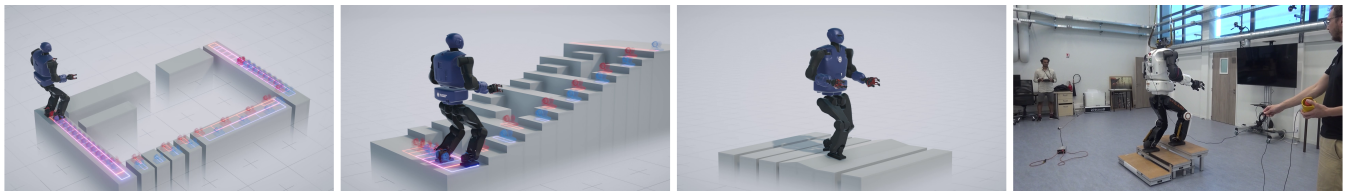
NAS: N-step computation of All Solutions to the footstep planning problem

Jiayi Wang[1*], Saeid Samadi[1*], Hefan Wang[1*], Pierre Fernbach[2], Olivier Stasse[3,4], Sethu Vijayakumar[1] and Steve Tonneau[1†]



***Abstract*— How many ways are there to climb a staircase in a given number of steps? Infinitely many, if we focus on the continuous aspect of the problem. A finite, possibly large number if we consider the discrete aspect, *i.e.* on which surface which effectors are going to step and in what order. We introduce NAS, an algorithm that considers both aspects simultaneously and computes *all* the possible solutions to such a contact planning problem, under standard assumptions. To our knowledge NAS is the first algorithm to produce a globally optimal policy, efficiently queried in real time for planning the next footsteps of a humanoid robot.**

Our empirical results (in simulation and on the Talos platform) demonstrate that, despite the theoretical exponential complexity, optimisations reduce the practical complexity of NAS to a manageable bilinear form, maintaining completeness guarantees and enabling efficient GPU parallelisation. NAS is demonstrated in a variety of scenarios for the Talos robot, both in simulation and on the hardware platform. Future work will focus on further reducing computation times and extending the algorithm's applicability beyond gaited locomotion. Our video is available at `https://youtu.be/I5yFe0ez0sI`

I. Introduction

In legged robotics, contact planning involves determining where an effector, such as a foot, should make contact with the environment to move. This process typically calculates a sequence of contacts to solve a particular locomotion problem instance, aiming for a **single** optimal solution. If this solution becomes unfeasible due to changes in the scene or drifts in state estimation during execution, a new plan must be computed. This paper aims at avoiding this costly re-computation by calculating **all possible solutions for a given instance**, specifically addressing bipedal locomotion.

Contact planning is crucial for any framework designed to generate legged motions, whether the goal is gaited locomotion (like footstep planning) or more complex loco-manipulation, which involves using all end-effectors in an arbitrary order. The challenge lies in handling non-linear geometric constraints (such as joint limits and collisions) and dynamic constraints, making it a high-dimensional, non-linear combinatorial problem. While efficient heuristics exist for addressing simplified cases such as locomotion on flat ground [1], the problem becomes more complicated when:

- possible contact surfaces are not continuously connected (e.g., the steps of a staircase or stepping stones);
- Several steps need to be planned ahead to avoid dead-ends, causing an exponential increase in combinations.

Sampling-based approaches have been proposed to tackle the problem [2], as well as other graph-search techniques such as A^* or Mixed-Integer Programming [3], [4]. However, it is often required to re-plan contact plans online (at a 10Hz frequency [5], [6], [7]) to account for execution and state estimation errors, especially in dynamic environments. The worst-case exponential complexity of these approaches then requires to significantly reduce the planning horizon n, that is the number of future steps that can be planned in advance. This is problematic in environments containing dead-ends as a short planning horizon does not allow to escape them.

Deep Reinforcement Learning (DRL) has shown promise in overcoming the need for simplifying assumptions, allowing robots to traverse challenging terrains robustly. However, so far DRL methods do not consider a long planning horizon. Preliminary research suggests that this horizon could be learned if combined with optimal control, but the generalisation is not trivial as these methods are supervised [8], [9].

We propose to explicitly characterise all feasible solutions for a contact planning problem instance, up to a number n of steps. This is achieved by observing that, under common assumptions, the set of feasible contact positions reachable in at-most n steps can be computed recursively through the computation of Minkowski sums and polytope intersections similarly to the idea of backward reachability analysis in control theory [10], [11]. This effectively results in an optimal policy for computing minimum-step contact sequences given any state of the robot. NAS can also be implemented as a

*Joint first authors. †email: `stonneau@ed.ac.uk`
[1]University of Edinburgh, School of Informatics, UK
[2]TOWARD S.A.S., Toulouse, France
[3]LAAS-CNRS, Université de Toulouse, Toulouse, 31400, France
[4]Artificial and Natural Intelligence Toulouse Institute, Toulouse, France
This work was supported by a Tata Consulting Services grant, Dynamograde (ANR-21-LCV3-0002), ROBO-TEX 2.0 (ROBOTEX ANR-10-EQPX-44-01 and TIRREX-ANR-21-ESRE-0015), the JST Moonshot R&D Grant No. JPMJMS2031, ANITI (ANR-19-P3IA-0004) and The Alan Turing Institute.

A^* for single-query planning, but we argue that outside of the theoretical interest, computing a globally optimal policy enables robust online replanning for disturbance adaption.

Our main contribution is NAS (N-step computation of All Solutions), an algorithm to compute the entire feasible space for a contact planning problem, resulting in a real-time optimal policy for footstep planning.

Although the theoretical complexity of the algorithm is exponential, we empirically show that it behaves as $O(m*n)$, with m the number of candidate contact surfaces and n the number of steps planned, thanks to the introduction of a node merging procedure that preserves the completeness of our approach. We also demonstrate the real-time capabilities of the framework to recompute a globally optimal plan.

In the context of this paper, our results are restricted to gaited locomotion and require to discretise the yaw orientation. Yet the method is not theoretically limited to these use cases. We aim at extending the approach as well as optimising the computation time of the algorithm since it is immediately compatible with GPU parallelisation.

II. State of the art

The contact planning problem is a special instance of the motion planning problem, where the objective is to find a collision-free path connecting two configurations of the robot [12]. Planning motions for a legged robot additionally involves planning the discrete change of contacts required to actuate the motion [13]. This problem is high-dimensional and subject to discrete combinatorics that make it hard to solve. The central question is to model the contact interactions in a way that reduces the combinatorics.

