Multi-contact Posture Generation using Vector Field Inequalities

Daniel S. J. Derwent[0000−0001−9002−4257], Simon Watson[0000−0001−9783−0147], and Bruno Vilhena Adorno[0000−0002−5080−8724]

Manchester Centre for Robotics and AI, University of Manchester, Oxford Rd, Manchester M13 9PL, UK
daniel.johnson-2@manchester.ac.uk,simon.watson@manchester.ac.uk, bruno.adorno@manchester.ac.uk

Abstract We present a novel method for solving posture generation problems in multi-contact motion planning for legged robots. Our approach builds on the state of the art by generating not only optimal contact placement locations but also simultaneously verifying the existence of a feasible trajectory allowing the robot to make those contacts. By optimising the robot's velocity rather than its configuration, we are able to replace what would otherwise be a highly constrained non-linear optimisation problem with a series of linearly constrained quadratic programs, which are comparatively much faster to solve. We implement our posture generator as part of a receding horizon multi-contact planning algorithm to generate several motion plans in challenging environments, including chimney climbing and negotiating narrow passages by forming contacts on walls. Using Bayesian data analysis, we find that the mean execution time of our planner is faster than the state of the art in all scenarios tested (ranging from 10 seconds to 10 minutes faster), while in two of four scenarios it returns shorter paths (ranging from 23.4 stance changes longer to 53.9 stance changes shorter).

Keywords: Contact planning · Legged robots · Vector-field inequalities.

1 Introduction

A unique challenge of legged motion planning is the need to plan how the robot will make and break contacts with its environment. If the desired form of motion is known and approximately cyclical (*e.g.*, walking across uneven terrain), it may suffice to use a pre-specified or adaptive gait. However, if robots are to perform acyclic motions such as navigating sparse irregular footholds [2] or very rough terrain where the precise contact locations are critical [7], then they must explicitly plan where, and in what sequence, individual contacts should be made or broken. This is referred to as multi-contact motion planning.

To guarantee that a given contact combination (*i.e.,* a *stance* [2]) is feasible, multi-contact planners must find safe whole-body configurations (referred to as *witness postures* [4]) that allow the robot to realise the intended stance while respecting certain constraints (*e.g.*, avoiding collisions, maintaining balance, *etc.*).

The task of finding a witness posture for a given stance is known as the '*posture generation problem*'.

1.1 Related Works

Multi-contact planners are typically divided into Motion-Before-Contact (MBC) and Contact-Before-Motion (CBM) approaches [2]. MBC algorithms plan a collision-free torso trajectory and then solve the posture generation problem to find witness postures for each resulting torso pose. For example, the planner in [14] generates a large offline dataset of randomly sampled limb configurations that is searched at runtime to assemble witness postures. MBC planners typically execute faster than CBM planners, needing only to consider the robot's torso pose rather than the full configuration. However, MBC approaches must assume what kinds of motion the robot can execute and constrain the trajectory search to regions where such motions are most likely possible, leading to a loss of generality (*e.g.*, [14] does not consider configurations where all contacts are on vertical surfaces).

Alternatively, CBM algorithms plan a series of stances that are combined to form an overall motion, solving the posture generation problem for each stance to verify their feasibility. Early CBM planners solve the posture generation problem using random sampling [2,6] that is biased in later works by user-defined primitives [7]. Another approach by Mordatch *et al.* uses numerical optimisation to select optimal contacts from a set of candidate footholds while simultaneously generating witness postures for each [11]. However, these approaches are all limited by their reliance on possible footholds being pre-surveyed.

The CVBFP algorithm [4] improves upon that in [11] by allowing the optimiser to choose contact locations from anywhere on a given surface. However, CVBFP assumes that whole-body trajectories between witness postures exist without verifying this to be the case. This is partially addressed by the multi-stage framework presented in [5], wherein the first stage computes a sequence of stances and witness postures, and the second stage generates a whole body trajectory which executes the motion. However, because the stance sequence is planned prior to the trajectory, it remains possible that a stance sequence may be returned for which a feasible trajectory cannot be found.

1.2 Statement of Contributions

We propose a novel posture generator that improves upon the state-of-the-art by:

- Guaranteeing that kinematically feasible whole-body trajectories between witness postures exist *before* they are added to the search tree, verifying assumptions made in [4,5].
- Generating the stance, witness posture, and whole-body trajectory *simultaneously*, removing the need for any additional stages of planning.

We also incorporate our posture generator into a novel receding horizon planning architecture that allows the robot to iteratively re-plan as it explores. We demonstrate our approach by planning several challenging motions for the Corin hexapod [15], including scenarios where some or all of the robot's contacts are on vertical surfaces (see Fig. 1). Finally, we use Bayesian data analysis techniques to compare our planner's performance to that of CVBFP [4].

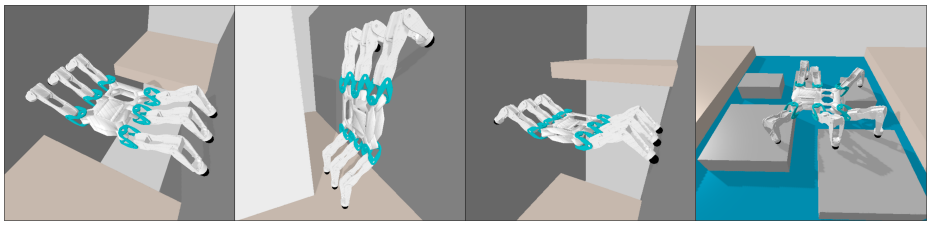


Figure 1: Simulated example scenarios used in this work. We refer to these scenarios (from left to right) as '*chimney walking*', '*wall walking*', '*chimney climbing*', and '*stepping stones*'.

