# Learning Feasible Transitions for Efficient Contact Planning

**Rikhat Akizhanov** RIKHAT.AKIZHANOV@MBZUAI.AC.AE
*Mohamed bin Zayed University of Artificial Intelligence (MBZUAI), UAE*

**Victor Dhédin** VICTOR.DHEDIN@TUM.DE
*Munich Institute of Robotics and Machine Intelligence, Technical University of Munich, Germany*

**Majid Khadiv** MAJID.KHADIV@TUM.DE
*Munich Institute of Robotics and Machine Intelligence, Technical University of Munich, Germany*

**Ivan Laptev** IVAN.LAPTEV@MBZUAI.AC.AE
*Mohamed bin Zayed University of Artificial Intelligence (MBZUAI), UAE*

**Editors:** N. Ozay, L. Balzano, D. Panagou, A. Abate

## Abstract

In this paper, we propose an efficient contact planner for quadrupedal robots to navigate in extremely constrained environments such as stepping stones. The main difficulty in this setting stems from the mixed nature of the problem, namely discrete search over the steppable patches and continuous trajectory optimization. To speed up the discrete search, we study the properties of the transitions from one contact mode to another. In particular, we propose to learn a dynamic feasibility classifier and a target adjustment network. The former predicts if a contact transition between two contact modes is dynamically feasible. The latter is trained to compensate for misalignment in reaching a desired set of contact locations, due to imperfections of the low-level control. We integrate these learned networks in a Monte Carlo Tree Search (MCTS) contact planner. Our simulation results demonstrate that training these networks with offline data significantly speeds up the online search process and improves its accuracy.

**Keywords:** legged robot, contact planning, motion planning, deep learning

## 1. Introduction

The main advantage of legged robots over wheeled robots is their ability to traverse highly irregular and sparse terrains. However, generating motions on such terrains is particularly challenging for both model-based and learning-based methods. Model-based method has two major approaches which either simply consider the geometry of the environment as a nonlinear constraint in the optimal control problem Tassa et al. (2012); Mordatch et al. (2012); Posa et al. (2014); Winkler et al. (2018), or segment the environment to a set of disconnected convex patches Deits and Tedrake (2015) that adds a discrete variable of selecting the proper patch into the motion generation problem, resulting in a mixed-integer optimization problem Deits and Tedrake (2014); Aceituno-Cabezas et al. (2017); Lin et al. (2020); Ponton et al. (2021). Both approaches are limited in the case of locomotion in highly constrained environments like stepping stones. The first approach is often stuck in local minima for terrains with extreme height variations Winkler et al. (2018). The second approach often relies on simplifications that do not take into account the full dynamics of the robot Deits and Tedrake (2014); Ponton et al. (2021); Tonneau et al. (2020), which limits the robot to perform quasi-static maneuvers.

Environments with sparse steppable locations also pose problems for learning-based methods Grandia et al. (2023) and reinforcement learning (RL) methods in particular. To address the

challenge of sparse environments, recent methods deploy teacher/student frameworks Zhang et al. (2023) and curriculum learning where the difficulty of the problem is gradually increased over several training rounds. As an alternative to pure learning approaches, other recent methods deploy a hybrid strategy combining model-based and learning-based methods. For example, Jenelten et al. (2024) deploys trajectory optimization to guide RL exploration by providing dense reward signals as in Bogdanovic et al. (2022). However, this approach is expensive as the decision-making problem must be solved twice (once using TO and once RL) and still requires numerous samples. Several other methods attempt to speed up the planning at runtime by learning optimal control outputs during training. In Lin et al. (2020); Meduri et al. (2021), a network is trained to predict robot capturability under disturbances and then used at runtime to produce footstep sequences. In Bratta et al. (2024), a model is trained to rank promising footstep locations based on a given cost function. At runtime, the model is used jointly with trajectory optimization as an acyclic online planner to select the best foothold location. Although these two methods enable successful navigation in sparse environments, they provide quasi-static solutions and do not account for imperfections of the low-level control as well as full dynamics of the robot, which hinders performance in extremely constrained environments.

In this work, we address the challenge of contact planning for dynamic maneuvers of legged robots in highly constrained environments by learning both dynamic transitions and low-level control compensation. Our approach makes use of Monte Carlo Tree Search (MCTS), a method recently demonstrated to scale effectively and outperform Mixed-Integer Quadratic Programming (MIQP) in dexterous manipulation Zhu et al. (2023) and gait discovery in locomotion Amatucci et al. (2022). Our work extends the framework of Dhédin et al. (2024) where MCTS is used jointly with whole-body nonlinear model predictive control (NMPC) to find dynamically feasible contact plans on stepping stones. In particular, we propose to learn a dynamic feasibility classifier to predict if a contact transition between two contact modes is dynamically feasible. We also introduce a target adjustment network to compensate for imperfections of the low-level control. We experimentally compare our method to Dhédin et al. (2024) and demonstrate significantly increased success rate and speed of our approach when tested in various randomized stepping stones environments on a Go2 quadruped platform. We also compare our framework with the state-of-the-art RL locomotion policy based on Zhang et al. (2023) and show that our method enables locomotion on very challenging terrain where the RL policy fails. In summary, this work proposes the following contributions:

- We propose a dynamic state predictor that estimates the state of the robot for a given contact mode transition, and a feasibility classifier to evaluate if this transition is dynamically feasible. This enables efficient pruning in the tree search.
- We introduce a learning-based method to compensate for inaccuracies of a given low-level controller by regressing its errors with respect to target contact locations.
- We propose to improve the search efficiency of MCTS by incorporating *safety* and *accuracy* heuristics, based on the confidence level of the learned classifier and residual networks.
- Our extensive experiments in randomized environments demonstrate advantages of the proposed approach in terms of speed and accuracy compared to the state of the art.

