Inverse Dynamics vs. Forward Dynamics in Direct Transcription Formulations for Trajectory Optimization

Henrique Ferrolho[1], Vladimir Ivan[1], Wolfgang Merkt[2], Ioannis Havoutis[2], Sethu Vijayakumar[1]

***Abstract*— Benchmarks of state-of-the-art rigid-body dynamics libraries have reported better performance for solving the inverse dynamics problem than the forward alternative. Those benchmarks encouraged us to question whether this computational advantage translates to direct transcription formulations, where calculating the rigid-body dynamics and their derivatives often accounts for a significant share of computation time. In this work, we implement an optimization framework where both approaches for enforcing the system dynamics are available. We evaluate the performance of each approach for systems of varying complexity, and for domains with rigid contacts. Our tests revealed that formulations employing inverse dynamics converge faster, require less iterations, and are more robust to coarse problem discretization. These results suggest that inverse dynamics should be the preferred approach to enforce nonlinear system dynamics in simultaneous methods, such as direct transcription.**

I. Introduction

Direct transcription [1] is an effective approach to formulate and solve trajectory optimization problems. Direct transcription works by converting the original trajectory optimization problem (which is *continuous* in time) into a numerical optimization problem that is *discrete* in time, and which in turn can be solved using an off-the-shelf nonlinear programming (NLP) solver. First, the trajectory is divided into segments and then, at the beginning of each segment, the system state and control inputs are explicitly discretized—these are the decision variables of the optimization problem. Due to this discretization approach, direct transcription falls under the class of *simultaneous* methods. Finally, a set of mathematical constraints is defined to enforce path constraints and boundary constraints, e.g., initial and final conditions, or intermediate goals. In dynamic trajectory optimization, there exists a specific set of constraints dedicated to enforce the equations of motion of the system: the so-called *defect constraints*. This paper focuses on different ways of defining these constraints, and the implications of the choice of each formulation.

The dynamics defects are one of the most important constraints in optimization problems when planning highly dynamic motions for complex systems, such as legged robots. Satisfaction of these constraints ensures that the computed motion is reliable and physically consistent with the nonlinear dynamics of the system. The dynamics defect constraints are usually at the very core of optimal control formulations, and require computing rigid-body dynamics and their derivatives—which account for a significant portion of the optimization computation time. Therefore, it is of utmost importance to use an algorithm that allows to compute the dynamics of the system reliably, while achieving low computational time.

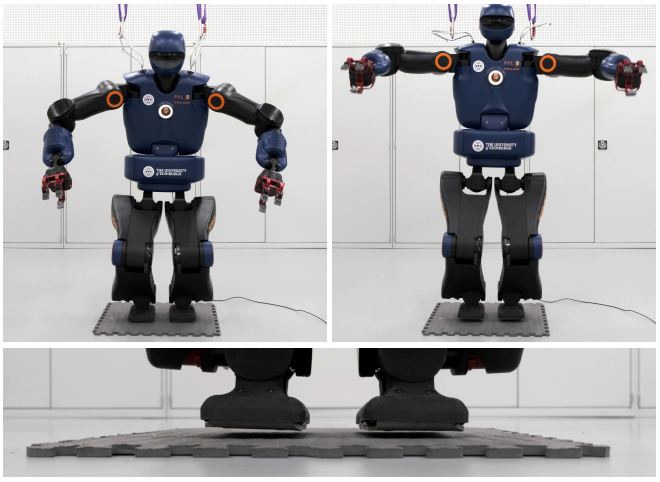


Fig. 1: Snapshots of the humanoid TALOS [2] jumping.