A. Combinatorics models for gaited locomotion

Focusing on bipedal gaited locomotion, Chestnutt et al. explicitly deal with the combinatorics by reducing the complexity of the problem through the discretisation of the action space [3], a path further explored recently [14]. They predetermine a set of actions that the robot could perform for a fixed position of each end-effector (expressed as target positions for the other end-effector) and use the A^* algorithm to plan an optimal path to the goal within this action space.

Deits et al. replaced the discrete action space with a continuous one, using the classic notion of reachable workspace to linearly constrain the end-effector positions relatively to each other, though the orientation of the robot remains discretised [4]. A^* is not immediately applicable with a continuous action space, so Mixed-Integer Programming (MIP) is used to compute a globally optimal solution. MIP has been extended to quadrupeds [15], [5] and recently the quasi-static constraint on the locomotion [16], [6] was overcome. To reduce the computational burden of MIP, we proposed a relaxation of the combinatorics using L1-norm minimisation [17], [18], which does not guarantee the convergence to a solution. Other relaxations have since been proposed [19].

Our approach leverages both A^* and MIP formulations. We use a dynamic programming approach as in A^*, but it is compatible with a continuous action space. Furthermore, we compute the entire feasible space and not a single solution.

B. Combinatorics models for multi-contact locomotion

Several contributions have also tackled the more general multi-contact problem, where no assumptions are made on the gait followed by the robot and all end-effectors (such as hands) are possibly involved in the contact creations. They are also graph-search methods and have been demonstrated in challenging scenarios, including climbing, chair egress, or tunnel crawling [20], [21], [22]. As for the gaited locomotion case, the reachable workspace has been used in this context to reduce the dimensionality of the problem and its combinatorics [2], [23]. Kumagai et al. [24] built on this and the notion of contact sustainability [25], [26] to propose an A^* algorithm that efficiently tackles the multi-contact planning problem which we consider to be the state of the art. While we also use dynamic programming, our objective is the computation of all solutions through recursive computation of all reachable states, as similarly done in backward reachability analysis [10], [11]. For robotics, reachability analysis has been used for motion planning under uncertainty [27], [28], [29], while we focus on the contact planning problem.

C. Towards combinatorics-free contact locomotion?

A variety of approaches have proposed to work around the discrete aspect of combinatorics and relax the contact planning problem into a continuous trajectory optimisation one [30], [31], [32], [33]. The potential advantage is clear, as the combinatorics is responsible for the exponential complexity of the problem. While these approaches do not guarantee the convergence to feasible solutions, recent work in manipulation suggests that smoothing allows the discovery of solutions to complex scenarios [34], [35], [36].

Deep Reinforcement Learning (DRL) techniques are attractive as they learn a policy efficiently queried online, and have successfully demonstrated their ability to tackle multi-contact locomotion without explicitly modelling the contact decisions [37], [38]. Yet the recent inclusion of model-based optimisation within the training framework has empirically demonstrated the interest in using optimal control with a dynamics model (including contacts) in terms of generalisation and robustness [39]. A better characterisation of the feasible space for the locomotion problem could alleviate this computational burden and enable the learning of a longer horizon as suggested by [9].

III. Definitions, notations and problem statement

A. Problem statement

In the present work, we focus on planning contact sequences for gaited bipedal locomotion tasks. A simple use

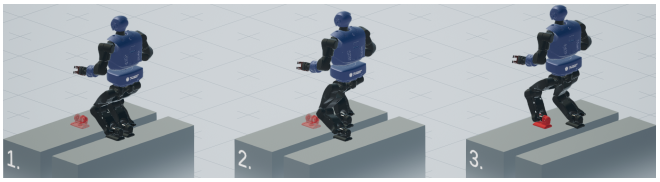


Fig. 1: A 2-step plan for the left foot to reach the red target.

case for our problem is illustrated in Fig. 1. The figure demonstrates one feasible contact plan that brings the robot from its starting configuration to a goal position, expressed as a terminal constraint on the position of the left foot. Our goal is to capture the infinitely many contact plans that solve this problem. We assume that the right and left foot alternate in creating contacts towards the goal. For now, we assume that the orientation of the feet around the axis z is fixed for the duration of the planning. The orientation can change along the x and y axes to fit to the contact surfaces.

To simplify the formulation, our equations assume that the left foot always acts as the end-effector reaching the target to complete the task. This assumption is solely for clarity in understanding the algorithm and does not limit our approach.

Given a target goal set $\mathcal{G}$, the environment, the kinematic constraints of the robot, and a maximum number of steps n, our objective is to compute the set of all possible contact sequences that bring the robot to $\mathcal{G}$ in at most n steps.

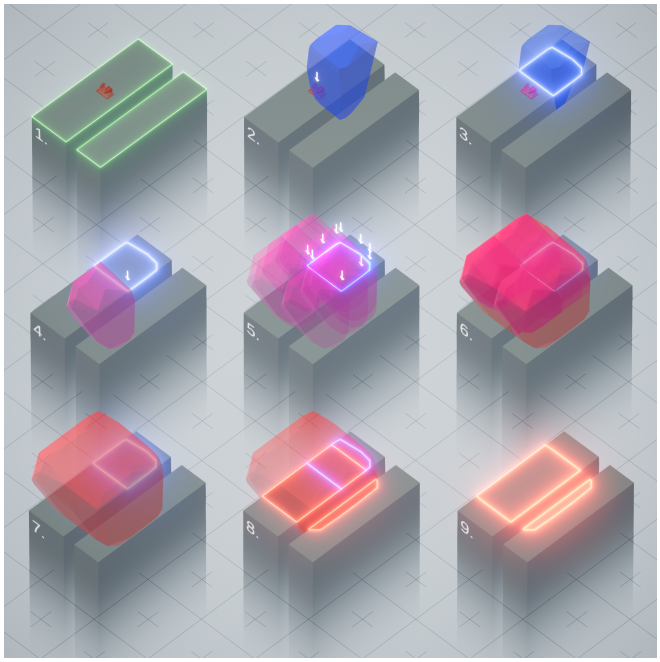


Fig. 2: Two-step feasibility computation. **1**. Red: The goal for this contact planning problem is to find a feasible contact sequence such that the left foot reaches the red position $\mathbf{p}_l$, thus we define $\mathcal{G} = \{\mathbf{p}_l\}$. Green: The environment $\mathcal{S}$ is the union of 2 convex polygons. **2**. All positions of the right foot such that $\mathbf{p}_l$ can be reached by the left foot in one step are bounded by the blue polytope ${}^{\mathcal{G}}\mathcal{R}$. It is obtained by translating the antecedent polytope ${}^{l}\mathcal{A}_r$ by $\mathbf{p}_l$. **3**. One-step feasible set for the right foot ${}^{\mathcal{G}}\mathcal{F}$. **4**. Reachability polytope ${}^{r}\mathcal{A}_l$ for the position of the right foot indicated by the arrow. **5**.${}^{r}\mathcal{A}_l$ translated by each extreme point of ${}^{\mathcal{G}}\mathcal{F}$. **6**. & **7**. Minkowski sum of ${}^{r}\mathcal{A}_l$ and ${}^{\mathcal{G}}\mathcal{F}$. **8**. & **9**. Two-step feasible set for the right foot, composed of 2 convex polygons.