The rest of the paper is organized as follows. We first formulate the problem of navigation on stepping stones and present the general MCTS framework in Section 2. We then describe the

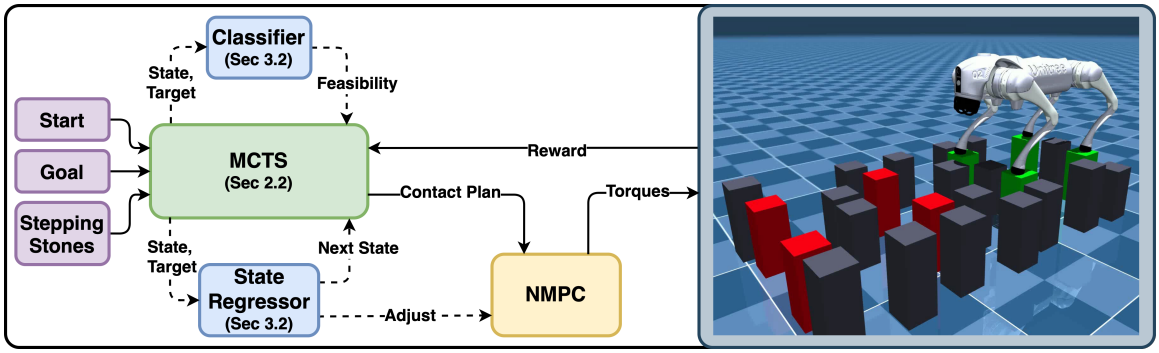



Figure 1: Framework overview with the block diagram. In the MCTS (Section 2.2), a state predictor and a feasibility classifier are jointly called to prune for paths that are likely not to be dynamically feasible (Section 3.2). While running the NMPC in simulation to check for the dynamic feasibility of a contact plan, target contact locations given to the NMPC are adjusted to compensate for the low-level controller inaccuracies (Section 3.3).

proposed improvements in Section 3. We evaluate the performance of our algorithm and compare it to other baselines in Section 4. Section 5 summarizes our findings and outlines future work.

## 2. Problem formulation

We address the problem of dynamic locomotion on stepping stones with a legged robot. Our problem consists of reaching a goal position from a start position in a given stepping stones environment (Fig. 1). The output is a sequence of stepping stones to step on in order to reach the goal, as well as low-level adjustments to land robot closer to stones' center.

### 2.1. Notations

We consider a set of contact locations $\mathcal{C}^W$ in world frame $W$ that are the centers of $N_{stones}$ stepping stones of radius $r_{stones}$. A state in the tree search $\mathbf{s}$ corresponds to a set of stepping stones for each $N_e$ end-effectors, denoted by discrete indices ($\mathbf{s} \in [\![1, N_{stones}]\!]^{N_e}$). The goal state is noted $\mathbf{s}^g$. $\mathcal{C}^W_{\mathbf{s}} \in \mathbb{R}^{N_e \times 3}$ corresponds to the $N_e$ contact locations in $W$ associated to the state $\mathbf{s}$. $\mathcal{C}^W_{\mathbf{s}}[j] \in \mathbb{R}^3$ correspond to the location of the $j$th end effector.

We denote $\mathbf{S} = [\mathbf{p}^W_B, \mathbf{q}^W_B, \mathbf{q}_j, \mathbf{v}^W_B, \mathbf{w}^W_B]$ the robot state. With $B$ the base frame, $\mathbf{p}^W_B$ and $\mathbf{q}^W_B$ the position and orientation (as a quaternion) of the base, $\mathbf{q}_j$ the joint position, $\mathbf{v}^W_B$ the linear velocity and $\mathbf{w}^W_B$ the angular velocity. In the following, $\bar{\mathbf{S}}$ is the state $\mathbf{S}$ without the absolute position. $\mathcal{E}^B \in \mathbb{R}^{N_e \times 3}$ corresponds to the end-effectors locations in the base frame.

### 2.2. Contact planning with MCTS

We formalize our contact planning problem as a Markov Decision Process (MDP). In this MDP, a state $\mathbf{s}$ is a set of contact locations for the end-effectors, represented by their corresponding discrete stepping stones index. The action $\mathbf{a}$ selects the next locations for each end-effector and brings the system to a new state $\mathbf{s}' = f(\mathbf{s}, \mathbf{a})$. To solve this problem, we follow the MCTS formulation in Dhédin et al. (2024) and create a search tree $\mathcal{T} = (\mathcal{V}.\mathcal{E})$, where the set of nodes $\mathcal{V}$ contains the

visited states and the set of edges contains the visited transitions $(\mathbf{s} \xrightarrow{\mathbf{a}} \mathbf{s}')$. Each transition maintains the state-action value $Q(\mathbf{s}, \mathbf{a})$ and the number of visits $N(\mathbf{s}, \mathbf{a})$.

