Robotica (2024), **42**, pp. 2761–2780
doi:10.1017/S0263574724001097

CAMBRIDGE UNIVERSITY PRESS

RESEARCH ARTICLE

Dynamic modeling and simulation of a torque-controlled spatial quadruped robot

Daniel Teixeira de Paula, Eduardo Paciencia Godoy and Mauricio Becerra-Vargas

São Paulo State University (Unesp), Institute of Science and Technology, Sorocaba, São Paulo, Brasil
Corresponding author: Daniel Teixeira de Paula; Email: daniel.t.paula@unesp.br

Received: 8 January 2024; **Revised:** 30 April 2024; **Accepted:** 31 May 2024; **First published online:** 11 September 2024

Keywords: quadruped robot; dynamic model; legged locomotion; contact dynamics; prototyping

Abstract

Evolution has shown that legged locomotion is most adequate for tasks requiring versatile movement on land, allowing animals to traverse a wide variety of environments ranging from natural terrain to artificial, man-made landscapes with great ease. By employing well-designed control schemes, this ability could be replicated for legged robots, enabling them to be used in critical situations that still pose great danger to human integrity, such as search and rescue missions, inspection of hazardous areas, and even space exploration. This work characterizes the quadruped robot and contact dynamics that will compose our in-house simulator to be used for prototyping locomotion control schemes applied to quadruped robots. The proposed simulator computes the robot dynamics using the Recursive Newton-Euler and Composite-Rigid-Body algorithms with a few modifications to make certain aspects relevant for contact detection and control more easily accessible; furthermore, a compliant contact force method alongside stick-slip friction modeled the contact dynamics. To allow the robot to move, a simple PD-independent joint controller was implemented to track a desired leg trajectory. With the same robot and controller implemented using the MuJoCo simulation software, this work evaluates the proposed simulator by comparing characteristic locomotion signals such as the trunk pose and the ground reaction forces. Results showed similar behavior for both simulators, especially with regard to the contact detection, despite the significantly different contact models. Lastly, final remarks to enhance our simulator's performance are suggested to be explored in future works.

1. Introduction

Over the past decade, legged robotics has seen a dramatic growth in popularity largely due to showcases of impressive acrobatic stunts brought by many research groups as well as an ever-growing number of companies. However, beyond feats of agility, one major aspect that sparks the interest in legged robots is their capability to traverse natural (e.g., dirt, sand, snow, and gravel) or man-made landscapes (e.g., rubble and stairs) with great ease. Mastering locomotion would allow these robots to be effectively used in critical applications which currently still pose great danger to human lives, such as search and rescue operations in disaster areas, inspection of hazardous areas, and even space exploration. Legged locomotion is achieved by not only having well-built legs and body but also control systems that take locomotion principles into account. Effective control design inherently requires some knowledge of the expected system behavior to achieve a desired effect. This aspect is especially relevant for model-based control, where a model of the system dynamics is used to generate the control signals.

The model of the system dynamics can be obtained by applying first principles (e.g., Newton-Euler and Euler-Lagrange), using derived algorithms (e.g., Articulated Body Algorithm [1]), or by performing system identification. In any case, the chosen method should reflect the project requirements, which must take into account aspects such as system complexity and controller restrictions.

In recent years, Model Predictive Control (MPC) has become a popular scheme for legged locomotion planning and control, being extensively used by well-known research groups in the field [2–9].

© The Author(s), 2024. Published by Cambridge University Press

https://doi.org/10.1017/S0263574724001097 Published online by Cambridge University Press

With MPC, dynamic model complexity plays a critical role in run-time performance. Neunert et al. [8] employ a whole-body nonlinear MPC with an explicit soft contact model implemented by the Control Toolbox [10] and full robot dynamics obtained from the RobCoGen [11, 12] code generation framework. Hardware experiments done with two robots, HyQ [13] and ANYmal [14], showed that despite the use of full robot dynamics, software design optimization and simplified contact dynamics allowed the proposed method to run at update rates over an order of magnitude higher than existing methods. As an alternative to full robot dynamics, approximations such as the centroidal [2], single rigid body [3–5, 7, 9], and template-based dynamics (e.g., spring-loaded inverted pendulum and cart-table with flywheel [6]) have been used to reduce complexity at the cost of accuracy. Rathod et al. [9] employ nonlinear MPC with an implicit hard contact model and single rigid body dynamics for locomotion planning. The use of a simplified model was key to decrease computational cost and allow for faster online re-planning that is required to deal with dynamic environments. Di Carlo et al. [3] go a step further into simplifying the model, using MPC with linearized single rigid body dynamics. This choice allowed the controller to be formulated as a single convex optimization problem, which greatly improved solution speed and reliability as demonstrated in hardware experiments done with the MIT Cheetah 3 robot [2]. One way to mitigate the reduced accuracy originated from using simplified models while not exactly fulfilling the simplifying assertions (e.g., massless legs) is by improving robustness. Grandia et al. [4] and Minniti et al. [7] incorporate robustness mechanisms into MPC while using a *kinodynamic* model. Even though this model considers only the floating-base dynamics (single rigid body) along with the kinematics of each leg, the added robustness was able to compensate for modeling mismatches and external disturbances.

Contrasting with MPC, Whole Body Control (WBC), a powerful legged locomotion control scheme, strives for high model accuracy to generate consistent output signals, typically requiring the use of complete robot dynamics. Fahmi et al. [15] use WBC to directly compute the actuation torques based on user velocity input and planned contact sequence while considering the full robot dynamics and an implicit hard contact model. Compared to their previous work using a simplified model (centroidal dynamics), hardware experiments showed enhanced locomotive capability, with the robot successfully traversing a wide range of challenging terrain. On the other hand, Kim et al. [5] employ WBC to compute the actuation torques based on the planned reaction force profile instead of body trajectories. This choice accommodates locomotion patterns with significant aerial phases, an important characteristic for high-speed running as demonstrated by experiments using the MIT Mini Cheetah [16].

Few of the recent works from well-known research groups in the field utilize a dynamic model provided by readily available external libraries (e.g., RBDL [17]) or simulation software (e.g., MuJoCo [18], Raisim [19]), with most relying on custom in-house solutions. "Poor" computational performance with respect to specific practical requirements and inflexible model complexity of existing solutions are among the reasons which may justify this dependence. However, employing ready-to-use software still holds the potential benefit of abstraction, which can overcome the need to worry about implementation details, saving time, and effort. Abstraction works in our favor if it can be asserted that the software will function as intended, a condition that can be fulfilled by offering enough clarity with regard to its inner workings, such as providing comprehensive and well-founded documentation, practical working examples, and referring to comparative analyses with well-established methods.

In this paper, we characterize the quadruped robot and contact dynamics that will compose our in-house simulator to be used for prototyping locomotion control schemes applied to quadruped robots. The proposed simulator computes the robot dynamics using the Recursive Newton-Euler Algorithm (RNEA) along with the Composite-Rigid-Body Algorithm (CRBA), similar to their definition in ref. [1] but with a few modifications to make certain aspects relevant for contact detection and control more easily accessible. With respect to the contact dynamics, it uses a compliant (nonlinear spring-damper) contact force model alongside stick-slip friction. Evaluation of the proposed simulator was done by implementing a simple PD-independent joint controller to track a desired leg trajectory, synthesized by a foot trajectory generator we proposed in our previous work [20], and comparing characteristic locomotion signals such as the trunk pose and the contact forces, with the same robot and controller implemented using the MuJoCo simulation software. Results showed very similar behavior for both

https://doi.org/10.1017/S0263574724001097 Published online by Cambridge University Press

simulators, especially with regard to the contact detection, despite the significantly different contact models.

1.1. Main contributions

Since our objective is to develop a ready-to-use software that embraces the benefits of abstraction while asserting that it will function as intended, our work's main contribution can be stated as defining a transparent and well-founded simulator that is capable of being used for prototyping locomotion control schemes applied to legged robots. Additionally, with a focus on easing the development of model-based control schemes, we consider two minor contributions: (1) the definition of algorithms derived from the RNEA that separately compute the quadratic velocity and gravity terms, making them easily accessible for use in specialized control schemes (e.g., gravity compensation); and (2) definition of an algorithm to compute the (geometric) Jacobian matrix of the feet, typically required when a task space specification (e.g., desired cartesian coordinates of the feet) must be converted into a joint torque command, by means of a recursive formulation that relies on traversing the support path of each leg of the robot.

2. Definitions and notation

In this work, scalar quantities are depicted with medium italic lowercase Latin or Greek letters (e.g., gravitational acceleration g and friction coefficient μ), vectors are depicted with bold-face upright (or italic for three-dimensional vectors) lowercase Latin or Greek letters (e.g., spatial velocity $\mathbf{v}$, position $\boldsymbol{p}$, and angular velocity $\boldsymbol{\omega}$), and matrices are depicted with bold-face upright uppercase Latin letters (e.g., rotation matrix $\mathbf{R}$). As an exception, bold-face lowercase Latin or Greek letters with a tilde represent the skew-symmetric matrix (as defined in Appendix A) correspondent to that vector (e.g., $\tilde{\boldsymbol{\omega}}$ is the skew-symmetric matrix of vector $\boldsymbol{\omega}$). In the absence of the right superscript, which denotes the frame of reference, consider the variable expressed in its respective body frame. Additionally, $\mathbf{0}$ represents either a zero vector or zero-filled matrix (depending on the context), while $\mathbf{1}$ represents the identity matrix.