2 Mathematical Preliminaries

The proposed posture generator relies heavily on geometrical primitives, such as planes, lines, and points, in addition to rigid transformations, twists, and wrenches, all of which are elegantly represented using dual quaternion algebra. Additionally, several robot modelling and control techniques that are useful for solving posture generation problems have been developed that utilize the strong algebraic properties of dual quaternion algebra, making this an attractive choice. Quaternions belong to the set $\mathbb{H} \triangleq \{h_1 + \hat{i}h_2 + \hat{j}h_3 + \hat{k}h_4 \ : \ h_1, h_2, h_3, h_4 \in \mathbb{R}\}$, where $\hat{i}$, $\hat{j}$, and $\hat{k}$ denote imaginary units such that $\hat{i}^2 = \hat{j}^2 = \hat{k}^2 = \hat{i}\hat{j}\hat{k} = -1$. Quaternions can represent 3D orientations and rotations using elements of the subset $\mathbb{S}^3 \triangleq \{\boldsymbol{h} \in \mathbb{H} \ : \ \|\boldsymbol{h}\| = 1\}$, as well as positions using elements of the subset $\mathbb{H}_p \triangleq \{\boldsymbol{h} \in \mathbb{H} \ : \ \mathrm{Re}\,(\boldsymbol{h}) = 0\}$ [16]. *Dual* quaternions extend quaternions and belong to the set $\mathcal{H} \triangleq \{\boldsymbol{h} + \varepsilon\boldsymbol{h}' \ : \ \boldsymbol{h}, \boldsymbol{h}' \in \mathbb{H}, \varepsilon^2 = 0, \varepsilon \neq 0\}$. The set of *unit* dual quaternions, $\underline{\boldsymbol{\mathcal{S}}} \triangleq \{\underline{\boldsymbol{h}} \in \mathcal{H} \ : \ \|\underline{\boldsymbol{h}}\| = 1\}$, represent rigid transformations in 3D space, and any $\underline{\boldsymbol{x}} \in \underline{\boldsymbol{\mathcal{S}}}$ can be written as $\underline{\boldsymbol{x}} = \boldsymbol{r} + \varepsilon\frac{1}{2}\boldsymbol{p}\boldsymbol{r}$, where $\boldsymbol{r} \in \mathbb{S}^3$ and $\boldsymbol{p} \in \mathbb{H}_p$ denote the rotation and translation components of the rigid transformation respectively [16]. Analogously to $\mathbb{H}_p$, we also define the subset $\mathcal{H}_p \triangleq \{\boldsymbol{h} + \varepsilon\boldsymbol{h}' : \boldsymbol{h}, \boldsymbol{h}' \in \mathbb{H}_p\}$, which is useful for modelling certain primitives.

The coefficients of elements of any of the aforementioned sets can be bijectively mapped into vectors. For example,

$$\mathrm{vec}_3\left(\hat{i}h_1+\hat{j}h_2+\hat{k}h_3\right) = \begin{bmatrix} h_1 & h_2 & h_3 \end{bmatrix}^T, \quad \mathrm{vec}_4\left(h_1+\hat{i}h_2+\hat{j}h_3+\hat{k}h_4\right) = \begin{bmatrix} h_1 & h_2 & h_3 & h_4 \end{bmatrix}^T.$$

This is particularly convenient when using task-space constrained controllers with geometrical constraints because they can be formulated as quadratic programs with linear constraints in the control inputs. For example, consider the

robot configuration $\boldsymbol{q}(t) \triangleq \left[\operatorname{vec}_4(\boldsymbol{r}(t))\ \operatorname{vec}_3(\boldsymbol{p}(t))\ \boldsymbol{\theta}(t)\right]^T$ where $\boldsymbol{r}(t) \in \mathbb{S}^3$ and $\boldsymbol{p}(t) \in \mathbb{H}_p$ denote the torso orientation and position at time t, respectively, such that the torso pose may be given as $\underline{\boldsymbol{x}}(t) = \boldsymbol{r}(t) + \varepsilon\frac{1}{2}\boldsymbol{p}(t)\boldsymbol{r}(t)$, and $\boldsymbol{\theta}(t)$ is the vector of joint configurations. Consider a task vector $\boldsymbol{x} \triangleq \boldsymbol{x}(\boldsymbol{q})$, which might represent, for instance, the robot pose, position, or orientation, and a constant desired task vector $\boldsymbol{x}_d \in \mathbb{R}^m$.We can define an error vector $\tilde{\boldsymbol{x}} \triangleq \boldsymbol{x} - \boldsymbol{x}_d$ and minimise the error using the control law

$$\begin{aligned} \boldsymbol{u} \in \underset{\dot{\boldsymbol{q}}}{\operatorname{argmin}} \quad & \left\|\boldsymbol{J}_{\tilde{\boldsymbol{x}}}\dot{\boldsymbol{q}} + \eta_o\tilde{\boldsymbol{x}}\right\|_2^2 + \lambda^2\left\|\dot{\boldsymbol{q}}\right\|_2^2 \\ \text{subject to} \quad & \boldsymbol{W}(\boldsymbol{q})\,\dot{\boldsymbol{q}} \preceq \boldsymbol{w}(\boldsymbol{q}), \end{aligned} \tag{1}$$

where $\boldsymbol{J}_{\tilde{\boldsymbol{x}}} = \partial\tilde{\boldsymbol{x}}/\partial\boldsymbol{q}$, $\eta_o \in (0,\infty)$, $\lambda \in [0,\infty)$, and $\boldsymbol{W}(\boldsymbol{q}) \in \mathbb{R}^{\ell\times n}$ and $\boldsymbol{w}(\boldsymbol{q}) \in \mathbb{R}^\ell$ impose ℓ linear constraints on $\dot{\boldsymbol{q}}$ [10].

The vector field inequality (VFI) framework allows us to transform non-linear geometric constraints on $\boldsymbol{q}$ into linear constraints on $\dot{\boldsymbol{q}}$, which ensure that the original constraints on $\boldsymbol{q}$ are met [10]. This requires only a differentiable signed distance function $d \triangleq d(\boldsymbol{q})$ between a geometrical entity kinematically coupled to the robot and another geometrical primitive in the task space, and the Jacobian matrix $\boldsymbol{J}_d \in \mathbb{R}^{1\times n}$ such that $\dot{d} = \boldsymbol{J}_d\dot{\boldsymbol{q}}$ [10]. To keep a robot entity outside of a given region, *e.g.,* for collision avoidance, we can define $\tilde{d}(\boldsymbol{q}) \triangleq d(\boldsymbol{q}) - d_{\text{safe}}$ and formalise this requirement as $\tilde{d}(\boldsymbol{q}) \geq 0$. Using the VFI framework, we can restate this as the constraint

$$\dot{\tilde{d}}(\boldsymbol{q}) \geq -\eta_d\tilde{d}(\boldsymbol{q}) \iff -\boldsymbol{J}_d\dot{\boldsymbol{q}} \leq \eta_d\tilde{d}(\boldsymbol{q}), \tag{2}$$

where $\eta_d \in (0,\infty)$ determines the maximum approach velocity [10]. Similarly, to keep a robot entity inside a given region, *e.g.*, keeping the centre of mass inside a support polygon, we restate the requirement as $\tilde{d}(\boldsymbol{q}) \leq 0$ to obtain the constraint $\boldsymbol{J}_d\dot{\boldsymbol{q}} \leq -\eta_d\tilde{d}(\boldsymbol{q})$. Robot entities and zones of interest (*i.e.*, restricted zones and safe zones) can take on a wide variety of forms, with distance functions and corresponding Jacobians defined in the literature for combinations of points, lines, planes and many others [10,12,13]. Throughout this paper, the relevant signed distance function between two primitives $\underline{\boldsymbol{a}}, \underline{\boldsymbol{b}} \in \mathcal{H}$ is denoted by $d_{a,b}$ and the *square* distance by $D_{a,b}$. In both cases the Jacobian matrix is denoted $\boldsymbol{J}_{a,b}$. The formulations for each distance function and Jacobian are found in [10], unless otherwise stated.

3 Proposed Posture Generation Approach

The core idea behind our approach is to use a VFI-constrained controller like (1), which drives the robot into an appropriate stance/witness posture. This allows us to simultaneously generate new stances while implicitly generating the witness posture and whole-body trajectory, which can then be tracked by a closed-loop controller on the real robot.