MCTS grows a search tree iteratively using the following steps: the **Selection** phase begins at the root node (initial state) and successively chooses child nodes until a leaf node is reached (an unexpanded node or terminal state). If all children of a node are expanded, a child node based on the highest Upper Confidence Bound (UCB) score is selected Kocsis and Szepesvári (2006). The **Expansion** phase adds the successor states to the tree by enumerating all possible actions, if the selected state is not terminal. The **Simulation** phase performs random actions from one of the successor states for a predefined number of steps and evaluates the reward at the end of this phase. The **Back-propagation** phase updates the state-action values and visit counts for all states along the selected and expanded path, using the reward obtained from the simulation.

Once a sequence from start to goal is found, NMPC (we use the formulation in Meduri et al. (2023)) is rolled-out in simulation to evaluate the dynamic feasibility of the whole sequence. For each transition $(\mathbf{s} \xrightarrow{\mathbf{a}} \mathbf{s}')$ in the sequence, the contact locations for each end-effector corresponding to state $\mathbf{s}'$ are set as the new target contacts in the NMPC. If the whole plan is not feasible (the robot didn't reach the goal), the opposite of the reward is back-propagated. This penalizes transitions leading to failure, as it lowers the value and the UCB score of the nodes in the infeasible sequence. The NMPC is not used to check the dynamic feasibility of each transition in the tree expansion, as it would be too time-consuming. It is only used to check a full path to the terminal state, if such a kinematically feasible path is found.

To speed up the search, Dhédin et al. (2024) proposed a few adjustments. In the selection and simulation step, a heuristic guides the search towards states closer to the goal (shortest path). In the expansion step, only a kinematically feasible set of contact locations is expanded to prune the tree: states corresponding to a configuration where the legs are crossed or too far from the current contact locations are pruned. We refer to this as *kinematic* pruning. It will be our baseline in the experiments and corresponds to the algorithm proposed by Dhédin et al. (2024).

## 3. Method

In this section, we propose three major modifications to the MCTS contact planner to improve its accuracy and to speed up the search process, see Fig. 1 for an overview.

As the first modification, we propose to train a classifier `c` and a state predictor `m` that are evaluated during the search in the expansion step to prune the dynamically infeasible transitions. The classifier takes as input the robot state $\bar{\mathbf{S}}$ and the contact locations of the transition, i.e. $\mathcal{C}^B_{\mathbf{s}}$ and $\mathcal{C}^B_{\mathbf{s}'}$. During the search, the state of the robot after a transition is unknown. Therefore, $\bar{\mathbf{S}}$ is predicted jointly by a state predictor `m`. Only states with a classification score higher than a threshold $t_{feasible}$ are considered feasible. This is what we call *dynamic* pruning in the rest of the paper. Details about the classifier and predictor networks are given in Sections 3.2.

As the second modification, we introduce a *target adjustment network* `t` to reduce the inaccuracies of the NMPC controller during the simulation. The network is trained so that the achieved contact locations (see Fig. 2) are closer to the target locations given to the NMPC, as presented in Section 3.3. This is notably useful in the case of small stepping stones, where accuracy is needed, as demonstrated in Section 4.2.

As the third modification, we propose to use the confidence of the classifier and state predictor networks to inform the search. We take into account the chances of dynamically succeeding in a

transition based on the confidence level of the neural networks. We greedily use the heuristic in the selection and simulation steps of MCTS by selecting the state with maximum heuristic value. This is detailed in Section 3.4.

Algorithms 1 and 2 provide a pseudocode for the modified expansion and simulation steps. For clarity, operations on the successor states are performed in a *for* loop, but in practice, they are performed in a batched manner for efficiency purposes.

To simplify the notations in the following, we note $\bar{\mathbf{S}}_{\mathcal{P}}$ the predicted robot state associated with the search path $\mathcal{P}$. $\mathcal{C}_{\mathbf{s}}^{B_{\mathcal{P}}}$ corresponds to the contact locations associated with states $\mathbf{s}$ in the predicted base frame $B_{\mathcal{P}}$. For a transition $\mathbf{s} \rightarrow \mathbf{s}'$, we note $\mathtt{c}(\mathbf{s}, \mathbf{s}')$ the classifier taking as inputs the predicted state $\bar{\mathbf{S}}_{\mathcal{P}}$, the current contact locations $\mathcal{C}_{\mathbf{s}}^{B_{\mathcal{P}}}$ and the target contact locations $\mathcal{C}_{\mathbf{s}'}^{B_{\mathcal{P}}}$. We use a similar notation for $\mathtt{m}(\mathbf{s}, \mathbf{s}')$ and $\mathtt{t}(\mathbf{s}, \mathbf{s}')$.