2.1. Quadruped robot

The quadruped robot model described in this section is based on the 18 degrees-of-freedom (DOF) torque-controlled MIT Mini Cheetah [16], a small and low-cost robot designed and built for dynamic locomotion with readily available geometrical and inertial data through the MIT Biomimetic Robotics Lab GitHub repository [21]. Figure 1 presents a three-dimensional rendering of the quadruped robot showing its geometrical structure, two of its frames of reference (the trunk frame and the world frame), and the characteristic DOF of the leg. The trunk frame {0} represents a frame of reference attached to the robot's trunk, while the world frame $\{W\}$ is an inertial frame of reference attached to the ground. The four legs are identical, each having a 3-DOF open kinematic chain topology with the base attached to the robot's trunk.

As illustrated in Fig. 1, from the base to the feet of a single leg, we have the abduction/adduction joint, the hip joint, and the knee joint. In this work, each leg is identified by a unique two-letter acronym based on the relative position of its base frame with respect to the trunk frame; that is, LF, RF, LH, and RH, referring to the left foreleg, to the right foreleg, to the left hindleg, and to the right hindleg, respectively.

2.2. Rigid bodies and connectivity

Figure 2a presents a simplified diagram with the corresponding numbers used to represent the 13 rigid bodies ($n_b = 13$) that compose the quadruped robot. The system topology can be expressed as a *connectivity graph*, an undirected and connected graph where the nodes represent bodies and line segments

https://doi.org/10.1017/S0263574724001097 Published online by Cambridge University Press

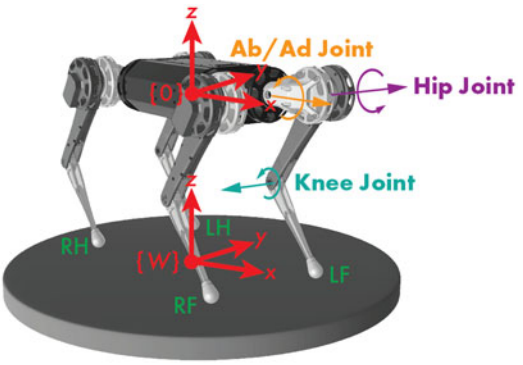


Figure 1. *Three-dimensional rendering of the quadruped robot model with two of its frames of reference and the characteristic degrees-of-freedom of the leg.*

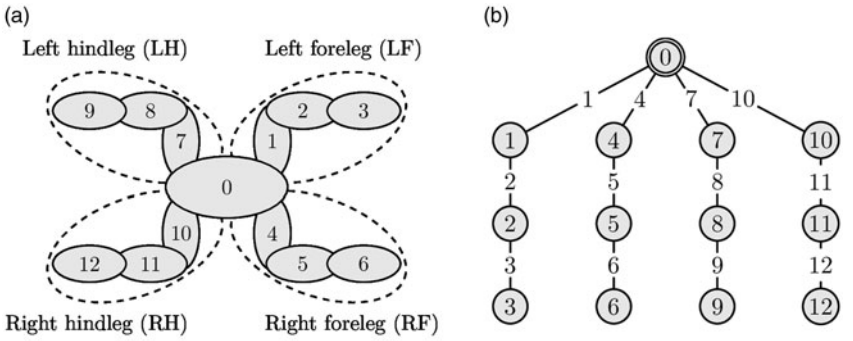


Figure 2. *Simplified diagrams showing the body numbering scheme and the connectivity graph respective to the quadruped robot model.*

represent the joints. The floating-base connectivity graph respective to the quadruped robot is presented in Fig. 2b. Note that the robot has 13 joints numbered from 0 to 12, where joint 0 (not shown in the graph) is a spatial 6-DOF joint while the remaining are all revolute joints.

The floating-base system with open-chain limbs that composes the quadruped robot has a tree-structured kinematic chain where each branch of the tree can be intuitively traversed using recursion. A systematic method to index the bodies in the rigid-body system must be established to convert this recursion into an iterative process, which is more computationally efficient. Based on the connectivity graph, the predecessor array PRED and successor array SUCC can be defined. Starting from joint 1, each element of the predecessor array PRED(i) represents the body preceding joint i, while for the successor array each element SUCC(i) represents the body succeeding joint i. From the connectivity graph (Fig. 2b), the predecessor and successor arrays can be defined as:

$$\begin{aligned} \text{PRED} &= \{0, 1, 2, 0, 4, 5, 0, 7, 8, 0, 10, 11\}, \\ \text{SUCC} &= \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12\}. \end{aligned} \tag{1}$$

When combined, the predecessor and successor arrays describe all the connected pairs of rigid bodies in the tree-structured kinematic chain. One such combination yields the parent array λ. Each element $\lambda(i)$ represents the parent of body i, being calculated as [1]:

$$\lambda(i) = \min\big(\text{PRED}(i), \text{SUCC}(i)\big), \quad i \in [1, n_b - 1]. \tag{2}$$

https://doi.org/10.1017/S0263574724001097 Published online by Cambridge University Press

Applying relation (2) to predecessor and successor arrays defined in (1) results in the parent array:

$$\lambda = \{0, 1, 2, 0, 4, 5, 0, 7, 8, 0, 10, 11\}. \tag{3}$$

Three other useful parameters which can be defined are the support set $\kappa(i)$, the set of all joints in between body i and the root, the child set $\mu(i)$, the set of children of body i, and the subtree set $\nu(i)$, the set of bodies in the subtree rooted at body i.

2.3. Traversing the kinematic chain

Given the robot's tree-structured kinematic chain, traversal is done using either an outward or an inward pass. In an outward pass, we start at the root, going from parent body $\lambda(i)$ to child body i until all the leaf nodes have been reached. For an inward pass, we begin at the leaf nodes and follow the opposite path, going from child body i to its parent body $\lambda(i)$ until all paths have arrived at the root. Only a single outward pass is used for the kinematics, while both an inward and outward pass are required for the dynamics.

2.4. Motion and force transforms

Let $\{A\}$ and $\{B\}$ be coordinate frames fixed to a single rigid body and let $\mathbf{v}_A = (\boldsymbol{\omega}_A, \boldsymbol{v}_A)$ and $\mathbf{v}_B = (\boldsymbol{\omega}_B, \boldsymbol{v}_B)$ be their respective spatial velocity vectors, where $\boldsymbol{v}$ is the linear velocity and $\boldsymbol{\omega}$ is the angular velocity. The spatial velocity $\mathbf{v}_B$ can be computed from $\mathbf{v}_A$ by means of the motion transformation matrix $\mathbf{X}_A^B$, which applies the required translation and rotation procedures to transform the spatial velocity vector from $\{A\}$ to $\{B\}$; that is,

$$\mathbf{v}_B = \mathbf{X}_A^B \mathbf{v}_A. \tag{4}$$

Given the rotation matrix $\mathbf{R}_A^B$ and the absolute position vectors of the origins of each frame $\boldsymbol{p}_{O_A}^A$ and $\boldsymbol{p}_{O_B}^A$ expressed in $\{A\}$, the motion transformation matrix $\mathbf{X}_A^B$ can be expressed as:

$$\mathbf{X}_A^B = \begin{bmatrix} \mathbf{R}_A^B & \mathbf{0}_{3\times 3} \\ -\mathbf{R}_A^B \tilde{\boldsymbol{r}}_{O_A O_B}^A & \mathbf{R}_A^B \end{bmatrix}, \tag{5}$$

where $\mathbf{0}_{3\times 3}$ is a zero-filled 3-by-3 matrix and $\tilde{\boldsymbol{r}}_{O_A O_B}^A$ is the skew-symmetric matrix (Appendix A) associated with the displacement vector $\boldsymbol{r}_{O_A O_B}^A = \boldsymbol{p}_{O_B}^A - \boldsymbol{p}_{O_A}^A$.

With respect to spatial forces, a similar relation can be established. The force transformation matrix $\overline{\mathbf{X}}_A^B$ correlates spatial force vector $\mathbf{f}_B = (\boldsymbol{n}_B, \boldsymbol{f}_B)$ to $\mathbf{f}_A = (\boldsymbol{n}_A, \boldsymbol{f}_A)$, where $\boldsymbol{n}$ is the moment and $\boldsymbol{f}$ is the force acting at origin of their respective frame of reference, such that

$$\mathbf{f}_B = \overline{\mathbf{X}}_A^B \mathbf{f}_A. \tag{6}$$

Following the same assumptions used to compute the motion transformation matrix (5), the force transformation matrix $\overline{\mathbf{X}}_A^B$ can be expressed as:

$$\overline{\mathbf{X}}_A^B = \begin{bmatrix} \mathbf{R}_A^B & -\mathbf{R}_A^B \tilde{\boldsymbol{r}}_{O_A O_B}^A \\ \mathbf{0}_{3\times 3} & \mathbf{R}_A^B \end{bmatrix}. \tag{7}$$

Note that the force transformation matrix can be computed from the motion transformation matrix and vice versa, that is,

$$\overline{\mathbf{X}}_A^B = \left(\mathbf{X}_A^B\right)^{-\mathrm{T}} = \left(\mathbf{X}_B^A\right)^{\mathrm{T}}. \tag{8}$$

https://doi.org/10.1017/S0263574724001097 Published online by Cambridge University Press

Algorithm 1. Kinematics computation

1: **for** $i = 1, 2, \ldots, n_b - 1$ **do**
2: $\quad j \leftarrow \lambda(i)$
3: $\quad \mathbf{X}_j^i \leftarrow \mathbf{X}_{j,i}^i \mathbf{X}_j^{j,i}$
4: $\quad$ **if** $j \neq 0$ **then**
5: $\quad\quad \mathbf{X}_0^i \leftarrow \mathbf{X}_j^i \mathbf{X}_0^j$
6: $\quad$ **end if**
7: $\quad \mathbf{v}_i \leftarrow \mathbf{X}_j^i \mathbf{v}_j + \mathbf{S}_i \dot{\mathbf{q}}_i$
8: **end for**