B. Surfaces, reachable areas, and kinematic constraints

We use convex polytopes to describe the environment and the reachable workspace of the end-effectors of the robot. The set of feasible solutions to a contact planning problem is described as a union of polytopes. They are either 3D polyhedra (Fig. 2.2) or 2D polygons in a 3D space (Fig. 2.3). A polytope $\mathcal{P}$ is defined as the convex hull of its vertices:

$$\mathcal{P} := \{\mathbf{p} \in \mathbb{R}^3 | \exists \boldsymbol{\lambda} \in \mathbb{R}^{+d}, \|\boldsymbol{\lambda}\|_1 = 1 \wedge \mathbf{p} = \mathbf{P}\boldsymbol{\lambda}\}. \quad (1)$$

where $\mathbf{P} \in \mathbb{R}^{3\times d}$ is a matrix obtained by concatenating the d extreme vertices of $\mathcal{P}$ and $\boldsymbol{\lambda}$ is a unit weighting vector of the extreme points. In the following, we implicitly define the matrix $\mathbf{L}$ for any polytope $\mathcal{L}$ that we define.

The environment is represented as a union of m disjoint convex contact surfaces $\mathcal{S} = \bigcup_{j=1}^{m} \mathcal{S}^j$ (Fig. 2.1 - green).

The kinematic constraints of the robot are linearised as commonly done in graph-based approaches. We define ${}^{r}\mathcal{K}_l$ as the polytope describing the set of reachable positions for the left foot assuming the right foot is located at the origin. We similarly define ${}^{l}\mathcal{K}_r$ for the right foot.

The antecedent constraints also need to be defined because our algorithm works backward. We define the ${}^{l}\mathcal{A}_r$ (Fig. 2.3 - blue) as the set of right foot positions from which a left foot position can be placed at the origin of the world. ${}^{r}\mathcal{A}_l$ is similarly defined for the left positions such that a right foot position at the origin is reachable (Fig. 2.4 - red). ${}^{r}\mathcal{A}_l$ and ${}^{l}\mathcal{A}_r$ are obtained by applying a central symmetry to the vertices of ${}^{r}\mathcal{K}_l$ and ${}^{l}\mathcal{K}_r$.

The goal $\mathcal{G}$ is the polytope defining the task for a robot. The plan is successful when either end-effector reaches $\mathcal{G}$. $\mathcal{G}$ is a subset of a contact surface $\mathcal{S}^j$, and can be a degenerated polytope (i.e. a single point, as in Fig. 2.1 - red).

IV. Overview

We propose to compute the set of feasible contact sequences to $\mathcal{G}$ using a dynamic programming algorithm. Starting from $n = 0$, we recursively compute all feasible states from which we can reach $\mathcal{G}$ in n steps, until a termination condition is met, either if the current state of the robot is reached or a user-defined n is reached.

By definition, the feasible set $\mathcal{F}_0$ for $n = 0$ (*i.e.* the target is reached without making any steps) is $\mathcal{G}$ (Fig. 2.1). We then compute the reachable set $\mathcal{R}_1$ of all positions that can bring the end-effector to $\mathcal{G}$ in one step, for $n = 1$ (Fig. 2.2).

Most of the positions in $\mathcal{R}_1$ are not contact positions, so to compute the feasible set $\mathcal{F}_1$ we intersect $\mathcal{R}_1$ with $\mathcal{S}$, giving $\mathcal{F}_1 = \bigcup_{j=1}^{m} \mathcal{F}_1^j$, with $\mathcal{F}_1^j = \mathcal{R}_1 \cap \mathcal{S}^i$ (Fig. 2.3).

Each non-empty $\mathcal{F}_1^j$ describes a 2D polygon such that for any position of the right foot on the polygon, there exists a one-step sequence that results in the left foot inside $\mathcal{G}$. Each $\mathcal{F}_1^j$ corresponds to a node added as child to $\mathcal{F}_0$ in a tree $\mathcal{T}$.

We can then proceed similarly to compute the feasibility set $\mathcal{F}_2^j$ associated with each $\mathcal{F}_1^j$ (Fig. 2.4-9), and recursively compute the feasibility sets until we reach $\mathcal{F}_n$.

From any node of $\mathcal{T}$, the contact sequence to $\mathcal{G}$ is obtained by recursively selecting the parent node until $\mathcal{G}$ is reached.

At runtime, to compute a contact sequence from a given state of the robot, we use a k-d tree to efficiently search for the node corresponding to the state of the robot.

V. 1-STEP FEASIBILITY

Similarly to other Dynamic Programming algorithms, we reason from the goal $\mathcal{G}$. The *antecedent* states form the set of all positions of an end-effector from which $\mathcal{G}$ can be reached in exactly one step. We rely on Fig. 2. to illustrate the procedure. It describes the computation of the 2-step feasible set for the problem in Fig. 1 and contains all the possible cases that can occur when computing the *antecedent* states.

A. Reachability from a given position (Fig. 2.1-2)

For a given position $\mathbf{p}_e$ of an end-effector e on flat ground, the set of positions of the other effector $\overline{e}$ such that $\mathbf{p}_e$ is reachable in one step is by definition ${}^{e}\mathcal{A}_{\overline{e}}$ translated by $\mathbf{p}_e$:

$$
\begin{aligned}
{}^{\mathbf{p}_e}\mathcal{R} := \{\mathbf{p}_{\overline{e}} \in \mathbb{R}^3 | \exists \boldsymbol{\lambda} \in \mathbb{R}^{+d}, \\
||\boldsymbol{\lambda}|| = 1 \wedge \mathbf{p}_{\overline{e}} =^{e} \mathbf{A}_{\overline{e}}\boldsymbol{\lambda} + \mathbf{p_e} =^{\mathbf{p}_e} \mathbf{R}\boldsymbol{\lambda}\}\,.
\end{aligned}
$$