**Algorithm 1** Pseudocode for the modified **expansion** step in MCTS

**Input:** Tree $\mathcal{T}$, current search path $\mathcal{P}$, selected node $\mathbf{s}$, state predictor network $\mathtt{m}$, feasibility classifier $\mathtt{c}$, threshold $t_{feasible}$, maximum transition length $d_{max}$
**Output:** Feasible successor states added to the tree
$\mathbf{S} \leftarrow$ predict_robot_state$(\mathcal{P}, \mathtt{m})$
**foreach** $\mathbf{s}' \in$ *Sucessors*$(\mathcal{T}, \mathbf{s})$ **do**
  `// kinematic pruning`
  **if** *crossing_leg*$(\mathbf{s}')$ *or transition_length*$(\mathbf{s}, \mathbf{s}') > d_{max}$ **then**
    Discard $\mathbf{s}'$; **continue**
  `// dynamic pruning`
  **if** *not dyn_feasible*$(\mathtt{c}, \mathbf{s}, \mathbf{s}', \mathbf{S}, t_{feasible})$ **then**
    Discard $\mathbf{s}'$; **continue**
  `// Add s' to the tree as a child node`
  add_successor$(\mathcal{T}, \mathbf{s}, \mathbf{s}')$
**end**

### 3.1. Dataset collection

To train the networks, we perform an offline data collection process with NMPC performing locomotion on flat terrain. The NMPC tries to follow a contact plan with randomized contact locations with a predefined gait. Before each gait cycle, we record:

- the state of the robot base $\mathbf{S}_i$ before the start of the cycle (noted $B_i$)
- the current end-effector positions $\mathcal{E}_i^{B_i}$
- the next target contact locations $\mathcal{E}^{*B_i}_{i+1}$

A transition is considered successful if the robot is not in collision and if the position error between the targeted and achieved contact locations is less than a threshold $e_{max} = 8cm$. In that case, we assign a label $\mathbf{y}_i = 1$ to the transition, otherwise $\mathbf{y}_i = 0$. Note that the joint velocities are not considered in the robot state $\mathbf{S}$. Considering them did not improve the performances of the classifier, as shown in Section 4.1.

**Algorithm 2** Pseudocode for the modified **simulation** step in MCTS

**Input:** Tree $\mathcal{T}$, search path $\mathcal{P}$, selected state $\mathbf{s}$, heuristic function $h$, target adjustment network $\mathtt{t}$
**Output:** Reward $r$

$\mathcal{P}_{sim} \leftarrow \mathcal{P}$
**for** $N_{sim}$ *steps* **do**
  // Choose next state greedily and store maximum heuristic value
  $\mathbf{s} \leftarrow \arg\max_{\mathbf{s}' \in \text{Successors}(\mathcal{T},\mathbf{s})} h(\mathbf{s},\mathbf{s}')$ ; $h^* \leftarrow \max_{\mathbf{s}' \in \text{Successors}(\mathcal{T},\mathbf{s})} h(\mathbf{s},\mathbf{s}')$
  // Add $\mathbf{s}$ to the simulation path
  $\mathcal{P}_{sim} \leftarrow [\mathcal{P}_{sim}, \mathbf{s}]$
  **if** $\mathbf{s} = \mathbf{s}_g$ **then**
    **break**
**end**
$r \leftarrow h^*$
**if** $\mathbf{s} = \mathbf{s}_g$ **then**
  // Roll-out the NMPC in simulation over the whole contact sequence.
  // Adjust the target contact locations at runtime with $\mathtt{t}$
  $sucess = \text{dynamic\_sim}(\text{NMPC}, \mathcal{P}_{sim}, \mathtt{t})$
  **if** *not*$(sucess)$ **then**
    $r \leftarrow -h^*$
**return** $r$

### 3.2. Feasibility classifier and state predictor

The classifier $\mathtt{c}$ is trained with supervised learning on samples $([\bar{\mathbf{S}}_i, \mathcal{E}_i^{B_i}, \mathcal{E}^{*B_i}_{i+1}], \mathbf{y}_i)$. The network outputs the logits probability (or score) to succeed the transition.

The state predictor $\mathtt{m}$ is trained with supervised learning on samples $([\bar{\mathbf{S}}_i, \mathcal{E}_i^{B_i}, \mathcal{E}_{i+1}^{B_i}], \bar{\mathbf{S}}_{i+1})$, the output being here the state after transition $i$. The state predictor is only trained on transitions that succeeded ($\mathbf{y}_i = 1$). Note that in this case, the two set of contact locations given in input correspond to the *achieved* contact locations. For the classifier, the target contact locations are given in input, as the contact state may not be achieved in cases of failure (see Fig. 2).

During the search, for a transition $\mathbf{s} \rightarrow \mathbf{s}'$, we provide as input to the networks $[\bar{\mathbf{S}}_{\mathcal{P}}, \mathcal{C}_{\mathbf{s}}^{B_{\mathcal{P}}}, \mathcal{C}_{\mathbf{s}'}^{B_{\mathcal{P}}}]$. This requires to estimate the absolute base position $\mathbf{p}_{B_{\mathcal{P}}}^W$ in order to express the stepping stones locations $\mathcal{C}^W$ in base frame. However, the absolute position of the base is not predicted by the network, as all position inputs are expressed in the local frame. We propose to estimate the translation induced by transition $i$, noted $\mathbf{t}_{B_{i+1}}^{B_i}$, as one can compute the absolute base position with $\mathbf{p}_{B_{i+1}}^W = \mathbf{p}_{B_i}^W + \mathbf{R}_{B_i}^W \mathbf{t}_{B_{i+1}}^{B_i}$. $\mathbf{R}_{B_i}^W$ is the orientation of the base as a rotation matrix and is predicted by the network. To do so, we first compute the end-effector positions in the predicted joint configuration with forward kinematics, noted $\tilde{\mathcal{E}}_i^{B_i}$. Then, we compute the translation that moves the base so that the end-effectors are as close as possible to the target contact locations in the current configuration i.e. the translation minimizes the average distance between $\tilde{\mathcal{E}}_i^{B_i}$, and the target locations $\mathcal{E}^{*B_i}_{i+1}$:

$$\mathbf{t}_{B_{i+1}}^{B_i} = \frac{1}{N_e} \sum_{j=1}^{N_e} (\mathcal{E}^{*B_i}_{i+1}[j] - \tilde{\mathcal{E}}_i^{B_i}[j]) \tag{1}$$

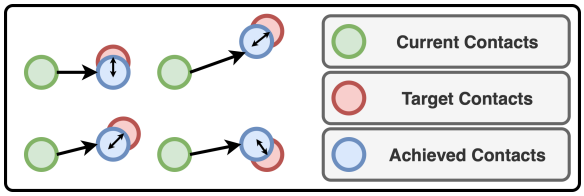



Figure 2: Illustration of current, target and achieved contacts for a jump with a quadruped robot. The NMPC doesn't precisely achieve target position. The residual is defined as the difference between target and achieved contacts.

### 3.3. Compensating for the low-level controller inaccuracies

The NMPC controller is not perfectly accurate in terms of reaching exactly the targeted contact locations. This is illustrated on Fig. 2. While this accuracy is not crucial for locomotion on flat terrain, it becomes critical to navigate on highly sparse terrains such as stepping stones. To tackle this, we trained a *target adjustment network* $\mathtt{t}$ to compensate for the inaccuracies of the low-level controller by adjusting the targeted contact positions. The network is trained on samples $([\bar{\mathbf{S}}_i, \mathcal{E}_i^{B_i}, \mathcal{E}_{i+1}^{B_i}], \mathcal{E}_{i+1}^{B_i} - \mathcal{E}^{*B_i}_{i+1})$. The output of the network is the residual to add to the target $\mathcal{E}^{*B_i}_{i+1}$ (input to the NMPC) in order to reach the achieved location $\mathcal{E}_{i+1}^{B_i}$. Indeed, if now the target is $\mathcal{E}_{i+1}^{B_i}$, the controller will reach exactly those locations by giving $\mathcal{E}^{*B_i}_{i+1}$ as input.

At runtime, we call the network before each gait cycle, add the residuals to the target contact locations, and provide the result to the NMPC. Note that the state predictor network and the target adjustment network have similar inputs. In practice, we use one network that outputs both the predicted state and residuals. We refer to them as one network in the result section.

### 3.4. Heuristics and reward

The proposed heuristic is composed of three terms. The first one guides the search in the direction of the goal as introduced by Dhédin et al. (2024), denoted by $h_{goal} \in [0, 1]$. The second one accounts for the chance of succeeding the desired gait, specified as $h_{safety} \in [0, 1]$. The third one accounts for the amount of correction of the low-level controller, i.e. $h_{accuracy} \in [0, 1]$. The final heuristic is defined as follows, $h = h_{goal} + \alpha h_{safety} + \beta h_{accuracy}$ with $\alpha, \beta \in \mathbb{R}$. Since $h_{goal}$ is higher for states closer to the goal, we also chose to use $h$ as our reward function. This is detailed in Algorithm 2. The heuristics $h_{goal}$, $h_{safety}$ and $h_{accuracy}$ are defined as follows

$$h_{goal}(\mathbf{s}, \mathbf{a}) = \sigma_g\left(\frac{1}{N_e}\sum_{j=1}^{n_e}\left(1 - \frac{||\mathcal{C}_{\mathbf{s}}^W[j] - \mathcal{C}_{\mathbf{s}_g}^W[j]||_2}{d_{max}}\right)\right) \tag{2}$$

where $d_{max}$ is the maximum distance between a pair of contact locations in the map,

$$h_{safety}(\mathbf{s}, \mathbf{s}') = \sigma_s(\mathtt{c}(\mathbf{s}, \mathbf{s}')) \tag{3}$$

where $\mathtt{c}(\mathbf{s}, \mathbf{s}')$ is the classification score, as we assume the higher score implies the higher confidence of the classifier,

$$h_{accuracy}(\mathbf{s}, \mathbf{s}') = \sigma_a\left(\frac{1}{\Delta_{res}(\mathtt{t}(\mathbf{s}, \mathbf{s}')) + 10^{-12}}\right) \tag{4}$$

where $\Delta_{res}(\mathtt{t}(\mathbf{s}, \mathbf{s}'))$ is the sum of the norm of the residual values predicted by the target adjustment network. Higher heuristics values correspond to transitions that requires less adjustment and should

more likely be feasible. $\sigma_g$, $\sigma_s$, $\sigma_a$ are sigmoid-based functions (noted $\sigma$) with linear scaling to map heuristics values in $[0, 1]$. They are defined such as: $\sigma_g(x) = \sigma(5x)$, $\sigma_s(x) = \sigma(x/5)$, $\sigma_a(x) = \sigma(x/5)$.