3. Tree-structured kinematics

The kinematics requires an outward pass to traverse the kinematic chain and calculate the spatial velocity of each body in the system with respect to the root node. Computing the motion transformation matrix (5) can be done in two consecutive stages: (i) a translation to the location of the next active joint, here represented by the fictitious joint frame $\{\lambda(i), i\}$ which represents joint i fixed to body $\lambda(i)$, followed by (ii) a rotation to account for the joint motion (and possible offset). Letting $j = \lambda(i)$ represent the parent body, the first stage transform $\mathbf{X}_j^{j,i}$ can be defined as:

$$\mathbf{X}_j^{j,i} = \begin{bmatrix} \mathbf{1}_3 & \mathbf{0}_{3\times3} \\ -\tilde{\boldsymbol{r}}_{O_j O_i}^j & \mathbf{1}_3 \end{bmatrix}, \tag{9}$$

where $\mathbf{1}_3$ is a 3×3 identity matrix. The following second stage transform $\mathbf{X}_{j,i}^i$ can be expressed as:

$$\mathbf{X}_{j,i}^i = \begin{bmatrix} \mathbf{R}_{j,i}^i & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{R}_{j,i}^i \end{bmatrix}. \tag{10}$$

Note that the first stage yields a constant matrix that depends only on the system's geometry, while the second stage is a function of the corresponding joint variable (e.g., the joint angle for revolute joints). The resulting motion transformation $\mathbf{X}_j^i$ from the parent body frame $\{j\}$ to the current body frame $\{i\}$ is given by:

$$\mathbf{X}_j^i = \mathbf{X}_{j,i}^i \mathbf{X}_j^{j,i}. \tag{11}$$

With this transform, the motion transformation matrix $\mathbf{X}_0^i$ from body 0 (in this case the root node) to each body i can be computed as:

$$\mathbf{X}_0^i = \mathbf{X}_j^i \mathbf{X}_0^j. \tag{12}$$

Given the joint motion subspace matrix $\mathbf{S}_i$, which confines the spatial joint velocity i to the motion subspace defined by the joint DOF, the spatial velocity $\mathbf{v}_i$ of body i can be formulated as:

$$\mathbf{v}_i = \mathbf{X}_j^i \mathbf{v}_j + \mathbf{S}_i \dot{\mathbf{q}}_i, \tag{13}$$

where $\dot{\mathbf{q}}_i$ is the joint velocity vector. In the case of a spatial 6-DOF joint, $\mathbf{S}_i = \mathbf{1}_6$ and the joint velocity is a six-dimensional vector (angular and linear velocity), while for a revolute joint, the joint velocity is a scalar (equivalent to the joint angle derivative) and $\mathbf{S}_i = [\,0\;0\;1\;0\;0\;0\,]^{\mathrm{T}}$ assuming the axis of rotation and its sense matches the z-axis of the body coordinate system.

Algorithm 1 presents the outward pass traversal used to compute each motion transformation matrix $\mathbf{X}_j^i$ and $\mathbf{X}_0^i$ as well as the spatial velocity $\mathbf{v}_i$. Note that the conditional statement in line 4 is required only because the expression in line 3 already handles the transforms relative to the bodies directly connected to the body 0.

https://doi.org/10.1017/S0263574724001097 Published online by Cambridge University Press

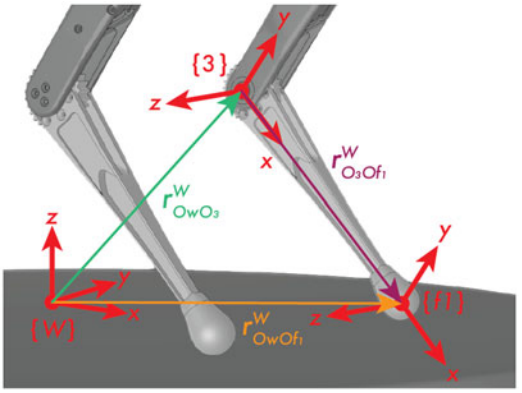


Figure 3. *Foot frame pose components for the left foreleg in the quadruped robot.*

3.1. Transform to the world frame

To correlate the motion transformation matrices to the inertial world frame, we define the motion transformation matrix $\mathbf{X}^0_W$ from the world frame $\{W\}$ to the trunk frame $\{0\}$ as:

$$\mathbf{X}^0_W = \begin{bmatrix} \mathbf{R}^0_W & \mathbf{0}_{3\times3} \\ -\mathbf{R}^0_W \tilde{\boldsymbol{r}}^W_{O_W O_0} & \mathbf{R}^0_W \end{bmatrix}, \tag{14}$$

where $\boldsymbol{r}^W_{O_W O_0}$ represents the displacement vector from the world frame to the trunk frame (cartesian coordinates of the trunk frame origin expressed in the world frame) and $\mathbf{R}^W_0 = \left(\mathbf{R}^0_W\right)^{\mathrm{T}}$ represents the orientation of the trunk frame with respect to the world frame. With $\mathbf{X}^0_W$, the motion transformation matrix from the world frame $\{W\}$ to each body i can be computed by pre-multiplying $\mathbf{X}^i_0$, that is,

$$\mathbf{X}^i_W = \mathbf{X}^i_0 \mathbf{X}^0_W = \begin{bmatrix} \mathbf{R}^i_W & \mathbf{0}_{3\times3} \\ -\mathbf{R}^i_W \tilde{\boldsymbol{r}}^W_{O_W O_i} & \mathbf{R}^i_W \end{bmatrix}. \tag{15}$$

3.2. Foot position

As the contact dynamics algorithm uses the foot position for contact detection and contact force generation, its relation to the joint variables must be established. Let the feet index array φ represent the bodies rigidly attached to each foot frame, where $i = \varphi(k)$ is the body correspondent to foot k, and let the foot frame $\{fk\}$ be located where the ground contact forces will act upon, with its orientation being the same as its respective body frame (as shown in Fig. 3 for the left foreleg). Then, the motion transformation matrix from the body frame $\{i\}$ to the foot frame $\{fk\}$ can be seen as a pure translation, that is,

$$\mathbf{X}^{fk}_i = \begin{bmatrix} \mathbf{1}_3 & \mathbf{0}_{3\times3} \\ -\tilde{\boldsymbol{r}}^i_{O_i O_{fk}} & \mathbf{1}_3 \end{bmatrix}, \tag{16}$$

where $|\boldsymbol{r}^i_{O_i O_{fk}}|$ is equivalent to the length of the last link of leg k. As a result, the motion transformation matrix from the world frame to the foot frame can be expressed as:

$$\mathbf{X}^{fk}_W = \mathbf{X}^{fk}_i \mathbf{X}^i_W = \begin{bmatrix} \mathbf{R}^i_W & \mathbf{0}_{3\times3} \\ -\mathbf{R}^i_W \tilde{\boldsymbol{r}}^W_{O_W O_{fk}} & \mathbf{R}^i_W \end{bmatrix}. \tag{17}$$

https://doi.org/10.1017/S0263574724001097 Published online by Cambridge University Press

Note that the foot displacement vector $\boldsymbol{r}^{W}_{O_W O_{fk}}$ can be extracted from the first block-column, second block-row term of (17) by deconstructing its skew-symmetric matrix which can be formulated as:

$$\tilde{\boldsymbol{r}}^{W}_{O_W O_{fk}} = -\left(\mathbf{R}^{i}_{W}\right)^{\mathrm{T}} \left(-\mathbf{R}^{i}_{W} \tilde{\boldsymbol{r}}^{W}_{O_W O_{fk}}\right). \tag{18}$$

4. Tree-structured dynamics

The proposed simulator builds the quadruped robot's equations of motion (EoM) using the RNEA to compute the bias force (i.e., quadratic velocity, gravitational, and external wrench terms) and the CRBA to compute the inertia matrix. Our approach is very similar to the *inertia matrix method* presented in ref. [1] applied to floating-base robots, with the exception that the RNEA was partitioned into separate modules to isolate the contribution of the quadratic velocity terms, gravitational terms, and external forces, making them more accessible for future control applications. Note that throughout this section, we consider that the velocity transformation matrices and spatial velocities of each body are readily available, being previously computed using Algorithm 1. In this section, unless explicitly stated otherwise, consider index j as representing the parent body index, that is, $j = \lambda(i)$.