The posture generator receives the initial configuration and stance, the foot being moved (the *foot of interest*, or FOI) and the surface to place it upon. The posture generator works in three stages—the lifting stage, transition stage, and placement stage. The lifting stage finds a configuration where the robot can safely break the existing contact with the FOI. The transition stage then moves the FOI to the desired contact surface, and the placement stage optimises its location on that surface. Our formulation considers all three stages as part of the same movement, which is simpler than [4] and [5], where lifting and placing are distinct actions generated by separate posture generator calls. Often the FOI can be lifted from the initial configuration, in which case the lifting stage is skipped. Likewise, if the FOI already lies on the desired surface then the transition stage is also skipped.

Thanks to our simpler formulation, we replace one non-linear optimisation problem in the configuration space [4,5] with a series of linearly constrained quadratic programs in the tangent space, which considerably reduces execution times. Once the control input for a given sampling period is computed, we use numerical integration to generate the next configuration, recalculate the functions of $\boldsymbol{q}$ (*e.g.*, the Jacobian matrices), and then generate the next control input. This continues until either the stage's stopping criteria are satisfied or an evaluation limit is reached.

The VFIs guarantee that if the robot entities are outside of their restricted zones at time $t = 0$, then they will continue to be so for all times $t > 0$ [10] when operating in *continuous time*. Since we use *discrete* numerical integration steps, the robot can briefly enter the restricted zone between consecutive time steps, potentially resulting in unsafe behaviour or an unfeasible optimisation problem. To address this, we artificially inflate the boundaries of each restricted zone by a buffer $b_d \in (0, \infty)$, which can be given as a function of the robot's maximum speed and the integration period, and terminate the optimisation if the non-inflated boundaries are crossed. We thus re-write the constraint (2) for keeping outside a given region as

$$\dot{\tilde{d}}(\boldsymbol{q}) \geq -\eta_d \left(\tilde{d}(\boldsymbol{q}) - b_d \right) \implies -\boldsymbol{J}_d \dot{\boldsymbol{q}} \leq \eta_d \left(\tilde{d}(\boldsymbol{q}) - b_d \right). \tag{3}$$

Analogously, the tightened constraint to stay inside a safe region is given as

$$\dot{\tilde{d}}(\boldsymbol{q}) \leq -\eta_d \left(\tilde{d}(\boldsymbol{q}) + b_d \right) \implies \boldsymbol{J}_d \dot{\boldsymbol{q}} \leq -\eta_d \left(\tilde{d}(\boldsymbol{q}) + b_d \right). \tag{4}$$

Furthermore, to ensure that there is always at least one solution to problem (1), we design our constraints such that they are always satisfied by $\dot{\boldsymbol{q}} = \boldsymbol{0}$. When $\boldsymbol{q}$ violates the inflated boundaries but does not violate the original boundaries (*i.e.,* $0 \leq \tilde{d}(\boldsymbol{q}) < b_d$), we have $\tilde{d}(\boldsymbol{q}) - b_d < 0$ and $\dot{\boldsymbol{q}} = \boldsymbol{0}$ does not satisfy (3). To overcome this problem, we add a slack variable $s_d \in [0, \max(0, -\eta_d(\tilde{d}(\boldsymbol{q}) - b_d))]$ to the constraint, resulting in

$$-\boldsymbol{J}_d \dot{\boldsymbol{q}} \leq \eta_d \left(\tilde{d}(\boldsymbol{q}) - b_d \right) + s_d. \tag{5}$$

Therefore, $0 \leq \tilde{d}(\boldsymbol{q}) < b_d$ implies $s_d = -\eta_d(\tilde{d}(\boldsymbol{q}) - b_d)$, and (5) becomes $-\boldsymbol{J}_d\dot{\boldsymbol{q}} \leq 0$, to which $\dot{\boldsymbol{q}} = \boldsymbol{0}$ is a valid solution, disallowing any further progress towards the restricted zone. Analogously, when staying inside a safe region but beyond the tightened boundary, we have that $-b_d < \tilde{d}(\boldsymbol{q}) \leq 0$ and the slack variable is rewritten as $s_d \in [0, \max(0, \eta_d(\tilde{d}(\boldsymbol{q}) + b_d))]$ so that (4) is relaxed as $\boldsymbol{J}_d\dot{\boldsymbol{q}} \leq -\eta_d\left(\tilde{d}(\boldsymbol{q}) + b_d\right) + s_d$, making $\dot{\boldsymbol{q}} = \boldsymbol{0}$ a feasible solution. Finally, problem (1) is rewritten as

$$\begin{aligned} \boldsymbol{u} \in \underset{\dot{\boldsymbol{q}}, \boldsymbol{s}}{\operatorname{argmin}} \quad & \Psi\left(\dot{\boldsymbol{q}}\right) + \Phi\left(\dot{\boldsymbol{q}}, \boldsymbol{s}\right) \\ \text{subject to} \quad & \dot{\boldsymbol{q}} \in \mathcal{C}_{\dot{\boldsymbol{q}}} \text{ and } \boldsymbol{s} \in \mathcal{C}_{\boldsymbol{s}} \end{aligned} \tag{6}$$

with

$$\Phi\left(\dot{\boldsymbol{q}}, \boldsymbol{s}\right) = \lambda^2 \left\|\dot{\boldsymbol{q}}\right\|_2^2 + \alpha^2 \left\|\boldsymbol{s}\right\|_2^2, \tag{7}$$

where $\alpha, \lambda \in [0, \infty)$ and $\boldsymbol{s} = \begin{bmatrix} s_1 & \cdots & s_\ell \end{bmatrix}^T$, with s_i denoting the ith slack variable. The task function $\Psi\left(\dot{\boldsymbol{q}}\right)$ defines the task for a given stage in the form of (1), whereas $\mathcal{C}_{\dot{\boldsymbol{q}}}$ and $\mathcal{C}_{\boldsymbol{s}}$ denote the sets of variables $\dot{\boldsymbol{q}}$ and $\boldsymbol{s}$, respectively, that respect the relevant task constraints.

4 Constraints and Objectives

4.1 Common Constraints

Five types of constraints apply to all stages of the posture generator: avoiding collisions and self-collisions; maintaining balance; preventing contacts from sliding; respecting the limits and underlying topologies of the optimisation variables; and avoiding excessive torso tilting.