In the general case, if $\mathbf{Q}$ is the rotation matrix of minimum distance (in the log sense) that aligns the robot's root frame z axis with the contact surface normal, we write:

$$
\begin{aligned}
{}^{\mathbf{p}_e}\mathcal{R} := \{\mathbf{p}_{\overline{e}} \in \mathbb{R}^3 | \exists \boldsymbol{\lambda} \in \mathbb{R}^{+d}, \\
||\boldsymbol{\lambda}|| = 1 \wedge \mathbf{p}_{\overline{e}} = \mathbf{Q}^{e}\mathbf{A}_{\overline{e}}\boldsymbol{\lambda} + \mathbf{p_e} =^{\mathbf{p}_e} \mathbf{R}\boldsymbol{\lambda}\}\,.
\end{aligned}
$$

B. Reachability from a set of positions (Fig. 2.4-7)

More interestingly, we can compute the antecedent set of a convex polytope $\mathcal{R}_e$ of positions of e that share the same contact normal. This set ${}^{\mathcal{R}_e}\mathcal{R}$ is all the positions for $\overline{e}$ allowing to create a contact with e inside $\mathcal{R}_e$ in one step. It is thus the union of all antecedents at every point of $\mathcal{R}_e$:

$$
{}^{\mathcal{R}_e}\mathcal{R} := \{\mathbf{Q}^{e}\mathbf{A}_{\overline{e}}\boldsymbol{\lambda} + \mathbf{R}_e\boldsymbol{\lambda}_1,\ ||\boldsymbol{\lambda}|| = ||\boldsymbol{\lambda}_1|| = 1\}\,. \quad (2)
$$

With $\boldsymbol{\lambda}$ and $\boldsymbol{\lambda}_1$ positive vectors of appropriate dimensions. Eq(2) denotes the Minkowski sum of the two convex sets $\mathcal{R}_e$ and ${}^{e}\mathcal{A}_{\overline{e}}$ (the latter being rotated by $\mathbf{Q}$). Therefore, ${}^{\mathcal{R}_e}\mathcal{R}$ is convex since the Minkowski sum preserves convexity.

C. Computing the 1-step feasible set (Fig. 2.7-9)

${}^{\mathcal{R}_e}\mathcal{F}$ is the subset of ${}^{\mathcal{R}_e}\mathcal{R}$ that results in contact locations. ${}^{\mathcal{R}_e}\mathcal{F}$ is obtained by intersecting ${}^{\mathcal{R}_e}\mathcal{R}$ with the contact surfaces of the environment and, whenever a collision is detected, computing the resulting intersected surfaces. ${}^{\mathcal{R}_e}\mathcal{F}$ is thus composed of the union of all resulting surfaces:
${}^{\mathcal{R}_e}\mathcal{F} = \bigcup_{j=1}^{m} {}^{\mathcal{R}_e}\mathcal{F}^j$, with ${}^{\mathcal{R}_e}\mathcal{F}^j = {}^{\mathcal{R}_e}\mathcal{R} \cap \mathcal{S}^i$.

If the set of positions given as input is a convex polytope (as is the case when we start from $\mathcal{G}$), all resulting surfaces are convex polygons, since the intersection of 2 convex polytopes is a convex polytope. This means that at any point of the expansion, **the feasible set is always a union of convex polytopes** (possibly degenerated into a point).

VI. THE NAS ALGORITHM

To compute the N-Step feasible space, that is the set of all possible positions from which the robot can reach $\mathcal{G}$ in at most n steps, we recursively compute the feasible set for all the steps from 1 to n in order to populate a tree $\mathcal{T}$.

A. Tree description and initialisation

Each node of $\mathcal{T}$ contains information about the end-effector currently in contact, the surface in contact, as well as the subset of the surface covered by this node. It also contains a link to its parent in the $\mathcal{T}$ as shown in struct 1.

struct 1 NODE

$effectorId$: ENUM
$parent$: NODE* // parent node
$surfaceId$: INT
$extremePoints$: POINT LIST //polygon description

$\mathcal{T}$ is implemented as an array of Node lists. Nodes are indexed by their depth in the tree, which is the number of steps required to reach the target from the node.

B. The NAS algorithm

Algorithm 2 NAS

function N_STEPS_FEASIBILITY($\mathcal{T}$, n)
  if $n == 0$ **then return** $\mathcal{T}$ **// (Fig. 2.1)**
  for each leaf node $node$ in $\mathcal{T}$ **do**
    $feasibleNodes \leftarrow$ FEASIBLE_NODES($node$)
    ADD_LEAVES($\mathcal{T}$, $node$, $feasibleNodes$)
  return N_STEPS_FEASIBILITY($\mathcal{T}$, $n-1$)

function FEASIBLE_NODES($node$)
  $effId \leftarrow$ OTHER($node.effectorId$)
  $nodeLists \leftarrow []$
  $\mathcal{R} \leftarrow$ REACH_POLYTOPE($node$) // (Fig. 2.2)
  for each surface s in $\mathcal{S}$ **do**
    $feasible\_s \leftarrow$ INTERSECT($\mathcal{R}, s$) // (Fig. 2.3,8)
    if NOT_EMPTY($feasible\_s$) **then**
      $child \leftarrow$ NODE($effId, node, s.Id, feasible\_s$)
      $nodeLists.add(child)$
  return $nodeLists$

NAS consists in initialising $\mathcal{T}$ with a single node $\mathcal{G}$ before calling N_STEPS_FEASIBILITY (Algorithm 2). N_STEPS_FEASIBILITY is a recursive function that computes the expansion of $\mathcal{T}$ for 1 step before calling itself. Each call computes the 1-step feasible set associated with each leaf node (method FEASIBLE NODES), creates children nodes, and adds them to the leaf node (method ADD_LEAVES).

FEASIBLE_NODES first computes which end-effector is selected for the expansion (method OTHER). The method REACH_POLYTOPE computes the volume $\mathcal{R}$ from which the node can be reached as per eq.(2). $\mathcal{R}$ is then intersected with each potential contact surface from $\mathcal{S}$ (method INTERSECT). Each non-empty intersection results in a new node.

C. Discrete handling of the rotation

NAS can be extended to handle a rotation of the foot around the z axis, if we discretise the possible orientations [4]. The updated function FEASIBLE_NODES is given by Algorithm 3. ROTATE rotates $\mathcal{R}$ around the z axis. Additionally, the current rotation angle of the feet needs to be added to the node structure to know the end-effector orientation at the current state. This algorithm is presented here for completeness, but our experiments focus on Algorithm 2.