4.1. Quadratic velocity terms

As the name implies, the quadratic velocity terms are quadratic (nonlinear) functions of spatial velocity which comprise wrenches in the EoM. In our case, these terms originate only from inertial effects (i.e., centrifugal and Coriolis forces). For body i with spatial velocity $\mathbf{v}_i$, the spatial acceleration $\mathbf{a}_i$ due to the quadratic velocity terms, assuming that the motion subspace matrix $\mathbf{S}_i$ is constant, can be computed as:

$$\mathbf{a}_i = \mathbf{X}^{i}_{j}\mathbf{a}_j + \mathbf{v}_i * \mathbf{S}_i\dot{\mathbf{q}}_i, \tag{19}$$

where $*$ is the twist cross-product operator described in Appendix A. From the spatial velocity $\mathbf{v}_i$ and the spatial acceleration $\mathbf{a}_i$, the wrench $\mathbf{f}_i$, which accounts for the contribution of body i to the quadratic velocity terms, results from

$$\mathbf{f}_i = \mathbf{I}_i\mathbf{a}_i + \mathbf{v}_i \circledast \mathbf{I}_i\mathbf{v}_i, \tag{20}$$

where $\circledast$ is the wrench cross-product operator described in Appendix A and $\mathbf{I}_i$ is the spatial inertia tensor of body i referred to its respective body frame.[1]

The preceding computations can be done in the outward pass alongside the forward kinematics. To obtain the net force acting on each body due to quadratic velocity terms, an additional inward pass is required to propagate wrenches applied to each child body in the kinematic chain. This operation can be interpreted as an update to the quadratic velocity terms, being expressed as:

$$\mathbf{f}_j \leftarrow \mathbf{f}_j + \left(\mathbf{X}^{i}_{j}\right)^{\mathrm{T}}\mathbf{f}_i. \tag{22}$$

With the help of the joint motion subspace matrix $\mathbf{S}_i$, it is possible to convert the quadratic velocity terms into their joint-space counterpart $\mathbf{c}_i$, such that

$$\mathbf{c}_i = \mathbf{S}_i^{\mathrm{T}}\mathbf{f}_i. \tag{23}$$

Algorithm 2 summarizes the joint-space quadratic velocity terms computation, defining the function used to compute the system joint-space quadratic velocity vector $\mathbf{c} = (\mathbf{c}_0, \mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_{12})$. It is worth

[1] The spatial inertia tensor $\mathbf{I}_i$ of body i stores its mass m_i and its inertia tensor $\boldsymbol{I}_i$, both constant terms obtained from the material and geometrical properties of each body, being defined as:

$$\mathbf{I}_i = \begin{bmatrix} \boldsymbol{I}_i & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & m_i\mathbf{1}_3 \end{bmatrix}. \tag{21}$$

https://doi.org/10.1017/S0263574724001097 Published online by Cambridge University Press

Algorithm 2. Quadratic velocity terms

$\mathbf{a}_0 \leftarrow (\mathbf{0}_3, -\tilde{\boldsymbol{\omega}}_0 \boldsymbol{v}_0)$
$\mathbf{f}_0 \leftarrow \mathbf{I}_0\mathbf{a}_0 + \mathbf{v}_0 \circledast \mathbf{I}_0\mathbf{v}_0$
for $i = 1, 2, \ldots, n_b - 1$ **do**
  $j \leftarrow \lambda(i)$
  $\mathbf{a}_i \leftarrow \mathbf{X}_j^i\mathbf{a}_j + \mathbf{v}_i * \mathbf{S}_i\dot{\mathbf{q}}_i$
  $\mathbf{f}_i \leftarrow \mathbf{I}_i\mathbf{a}_i + \mathbf{v}_i \circledast \mathbf{I}_i\mathbf{v}_i$
end for
for $i = n_b - 1, \ldots, 2, 1$ **do**
  $j \leftarrow \lambda(i)$
  $\mathbf{c}_i \leftarrow \mathbf{S}_i^{\mathrm{T}}\mathbf{f}_i$
  $\mathbf{f}_j \leftarrow \mathbf{f}_j + \left(\mathbf{X}_j^i\right)^{\mathrm{T}}\mathbf{f}_i$
end for
$\mathbf{c}_0 \leftarrow \mathbf{S}_0^{\mathrm{T}}\mathbf{f}_0$

noting that at the start of the algorithm, we explicitly initialize the trunk spatial acceleration with only its the velocity-product term.

4.2. Gravitational terms

The procedure required to compute the gravitational forces is very similar to the quadratic velocity terms computation, relying on an outward pass to account for the individual contribution of each body and an inward pass to propagate the succeeding wrenches and generate the joint-space terms. The outward pass involves computing the spatial acceleration $\mathbf{a}_i$ and the resulting wrench $\mathbf{f}_i$ such that

$$\mathbf{a}_i = \mathbf{X}_j^i\mathbf{a}_j, \quad \mathbf{f}_i = \mathbf{I}_i\mathbf{a}_i, \tag{24}$$

while the inward pass updates each wrench exactly as (22), with the joint-space gravity terms being expressed as:

$$\mathbf{g}_i = \mathbf{S}_i^{\mathrm{T}}\mathbf{f}_i. \tag{25}$$

Algorithm 3 summarizes the joint-space gravity terms computation, defining the function used to compute the system joint-space gravitational forces vector $\mathbf{g} = (\mathbf{g}_0, \mathbf{g}_1, \mathbf{g}_2, \ldots, \mathbf{g}_{12})$. Note that as in ref. [1], gravity is not added as an external force but instead is introduced through a fictitious upward acceleration applied to the trunk $\mathbf{a}_0 = -(\mathbf{0}_3, \mathbf{R}_W^0\boldsymbol{g}_0)$, where $\boldsymbol{g}_0 = (0, 0, -9.807)\ \mathrm{m/s^2}$. Since the gravity acceleration vector $\boldsymbol{g}_0$ is expressed in the world frame, it must first be transformed into the body frame to be included in the algorithm.

4.3. External wrenches

Computing the external wrenches (Algorithm 4) is even simpler than the gravitational terms. The outward pass computes the resulting external wrenches $\mathbf{f}_i$ applied to body i at O_i expressed in frame $\{i\}$ from those applied at O_{F_i} expressed in frame $\{F_i\}$, that is, $\mathbf{f}_{\mathrm{ext}i}^{F_i}$. The inward pass performs the wrench propagation and generates the joint-space terms $\boldsymbol{\tau}_{ei}$. In this work, only the external forces originated from the contact dynamics and the active joint torques $\boldsymbol{\tau}$ are accounted for, with a detailed discussion regarding how to compute the former presented in Section 5. Note that the active joint torques $\boldsymbol{\tau}$ are directly added to the EoM using the selection matrix of actuated joints $\mathbf{S}_{\mathrm{act}} = [\, \mathbf{0}_{12\times 6}\ \mathbf{1}_{12}\,]$.

There is an alternative method to compute the joint-space terms $\boldsymbol{\tau}_{\mathrm{e}}$ that relies on multiplying the transpose of the system Jacobian matrix $\mathbf{J}$ (as derived in Appendix B for the contact forces applied to

https://doi.org/10.1017/S0263574724001097 Published online by Cambridge University Press

Algorithm 3. Gravitational terms

$\mathbf{a}_0 \leftarrow -(\mathbf{0}_3, \mathbf{R}_W^0 \boldsymbol{g}_0)$
$\mathbf{f}_0 \leftarrow \mathbf{I}_0 \mathbf{a}_0$
for $i = 1, 2, \ldots, n_b - 1$ **do**
  $j \leftarrow \lambda(i)$
  $\mathbf{a}_i \leftarrow \mathbf{X}_j^i \mathbf{a}_j$
  $\mathbf{f}_i \leftarrow \mathbf{I}_i \mathbf{a}_i$
end for
for $i = n_b - 1, \ldots, 2, 1$ **do**
  $j \leftarrow \lambda(i)$
  $\mathbf{g}_i \leftarrow \mathbf{S}_i^{\mathrm{T}} \mathbf{f}_i$
  $\mathbf{f}_j \leftarrow \mathbf{f}_j + \left(\mathbf{X}_j^i\right)^{\mathrm{T}} \mathbf{f}_i$
end for
$\mathbf{g}_0 \leftarrow \mathbf{S}_0^{\mathrm{T}} \mathbf{f}_0$

Algorithm 4. External wrenches

for $i = 0, 1, \ldots, n_b - 1$ **do**
  $\mathbf{f}_i \leftarrow \overline{\mathbf{X}}_{F_i}^i \mathbf{f}_{\mathrm{ext}i}^{F_i}$
end for
for $i = n_b - 1, \ldots, 2, 1$ **do**
  $j \leftarrow \lambda(i)$
  $\boldsymbol{\tau}_{\mathrm{e}i} \leftarrow \mathbf{S}_i^{\mathrm{T}} \mathbf{f}_i$
  $\mathbf{f}_j \leftarrow \mathbf{f}_j + \left(\mathbf{X}_j^i\right)^{\mathrm{T}} \mathbf{f}_i$
end for
$\boldsymbol{\tau}_{\mathrm{e}0} \leftarrow \mathbf{S}_0^{\mathrm{T}} \mathbf{f}_0$

the feet) to the applied wrenches $\mathbf{f}_{\mathrm{ext}} = (\mathbf{f}_{\mathrm{ext}1}, \mathbf{f}_{\mathrm{ext}2}, \ldots)$, that is, $\boldsymbol{\tau}_{\mathrm{e}} = \mathbf{J}^{\mathrm{T}} \mathbf{f}_{\mathrm{ext}}$. Though this method is not used in our simulator, having the Jacobian matrix available is required to implement control schemes based on impedance control which are commonly employed in legged robotics locomotion controllers.