Collisions avoidance and self-collision constraints: We describe the surfaces in the environment as a set $\mathit{\Pi}_{\text{obj}}\left(\boldsymbol{q}\right) \subseteq \left\{\underline{\boldsymbol{\pi}}_{O_1}, \ldots, \underline{\boldsymbol{\pi}}_{O_n}\right\}$, wherein each element $\underline{\boldsymbol{\pi}}_{O_i} \in \underline{\boldsymbol{\mathcal{S}}}$ is a plane, such that $\underline{\boldsymbol{\pi}}_{O_i} = \boldsymbol{n}_{O_i} + \varepsilon d_{O_i}$, with $\boldsymbol{n}_{O_i} \in \mathbb{S}^3 \cap \mathbb{H}_p$ being the plane normal and d_{O_i} being the distance between the plane and the origin of the reference frame [16]. Since the planes are infinite, to represent non-convex free spaces (such as the wall walking environment shown in Fig. 1), the planes in $\mathit{\Pi}_{\text{obj}}$ are selected based on the robot's configuration $\boldsymbol{q}$, similar to the approach in [12]. We define $\mathcal{A} \subset \mathcal{H}$ as the set of dual quaternion primitives representing each robot body, which may be planes, lines or points [16]. Hence, the collision avoidance constraints are defined as

$$-\boldsymbol{J}_{\underline{\boldsymbol{a}}_i\underline{\boldsymbol{\pi}}_{O_j}}\dot{\boldsymbol{q}} \leq \eta_d\left(\tilde{d}_{\underline{\boldsymbol{a}}_i\underline{\boldsymbol{\pi}}_{O_j}} - b_d\right) + s_{\underline{\boldsymbol{a}}_i\underline{\boldsymbol{\pi}}_{O_j}},\ \forall \underline{\boldsymbol{a}}_i \in \mathcal{A},\ \forall \underline{\boldsymbol{\pi}}_{O_j} \in \mathit{\Pi}_{\text{obj}}(\boldsymbol{q}). \tag{8}$$

The process for self-collision avoidance is similar. We denote by $\mathcal{B}(\underline{\boldsymbol{a}}) \subset \mathcal{A}$ the subset of primitives in $\mathcal{A}$ which are forbidden from colliding with $\underline{\boldsymbol{a}} \in \mathcal{A}$. The self-collision constraint is thus written as

$$-\boldsymbol{J}_{\underline{\boldsymbol{a}}\underline{\boldsymbol{b}}}\dot{\boldsymbol{q}} \leq \eta_d\left(\tilde{d}_{\underline{\boldsymbol{a}}\underline{\boldsymbol{b}}} - b_d\right) + s_{\underline{\boldsymbol{a}}\underline{\boldsymbol{b}}},\ \forall \underline{\boldsymbol{b}} \in \mathcal{B}(\underline{\boldsymbol{a}}),\ \forall \underline{\boldsymbol{a}} \in \mathcal{A}. \tag{9}$$

Balance constraint: We use the generalised support polygon proposed by Bretl and Lall [3] that describes the set of (x, y)-coordinates that the robot's centre of mass $\boldsymbol{p}_C \in \mathbb{H}_p$ must occupy to maintain static balance without contact sliding for contacts on arbitrary surfaces. We define a set of planes $\mathit{\Pi}_{\text{balance}} \triangleq \{\underline{\boldsymbol{\pi}}_{B_1}, \ldots, \underline{\boldsymbol{\pi}}_{B_m}\}$ describing a vertical prism whose (x, y) cross section is the support polygon. Hence, we obtain the VFI constraint

$$\boldsymbol{J}_{\boldsymbol{p}_C \underline{\boldsymbol{\pi}}_{B_i}} \dot{\boldsymbol{q}} \leq -\eta_d \left(\tilde{d}_{\boldsymbol{p}_C \underline{\boldsymbol{\pi}}_{B_i}} + b_d \right) + s_{\boldsymbol{p}_C \underline{\boldsymbol{\pi}}_{B_i}}, \forall \underline{\boldsymbol{\pi}}_{B_i} \in \mathit{\Pi}_{\text{balance}}. \tag{10}$$

Preventing foot sliding: We constrain each foot to remain inside a sphere centred on its desired location. Defining the set $\mathcal{P}_{\text{feet}} \subset \mathbb{H}_p$ containing the position of the frame attached to each foot, and denoting by $\boldsymbol{p}_{d_i} \in \mathbb{H}_p$ the desired location of foot i, we write

$$\boldsymbol{J}_{\boldsymbol{p}_i \boldsymbol{p}_{d_i}} \dot{\boldsymbol{q}} \leq -\eta_d \left(\tilde{D}_{\boldsymbol{p}_i \boldsymbol{p}_{d_i}} + b_d \right) + s_{\boldsymbol{p}_i \boldsymbol{p}_{d_i}}, \forall \boldsymbol{p}_i \in \mathcal{P}_{\text{feet}}, \tag{11}$$

where $\tilde{D}_{\boldsymbol{p}_i \boldsymbol{p}_{d_i}}(\boldsymbol{q}) \triangleq D_{\boldsymbol{p}_i \boldsymbol{p}_{d_i}}(\boldsymbol{q}) - R^2_{\text{sphere}}$, with R_{sphere} being the sphere's radius.

Respecting variable limits and topologies: Since the configuration vector $\boldsymbol{q}$ includes the term $\boldsymbol{r} \in \mathbb{S}^3$, the optimisation must be constrained to ensure that $\dot{\boldsymbol{q}}$ respects the properties of the underlying topology of unit quaternions—*i.e.,* the condition $\|\boldsymbol{r}\| = 1$ must be maintained, which is equivalent to $\text{vec}_4(\boldsymbol{r})^T \text{vec}_4(\boldsymbol{r}) = 1$. By taking the time derivative of this expression, we obtain the constraint

$$\text{vec}_4(\dot{\boldsymbol{r}})^T \text{vec}_4(\boldsymbol{r}) = 0. \tag{12}$$

We also constrain $\boldsymbol{s}$ to respect its limits, and constrain $\dot{\boldsymbol{q}}$ to respect limits on $\boldsymbol{q}$ (similar to [13]). Given limits on $\boldsymbol{q}$ in the form $\boldsymbol{q}_{\min}, \boldsymbol{q}_{\max} \in \mathbb{R}^n$, we also define $\boldsymbol{s}_{\max} \in \mathbb{R}^\ell$ as an upper limit on $\boldsymbol{s}$ such that each element i of $\boldsymbol{s}_{\max}$ is given by $s_{\max_i} = \max\left(-\eta_d(\tilde{d}_i - b_d), \eta_d(\tilde{d}_i + b_d)\right)$. Thus we write the constraints

$$\eta_d \left(\boldsymbol{q}_{\min} - \boldsymbol{q} + b_d \right) \leq \dot{\boldsymbol{q}} \leq \eta_d \left(\boldsymbol{q}_{\max} - \boldsymbol{q} - b_d \right), \qquad \boldsymbol{0}_\ell \leq \boldsymbol{s} \leq \boldsymbol{s}_{\max}, \tag{13}$$

where $\boldsymbol{0}_\ell \in \mathbb{R}^\ell$ is a vector of zeros.