In the study of the dynamics of open-chain robots, the *forward dynamics* problem determines the joint accelerations resultant from a given set of joint forces and torques applied at a given state. On the other hand, the *inverse dynamics* problem determines the joint torques and forces required to meet some desired joint accelerations at a given state. In trajectory optimization, most direct formulations use forward dynamics to enforce dynamical consistency [3]. However, for most dynamics libraries, benchmark results have shown that implementations to solve the *inverse dynamics* problem (e.g., the Recursive Newton-Euler Algorithm) outperform implementations for solving the *forward dynamics* problem (e.g., the Articulated Body Algorithm) [4], [5]. For example, for the humanoid robot TALOS [2], the library Pinocchio [6] solves the inverse dynamics problem in just $4\,\mu s$, while the forward dynamics problem takes $10\,\mu s$. The benchmark difference is observed for a single call to either method computing the dynamics of the system, but it motivated us to question whether the computational advantage of inverse dynamics would translate to trajectory optimization problems—where the dynamics problem needs to be solved several times while computing the defect constraints. Moreover, there is biological evidence suggesting that inverse dynamics

[1]School of Informatics, University of Edinburgh, United Kingdom.
[2]Oxford Robotics Institute, University of Oxford, United Kingdom.
This research is supported by the EPSRC UK RAI Hub for Offshore Robotics for Certification of Assets (ORCA, grant reference EP/R026173/1), EU H2020 project Memory of Motion (MEMMO, project ID: 780684), and the EPSRC Centre for Doctoral Training in Robotics and Autonomous Systems (EPSRC, grant reference EP/L016834/1).
Email address: henrique.ferrolho@ed.ac.uk

is employed by the nervous system to generate feedforward commands [7], while other studies support the existence of a forward model [8]—which increases our interest in the topic.

In this work, we present a trajectory optimization framework for domains with rigid contacts, using a direct transcription approach. Particularly, our formulation allows to define dynamics defect constraints employing either forward dynamics or inverse dynamics. We defined a set of evaluation tasks across different classes of robot platforms, including fixed- and floating-base systems, with point and surface contacts. Our results showed that inverse dynamics leads to significant improvements in computational performance when compared to forward dynamics—supporting our initial hypothesis.

II. Related Work

RigidBodyDynamics.jl (RBD.jl) [9], RBDL [10], Pinocchio [6], and RobCoGen [11] are all state-of-the-art software implementations of key rigid-body dynamics algorithms. Recently, Neuman *et al.* [5] benchmarked these libraries and revealed interesting trends. One such trend is that implementations of inverse dynamics algorithms have faster runtimes than forward dynamics.[1] Koolen and Deits [4] also compared RBD.jl with RBDL, and their results showed that solving inverse dynamics was at least two times faster than solving forward dynamics for the humanoid robot Atlas. Both of these studies only consider computation time of rigid-body dynamics; they do not provide insight into how these algorithms perform when used in trajectory optimization.

Lee *et al.* [12] have proposed Newton and quasi-Newton algorithms to optimize motions for serial-chain and closed-chain mechanisms using inverse dynamics. However, they used relatively simple mechanisms for which analytic derivatives can be obtained. In this work, we are interested in dynamic motions of complex mechanisms in domains with contact, e.g. humanoids with a floating-base, and for which the derivation of analytic derivatives is an error-prone process, involving significant effort.

In the same spirit, Erez and Todorov [13] generated a running gait for a humanoid based on inverse dynamics under external contacts. This method allowed them to formulate an *unconstrained* optimization where all contact states can be considered equally, contact timings and locations are optimized, and reaction forces are computed using a smooth and invertible contact model based on convex optimization. However, their approach requires "helper forces", as well as tuning of contact smoothness and of the penalty parameters on the helper forces to achieve reasonable-looking behavior. In contrast, our approach does not require helper forces or any tuning whatsoever; we consider contact forces as decision variables (an approach known as planning "through contact") and model contacts rigidly. Another difference is that we formulate a *constrained* optimization problem and enforce the nonlinear system dynamics with hard constraints, which results in high-fidelity motions. This is especially important for deploying motions on real hardware, where dynamic consistency and realism are imperative. The main focus of our paper is not the contact problem, and we assume contact locations and contact times are known *a priori*—an assumption not made in [13].

Finally, and to the best of our knowledge, there is no current work directly comparing inverse dynamics against forward dynamics in the context of direct methods. Posa *et al.* [14] also identified that a formal comparison is important, but missing so far. They argued that one of the reasons for this was that the field had not yet agreed upon a set of canonical and hard problems. In this paper, we tackle this issue, and compare the two approaches on robots of different complexity on a set of dynamic tasks.