## 4. Results

In this section we present results of the proposed framework applied to a Go2 quadruped robot. We use MuJoCo simulator Todorov et al. (2012) and control the robot in a stepping stone environment with varying stone sizes. We provide results for two different gaits, trotting and jumping. We first detail the training procedures of the different networks in Section 4.1 (the procedure is the same for both gaits). We then describe the test stepping stones environment in Section 4.2. We evaluate how dynamic pruning and correcting the inaccuracies of the low-level controller affect the performance in Sections 4.2 and 4.2. We compare our method to a state-of-the-art RL approach in Section 4.3. Finally, we show in Section 4.4 that the performance can be further improved using the proposed safety and accuracy heuristics.

### 4.1. Training the classifier and state predictor networks

Both networks are Mutli-Layer Perceptron (MLP) trained on datasets recorded from the same transitions as described in Section 3.1. The training and validation sets contain respectively 130000 and 13000 samples (different datasets for jumping and trotting with the same size). We augment the data during training by adding small Gaussian noise with a standard deviation of $\sigma^2_{aug}$ to the input, which improves the training performance. We trained the networks using the Adam optimizer with an initial learning rate of $0.001$ and an exponential learning scheduler with a rate of $0.98$. Training parameters, network architectures and performances are detailed in table 1.

Table 1: Training parameters and performance metrics for different MLP networks. BCE: Binary Cross-Entropy. MSE: Mean Squared Error.

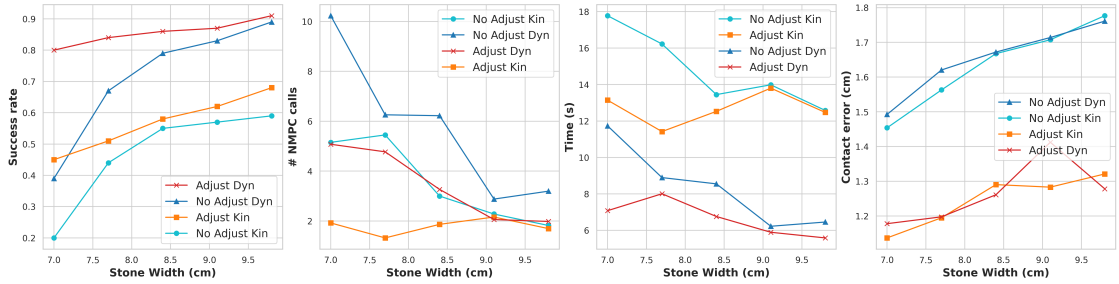
| **Network** | **N. Layers** | **H. dim.** | **Act.** | **Batch Size** | $\sigma^2_{aug}$ | **Loss** | **ROC AUC** | **Accuracy** |
|---|---|---|---|---|---|---|---|---|
| `c` (Jump) | 4 | 64 | ReLU | 512 | 1e-4 | BCE | 0.93 | 0.82 |
| `c` (Trot) | 4 | 64 | ReLU | 512 | 1e-4 | BCE | 0.95 | 0.85 |
| `m/t` | 3 | 128 | ReLU | 512 | 1e-4 | MSE | - | - |

### 4.2. Experimental Setup

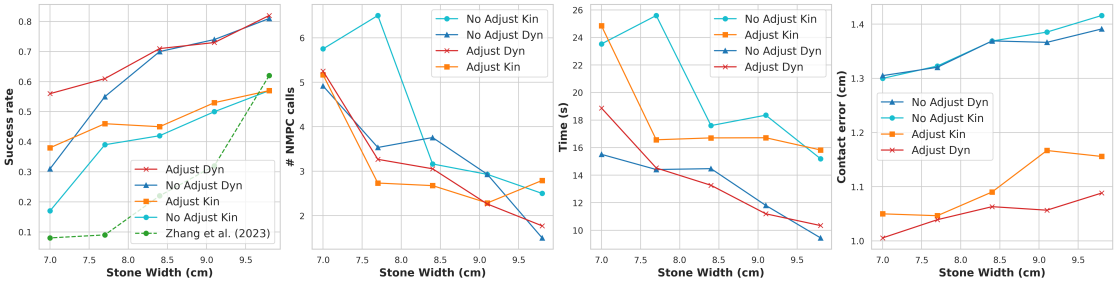
To evaluate the performance of our contact planner, we considered an environment with up to $N_{stones} = 35$ squared stones placed on a $7 \times 5$ grid. The stepping stones initially form a regular grid of spacing $(e_x, e_y)$ so that the feet lay on $4$ stepping stones in the initial configuration. The position of each stone is then displaced by $\epsilon_x(\frac{e_x}{2} - r)$ with $\epsilon_x \sim \mathcal{U}(-0.75, 0.75)$ in the $x$ direction (respectively for the direction $y$). $N_{removed} = 9$ stepping stones are randomly removed. Additionally, a small random noise $\epsilon_h \sim \mathcal{U}(-0.02, 0.02)$ is added to the height of the stones.