4.4. Joint-space inertia matrix

The CRBA used to derive the joint-space inertia matrix revolves around the concept of composite-rigid-body inertia, the spatial inertia matrix of the subtree rooted at a given body. For body i, its composite-rigid-body inertia $\mathbf{I}_{ci}$ is the spatial inertia matrix of the subtree rooted at body i represented by the child set $\mu(i)$, being defined as:

$$\mathbf{I}_{ci} = \mathbf{I}_i + \sum_{j \in \mu(i)} \left(\mathbf{X}_j^i\right)^{\mathrm{T}} \mathbf{I}_{cj} \mathbf{X}_j^i. \tag{26}$$

Having defined the composite-rigid-body inertia, each block-row i and block-column j submatrix $\mathbf{H}_{ij}$ from the leg joint-space inertia matrix $\mathbf{H}$ can be calculated as:

$$\mathbf{H}_{ij} \begin{cases} \left(\mathbf{F}_i^j\right)^{\mathrm{T}} \mathbf{S}_j & \text{if } i \in \nu(j), \\ \mathbf{H}_{ji}^{\mathrm{T}} & \text{if } j \in \nu(i), \\ \mathbf{0} & \text{otherwise}, \end{cases} \tag{27}$$

https://doi.org/10.1017/S0263574724001097 Published online by Cambridge University Press

Algorithm 5. Joint-space inertia

$\mathbf{H} \leftarrow \mathbf{0}$
for $i = 0, 1, \ldots, n_b - 1$ **do**
  $\mathbf{I}_{ci} \leftarrow \mathbf{I}_i$
end for
for $i = n_b - 1, \ldots, 2, 1$ **do**
  $j \leftarrow \lambda(i)$
  $\mathbf{I}_{cj} \leftarrow \mathbf{I}_{cj} + \left(\mathbf{X}_j^i\right)^{\mathrm{T}} \mathbf{I}_{ci} \mathbf{X}_j^i$
  $\mathbf{F}_i \leftarrow \mathbf{I}_{ci} \mathbf{S}_i$
  $\mathbf{H}_{ii} \leftarrow \mathbf{S}_i^{\mathrm{T}} \mathbf{F}_i$
  $j \leftarrow i$
  while $\lambda(j) \neq 0$ **do**
    $\mathbf{F}_i \leftarrow \left(\mathbf{X}_{\lambda(j)}^j\right)^{\mathrm{T}} \mathbf{F}_i$
    $j \leftarrow \lambda(j)$
    $\mathbf{H}_{ij} \leftarrow \mathbf{F}_i^{\mathrm{T}} \mathbf{S}_j$
    $\mathbf{H}_{ji} \leftarrow \mathbf{H}_{ij}^{\mathrm{T}}$
  end while
  $\mathbf{F}_i \leftarrow \left(\mathbf{X}_0^j\right)^{\mathrm{T}} \mathbf{F}_i$
end for

where $\mathbf{F}$ is a matrix whose block-columns $\mathbf{F}_i$ are the spatial forces required at the floating base to support unit accelerations about joint variable i [1], with each of these block-columns being defined as:

$$\mathbf{F}_i = \mathbf{I}_{ci} \mathbf{S}_i. \tag{28}$$

Finally, by applying the force transformation matrix to $\mathbf{F}_i$ we obtain $\mathbf{F}_i^j = \overline{\mathbf{X}}_i^j \mathbf{F}_i = \left(\mathbf{X}_j^i\right)^{\mathrm{T}} \mathbf{F}_i$.

As can be seen in Algorithm 5, the outward pass involves storing the spatial inertia terms for each body, while the inward pass computes the resulting composite-rigid-body inertia and the joint-space inertia matrix.

4.5. Joint-space equations of motion

With all its elements defined, the equations of motion can be formulated as:

$$\begin{bmatrix} \mathbf{I}_{c0} & \mathbf{F} \\ \mathbf{F}^{\mathrm{T}} & \mathbf{H} \end{bmatrix} \begin{bmatrix} \mathbf{a}_0 \\ \ddot{\mathbf{q}} \end{bmatrix} + \begin{bmatrix} \mathbf{c}_0 \\ \mathbf{c}_l \end{bmatrix} + \begin{bmatrix} \mathbf{g}_0 \\ \mathbf{g}_l \end{bmatrix} = \begin{bmatrix} \mathbf{0}_6 \\ \boldsymbol{\tau} \end{bmatrix} + \begin{bmatrix} \boldsymbol{\tau}_{e0} \\ \boldsymbol{\tau}_{el} \end{bmatrix}. \tag{29}$$

The first block-row portrays the dynamics of the trunk (floating base), with the second block-row characterizing the leg dynamics. By aggregating common terms as single matrices and vectors, the EoM can be expressed as:

$$\mathbf{M}\mathbf{a} + \mathbf{c} + \mathbf{g} = \mathbf{S}_{\mathrm{act}}^{\mathrm{T}} \boldsymbol{\tau} + \boldsymbol{\tau}_{\mathrm{e}}, \tag{30}$$

where $\mathbf{M}$, $\mathbf{c}$, and $\mathbf{g}$ represent the joint-space inertia matrix, quadratic velocity force vector, and gravitational force vector, respectively, and $\boldsymbol{\tau}_{\mathrm{e}}$ represents the joint-space external forces vector. Vector $\mathbf{a}$ represents the generalized acceleration vector which is composed of the trunk acceleration $\mathbf{a}_0$ along with the leg joint acceleration $\ddot{\mathbf{q}}$, that is, $\mathbf{a} = (\mathbf{a}_0, \ddot{\mathbf{q}})$.

https://doi.org/10.1017/S0263574724001097 Published online by Cambridge University Press

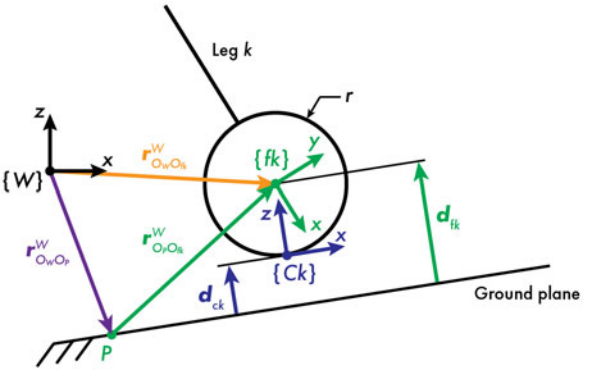


Figure 4. *Required components to compute the sphere-to-plane distance d_{ck} used for contact detection.*

5. Contact dynamics

Compliant contact dynamics is composed of two consecutive steps, contact detection and contact force generation. For contact detection, we use the sphere-to-plane distance d_{ck} between the spherical shell (with radius r, centered at the foot frame origin) and the ground plane to check whether the foot is in contact, with $d_{ck} \leq 0$ implying there is contact between foot k and the ground. As illustrated in Fig. 4, the sphere-to-plane distance d_{ck} can be obtained by computing the signed distance d_{fk} between the ground plane and foot frame origin and subtracting the spherical shell radius. The signed distance d_{fk} is computed as the projection of the foot position vector relative to point P on the ground plane onto the surface normal (defined by the z-axis of the contact frame $\hat{z}_{C_k}^W$), that is,

$$d_{fk} = \left(\hat{z}_{C_k}^W\right)^{\mathrm{T}} \boldsymbol{r}_{O_P O_{fk}}^W, \quad d_{ck} = d_{fk} - r. \tag{31}$$

We chose to model the ground as a horizontal plane, allowing point P to be defined as the origin of the world frame and the ground plane orientation to be equal to the world frame, that is, $\mathbf{R}_W^{C_k} = \mathbf{R}_{C_k}^W = \mathbf{1}_3$. Note that this procedure is also applicable when the ground plane is not horizontal.

Given the point contact assumption, only the linear force element from the external wrench acting on each leg k in contact is nonzero. This contact force is made up of three components, the normal component f_{n} which acts on the z-axis of the contact frame, and frictional components f_{tx} and f_{ty} which act on the x- and y-axis of the contact frame.

Respective to leg k, the contact frame $\{C_k\}$ is located at the point on the foot that is closest to the ground plane. Each contact frame is equivalent to frame $\{F_i\}$ in Algorithm 4, where i is the body respectively attached to foot k (e.g., $i = 3$ for $k = 1$). The force transformation matrix used to transform the contact wrench from $\{C_k\}$ to $\{i\}$ is defined as:

$$\overline{\mathbf{X}}_{C_k}^i = \left(\mathbf{X}_i^{C_k}\right)^{\mathrm{T}} = \left(\mathbf{X}_{fk}^{C_k} \mathbf{X}_i^{fk}\right)^{\mathrm{T}}, \tag{32}$$

where $\mathbf{X}_i^{fk}$ is a known constant factor obtained from (16) and $\mathbf{X}_{fk}^{C_k}$ is the motion transformation matrix from the foot frame to the contact frame, which can be expressed as:

$$\mathbf{X}_{fk}^{C_k} = \begin{bmatrix} \mathbf{R}_{fk}^{C_k} & \mathbf{0}_{3\times 3} \\ \tilde{\boldsymbol{r}}_{O_{C_k} O_{fk}}^{C_k} \mathbf{R}_{fk}^{C_k} & \mathbf{R}_{fk}^{C_k} \end{bmatrix}, \quad \boldsymbol{r}_{O_{C_k} O_{fk}}^{C_k} = \begin{bmatrix} 0 \\ 0 \\ r \end{bmatrix}, \tag{33}$$

where $\mathbf{R}_{fk}^{C_k} = \left(\mathbf{R}_W^{fk} \mathbf{R}_{C_k}^W\right)^{\mathrm{T}}$, with $\mathbf{R}_W^{fk}$ being the rotation matrix which can be extracted from (17) and $\mathbf{R}_{C_k}^W$ the rotation matrix which defines the ground plane orientation (in our case, $\mathbf{R}_{C_k}^W = \mathbf{1}_3$).

https://doi.org/10.1017/S0263574724001097 Published online by Cambridge University Press

5.1. Normal contact force

The normal contact force f_n results from the sum of an elastic and a damping term ($f_n = f_s + f_d$). The elastic term f_s is computed using a nonlinear Hertz model, that is,

$$f_s = -K\delta^e, \tag{34}$$

where K is the stiffness, $\delta = d_{ck}$ is the ground penetration, and e is the force exponent. The damping term f_d can be expressed as:

$$f_d = -c\dot{\delta}, \quad c = c(\delta, c_{max}, w), \tag{35}$$

where $\dot{\delta}$ is the ground penetration rate, c is the effective damping coefficient, c_{max} is the nominal damping coefficient, and w is the ramp-up distance. An effective damping coefficient is used to avoid the natural discontinuity on impact. This term is computed through a smoothstep function used to ramp up the damping coefficient from 0 to c_{max} over the range $\delta \in [-w, 0]$.