Preventing robot tilting: Finally, we constrain the robot's orientation to lie within a maximum tolerance from a desired value. Like CVBFP [4], our planner incorporates a guide path that may be generated autonomously or by the user. Therefore, the desired torso orientation is defined as that of the closest point on the guide path to the current configuration. We then obtain lines describing the x, y and z axes of the robot's torso frame with respect to the world frame ($\underline{\boldsymbol{l}}_x, \underline{\boldsymbol{l}}_y, \underline{\boldsymbol{l}}_z \in \mathcal{H}_p \cap \underline{\boldsymbol{\mathcal{S}}}$, respectively) as well as those for the desired torso orientation (denoted $\underline{\boldsymbol{l}}_{d_x}, \underline{\boldsymbol{l}}_{d_y}, \underline{\boldsymbol{l}}_{d_z} \in \mathcal{H}_p \cap \underline{\boldsymbol{\mathcal{S}}}$). Using the angular distance function between two lines $d_{\phi_{\underline{\boldsymbol{l}}_1} \phi_{\underline{\boldsymbol{l}}_2}}$ and its Jacobian matrix $\boldsymbol{J}_{\phi_{\underline{\boldsymbol{l}}_1} \phi_{\underline{\boldsymbol{l}}_2}}$ as described in [13], we write

$$\boldsymbol{J}_{\phi_{\underline{\boldsymbol{l}}_i} \phi_{\underline{\boldsymbol{l}}_{d_i}}} \dot{\boldsymbol{q}} \leq -\eta_d \left(\tilde{d}_{\phi_{\underline{\boldsymbol{l}}_i} \phi_{\underline{\boldsymbol{l}}_{d_i}}} + b_d \right) + s_{\phi_{\underline{\boldsymbol{l}}_i} \phi_{\underline{\boldsymbol{l}}_{d_i}}}, \forall \underline{\boldsymbol{l}}_i \in \{\underline{\boldsymbol{l}}_x, \underline{\boldsymbol{l}}_y, \underline{\boldsymbol{l}}_z\}. \tag{14}$$

4.2 Lifting Stage

In the lifting stage, the posture generator moves the robot into a configuration where the FOI is not required for the robot's balance and can thus be safely lifted. We compute two generalised support polygons [3], one *with* and one *without* the FOI, whose respective sets of planes are denoted $\mathit{\Pi}_{\text{with}}$ and $\mathit{\Pi}_{\text{without}}$, and we define the line $\underline{\boldsymbol{l}}_{\text{cent}} \in \mathcal{H}_p \cap \underline{\boldsymbol{\mathcal{S}}}$ that is parallel to the world frame's z-axis and intersects the centroid of the region $\mathit{\Pi}_{\text{without}}$. Thus, we define the task function for the lifting stage as $\Psi_L\left(\dot{\boldsymbol{q}}\right) = \left\| \boldsymbol{J}_{\boldsymbol{p}_C \underline{\boldsymbol{l}}_{\text{cent}}} \dot{\boldsymbol{q}} + \eta_o \tilde{d}_{\boldsymbol{p}_C \underline{\boldsymbol{l}}_{\text{cent}}} \right\|_2^2$, which drives the robot's CoM, $\boldsymbol{p}_C$, towards $\underline{\boldsymbol{l}}_{\text{cent}}$ until $\tilde{d}_{\boldsymbol{p}_C \underline{\boldsymbol{\pi}}_i} \leq -b_d$ for all $\underline{\boldsymbol{\pi}}_i \in \mathit{\Pi}_{\text{without}}$. If this condition is already met, then the lifting stage is skipped. We define the sets $\mathcal{C}_{\dot{\boldsymbol{q}}}$ and $\mathcal{C}_{\boldsymbol{s}}$ for the lifting stage as the values of $\dot{\boldsymbol{q}}$ and $\boldsymbol{s}$, respectively, that respect constraints (8) to (14). Note that (10) uses $\mathit{\Pi}_{\text{balance}} = \mathit{\Pi}_{\text{with}}$ for the lifting stage, but all subsequent stages use $\mathit{\Pi}_{\text{balance}} = \mathit{\Pi}_{\text{without}}$.

4.3 Transition Stage

In the transition stage, the FOI is brought into contact with the desired surface if it is not already. The desired contact surface is a plane $\underline{\boldsymbol{\pi}}_{\text{con}} \in \underline{\boldsymbol{\mathcal{S}}}$ with boundaries defined by the set of planes $\mathit{\Pi}_{\text{bound}}$. Thus, denoting the position of the FOI $\boldsymbol{p}_F \in \mathbb{H}_p$, we consider the transition stage satisfied if and only if $\tilde{d}_{\boldsymbol{p}_F \underline{\boldsymbol{\pi}}_i} \leq -b_d$ for all $\underline{\boldsymbol{\pi}}_i \in \mathit{\Pi}_{\text{bound}}$ and $\tilde{D}_{\boldsymbol{p}_F \underline{\boldsymbol{\pi}}_{\text{con}}} \leq -b_d$. The objective function for this stage depends on whether $\boldsymbol{p}_F$ respects the boundaries in $\mathit{\Pi}_{\text{bound}}$. If all the boundary planes are respected, then we constrain $\boldsymbol{p}_F$ to continue respecting them by applying the constraint

$$\boldsymbol{J}_{\boldsymbol{p}_F \underline{\boldsymbol{\pi}}_i} \dot{\boldsymbol{q}} \leq -\eta_d \left(\tilde{d}_{\boldsymbol{p}_F \underline{\boldsymbol{\pi}}_i} + b_d \right) + s_{\boldsymbol{p}_F \underline{\boldsymbol{\pi}}_i}, \, \forall \underline{\boldsymbol{\pi}}_i \in \mathit{\Pi}_{\text{bound}}, \tag{15}$$

while the task function $\Psi_{T_r}\left(\dot{\boldsymbol{q}}\right) \triangleq \left\| \boldsymbol{J}_{\boldsymbol{p}_F \underline{\boldsymbol{\pi}}_{\text{con}}} \dot{\boldsymbol{q}} + \eta_o \tilde{D}_{\boldsymbol{p}_F \underline{\boldsymbol{\pi}}_{\text{con}}} \right\|_2^2$ drives $\boldsymbol{p}_F$ towards $\underline{\boldsymbol{\pi}}_{\text{con}}$. Alternatively, if one or more boundary plane in $\mathit{\Pi}_{\text{bound}}$ is not respected, then constraint (15) is not applied and the task function is re-written as$\Psi_{T_n}\left(\dot{\boldsymbol{q}}\right) \triangleq \left\| \boldsymbol{J}_{\boldsymbol{p}_F \underline{\boldsymbol{l}}_{\text{con}}} \dot{\boldsymbol{q}} + \eta_o \tilde{D}_{\boldsymbol{p}_F \underline{\boldsymbol{l}}_{\text{con}}} \right\|_2^2 + \Psi_{T_r}\left(\dot{\boldsymbol{q}}\right)$, where $\underline{\boldsymbol{l}}_{\text{con}} \in \mathcal{H}_p \cap \underline{\boldsymbol{\mathcal{S}}}$ is a line through the centroid of the contact plane which is perpendicular to its surface, similar to that used in the lifting case. Thus this task function also brings $\boldsymbol{p}_F$ into the region described by $\mathit{\Pi}_{\text{bound}}$.

