot. Res.*, vol. 30, no. 7, pp. 846–894, 2011.

[11] A. Upadhyay and C. Ekenna, "Investigating heterogeneous planning spaces," in *Proc. IEEE Int. Conf. Simul., Model. Program. Auton. Robots*, 2018, pp. 108–115.

[12] J. Denny, M. Morales, S. Rodriguez, and N. M. Amato, "Adapting rrt growth for heterogeneous environments," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2013, pp. 1772–1778.

[13] P. Fankhauser, M. Bjelonic, C. Dario Bellicoso, T. Miki, and M. Hutter, "Robust rough-terrain locomotion with a quadrupedal robot," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2018, pp. 1–8.

[14] R. O. Chavez-Garcia, J. Guzzi, L. M. Gambardella, and A. Giusti, "Learning ground traversability from simulations," *IEEE Robot. Autom. Lett.*, vol. 3, no. 3, pp. 1695–1702, Jul. 2018.

[15] P. Krüsi, P. Furgale, M. Bosse, and R. Siegwart, "Driving on point clouds: Motion planning, trajectory optimization, and terrain assessment in generic nonplanar environments," *J. Field Robot.*, vol. 34, no. 5, pp. 940–984, 2017.

[16] L. Wellhausen, A. Dosovitskiy, R. Ranftl, K. Walas, C. Cadena Lerma, and M. Hutter, "Where should i walk? predicting terrain properties from images via self-supervised learning," *IEEE Robot. Autom. Lett.*, vol. 4, no. 2, pp. 1509–1516, Apr. 2019.

[17] L. D. Jackel, E. Krotkov, M. Perschbacher, J. Pippine, and C. Sullivan, "The DARPA LAGR program: Goals, challenges, methodology, and phase I results," *J. Field Robot.*, vol. 23, pp. 945–973, 2006.

[18] H. L. Chiang, J. Hsu, M. Fiser, L. Tapia, and A. Faust, "Rl-rrt: Kinodynamic motion planning via learning reachability estimators from rl policies," *IEEE Robot. Autom. Lett.*, vol. 4, no. 4, pp. 4298–4305, Oct. 2019.

[19] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2006.

[20] Y. Li, Z. Littlefield, and K. E. Bekris, "Asymptotically optimal sampling-based kinodynamic planning," *Int. J. Robot. Res.*, vol. 35, no. 5, pp. 528–564, 2016.

[21] I. A. Sucan, M. Moll, and L. E. Kavraki, "The open motion planning library," *IEEE Robot. Autom. Mag.*, vol. 19, no. 4, pp. 72–82, Dec. 2012.

[22] M. Hutter *et al.*, "Anymal - a highly mobile and dynamic quadrupedal robot," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2016, pp. 38–44.

[23] P. Fankhauser, C. Dario Bellicoso, C. Gehring, R. Dubé, A. Gawel, and M. Hutter, "Free gait — an architecture for the versatile control of legged robots," in *Proc. IEEE-RAS Int. Conf. Humanoid Robots (Humanoids)*, 2016, pp. 1052–1058.

[24] J. Guzzi, A. Giusti, G. A. Di Caro, and L. M. Gambardella, "Mighty thymio for university-level educational robotics," in *Proc. AAAI Conf. Artif. Intell.*, 2018, pp. 7952–7953.

[25] F. Mondada *et al.*, "Bringing robotics to formal education: The thymio open-source hardware robot," *IEEE Robotics and Automation Mag.*, vol. 24, no. 1, pp. 77–85, Mar. 2017.

[26] M. Wermelinger, P. Fankhauser, R. Diethelm, P. Krüsi, R. Siegwart, and M. Hutter, "Navigation planning for legged robots in challenging terrain," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2016, pp. 1184–1189.

[27] M. Otte and E. Frazzoli, "Rrtx: Asymptotically optimal single-query sampling-based motion planning witutt quick replanning," *Int. J. Robot. Res.*, vol. 35, no. 7, pp. 797–822, 2016.