Algorithm 3 FEASIBLE_NODES with yaw orientation

function FEASIBLE_NODES($node$)
  $effId \leftarrow$ OTHER($node.effectorId$)
  $nodeLists \leftarrow []$
  $\mathcal{R} \leftarrow$ REACH_POLYTOPE($node$)
  for each discrete angle value θ **do**
    $\mathcal{R}_\theta \leftarrow$ ROTATE($\mathcal{R}, \theta$)
    for each surface s in $\mathcal{S}$ **do**
      $feasible\_s \leftarrow$ INTERSECT($\mathcal{R}_\theta, s$)
      if NOT_EMPTY($feasible\_s$) **then**
        $\gamma \leftarrow node.angle + \theta$
        $child \leftarrow$ NODE($effId, node, s.Id, feasible\_s, \gamma$)
        $nodeLists.add(child)$
  return $nodeLists$

D. Properties of $\mathcal{T}$

1) n-step completeness: For a given robot state, defined as the position $\mathbf{p}_e$ of the active end-effector on a contact surface, if there exists a contact sequence leading to the target in n steps, there is necessarily a node in $\mathcal{T}$ that contains $\mathbf{p}_e$[1]. Otherwise, there is no valid up-to-n-steps contact sequence.

2) Minimum step optimality: For a given $\mathbf{p}_e$, there can be more than one matching node. The nodes with the lowest depth all denote a sequence with a minimum number of steps.

E. Optimisation of the algorithm

1) Optimising the tree generation: Unsurprisingly, NAS has a theoretical exponential complexity in $O(b^n)$, with $b \leq m$ the branching factor (or average number of successors per node). This is aligned with the worst case A^* and mixed-integer complexities. However, this complexity can be reduced through the use of what we call node merging.

The expansion of several leaf nodes at step j can lead to new leaf nodes at step $j+1$ covering exactly the same surface. This can happen when all points reachable on a given surface have been covered by the expansion. It is possible to reduce the number of branches by merging such nodes into a single one with several parents, without changing the completeness nor the optimally properties of $\mathcal{T}$: for bidepal locomotion the position of the parent end-effector has no influence on the expansion of the node, and since we keep track of all parents no path is lost. This optimisation can also be applied if more than 2 end-effectors are involved but it requires considering all end-effectors when merging.

[1]We abusively refer to a node containing a point or a state to indicate that the polygon associated with the node contains the point of interest.

2) Optimising the tree exploitation using a kd-tree: The algorithms used for contact planning will require identifying which node(s) of $\mathcal{T}$ "contain(s)" a given point $\mathbf{p}_e$, i.e. nodes such that $\mathbf{p}_e$ is included in their associated polygon. Finding such a node involves iterating through all the nodes sorted by their depth in $\mathcal{T}$ until encountering one that contains $\mathbf{p}_e$ (it will be an optimal one). This process has a linear complexity. To improve efficiency, we store the nodes in a separate k-d tree [40] for each end-effector. This allows for an average search complexity of $\log(h)$, with h the total number of nodes. The construction complexity of the k-d tree, in $O(h\log(h)$, is dominated by the complexity of constructing $\mathcal{T}$. Additionally, the k-d tree can efficiently return all the nodes containing $\mathbf{p}_e$.

F. Applications for NAS

Most of the following use cases rely on the search of a specific node in $\mathcal{T}$. The resulting complexities indicated assume that the search is implemented using the k-d tree for a complexity in $O(\log(h))$, with h the number of nodes in $\mathcal{T}$, as detailed in Section (VI-E).

1) $\mathcal{T}$ as an optimal policy for contact planning: For a given state of the robot $\mathbf{p}_e$, we can find any node that contains $\mathbf{p}_e$ (and the associated $effectorId$), then go up the parent node chain to compute a contact sequence to the target. Assuming the depth of the node is $k \leq n$, this results in a feasible contact plan $\mathbf{F} = [\mathcal{F}_{k-1,e}, \cdots, \mathcal{F}_{1,e}, \mathcal{F}_{0,e} \subset \mathcal{G}]$, where $\mathcal{F}_{i,e}$ is the feasible convex polygon given by the node at depth i in the selected sequence. This process has an average complexity of $O(n + \log(h))$.

To find a contact sequence with a minimal number of steps, we can select the first node at the lowest depth which contains $\mathbf{p}_e$. Alternatively, if $\mathbf{p}_e$ is known at expansion time, the algorithm terminating condition can be modified to stop the expansion whenever a node containing $\mathbf{p}_e$ is generated (or until a maximum iteration is reached, meaning that the problem has no solution). This node will always result in a path that involves the minimum number of steps to the target.

2) Computing exact footstep locations: $\mathcal{T}$ is a policy for selecting contact surfaces, but does not directly tell us where exactly on the surface the stepping should occur. By construction of the graph, any point included by a node is optimal regarding the minimum number of steps. This allows us to apply any selection technique for a valid contact position, as long as we choose a point reachable from the previous location of the robot. The closest reachable point from the Chebyshev center of the node can be selected for robustness for instance. If a given objective has to be minimised for the footstep plan, a linearly constrained optimisation problem can be solved. The user is free to consider as many steps as needed in the optimisation horizon n_{hor}: no matter n_{hor}, the positions selected will always be feasible and lead to a minimal number of steps. From a current position $\mathbf{p}_0$ within a node at depth $k > 0$, an example of convex program is:

$$
\begin{aligned}
\textbf{find} \quad & \mathbf{X} = [\mathbf{p}_1, \cdots, \mathbf{p}_{n_{hor}}] \in \mathbb{R}^{3 \times n_{hor}} \\
\textbf{min} \quad & l(\mathbf{X}) \\
\textbf{s.t.} \quad & \forall i, 1 \leq i \leq n_{hor} : \\
& \mathbf{p}_i \in {}^{\mathbf{p}_{i-1}}\mathcal{R} \cap \mathcal{F}_{k-i,0}
\end{aligned} \tag{3}
$$

With l a convex objective, $1 \leq n_{hor} \leq k$ and ${}^{\mathbf{p}_i}\mathcal{R}$ the reachable workspace from the contact $\mathbf{p}_i$.

3) Online replanning using $\mathcal{T}$: At any point during the motion of the robot, the contact plan can be efficiently updated if the situation invalidates it. Whenever the robot makes a new step, we can check whether the contact location $\mathbf{p}_i$ belongs to the planned $\mathcal{F}_{i-1,e}$. If not (e.g. due to a perturbation of the hardware) we can immediately query $\mathcal{T}$ again and obtain an updated path from $\mathbf{p}_i$.