A. Statement of Contributions

The main contributions of this work are:

1) A direct transcription formulation that uses *inverse dynamics* to enforce physical consistency, for constrained trajectory optimization in domains with rigid contacts.
2) Evaluation of the performance of direct transcription formulations using either forward or inverse dynamics, for different classes of robot platforms: a fixed-base manipulator, a quadruped, and a humanoid.
3) Comparison of performance for different linear solvers, and across strategies to handle the barrier parameter of the interior point optimization algorithm.

We have also validated the computed trajectories in full-physics simulation and in real-life hardware experiments.

III. Trajectory Optimization

A. Robot Model Formulation

We formulate the model of a legged robot as a free-floating base B to which limbs are attached. The motion of the system can be described with respect to (w.r.t.) a fixed inertial frame I. We represent the position of the free-floating base w.r.t. the inertial frame, and expressed in the inertial frame, as ${}_I\boldsymbol{r}_{IB} \in \mathbb{R}^3$; and the orientation of the base as $\boldsymbol{\psi}_{IB} \in \overline{\mathbb{R}}^3$, using modified Rodrigues parameters (MRP) [15], [16].[2,3] The joint angles describing the configuration of the limbs of the robot (legs or arms) are stacked in a vector $\boldsymbol{q}_j \in \mathbb{R}^{n_j}$, where n_j is the number of actuated joints. The generalized coordinates vector $\boldsymbol{q}$ and the generalized velocities vector $\boldsymbol{v}$ of this floating-base system may therefore be written as

$$\boldsymbol{q} = \begin{bmatrix} {}_I\boldsymbol{r}_{IB} \\ \boldsymbol{\psi}_{IB} \\ \boldsymbol{q}_j \end{bmatrix} \in \mathbb{R}^3 \times \overline{\mathbb{R}}^3 \times \mathbb{R}^{n_j}, \quad \boldsymbol{v} = \begin{bmatrix} \boldsymbol{\nu}_B \\ \dot{\boldsymbol{q}}_j \end{bmatrix} \in \mathbb{R}^{n_v}, \quad (1)$$

where the twist $\boldsymbol{\nu}_B = \begin{bmatrix} {}_I\boldsymbol{v}_B & {}_B\boldsymbol{\omega}_{IB} \end{bmatrix}^\top \in \mathbb{R}^6$ encodes the linear and angular velocities of the base B w.r.t. the inertial frame expressed in the I and B frames, and $n_v = 6 + n_j$.

[1] RobCoGen is an exception to this observation as it implements a hybrid dynamics solver which has a higher computational cost, and is significantly different from the implementations used by the other libraries.

[2] $\overline{\mathbb{R}} = \mathbb{R} \cup \{-\infty, +\infty\}$ is the *affinely extended set of real numbers*. [16]

[3] The MRP encode a 3D rotation with the stereographic projection of a Hamiltonian unit quaternion. This representation is a particularly good choice for differentiation and optimization, because the derivatives of the rotation matrix w.r.t. the MRP are rational functions.

For fixed-base manipulators, the generalized vectors of coordinates and velocities can be simplified to $\boldsymbol{q} = \boldsymbol{q}_j \in \mathbb{R}^{n_j}$ and $\boldsymbol{v} = \dot{\boldsymbol{q}}_j \in \mathbb{R}^{n_j}$, due to the absence of a free-floating base.

B. Problem Formulation

We tackle the motion planning problem using trajectory optimization; more specifically, using a *direct transcription* approach. The original problem is *continuous* in time, so we start by converting it into a numerical optimization problem that is *discrete* in time. For that, we divide the trajectory into N equally spaced segments

$$t_I = t_1 < t_2 < \cdots < t_M = t_F, \tag{2}$$

where t_I and t_F are the start and final instants, respectively. This division results in $M = N + 1$ discrete *mesh points*, for each of which we explicitly discretize the states of the system, as well as the control inputs. Let $x_k \equiv x(t_k)$ and $u_k \equiv u(t_k)$ be the values of the state and control variables at the k-th mesh point. We treat $x_k \triangleq \{\boldsymbol{q}_k, \boldsymbol{v}_k\}$ and $u_k \triangleq \{\boldsymbol{\tau}_k, \boldsymbol{\lambda}_k\}$ as a set of NLP variables, and formulate the trajectory optimization problem as