As a result of taking the derivative of (31), the ground penetration rate $\dot{\delta}$ is found to be equivalent to the foot linear velocity component in the z-axis of the contact frame, that is,

$$\dot{\delta} = \left(\hat{z}_{C_k}^W\right)^{\mathrm{T}} \boldsymbol{v}_{fk}^W. \tag{36}$$

Note that $\boldsymbol{v}_{fk}^W$ is the linear velocity component of the spatial velocity vector $\mathbf{v}_{fk}^W = (\boldsymbol{\omega}_{fk}^W, \boldsymbol{v}_{fk}^W)$, which can be computed from the spatial velocity of the last link i of leg k such that

$$\mathbf{v}_{fk}^W = \begin{bmatrix} \mathbf{R}_{fk}^W & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{R}_{fk}^W \end{bmatrix} \mathbf{X}_i^{fk} \mathbf{v}_i, \tag{37}$$

where $\mathbf{X}_i^{fk}$ is a known constant factor (16) and $\mathbf{R}_{fk}^W$ is obtained from the inverse of (17).

5.2. Friction contact force

The frictional components f_x and f_y of the contact force are computed using a Coulomb friction model such that

$$\left.\begin{aligned} f_{tx} &= \mu_x f_n \\ f_{ty} &= \mu_y f_n \end{aligned}\right\}, \quad \mu = \mu(v_{rel}, \mu_s, v_s, \mu_d, v_d), \tag{38}$$

where μ is the effective friction coefficient, v_{rel} is the relative velocity (either x or y component of $\boldsymbol{v}_{C_k}$, the linear component of the contact frame velocity defined as $\mathbf{v}_{C_k} = \mathbf{X}_i^{C_k} \mathbf{v}_i$), μ_d is the dynamic friction coefficient, v_d is the stiction-sliding transition velocity, μ_s is the static friction coefficient, and v_s is the velocity where $\mu = \mu_s$. The effective friction coefficient is computed through a smoothstep function used to transition from the negative to the positive static friction coefficient (which occurs when the relative velocity sense is reversed) and from the static to the sliding friction coefficient.

6. Simulation setup

To evaluate the proposed method, the algorithms were implemented in Simulink using a Level-2 MATLAB S-Function block. Although the algorithms do not imply any specific coding language, Simulink was used due to the authors' familiarity with the tool. Given that it can model continuous state behavior including state dynamics, the MATLAB S-Function block is the preferred method to implement continuous-time dynamic systems, such as our rigid-body mechanical system, within Simulink.

https://doi.org/10.1017/S0263574724001097 Published online by Cambridge University Press

6.1. State-space formulation

Implementing the system in Simulink required a state-space representation of the EoM. Let $\mathbf{x}$ be the state vector such that

$$\mathbf{x} = (\mathbf{x}_0, \mathbf{q}, \mathbf{v}_0, \dot{\mathbf{q}}), \tag{39}$$

where $\mathbf{x}_0$ are the spatial coordinates (orientation and position) of the trunk, $\mathbf{q}$ are the leg joint angles, $\mathbf{v}_0$ is the spatial velocity (angular and linear velocity) of the trunk, and $\dot{\mathbf{q}}$ are the leg joint velocities. The state vector time derivative is defined as:

$$\dot{\mathbf{x}} = (\dot{\mathbf{x}}_0, \dot{\mathbf{q}}, \mathbf{a}_0, \ddot{\mathbf{q}}). \tag{40}$$

Note that the derivative of the orientation is not equivalent to the angular velocity, that is, $\dot{\mathbf{x}}_0 \neq \mathbf{v}_0$. Given that the state derivative must be a function of the inputs, of the states, and possibly of time, this must be addressed by performing a conversion from angular velocity (which is part of the state vector) to Euler Angle rate. For a body-fixed ZYX Euler Angle sequence $\boldsymbol{\xi} = (\psi, \theta, \phi)$, the conversion matrix $\mathbf{G}$ from Euler Angle rate $\dot{\boldsymbol{\xi}}$ to angular velocity $\boldsymbol{\omega}_0$ is formulated as:

$$\mathbf{G} = \begin{bmatrix} 0 & -\sin(\psi) & \cos(\theta)\cos(\psi) \\ 0 & \cos(\psi) & \cos(\theta)\sin(\psi) \\ 1 & 0 & -\sin(\theta) \end{bmatrix}, \quad \boldsymbol{\omega}_0 = \mathbf{G}\dot{\boldsymbol{\xi}}. \tag{41}$$

Taking the inverse of this conversion matrix, $\dot{\mathbf{x}}_0$ can be expressed as a function of $\mathbf{v}_0$ such that

$$\dot{\mathbf{x}}_0 = \mathbf{T}\mathbf{v}_0, \quad \mathbf{T} = \begin{bmatrix} \mathbf{G}^{-1} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{1}_3 \end{bmatrix}. \tag{42}$$

With the state vector and its first time derivative defined, by substituting the EoM (30) into the last two block-rows of (40), the state-space equations of the system can be expressed as:

$$\dot{\mathbf{x}} = \begin{bmatrix} \mathbf{T}\mathbf{v}_0 \\ \ddot{\mathbf{q}} \\ \mathbf{M}^{-1}\left(\mathbf{S}_{\text{act}}^{\text{T}}\boldsymbol{\tau} + \boldsymbol{\tau}_{\text{e}} - (\mathbf{c}+\mathbf{g})\right) \end{bmatrix}. \tag{43}$$

6.2. Locomotion control

To make the quadruped robot move, a foot trajectory was established and a control scheme to track it was defined. The former was implemented using a foot trajectory generator defined in our previous work [20], while the latter was defined to be a PD-independent joint controller for each leg implemented using a MATLAB Function block. The foot trajectory generator is comprised of a Central Pattern Generator to synthesize rhythmic signals matching a quadruped gait (in this case, the trot gait) that drive Bézier curves respective to the two stages of locomotion (i.e., stance and swing phase). Given that the resulting curves are planar, inverse kinematics was used to compute the hip and knee joint angles, with the abduction/adduction joints always set to 0. Regarding control, for active joint i, its control law can be expressed as:

$$\tau_i = k_{\text{p}}(q_i - q_{\text{d}i}) + k_{\text{d}}\dot{q}_i. \tag{44}$$

where k_{p} is the proportional gain, k_{d} is the derivative gain, q_i is the measured joint angle, $q_{\text{d}i}$ is the desired joint angle computed from the inverse kinematics of the leg trajectory, and $\dot{q}_i$ is the measured joint velocity. Note that the derivative component was added as a feedforward term to increase stability.

https://doi.org/10.1017/S0263574724001097 Published online by Cambridge University Press

Table I. *Simulation parameters used for our simulator and the MuJoCo physics engine.*

Parameter	Our simulator	MuJoCo
Integration method	Explicit-2nd-order Euler	Implicit-1st-order Euler
Time step	1×10^{-4} s	1×10^{-4} s
Simulation time	10 s	10 s
Dynamic friction coeff. (μ_d)	2.0	1.6
Stiction-sliding trans. vel. (v_d)	0.2 mm/s	–
Static friction coeff. (μ_s)	2.0	1.6
Stiction velocity (v_s)	0.1 mm/s	–
Contact stiffness (K)	1.0×10^4 N/m	1.6×10^3 N/m[†]
Contact exponent (e)	2.2	–
Nom. contact damping (c_{max})	10 N/(m/s)	100 N/(m/s)[†]
Ramp-up distance (w)	0.1 mm	–
Proportional gain (k_p)	40.0 N.m	40.0 N.m
Derivative gain (k_d)	1.0 N.m.s	1.0 N.m.s

[†] For MuJoCo, stiffness and damping terms have a distinct meaning from those defined for our simulator's contact model. MuJoCo employs these parameters alongside an impedance parameter [18] to compute the regularization term used to incorporate the geometric constraints that model the contact dynamics.

6.3. Baseline behavior

A comparative analysis with same robot implemented using the readily available physics engine MuJoCo [18] was performed to evaluate the implementation of the algorithms. While the quadruped robot model was the same in both software, the contact model was not. Unlike the compliant method described in Section 5, MuJoCo uses an augmented non-smooth method that models contact as geometric constraints with an added regularization term. Even though different methods were used to model the contact dynamics, this analysis serves as a way to show if the system behavior was at least similar. Table I summarizes the simulation parameters used for our simulator and MuJoCo, including the integration method, time step, simulation time, contact parameters, and controller gains. Note that both approaches use a three-component contact dimension (point contact) and have an elliptic friction cone shape.

Since the simulation is composed of the robot and the contact dynamics, aspects related to each of these elements were analyzed. The trunk pose was used to represent the robot behavior as its observed state reflects the states from all the legs and their interaction with the ground. Regarding the contact dynamics, contact activation, a binary signal where 1 (one) indicates when contact forces are nonzero, and the measured ground reaction forces from a single leg were used. To quantify the disparity between MuJoCo and our simulator, both the respective time series and mean absolute error of each scalar component were used.

7. Results

Figure 5 shows the trunk's position and orientation obtained in simulation. Very similar trunk pose was observed in both simulators, registering (63, 17, 4.6) mm mean absolute error for the position components and (1.5, 1.3, 2.7) deg for the Euler Angles. Despite using the same robot and controller, the trunk pose was not expected to be a perfect match due to the distinct methods used to model the contact dynamics, which affect how contact force generation is performed and, consequently, how a particular motion evolves in time.