Finally, $\boldsymbol{p}_F$ is constrained to prevent it from crossing in-front of any preceding legs. For example, the middle left foot cannot cross in-front of the front left foot (designated the *blocking foot*). To define this constraint, we first define a plane $\underline{\boldsymbol{\pi}}_x$ that intersects the blocking foot with a normal vector parallel to the x-axis of the robot's torso frame, which points forward with respect to the body. Thus, we define the constraint

$$\boldsymbol{J}_{\boldsymbol{p}_F \underline{\boldsymbol{\pi}}_x} \dot{\boldsymbol{q}} \leq -\eta_d \left(\tilde{d}_{\boldsymbol{p}_F \underline{\boldsymbol{\pi}}_x} + b_d \right) + s_{\boldsymbol{p}_F \underline{\boldsymbol{\pi}}_x}. \tag{16}$$

As before, we define $\mathcal{C}_{\dot{\boldsymbol{q}}}$ and $\mathcal{C}_{\boldsymbol{s}}$ for the transition stage as the values of $\dot{\boldsymbol{q}}$ and $\boldsymbol{s}$, respectively, that respect constraints (8) to (16).

4.4 Placement Stage

The placement stage optimises the position of $\boldsymbol{p}_F$ on $\underline{\boldsymbol{\pi}}_{\text{con}}$ by minimising a potential field $U(\boldsymbol{q}) : \mathcal{Q} \to \mathbb{R}$ that is based on a guide path composed of line segments linking waypoint positions for each foot. Here, we consider only the potential field associated with the FOI, denoted $U_F(\boldsymbol{q})$, as no other foot is free to move. Let us denote the closest point on the relevant guide path to $\boldsymbol{p}_F$ as $\boldsymbol{p}_U \in \mathbb{H}_p$ and the point that terminates the line segment containing $\boldsymbol{p}_U$ as $\boldsymbol{p}_T \in \mathbb{H}_p$. Thus, $U_F(\boldsymbol{q})$ is given as $U_F(\boldsymbol{q}) \triangleq \beta D_{\boldsymbol{p}_F\boldsymbol{p}_U} + \gamma D_{\boldsymbol{p}_U\boldsymbol{p}_T}$, where $\beta, \gamma \in [0, \infty)$. Thus, to minimise $U_F(\boldsymbol{q})$, the posture generator forms contacts as close to the guide path as possible by minimising $D_{\boldsymbol{p}_F\boldsymbol{p}_U}$ while making as much progress towards the goal as possible by minimising $D_{\boldsymbol{p}_U\boldsymbol{p}_T}$. Writing the Jacobian of $U_F(\boldsymbol{q})$ with respect to $\boldsymbol{q}$ as $\boldsymbol{J}_{U_F}(\boldsymbol{q}) \triangleq \beta \boldsymbol{J}_{\boldsymbol{p}_F\boldsymbol{p}_U} + \gamma \boldsymbol{J}_{\boldsymbol{p}_U\boldsymbol{p}_T}$, the task function for the placement stage is given as $\Psi_P(\dot{\boldsymbol{q}}) = \|\boldsymbol{J}_{U_F}\dot{\boldsymbol{q}} + \eta_o U_F(\boldsymbol{q})\|_2^2$. Finally we limit the maximum square distance from $\boldsymbol{p}_F$ to $\underline{\boldsymbol{\pi}}_{\text{con}}$ by applying the constraint

$$\boldsymbol{J}_{\boldsymbol{p}_F\underline{\boldsymbol{\pi}}_{\text{con}}}\dot{\boldsymbol{q}} \leq -\eta_d \left(\tilde{D}_{\boldsymbol{p}_F\underline{\boldsymbol{\pi}}_{\text{con}}} + b_d\right) + s_{\boldsymbol{p}_F\underline{\boldsymbol{\pi}}_{\text{con}}}. \tag{17}$$

The sets $\mathcal{C}_{\dot{\boldsymbol{q}}}$ and $\mathcal{C}_{\boldsymbol{s}}$ are thus given for the placement stage as the values of $\dot{\boldsymbol{q}}$ and $\boldsymbol{s}$ respectively which respect constraints (8) to (17).

5 Other considerations

5.1 Numerical Integration

Once a control input is generated by using (6), the robot configuration is updated via numerical integration as follows. First, the configuration velocity vector $\dot{\boldsymbol{q}} \triangleq \left[\text{vec}_4\left(\dot{\boldsymbol{r}}(t)\right) \text{vec}_3\left(\dot{\boldsymbol{p}}(t)\right) \dot{\boldsymbol{\theta}}(t)\right]^T$ is extracted from $\boldsymbol{u} = (\dot{\boldsymbol{q}}, \boldsymbol{s})$, and the torso pose derivative is calculated as $\dot{\underline{\boldsymbol{x}}}(t) = \dot{\boldsymbol{r}}(t) + \varepsilon\frac{1}{2}\left(\dot{\boldsymbol{p}}(t)\boldsymbol{r}(t) + \boldsymbol{p}(t)\dot{\boldsymbol{r}}(t)\right)$. Then, the next torso pose $\underline{\boldsymbol{x}}(t+\tau)$ and joint angle vector $\boldsymbol{\theta}(t+\tau)$, where $\tau \in (0, \infty]$ is the integration step, are calculated as

$$\underline{\boldsymbol{x}}(t+\tau) = \exp\left(\tau\dot{\underline{\boldsymbol{x}}}(t)\underline{\boldsymbol{x}}(t)^*\right)\underline{\boldsymbol{x}}(t), \qquad \boldsymbol{\theta}(t+\tau) = \boldsymbol{\theta}(t) + \tau\dot{\boldsymbol{\theta}}(t), \tag{18}$$

where the (dual quaternion) exponential map and group operation are used in the first expression to ensure that the underlying topological space of unit dual quaternions [16] is respected. The pose $\underline{\boldsymbol{x}}(t+\tau)$ can hence be decomposed into $\boldsymbol{r}(t+\tau)$ and $\boldsymbol{p}(t+\tau)$, and used to form the overall configuration $\boldsymbol{q}(t+\tau)$. As mentioned in Section 3, the integration step may bring the robot into the restricted region, necessitating cancelling the control input generation. This happens most often with the contact sliding constraint (11), which is typically very tight. We partially address this by using the following constrained controller to correct drift in the foot positions resulting from the numerical step,

$$\begin{aligned} \boldsymbol{u} \in \underset{\dot{\boldsymbol{q}}}{\text{argmin}} \quad & \|\boldsymbol{J}_{\text{drift}}\dot{\boldsymbol{q}} + \eta_o \boldsymbol{D}_{\text{drift}}\|_2^2 + \lambda^2 \|\dot{\boldsymbol{q}}\|_2^2 \\ \text{subject to} \quad & \text{vec}_4\left(\dot{\boldsymbol{r}}\right)^T \text{vec}_4\left(\boldsymbol{r}\right) = 0, \end{aligned} \tag{19}$$

where $\boldsymbol{D}_{\text{drift}} \in \mathbb{R}^6$ is the stacked vector of square distances between each foot and its desired location, and $\boldsymbol{J}_{\text{drift}} = \partial \boldsymbol{D}_{\text{drift}} / \partial \boldsymbol{q}$. This reduces the frequency of violations but still necessitates small values of τ to avoid violating the remaining constraints on (6). We resolve this by varying τ between $\tau_{\min}$ and $\tau_{\max}$, making large steps where possible and smaller steps where necessary.