The formulation also allows to dynamically mark areas of the scene as impassible. To remove a contact surface from the feasible set, we mark all the nodes concerned by the surface as invalid. Upon re-planning, from the current state of the robot, we can iterate through the nodes found until we find a sequence that does not go through any invalid nodes. The average complexity for the search is lower than the search for the nodes containing the current position ($O(\log(h))$), plus at most v times the exploration of paths each of length of at most n, giving $O(n*v+\log(h))$, where v is the number of nodes that contain the current robot state.

4) Optimal trajectory optimisation: NAS can also be formulated as a single-query A^* algorithm with continuous nodes instead of discrete points, with an expansion that is not exhaustive but rather guided by a heuristic, for a likely better average complexity. The main advantage over a standard formulation is that the formulation is continuous and reduces the branching factor. We leave the comparison of both approaches for future work as the focus of this paper is on the characterisation of all solutions.

G. Implementation details

We implement the Minkowski sum of a polygon $\mathcal{S}$ and a polyhedron $\mathcal{P}$ as the convex hull of the polytopes obtained by translating $\mathcal{P}$ by each vertex of $\mathcal{S}$. The complexity of this operation is $O(k \log(k))$ where $k = k_{\mathcal{S}} * k_{\mathcal{P}}$ and $k_{\mathcal{S}}$ and $k_{\mathcal{P}}$ are the total number of vertices in $\mathcal{S}$ and $\mathcal{P}$. This operation remains efficient as $k_{\mathcal{S}}$ typically remains below 10. We compute the convex hulls using the SciPy [41]. Our code is implemented in Python, including collision detection and the k-d tree, with efficiency not being the primary concern.

To avoid computing plans that result in the foot partially out of a contact surface, we automatically scale down the surfaces given as input to NAS. This parametrisation is optional if the case is handled by the controller as in [42].

VII. EXPERIMENTS

We report quantitative information about the generation and run time of NAS that empirically demonstrate the validity of the approach on the Talos robot [43]. The experiments were run on an desktop computer equipped with an Intel i9 9900K CPU (3.6GHz) and 64GB RAM. The motions were synthesised using the PAL robotics controller. They are validated on the real robot or on the Gazebo simulator [44], synchronised in real-time with Unity 3D for rendering [45].

We designed 5 different scenes, 4 of which are shown in the teaser figure, with a number of surfaces varying from 4 to 43. One scene includes non-flat surfaces. Our video (`https://youtu.be/I5yFe0ez0sI`) demonstrates examples of minimum step motions computed with NAS, including an example of dynamic re-planning.

A. Tree generation analysis

The theoretical complexity of the graph generation is $O(b^n), b \leq m$. This is confirmed by Fig. 3, which presents the number of nodes in $\mathcal{T}$ when no optimisation is used in one representative use case. However, node merging optimisation results in a bilinear complexity $O(m * n)$ in all scenarios, as evidenced by Fig. 4. Even in unrealistic scenarios (such as planning 100 steps over a scene with more than 40 surfaces), the graph generation time remains below 5 minutes. This proves empirically the viability of NAS.

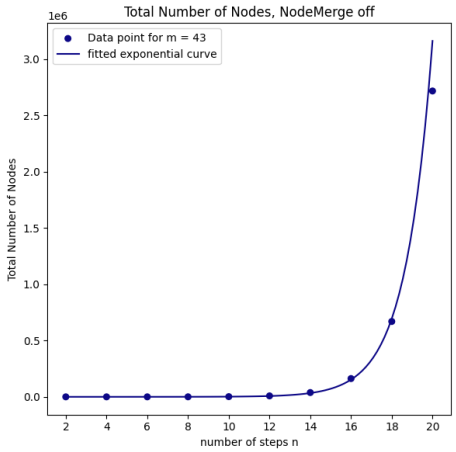


Fig. 3: Exponential growth of nodes without node merging.

B. Tree exploitation

For lack of space, we report briefly on the exploitation of the tree. The k-d tree query time is conditioned by the number of nodes, which in all the scenarios demonstrated remains below 10,000. In this context, the query time is below 24 ms in our scenarios and at worst 89 ms (for the scene with 43 surfaces), within our 10Hz requirement.

To compute the footstep sequences we solve eq(3) with the complete horizon (although we have established that this is not a requirement) and $l = 0$[2]. In all instances, the cumulated time to query the k-d tree and solve the QP remains below 100 ms. Here the resolution time only depends on n as the combinatorics is fixed: for instance, the 43 surfaces scene is solved in 6 ms ($n = 23$), while the re-planning scene is the longest to solve with 79 ms (and $n = 61$).

[2]Hence we only solve a feasibility problem.

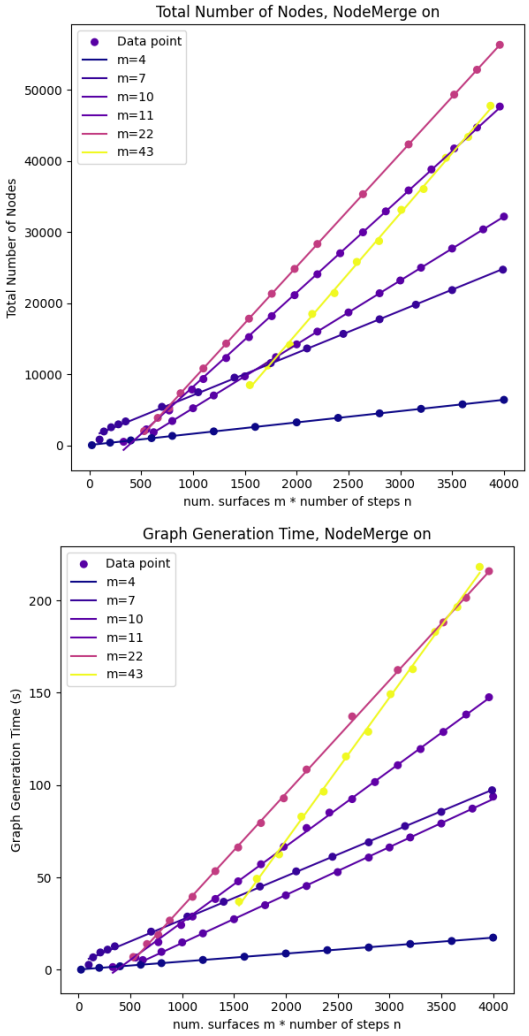


Fig. 4: In all instances, node merging allows the number of nodes and generation time for $\mathcal{T}$ to grow as $O(m * n)$.

VIII. Discussion

Our experiments show that NAS is applicable to typical planning problems for biped robots. The main benefit of the formulation is that it comes with strong guarantees, within the assumptions that are made when defining the problem.