We ran MCTS for a maximum of $10000$ iterations and stop the search when the first dynamically feasible contact plan is found (referred to as *success* later). We consider different metrics to evaluate the performance: the success rate, the number of NMPC simulations required to find a dynamically feasible solution (noted #NMPC calls), search time, and the position error between achieved and

targeted contact locations (noted contact error). In the tested scenarios, the contact sequences typically consist of 3–6 transitions, which are compatible with execution on real quadrupeds using receding-horizon control strategies. We consider in the experiments randomized environment with different stepping stone sizes (side length ranging from 7 cm to 9.7 cm). Statistics are averaged over 100 randomly selected environments for each stepping stone size. Results are shown in Fig. 3. In the following we discuss the results.




(*a*) Jumping Gait




(*b*) Trotting Gait

Figure 3: Performance of the search for two different gaits. *adjust*: experiments with target adjustment. *kin*: experiments with kinematic pruning only. *dyn*: experiments with kinematic and dynamic pruning. Note that we use the baseline heuristic for those experiments ($\alpha = 0$, $\beta = 0$). Results of Zhang et al. (2023) are added for comparison in trotting gate.

**MCTS Contact planner with target adjustment.** For both gaits, adjusting the target contact locations improves the success rate. This improvement is more significant for stepping stones with a small radius. The success rate almost doubles on the smallest stones (with dynamic pruning). With the target adjustment network, the low-level controller is about 0.3 cm more accurate on average on reaching the targeted contact locations.

**MCTS Contact Planner with Dynamic Pruning.** Dynamic pruning results in a more efficient search process. As can be seen in Fig. 3, it reduces the number of NMPC simulations to perform compared to kinematic pruning only (on average 4.1 against 5.8 for jumping). It reduces the time of the search by more than 50%, as time-consuming NMPC simulations are avoided and fewer nodes are added to the tree. Although kinematic checks are fast, they often consider many transitions feasible that are not dynamically possible. Therefore, a significant portion of search time is wasted on expanding and selecting nodes that do not yield dynamically viable solutions. Indeed, we observed that the number of MCTS iterations is about 10 times lower with dynamic pruning. Combined with target adjustment, dynamic pruning achieves the best success rate for both jumping and trotting gait.

### 4.3. Comparison with state-of-the-art RL approach

We replicate the work of Zhang et al. (2023) to train a quadruped robot to traverse a stepping stones environment using reinforcement learning. Since the original code is not publicly available, we validated our implementation by reproducing comparable results with Anymal-D quadruped in Isaac Lab Mittal et al. (2023). We then adapted the environment and learning procedure to the Unitree Go2 robot. The training follows a two-stage process: first, the agent is pre-trained in a "Stones Everywhere" setup, followed by fine-tuning to configurations similar to our experimental setup described in 4.2. We reduced the minimum stepping stone size to 12 cm. The agent was not able to learn for smaller stepping stones following the method of Zhang et al. (2023). As shown in the Fig. 3, the policy is not able to traverse stepping stones smaller than the one seen in training. On the smallest stones, our method has a success rate about 5 times higher than the RL policy from Zhang et al. (2023), suggesting its advantage in scenarios requiring high motion precision and dynamic feasibility where RL policies struggle.

### 4.4. MCTS Contact planner with safety and accuracy heuristics

We performed a grid search to find $\alpha$ and $\beta$ hyperparameters values that would improve the search performance (according to the metrics introduced before). Results on environments with the smallest stepping stone size are summarized in the Table 2. Chosen $(\alpha, \beta)$ values improve the search efficiency while keeping a similar success rate. This can be explained as the pruning strategy remains the same, and therefore, the tree search explores similar paths but in a different order. With the new heuristic, for the jumping gait, the number of NMPC calls is halved and the search time is divided by 3. Improvements were less significant for the trotting gait.

Table 2: Performance comparison with and without the proposed heuristic.

| Algorithm | Success Rate | #NMPC Calls | Time (s) | Iterations | Contacts Error (cm) |
|---|---|---|---|---|---|
| **Jumping** | | | | | |
| $\alpha = \beta = 0$ | 80% | 5.27 | 12.05 | 270.4 | 1.25 |
| $\alpha = 0.2, \beta = 0.4$ | 77% | 2.54 | 4.61 | 107.7 | 1.14 |
| **Trotting** | | | | | |
| $\alpha = \beta = 0$ | 56% | 3.6 | 14.5 | 360.5 | 1.19 |
| $\alpha = 0.4, \beta = 0.2$ | 52% | 3.11 | 13.9 | 513.4 | 1.16 |

## 5. Conclusions and future works

This paper proposes a method to improve contact planning for legged robots in sparse environments. By combining model-based and learning-based approaches, the method efficiently explores the search space for feasible contact plans. Offline training is used to prune dynamically infeasible transitions and correct low-level controller inaccuracies at runtime.

Experiments on randomized stepping stones showed faster search times and higher success rates than the baseline. The framework's precision enabled traversal of very small stepping stones, outperforming RL-based and MPC-based methods.

Future work includes implementing the framework on real robots, planning gait sequences for better performance, and extending it to loco-manipulation problems.

## References

Bernardo Aceituno-Cabezas, Carlos Mastalli, Hongkai Dai, Michele Focchi, Andreea Radulescu, Darwin G Caldwell, José Cappelletto, Juan C Grieco, Gerardo Fernández-López, and Claudio Semini. Simultaneous contact, gait, and motion planning for robust multilegged locomotion via mixed-integer convex optimization. *IEEE Robotics and Automation Letters*, 3(3):2531–2538, 2017.