$$\begin{aligned} \text{find} \quad & \boldsymbol{\xi} \\ \text{subject to} \quad & x_{k+1} = f(x_k, u_k) \\ & x_k \in \mathcal{X} \\ & u_k \in \mathcal{U}, \end{aligned} \tag{3}$$

where $\boldsymbol{\xi}$ is the vector of decision variables, $x_{k+1} = f(x_k, u_k)$ is the state transition function incorporating the nonlinear system dynamics, and $\mathcal{X}$ and $\mathcal{U}$ are sets of feasible states and control inputs enforced by a set of equality and inequality constraints. The decision variables vector $\boldsymbol{\xi}$ results from aggregating the generalized coordinates, generalized velocities, joint torques, and contact forces at every mesh point:[4]

$$\boldsymbol{\xi} \triangleq \{\boldsymbol{q}_1, \boldsymbol{v}_1, \boldsymbol{\tau}_1, \boldsymbol{\lambda}_1, \cdots, \boldsymbol{q}_N, \boldsymbol{v}_N, \boldsymbol{\tau}_N, \boldsymbol{\lambda}_N, \boldsymbol{q}_M, \boldsymbol{v}_M\}. \tag{4}$$

Similarly to Winkler *et al.* [17] and differently to Erez and Todorov [13], we transcribe the problem by only making use of hard constraints; and satisfaction of those constraints is a necessary requirement for the computed motions to be physically feasible and to complete the task successfully. This design decision is motivated by the fact that considering a cost function requires expert knowledge to carefully tune the weighting parameters that control the trade-off between different objective terms. Optimizing an objective function also requires additional iterations and computational time. Nonetheless, for the sake of completion, one of the experiments we present later in this paper does include and discuss the minimization of a cost function.

For tasks where the robot makes or breaks contacts with the environment, we assume contact locations and contact timings are known *a priori*. This assumption allows us to enforce zero contact forces for mesh points where the robot is not in contact with the environment, and therefore our formulation does not require any actual complementarity constraints. On the other hand, such assumption depends on pre-determined contact sequences specified either by a human or by a contact planner (such as [17], [18], [19]).

[4]The control inputs at the final state need not be discretized.

TABLE I: Summary of the formulated NLP constraints.

Constraint	Structure	Relation
Bounds on $\boldsymbol{\xi}$	Linear	Mixed
Friction Cones	Linear	Inequality
End-effector Poses	Nonlinear	Equality
System Dynamics	Nonlinear	Equality

C. Constraints

In this subsection we enumerate the constraints enforced by our problem. Their summary is given in Table I.

1) Bounds on the decision variables: We constrain the joint positions, velocities, and torques to be within their corresponding lower and upper bounds with

$$\begin{aligned} \boldsymbol{q}_{\text{lb}} \leq \boldsymbol{q}_k \leq \boldsymbol{q}_{\text{ub}} \qquad & \forall k = 1 : M \\ \boldsymbol{v}_{\text{lb}} \leq \boldsymbol{v}_k \leq \boldsymbol{v}_{\text{ub}} \qquad & \forall k = 1 : M \\ \boldsymbol{\tau}_{\text{lb}} \leq \boldsymbol{\tau}_k \leq \boldsymbol{\tau}_{\text{ub}} \qquad & \forall k = 1 : M - 1. \end{aligned} \tag{5}$$

2) Initial and final joint velocities: We enforce the initial and final velocities of every joint to be zero: $\boldsymbol{v}_1 = \boldsymbol{v}_M = \boldsymbol{0}$.

3) End-effector pose: We enforce end-effector poses with

$$f^{\text{fk}}(\boldsymbol{q}_k, i) = \boldsymbol{p}_i, \tag{6}$$

where $f^{\text{fk}}(\cdot)$ is the forward kinematics function, i refers to the i-th end-effector of the robot, and $\boldsymbol{p}_i \in SE(3)$ is the desired pose. Our formulation allows to define these constraints for a specific set of mesh points, which is convenient because we may not want to dictate a particular path for the feet to follow during swing phases. We can also selectively enforce only the position or the orientation component of a pose.