1) Handling multiple goals: As a dynamic programming approach NAS needs to be computed from the goal for the feasibility to be guaranteed. Expanding $\mathcal{T}$ from the starting position would allow to tackle multiple goals, but online-replanning would not be possible past the first step.

2) Computing n: The number of steps n can be initially automatically computed, as we stop the expansion as soon as the current robot position is covered by $\mathcal{T}$. However, if re-planning involves that the new optimal motion requires more than n steps, a new expansion phase needs to be recomputed online until the new value of n (again, optimal) is found.

3) Handling continuous yaw orientation: Our approach (and the state of the art) currently only allows handling the yaw orientation of the end-effectors in a discretised manner. We are working on a continuous formulation that involves extending the reachability formulation to four dimensions.

4) Scaling the approach to non-gaited loco-manipulation: NAS can be directly extended to any legged robots. However, the size of $\mathcal{T}$ will grow significantly as the result of introducing additional discrete choice, in particular if locomotion is not gaited, although node merging remains possible. Future work will establish under what conditions this extension is viable, specifically if the computation of $\mathcal{T}$ can be optimised.

5) Parallelisation: As a breadth-first algorithm, NAS is parallelisable, which could significantly improve the computation times. Furthermore, the operations involved in the expansion are all compatible with a GPU implementation, which could further improve this efficiency.

6) Interest for machine learning: We argue that the characterisation of the complete feasible space, even under simplifying assumptions, presents two advantages for learning:

- The tailoring of the search space to the close neighborhood of the feasible set, to improve sample efficiency;
- The complete combinatorics can be explored and relevant information regarding optimality and whole-body feasibility can be fed back to the training network.

IX. Conclusion

In this work, we present NAS, a dynamic programming algorithm for computing the feasible space of a contact planning problem. NAS computes a globally optimal policy for a given problem, which allows for real-time planning (and re-planning) of a feasible contact plan. Thanks to the node merging procedure we introduce, the computation of the feasible space is performed with a bilinear complexity.

The reachability-based formulation of NAS also enables the implementation of a novel, continuous A^* algorithm to solve the problem once, with optimality guarantees equivalent to recent Mixed-Integer formulations.

NAS is parallelisable and could be considered for improving the efficiency of machine learning algorithms thanks to the tight characterisation of the search-space for the problem.

X. Acknowledgements

The authors would like to thank Rajesh Subburaman for his help on hardware experiments.

References

[1] S. Kajita, F. Kanehiro, K. Kaneko, K. Fujiwara, K. Harada, K. Yokoi, and H. Hirukawa, "Biped walking pattern generation by using preview control of zero-moment point," in *IEEE Int. Conf. Rob. Autom. (ICRA)*, vol. 2, 2003, pp. 1620–1626 vol.2.

[2] S. Tonneau, A. Del Prete, J. Pettré, C. Park, D. Manocha, and N. Mansard, "An efficient acyclic contact planner for multiped robots," *IEEE Transactions on Robotics*, vol. 34, no. 3, pp. 586–601, 2018.

[3] J. Chestnutt, M. Lau, G. Cheung, J. Kuffner, J. Hodgins, and T. Kanade, "Footstep planning for the honda asimo humanoid," in *IEEE Int. Conf. Rob. Autom. (ICRA)*, 2005, pp. 629–634.

[4] R. Deits and R. Tedrake, "Footstep planning on uneven terrain with mixed-integer convex optimization," in *2014 IEEE-RAS international conference on humanoid robots*. IEEE, 2014, pp. 279–286.

[5] F. Risbourg, T. Corbères, P.-A. Léziart, T. Flayols, N. Mansard, and S. Tonneau, "Real-time footstep planning and control of the solo quadruped robot in 3d environments," in *IEEE/RSJ Int. Conf. Intell. Rob. Sys. (IROS)*, 2022, pp. 12 950–12 956.

[6] T. Corbères, C. Mastalli, W. Merkt, I. Havoutis, M. Fallon, N. Mansard, T. Flayols, S. Vijayakumar, and S. Tonneau, "Perceptive locomotion through whole-body mpc and optimal region selection," 2024.

[7] F. Jenelten, R. Grandia, F. Farshidian, and M. Hutter, "TAMOLS: Terrain-aware motion optimization for legged systems," *IEEE Trans. Robot. (T-RO)*, vol. 38, 2022.

[8] J. Wang, S. Kim, T. S. Lembono, W. Du, J. Shim, S. Samadi, K. Wang, V. Ivan, S. Calinon, S. Vijayakumar, and S. Tonneau, "Online multicontact receding horizon planning via value function approximation," *IEEE Trans. Robot. (T-RO)*, vol. 40, pp. 2791–2810, 2024.

[9] A. K. C. Ravi, V. Dhédin, A. Jordana, H. Zhu, A. Meduri, L. Righetti, B. Schölkopf, and M. Khadiv, "Efficient search and learning for agile locomotion on stepping stones," 2024. [Online]. Available: https://arxiv.org/abs/2403.03639

[10] S. Bansal, M. Chen, S. Herbert, and C. J. Tomlin, "Hamilton-jacobi reachability: A brief overview and recent advances," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE, 2017, pp. 2242–2253.

[11] M. Althoff, G. Frehse, and A. Girard, "Set propagation techniques for reachability analysis," *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 4, no. 1, pp. 369–395, 2021.

[12] T. Lozano-Pérez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," *Commun. ACM*, vol. 22, no. 10, p. 560–570, oct 1979. [Online]. Available: https://doi.org/10.1145/359156.359164

[13] T. Bretl, "Motion planning of multi-limbed robots subject to equilibrium constraints: The free-climbing robot problem," *The International Journal of Robotics Research*, vol. 25, pp. 317 – 342, 2006.

[14] R. J. Griffin, G. Wiedebach, S. McCrory, S. Bertrand, I. Lee, and J. Pratt, "Footstep planning for autonomous walking over rough terrain," in *IEEE-RASInt. Conf. on Hum. Rob.*, 2019, pp. 9–16.

[15] B. Aceituno-Cabezas, H. Dai, J. Cappelletto, J. C. Grieco, and G. Fernández-López, "A mixed-integer convex optimization framework for robust multilegged robot locomotion planning over challenging terrain," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 4467–4472.

[16] B. Ponton, M. Khadiv, A. Meduri, and L. Righetti, "Efficient multi-contact pattern generation with sequential convex approximations of the centroidal dynamics," *IEEE Trans. Robot. (T-RO)*, vol. 37, no. 5, pp. 1661–1679, 2021.