Lorznzo Amatucci, Joon-Ha Kim, Jemin Hwangbo, and Hae-Won Park. Monte carlo tree search gait planner for non-gaited legged system control. In *2022 International Conference on Robotics and Automation (ICRA)*, pages 4701–4707. IEEE, 2022.

Miroslav Bogdanovic, Majid Khadiv, and Ludo Righetti. Model-free reinforcement learning for robust locomotion using demonstrations from trajectory optimization. *Frontiers in Robotics and AI*, 9, 2022.

Angelo Bratta, Avadesh Meduri, Michele Focchi, Ludovic Righetti, and Claudio Semini. Contact-net: Online multi-contact planning for acyclic legged robot locomotion. In *2024 21st International Conference on Ubiquitous Robots (UR)*, pages 747–754, 2024. doi: 10.1109/UR61395.2024.10597477.

Robin Deits and Russ Tedrake. Footstep planning on uneven terrain with mixed-integer convex optimization. In *2014 IEEE-RAS international conference on humanoid robots*, pages 279–286. IEEE, 2014.

Robin Deits and Russ Tedrake. Computing large convex regions of obstacle-free space through semidefinite programming. In *Algorithmic Foundations of Robotics XI: Selected Contributions of the Eleventh International Workshop on the Algorithmic Foundations of Robotics*, pages 109–124. Springer, 2015.

Victor Dhédin, Adithya Kumar Chinnakkonda Ravi, Armand Jordana, Huaijiang Zhu, Avadesh Meduri, Ludovic Righetti, Bernhard Schölkopf, and Majid Khadiv. Diffusion-based learning of contact plans for agile locomotion, 2024. URL https://arxiv.org/abs/2403.03639.

Ruben Grandia, Fabian Jenelten, Shaohui Yang, Farbod Farshidian, and Marco Hutter. Perceptive locomotion through nonlinear model predictive control. *IEEE Transactions on Robotics*, 2023.

Fabian Jenelten, Junzhe He, Farbod Farshidian, and Marco Hutter. Dtc: Deep tracking control. *Science Robotics*, 9(86):eadh5401, 2024.

Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

Yu-Chi Lin, Ludovic Righetti, and Dmitry Berenson. Robust humanoid contact planning with learned zero- and one-step capturability prediction, 2020. URL https://arxiv.org/abs/1909.09233.

Avadesh Meduri, Majid Khadiv, and Ludovic Righetti. Deepq stepper: A framework for reactive dynamic walking on uneven terrain. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2099–2105. IEEE, 2021.

Avadesh Meduri, Paarth Shah, Julian Viereck, Majid Khadiv, Ioannis Havoutis, and Ludovic Righetti. Biconmp: A nonlinear model predictive control framework for whole body motion planning. *IEEE Transactions on Robotics*, 2023.

Mayank Mittal, Calvin Yu, Qinxi Yu, Jingzhou Liu, Nikita Rudin, David Hoeller, Jia Lin Yuan, Ritvik Singh, Yunrong Guo, Hammad Mazhar, Ajay Mandlekar, Buck Babich, Gavriel State, Marco Hutter, and Animesh Garg. Orbit: A unified simulation framework for interactive robot learning environments. *IEEE Robotics and Automation Letters*, 8(6):3740–3747, 2023. doi: 10.1109/LRA.2023.3270034.

Igor Mordatch, Emanuel Todorov, and Zoran Popović. Discovery of complex behaviors through contact-invariant optimization. *ACM Transactions on Graphics (ToG)*, 31(4):1–8, 2012.

Brahayam Ponton, Majid Khadiv, Avadesh Meduri, and Ludovic Righetti. Efficient multicontact pattern generation with sequential convex approximations of the centroidal dynamics. *IEEE Transactions on Robotics*, 37(5):1661–1679, 2021.

Michael Posa, Cecilia Cantu, and Russ Tedrake. A direct method for trajectory optimization of rigid bodies through contact. *The International Journal of Robotics Research*, 33(1):69–81, 2014.

Yuval Tassa, Tom Erez, and Emanuel Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4906–4913. IEEE, 2012.

Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033, 2012. doi: 10.1109/IROS.2012.6386109.

Steve Tonneau, Daeun Song, Pierre Fernbach, Nicolas Mansard, Michel Taix, and Andrea Del Prete. Sl1m: Sparse l1-norm minimization for contact planning on uneven terrain. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6604–6610. IEEE, 2020.

Alexander W Winkler, C Dario Bellicoso, Marco Hutter, and Jonas Buchli. Gait and trajectory optimization for legged systems through phase-based end-effector parameterization. *IEEE Robotics and Automation Letters*, 3(3):1560–1567, 2018.

Chong Zhang, Nikita Rudin, David Hoeller, and Marco Hutter. Learning agile locomotion on risky terrains. *arXiv preprint arXiv:2311.10484*, 2023.

Huaijiang Zhu, Avadesh Meduri, and Ludovic Righetti. Efficient object manipulation planning with monte carlo tree search. In *2023 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2023.