When a value of $\boldsymbol{u}$ is returned by solving (6), the posture generator calculates $\boldsymbol{q}(t+\tau)$ using $\tau = \tau_{\max}$. The controller in (19) is used to correct any contact drifts, and the resulting configuration is checked to verify that the constraints are satisfied. If so, then the posture generator progresses to the next optimisation. If, however, $\boldsymbol{q}(t+\tau)$ is invalid, then $\tau \leftarrow \tau - \Delta\tau$ and $\boldsymbol{q}(t+\tau)$ is recalculated. To maintain non-oscillatory numerical stability, we must ensure that $\eta_o \tau \leq 1$ [1], thus each time τ is changed, η_o is re-calculated as $\eta_o = \tau_\eta / \tau$, where $\tau_\eta \in (0, 1]$ is a user-defined constant. This continues until either a valid step is found or $\tau = \tau_{\min}$, in which case the posture generator terminates and returns a failure.

5.2 Receding Horizon Planning Strategy

The posture generator is implemented as part of our receding horizon contact planning (RHCP) algorithm. We do not give a full account of RHCP here, but it is described more fully in [8]. For the purposes of this paper, the main feature of RHCP is that it performs a local breadth-first search wherein one call to the posture generator is made for each combination of a foot and a contact surface in the environment. The resulting child nodes, each containing a feasible stance and an associated witness posture, are expanded in the same way to produce a second generation of children, and so on until a horizon depth $k_{\max}$ is reached. At that point, the node on the horizon whose witness posture best minimises the potential field is chosen, the first foot step towards the stance contained in that node is executed, and the process repeats until the robot reaches the goal.

6 Experiments and Results

Our planner and posture generator were compared to CVBFP in the four simulated scenarios depicted in Fig. 1. Both planners generated 130 motion plans for Corin in each scenario using the same starting configurations for both planners sampled from a normal distribution for each motion plan. RHCP used a two-step planning horizon. All tests used an Intel® Core™ i9-7900X Processor. Bayesian data analysis techniques were used to compare the number of stance changes in each plan and the total planning time, assuming that both metrics can be modelled by t-distributions [9]. Note that lifting and then placing a foot counts as two stance changes. The probability distributions of the difference in means and the effect size for each scenario are shown in Fig. 2. A region of practical equivalence (ROPE) of ± 0.1 was used to determine which effect sizes are statistically significant and is shown in *red* in Fig. 2.

In 57 wall walking tests and one stepping stones test, CVBFP failed to return a plan within two hours and was timed out. These cases are therefore excluded

from the analysis in Fig. 2. We hypothesise that in these cases CVBFP became stuck because the robot had assumed a configuration where its balance was critically dependent on a particular foot, preventing that foot from being lifted and thus preventing the robot from making forward progress. Backing up in the search tree could resolve this problem, but due to its policy of sibling node generation [4], CVBFP is slow to consider this option, preferring to try alternative placements for the remaining feet until it exhausts its options and is forced to backtrack. Our planner does not suffer from this problem because we only explore a single contact location for a given foot and surface pair, making RHCP comparatively quicker to give up on unproductive lines of exploration and accept backing up. Further research is required to provide more confidence in this hypothesis, but we note that similar problems were also encountered in our preliminary work [8].

As one might expect, given that our planner is local while CVBFP is global, Fig. 2a shows a statistically significant increase in the number of stance changes made in the chimney climbing and stepping stones scenarios, with RHCP making 16.6 and 23.4 more stance changes than CVBFP on average, respectively. However, in the chimney walking and wall walking scenarios, RHCP made 37.0 and 53.9 *fewer* stance changes on average, respectively. Additionally, Fig. 2b shows a statistically significant decrease in planning time for all scenarios, with RHCP completing approximately 10 s faster than CVBFP in the chimney climbing scenario, 5.5 minutes faster in the chimney walking and stepping stones scenarios, and 10 minutes faster in the wall walking scenario on average. Furthermore, as the slowest CVBFP results, those which timed out, have been excluded from the analysis in Fig. 2b the true improvement in execution time, particularly in the wall walking scenario, may be greater than that shown.

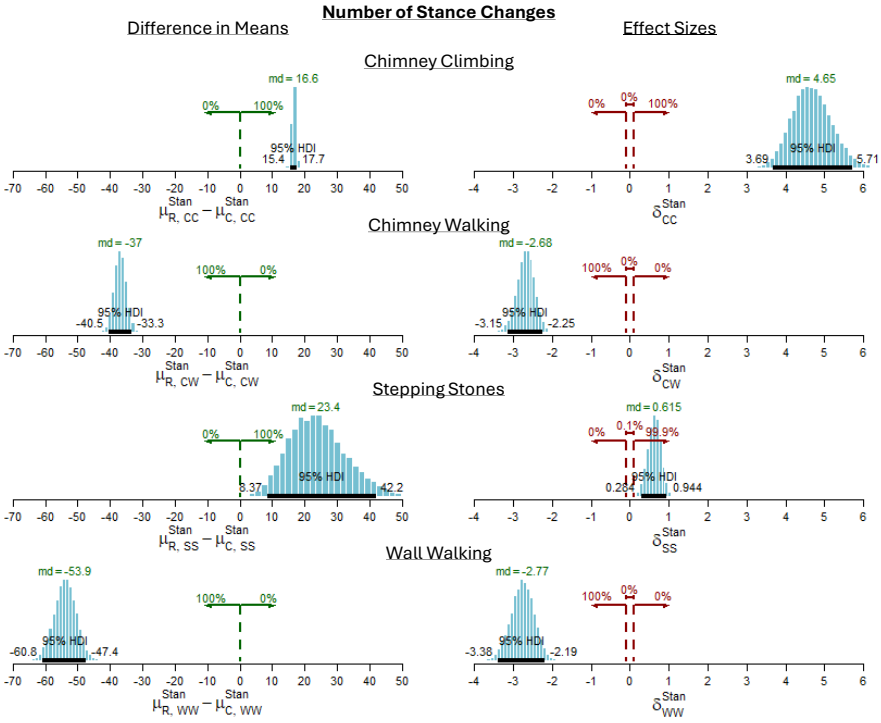
7 Conclusions

This paper presented a novel posture generator for legged robots to successfully generate several motions in challenging environments. With a two-step planning horizon, our approach was faster than the state-of-the-art in all scenarios tested, in some cases generating shorter paths despite being a local planner, all while adding new capabilities—namely, the ability to simultaneously plan whole-body trajectories.