[17] S. Tonneau, D. Song, P. Fernbach, N. Mansard, M. Taïx, and A. Del Prete, "Sl1m: Sparse l1-norm minimization for contact planning on uneven terrain," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2020.

[18] D. Song, P. Fernbach, T. Flayols, A. D. Prete, N. Mansard, S. Tonneau, and Y. J. Kim, "Solving footstep planning as a feasibility problem using l1-norm minimization," *IEEE Robot. Automat. Lett. (RA-L)*, vol. 6, no. 3, pp. 5961–5968, 2021.

[19] T. Marcucci, M. Petersen, D. von Wrangel, and R. Tedrake, "Motion planning around obstacles with convex optimization," *Science Robotics*, vol. 8, no. 84, p. eadf7843, 2023. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.adf7843

[20] A. Escande, A. Kheddar, S. Miossec, and S. Garsault, "Planning support contact-points for acyclic motions and experiments on hrp-2," in *Experimental Robotics*. Springer, 2009, pp. 293–302.

[21] K. Hauser, T. Bretl, J.-C. Latombe, K. Harada, and B. Wilcox, "Motion planning for legged robots on varied terrain," *The International Journal of Robotics Research*, vol. 27, no. 11-12, pp. 1325–1349, 2008.

[22] S. Wang and K. Hauser, "Unified multi-contact fall mitigation planning for humanoids via contact transition tree optimization," in *IEEE-RASInt. Conf. on Hum. Rob.*, 2018, pp. 1–9.

[23] S. Tonneau, P. Fernbach, A. Del Prete, J. Pettré, and N. Mansard, "2pac: Two-point attractors for center of mass trajectories in multi-contact scenarios," *ACM Trans. on Graph. (TOG)*, vol. 37, no. 5, 2018.

[24] I. Kumagai, M. Morisawa, S. Hattori, M. Benallegue, and F. Kanehiro, "Multi-contact locomotion planning for humanoid robot based on sustainable contact graph with local contact modification," *IEEE Robot. Automat. Lett. (RA-L)*, vol. 5, no. 4, pp. 6379–6387, 2020.

[25] S. Tonneau, "Motion planning and synthesis for virtual characters in constrained environments," Theses, INSA de Rennes, Feb. 2015. [Online]. Available: https://theses.hal.science/tel-01144630

[26] I. Kumagai, M. Morisawa, M. Benallegue, and F. Kanehiro, "Bipedal locomotion planning for a humanoid robot supported by arm contacts based on geometrical feasibility," in *IEEE-RASInt. Conf. on Hum. Rob.*, 2019, pp. 132–139.

[27] S. Lengagne, N. Ramdani, and P. Fraisse, "Planning and fast re-planning safe motions for humanoid robots," *IEEE Transactions on Robotics*, vol. 27, no. 6, pp. 1095–1106, 2011.

[28] S. B. Liu, H. Roehm, C. Heinzemann, I. Lütkebohle, J. Oehlerking, and M. Althoff, "Provably safe motion of mobile robots in human environments," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2017, pp. 1351–1357.

[29] J. Borquez, S. Peng, Y. Chen, Q. Nguyen, and S. Bansal, "Hamilton-jacobi reachability analysis for hybrid systems with controlled and forced transitions," in *Proc. of Robotics: Science and Systems*, 2023.

[30] I. Mordatch, E. Todorov, and Z. Popović, "Discovery of complex behaviors through contact-invariant optimization," *ACM Trans. Graph.*, vol. 31, no. 4, Jul. 2012. [Online]. Available: https://doi.org/10.1145/2185520.2185539

[31] K. Yunt and C. Glocker, "Trajectory optimization of mechanical hybrid systems using sumt," in *9th IEEE International Workshop on Advanced Motion Control, 2006.*, 2006, pp. 665–671.

[32] M. Posa, C. Cantu, and R. Tedrake, "A direct method for trajectory optimization of rigid bodies through contact," *The Int. J. of Rob. Res. (IJRR)*, vol. 33, 2014.

[33] A. W. Winkler, C. D. Bellicoso, M. Hutter, and J. Buchli, "Gait and trajectory optimization for legged systems through phase-based end-effector parameterization," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1560–1567, 2018.

[34] D. Layeghi, S. Tonneau, and M. Mistry, "Optimal control via combined inference and numerical optimization," in *IEEE Int. Conf. Rob. Autom. (ICRA)*. IEEE, 2022, pp. 3429–3435.

[35] T. Pang, H. T. Suh, L. Yang, and R. Tedrake, "Global planning for contact-rich manipulation via local smoothing of quasi-dynamic contact models," *IEEE Trans. Robot. (T-RO)*, 2023.

[36] Q. Le Lidec, F. Schramm, L. Montaut, C. Schmid, I. Laptev, and J. Carpentier, "Leveraging randomized smoothing for optimal control of nonsmooth dynamical systems," *Nonlinear Analysis: Hybrid Systems*, vol. 52, p. 101468, 2024.

[37] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter, "Learning agile and dynamic motor skills for legged robots," *Science Robotics*, vol. 4, no. 26, p. eaau5872, 2019. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.aau5872

[38] Z. Zhuang, Z. Fu, J. Wang, C. Atkeson, S. Schwertfeger, C. Finn, and H. Zhao, "Robot parkour learning," in *Conference on Robot Learning (CoRL)*, 2023.

[39] F. Jenelten, J. He, F. Farshidian, and M. Hutter, "Dtc: Deep tracking control," *Science Robotics*, vol. 9, no. 86, p. eadh5401, 2024. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.adh5401

[40] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. ACM*, vol. 18, no. 9, p. 509–517, Sep. 1975. [Online]. Available: https://doi.org/10.1145/361002.361007

[41] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[42] R. J. Griffin, G. Wiedebach, S. McCrory, S. Bertrand, I. Lee, and J. Pratt, "Footstep planning for autonomous walking over rough terrain," in *2019 IEEE-RAS 19th International Conference on Humanoid Robots (Humanoids)*. IEEE, 2019, pp. 9–16.

[43] O. Stasse, T. Flayols, R. Budhiraja, K. Giraud-Esclasse, J. Carpentier, J. Mirabel, A. Del Prete, P. Souères, N. Mansard, F. Lamiraux *et al.*, "Talos: A new humanoid research platform targeted for industrial applications," in *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*. IEEE, 2017, pp. 689–695.

[44] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, 2004, pp. 2149–2154 vol.3.

[45] Unity Technologies, *Unity3D Engine*, https://unity.com/, 2024, version 2024.1. [Online]. Available: https://unity.com/