Figure 6 shows the left forelimb's contact activation and the vertical (normal) ground reaction force. Very similar activation time was observed, with MuJoCo having a few spurious impulsive signals at 2.8, 4.6, 5.0, 5.9, and 6.4 s. Regarding the ground reaction forces, our approach resulted in slightly higher peak values, yielding a mean absolute contact force error of 5.65 N. The higher peak values can be

https://doi.org/10.1017/S0263574724001097 Published online by Cambridge University Press

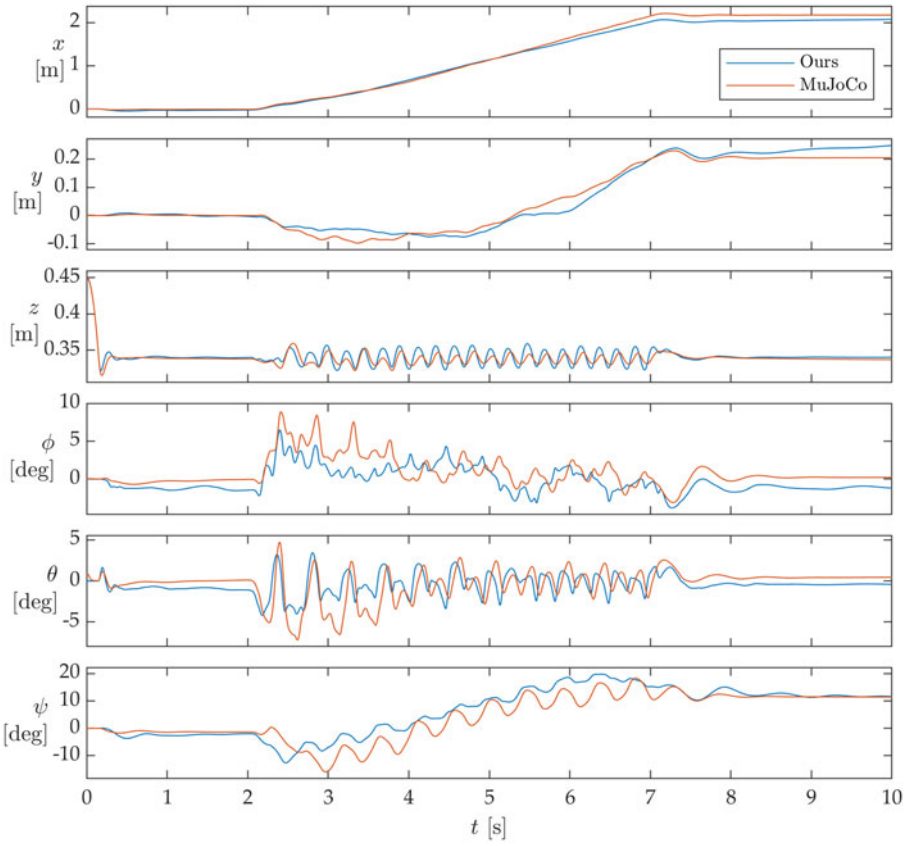


Figure 5. *Quadruped robot trunk position and orientation obtained in simulation.*

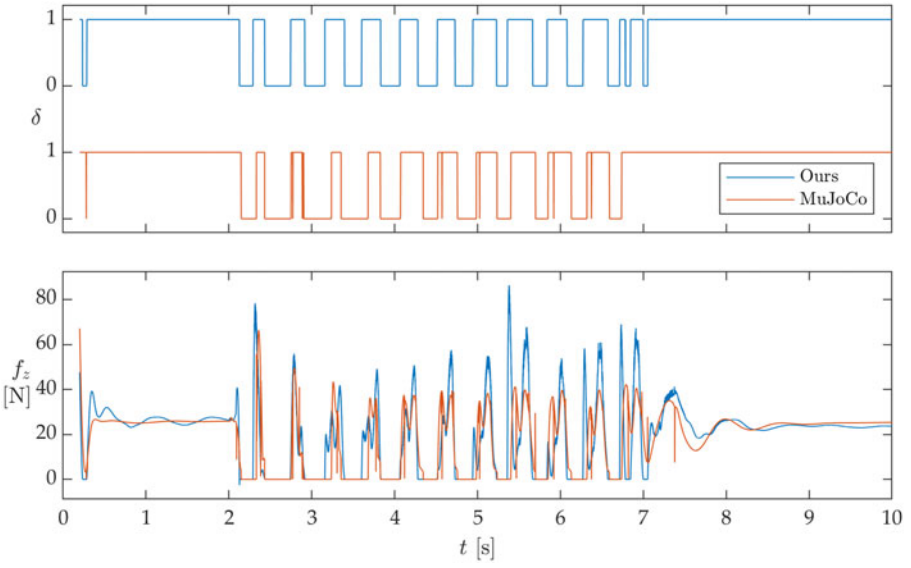


Figure 6. *Contact activation and vertical contact forces applied to the left foreleg obtained in simulation.*

https://doi.org/10.1017/S0263574724001097 Published online by Cambridge University Press

traced to the higher effective contact stiffness and lower damping defined for our contact model. Note that in MuJoCo, the stiffness and damping terms presented in Table I are used to compute a reference acceleration term, which is only a single component of the cost function that is optimized to generate the constraint forces and update the joint acceleration while accounting for the constraints. Also, with regard to processing time, simulating 10 s took 300 s using our implementation, while it took only 20 s with MuJoCo. This order of magnitude discrepancy can have a multitude of reasons, from the intrinsic overhead of the Level-2 MATLAB S-Function to the limitations of the contact dynamics method when using a high stiffness coefficient. Since simulation speed (though always desirable) was not a critical feature for this work, we restricted ourselves to only noting the issue here until we can address it in future works.

8. Conclusion

In this paper, we characterized the quadruped robot and contact dynamics that will compose our in-house simulator. The robot dynamics was derived using the RNEA and CRBA, ensuring each component of the bias term was explicit to allow for ease of usage in control systems. Contact dynamics was modeled using a compliant method composed of a nonlinear spring-damper model for the normal component and a smooth stick-slip model for the friction components. We used a simple PD-independent joint controller to track a desired leg trajectory and allow for the robot locomotion to be evaluated. When comparing the characteristic locomotion signals with the same robot and controller implemented using MuJoCo, we observed similar behavior for both simulators despite the significantly different contact models. Though activation timing was a very close match, the distinct methods used to generate the contact forces ended up causing differences in the contact forces and the resulting trunk pose. Despite our implementation being an order of magnitude slower than MuJoCo, the fact that it resulted in similar behavior shows that the underlying mechanics were sound, allowing for the next steps to be focused on improving aspects such as simulation speed. Future works can try to optimize the existing implementation to increase computational speed while maintaining an acceptable level of accuracy, making it applicable for both rapid prototyping and real-time legged robot control.

Author contributions. Conceptualization, provision of resources, software development, and data collection were carried out by Mauricio Becerra-Vargas. Data curation and analysis, visualization, and manuscript revision were performed by Daniel Teixeira de Paula. Supervision and project administration were done by Eduardo Paciencia Godoy. All authors have read and approved the final manuscript.

Financial support. This research received no specific grant from any funding agency, commercial, or not-for-profit sectors.

Competing interests. The authors declare no conflicts of interest exist.

Ethical approval. None.

References

[1] R. Featherstone. *Rigid Body Dynamics Algorithms* (Springer, New York, NY, 2008).
[2] G. Bledt, M. J. Powell, B. Katz, J. Di Carlo, P. M. Wensing and S. Kim, "MIT Cheetah 3: Design and Control of a Robust, Dynamic Quadruped Robot," **In:** *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (Madrid, Spain, IEEE, 2018) pp. 2245–2252.
[3] J. Di Carlo, P. M. Wensing, B. Katz, G. Bledt and S. Kim, "Dynamic Locomotion in the MIT Cheetah 3 Through Convex Model-Predictive Control," **In:** *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (Madrid, Spain, IEEE, 2018) pp. 1–9.
[4] R. Grandia, F. Farshidian, R. Ranftl and M. Hutter, "Feedback MPC for Torque-Controlled Legged Robots," **In:** *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (Macau, China, IEEE, 2019) pp. 4730–4737.
[5] D. Kim, J. D. Carlo, B. Katz, G. Bledt and S. Kim, Highly dynamic quadruped locomotion via whole-body impulse control and model predictive control, (2019).

https://doi.org/10.1017/S0263574724001097 Published online by Cambridge University Press