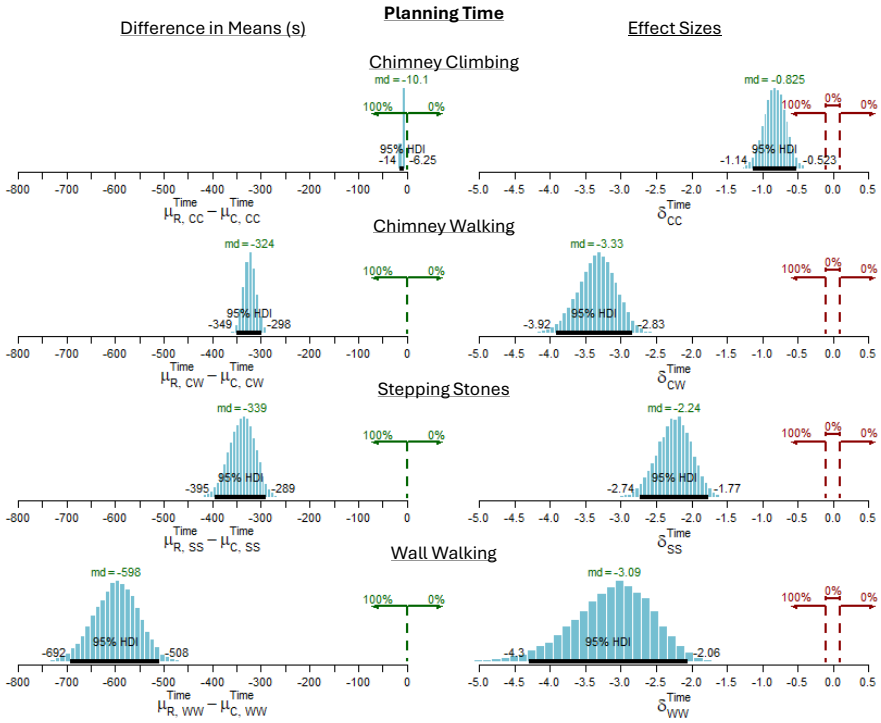
Future works will analyse in more detail how different environments and horizon depths affect the performance of the two planners and validate the paths produced by executing them in realistic simulations and on the physical robot.

Acknowledgments. This work was supported by a grant from the University of Manchester and by the Royal Academy of Engineering under the Research Chairs and Senior Research Fellowships programme. Corin was developed by Hassan Hakim Khalili and Wei Cheah [15]. R code to generate figure 2 was based on scripts released by John K. Kruschke, with adaptations by Ana Christina Almada Campos and the authors.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.



(a) Number of stance changes.



(b) Planning time.

Figure 2: Plots of probability distributions regarding the number of stance changes in the motions planned (*top*) and the typical planning time (*bottom*) for each environment. In each case, we show the difference in means $\mu_R - \mu_C$ between RHCP and CVBFP (*left*) as well as effect sizes σ (*right*).

References

1. Bjerkeng, M., Falco, P., Natale, C., Pettersen, K.Y.: Stability analysis of a hierarchical architecture for discrete-time sensor-based control of robotic systems. IEEE Transactions on Robotics **30**(3), 745–753 (2014). https://doi.org/10.1109/TRO.2013.2294882
2. Bretl, T.: Motion planning of multi-limbed robots subject to equilibrium constraints: The free-climbing robot problem. The International Journal of Robotics Research **25**(4), 317–342 (2006), https://doi.org/10.1177/0278364906063979
3. Bretl, T., Lall, S.: Testing static equilibrium for legged robots. IEEE Transactions on Robotics **24**(4), 794–807 (2008). https://doi.org/10.1109/TRO.2008.2001360
4. Escande, A., Kheddar, A., Miossec, S.: Planning contact points for humanoid robots. Robotics and Autonomous Systems **61**(5), 428–442 (2013), https://doi.org/10.1016/j.robot.2013.01.008
5. Ferrari, P., Rossini, L., Ruscelli, F., Laurenzi, A., Oriolo, G., Tsagarakis, N.G., Mingo Hoffman, E.: Multi-contact planning and control for humanoid robots: Design and validation of a complete framework. Robotics and Autonomous Systems **166**, 104448 (2023), https://doi.org/10.1016/j.robot.2023.104448
6. Hauser, K., Bretl, T., Latombe, J.C.: Non-gaited humanoid locomotion planning. In: 5th IEEE-RAS International Conference on Humanoid Robots, 2005. pp. 7–12 (2005), https://doi.org/10.1109/ICHR.2005.1573537
7. Hauser, K., Bretl, T., Harada, K., Latombe, J.C.: Using Motion Primitives in Probabilistic Sample-Based Planning for Humanoid Robots, pp. 507–522. Springer Berlin Heidelberg, Berlin, Heidelberg (2008), https://doi.org/10.1007/978-3-540-68405-3_32
8. Johnson, D.S., Adorno, B.V., Watson, S.: Receding horizon contact planning for advanced motions in hexapod robots. In: Towards Autonomous Robotic Systems Extended Abstracts: 24th Annual Conference, TAROS 2023 Cambridge, UK. pp. 34–37 (2023), https://research.manchester.ac.uk/en/publications/receding-horizon-contact-planning-for-advanced-motions-in-hexapod
9. Kruschke, J.K.: Bayesian estimation supersedes the t test. J. Exp. Psychol. Gen. **142**(2), 573–603 (May 2013)
10. Marinho, M.M., Adorno, B.V., Harada, K., Mitsuishi, M.: Dynamic active constraints for surgical robots using vector-field inequalities. IEEE Transactions on Robotics **35**(5), 1166–1185 (2019). https://doi.org/10.1109/TRO.2019.2920078
11. Mordatch, I., Todorov, E., Popović, Z.: Discovery of complex behaviors through contact-invariant optimization. ACM Trans. Graph. **31**(4) (Jul 2012), https://doi.org/10.1145/2185520.2185539
12. Pereira, M.S., Adorno, B.V.: Manipulation task planning and motion control using task relaxations. J. Control Autom. Electr. Syst. **33**(4), 1103–1115 (Aug 2022)
13. Quiroz-Omaña, J.J., Adorno, B.V.: Whole-body control with (self) collision avoidance using vector field inequalities. IEEE Robotics and Automation Letters **4**(4), 4048–4053 (2019). https://doi.org/10.1109/LRA.2019.2928783
14. Tonneau, S., Del Prete, A., Pettré, J., Park, C., Manocha, D., Mansard, N.: An Efficient Acyclic Contact Planner for Multiped Robots. IEEE Transactions on Robotics **34**(3), 586–601 (2018), https://doi.org/10.1109/TRO.2018.2819658
15. University of Manchester: Corin: Mobile hexapod for remote inspection and object manipulation (04 2022), https://uomrobotics.com/robots/corin.html
16. Vilhena Adorno, B.: Robot Kinematic Modeling and Control Based on Dual Quaternion Algebra — Part I: Fundamentals. (Feb 2017), https://hal.science/hal-01478225, working paper or preprint