[6] C. Mastalli, I. Havoutis, M. Focchi, D. G. Caldwell and C. Semini, "Motion planning for quadrupedal locomotion: Coupled planning, terrain mapping, and whole-body control," *IEEE Trans Robot* **36**(6), 1635–1648 (2020).
[7] M. V. Minniti, R. Grandia, F. Farshidian and M. Hutter, "Adaptive CLF-MPC with application to quadrupedal robots," *IEEE Robot Automa Lett* **7**(1), 565–572 (2022).
[8] M. Neunert, M. Stauble, M. Giftthaler, C. D. Bellicoso, J. Carius, C. Gehring, M. Hutter and J. Buchli, "Whole-body nonlinear model predictive control through contacts for quadrupeds," *IEEE Robot Automa Lett* **3**(3), 1458–1465 (2018).
[9] N. Rathod, A. Bratta, M. Focchi, M. Zanon, O. Villarreal, C. Semini and A. Bemporad, "Model predictive control with environment adaptation for legged locomotion," *IEEE Access* **9**, 145710–145727 (2021).
[10] M. Giftthaler, M. Neunert, M. Stäuble and J. Buchli, "The Control Toolbox - An Open-Source C++ Library for Robotics, Optimal and Model Predictive Control," **In:** *IEEE International Conference on Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR)*, (Brisbane, QLD, Australia, IEEE, 2018) pp. 123–129.
[11] M. Frigerio, J. Buchli and D. G. Caldwell, "Code Generation of Algebraic Quantities for Robot Controllers," **In:** *IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Vilamoura-Algarve, Portugal, IEEE, 2012) pp. 2346–2351.
[12] M. Giftthaler, M. Neunert, M. Stäuble, M. Frigerio, C. Semini and J. Buchli, "Automatic differentiation of rigid body dynamics for optimal control and estimation," *Adv Robotics* **31**(22), 1225–1237 (2017).
[13] C. Semini, N. G. Tsagarakis, E. Guglielmino, M. Focchi, F. Cannella and D. G. Caldwell, "Design of HyQ: A hydraulically and electrically actuated quadruped robot," *Proceed Inst Mech Eng, Part I: J Syst Control Eng* **225**(6), 831–849 (2011).
[14] M. Hutter, C. Gehring, D. Jud, A. Lauber, C. D. Bellicoso, V. Tsounis, J. Hwangbo, K. Bodie, P. Fankhauser, M. Bloesch, R. Diethelm, S. Bachmann, A. Melzer and M. Hoepflinger, "ANYmal - A Highly Mobile and Dynamic Quadrupedal Robot," **In:** *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, (Daejeon, Korea (South), IEEE, 2016) pp.38–44.
[15] S. Fahmi, C. Mastalli, M. Focchi and C. Semini, "Passive whole-body control for quadruped robots: Experimental validation over challenging terrain," *IEEE Robot Automa Lett* **4**(3), 2553–2560 (2019).
[16] B. Katz, J. D. Carlo and S. Kim, "Mini Cheetah: A Platform for Pushing the Limits of Dynamic Quadruped Control," **In:** *International Conference on Robotics and Automation (ICRA)*, (Montreal, QC, Canada, IEEE, 2019) pp.6295–6301.
[17] M. L. Felis, "RBDL: An efficient rigid-body dynamics library using recursive algorithms," *Auton Robot* **41**(2), 495–511 (2017).
[18] E. Todorov, T. Erez and Y. Tassa, "MuJoCo: A Physics Engine for Model-Based Control," **In:** *IEEE/RSJ International Conference on Intelligent Robots and Systems*, (Vilamoura-Algarve, Portugal, IEEE, 2012) pp. 5026–5033.
[19] J. Hwangbo, J. Lee and M. Hutter, "Per-contact iteration method for solving contact dynamics," *IEEE Robot Automa Lett* **3**(2), 895–902 (2018).
[20] D. T. De Paula, E. P. Godoy and M. Becerra-Vargas, "Towards Dynamic Quadruped Locomotion: Development of a CPG-driven Foot Trajectory Generator," **In:** *Proceedings of the 30th Mediterranean Conference on Control and Automation*, (Vouliagmeni, Greece, IEEE, 2022) pp. 988–993.
[21] MIT Biomimetic Robotics Lab, Cheetah-software, (2019). https://github.com/mit-biomimetics/Cheetah-Software.

Appendix

A. Definitions

A.1. Skew-symmetric matrix

Let $\boldsymbol{v}$ be a vector with components v_1, v_2, and v_3. The skew-symmetric matrix $\tilde{\boldsymbol{v}}$ is defined as:

$$\tilde{\boldsymbol{v}} = \begin{bmatrix} 0 & -v_3 & v_2 \\ v_3 & 0 & -v_1 \\ -v_2 & v_1 & 0 \end{bmatrix}. \tag{A1}$$

A.2. Twist cross product

Given the spatial velocity $\mathbf{v} = (\boldsymbol{\omega}, \boldsymbol{v})$ composed of the angular velocity $\boldsymbol{\omega}$ and linear velocity $\boldsymbol{v}$, the twist cross-product operator $\mathbf{v}*$ can be defined as:

$$\mathbf{v}* = \begin{bmatrix} \tilde{\boldsymbol{\omega}} & \mathbf{0}_{3\times3} \\ \tilde{\boldsymbol{v}} & \tilde{\boldsymbol{\omega}} \end{bmatrix}. \tag{A2}$$

https://doi.org/10.1017/S0263574724001097 Published online by Cambridge University Press

A.3. Wrench cross product

Given the spatial force $\mathbf{f} = (\boldsymbol{n}, \boldsymbol{f})$ composed of the couple moment $\boldsymbol{n}$ and linear force $\boldsymbol{f}$, the wrench cross-product operator $\mathbf{f}\circledast$ can be defined as:

$$\mathbf{f}\circledast = -(\mathbf{f}*)^{\mathrm{T}} = \begin{bmatrix} \tilde{\boldsymbol{n}} & \tilde{\boldsymbol{f}} \\ \mathbf{0}_{3\times 3} & \tilde{\boldsymbol{n}} \end{bmatrix}. \tag{A3}$$

B. Jacobian matrix computation

Let the geometric Jacobian matrix of the feet $\mathbf{J}_{\text{feet}}$ be a map from the joint velocities $\mathbf{w} = (\mathbf{v}_0, \dot{\mathbf{q}})$ to the feet frame velocities $\mathbf{v}_{\text{feet}}$, such that $\mathbf{v}_{\text{feet}} = \mathbf{J}_{\text{feet}}\mathbf{w}$, and let its transpose be a map from the forces applied to the foot frames $\mathbf{f}_{\text{feet}}$ to the joint-space torques $\boldsymbol{\tau}_{\text{e}}$, such that $\boldsymbol{\tau}_{\text{e}} = \mathbf{J}_{\text{feet}}{}^{\mathrm{T}}\mathbf{f}_{\text{feet}}$. For n_{J} joints and m feet, the geometric Jacobian matrix of the feet $\mathbf{J}_{\text{feet}}$ is defined as:

$$\mathbf{J}_{\text{feet}} = \begin{bmatrix} \mathbf{J}_{\text{feet},1} \\ \vdots \\ \mathbf{J}_{\text{feet},k} \\ \vdots \\ \mathbf{J}_{\text{feet},m} \end{bmatrix}, \quad \mathbf{J}_{\text{feet},k} = \begin{bmatrix} \mathbf{J}_{\text{feet},k1} & \cdots & \mathbf{J}_{\text{feet},kn_{\text{J}}} \end{bmatrix}, \quad k \in [1, m], \tag{B1}$$

where each block-row $\mathbf{J}_{\text{feet},k}$ is associated with the spatial velocity $\mathbf{v}_{fk}$ of foot k such that $\mathbf{v}_{fk} = \mathbf{J}_{\text{feet},k}\mathbf{w}$. Given that each foot is attached to a specific body in the kinematic tree, we define a feet index array φ that stores the bodies correspondent to each foot, where $\varphi(k)$ is the body index respective to foot k.

The spatial velocity $\mathbf{v}_{fk}$ can be computed from the spatial velocity of its attached body $\mathbf{v}_i$, where $i = \varphi(k)$, which can be expressed as a function of the joint velocities, that is,

$$\mathbf{v}_{fk} = \mathbf{X}_i^{fk}\mathbf{v}_i = \mathbf{X}_i^{fk}\sum_{j\in\kappa(i)} \mathbf{X}_j^i\mathbf{S}_j\dot{\mathbf{q}}_j, \tag{B2}$$

where $\kappa(i)$ is the support set of body i (which contains all the joints between body i and the root node, in this case the ground). Note that index j is not necessarily $\lambda(i)$; therefore, not all motion transformation matrices will be available from the standard kinematics loop presented in Algorithm 1. Here, we assume that the necessary modifications were done and that all motion transformation matrices are available.

From (B2), we can infer that each term multiplying a joint velocity $\dot{\mathbf{q}}_j$ is equivalent to a block-column $\mathbf{J}_{\text{feet},kj}$ of (B1), that is,

$$\mathbf{J}_{\text{feet},kj} = \mathbf{X}_i^{fk}\left(\mathbf{X}_j^i\mathbf{S}_j\right). \tag{B3}$$

This expression suggests that we can compute the Jacobian matrix by traversing the support path defined for each foot as presented in Algorithm 6. Note that the first transform $\mathbf{X}_i^{fk}$ is applied only once per foot to the entire block-row matrix $\mathbf{J}_{\text{feet},k}$. If the measured contact forces or the desired spatial velocities are expressed in the contact frame $\{C_k\}$, transform $\mathbf{X}_{fk}^{C_k}$ defined in (33) can be pre-multiplied to (B3).

https://doi.org/10.1017/S0263574724001097 Published online by Cambridge University Press

Algorithm 6. Jacobian computation

$\mathbf{J}_{\text{feet}} = \mathbf{0}$
for $k = 1, 2, \ldots, m$ **do**
    $i \leftarrow \varphi(k)$
    for $j \in \kappa(i)$ **do**
        $\mathbf{J}_{\text{feet},kj} \leftarrow \mathbf{X}_j^i \mathbf{S}_j$
    end for
    $\mathbf{J}_{\text{feet},k} \leftarrow \mathbf{X}_i^{fk} \mathbf{J}_{\text{feet},k}$
end for

Cite this article: D. Teixeira de Paula, E. P. Godoy and M. Becerra-Vargas (2024). "Dynamic modeling and simulation of a torque-controlled spatial quadruped robot", Robotica **42**, 2761–2780. https://doi.org/10.1017/S0263574724001097

https://doi.org/10.1017/S0263574724001097 Published online by Cambridge University Press