4) Contact forces: For mesh points where the robot is *not* in contact with the environment, we enforce the contact forces at the respective contact points to be zero: $\boldsymbol{\lambda}_k = \boldsymbol{0}$.

5) Friction constraints: We model friction at the contacts with linearized friction cones [20]. A point contact remains fixed as long as the contact force lives within the boundaries of its friction cone. We consider the set of points $\{C_i\}$ where the robot is in contact with its environment[5], and enforce each of the contact points to remain static with

$$\begin{aligned} |\boldsymbol{f}_i^c \cdot \boldsymbol{t}_i| &\leq (\mu_i/\sqrt{2})(\boldsymbol{f}_i^c \cdot \boldsymbol{n}_i) \\ |\boldsymbol{f}_i^c \cdot \boldsymbol{b}_i| &\leq (\mu_i/\sqrt{2})(\boldsymbol{f}_i^c \cdot \boldsymbol{n}_i) \\ \boldsymbol{f}_i^c \cdot \boldsymbol{n}_i &> 0, \end{aligned} \tag{7}$$

where $\boldsymbol{f}_i^c$ is the contact force, $\boldsymbol{n}_i$ and μ_i are the unit normal and the friction coefficient at the contact, and $(\boldsymbol{t}_i, \boldsymbol{b}_i)$ form the basis of the tangential contact plane such that $(\boldsymbol{t}_i, \boldsymbol{b}_i, \boldsymbol{n}_i)$ is a direct frame.

[5]Surface-to-surface contacts are characterized by an infinite number of contact points. For such cases, we approximate the contact with a finite number of points at key locations. E.g., the humanoid robot mentioned in this paper has flat feet soles, so we consider one contact point at each corner of the sole, i.e., four contact points per foot.

6) System dynamics: We enforce the continuous nonlinear system dynamics $\dot{x} = f(x, u)$ with *defect* constraints. The approach used to define these constraints is the main subject of this paper, and we will explain it in detail in the next section.

IV. System Dynamics

The equations of motion for a floating-base robot that interacts with its environment can be written as

$$\boldsymbol{M}(\boldsymbol{q})\dot{\boldsymbol{v}} + \boldsymbol{h}(\boldsymbol{q}, \boldsymbol{v}) = \boldsymbol{S}^\top \boldsymbol{\tau} + \boldsymbol{J}_s^\top(\boldsymbol{q})\boldsymbol{\lambda}, \tag{8}$$

where $\boldsymbol{M}(\boldsymbol{q}) \in \mathbb{R}^{n_v \times n_v}$ is the mass matrix, and $\boldsymbol{h}(\boldsymbol{q}, \boldsymbol{v}) \in \mathbb{R}^{n_v}$ is the vector of Coriolis, centrifugal, and gravity terms. On the right-hand side of the equation, $\boldsymbol{\tau} \in \mathbb{R}^{n_\tau}$ is the vector of joint torques commanded to the system, and the selection matrix $\boldsymbol{S} = [\boldsymbol{0}_{n_\tau \times (n_v - n_\tau)} \quad \mathbb{I}_{n_\tau \times n_\tau}]$ selects which degrees of freedom (DoF) are actuated. We consider that all limb joints are actuated, thus $n_\tau = n_j$. The vector $\boldsymbol{\lambda} \in \mathbb{R}^{n_s}$ denotes the forces and torques experienced at the contact points, with n_s being the total dimensionality of all contact wrenches. The support Jacobian $\boldsymbol{J}_s \in \mathbb{R}^{n_s \times n_v}$ maps the contact wrenches $\boldsymbol{\lambda}$ to joint-space torques, and it is obtained by stacking the Jacobians which relate generalized velocities to limb end-effector motion as $\boldsymbol{J}_s = [\boldsymbol{J}_{C_1}^\top \quad \cdots \quad \boldsymbol{J}_{C_{n_c}}^\top]^\top$, with n_c being the number of limbs in contact. For fixed-base manipulators that are not subject to contact forces, we can simplify the equations of motion: $\boldsymbol{M}(\boldsymbol{q})\dot{\boldsymbol{v}} + \boldsymbol{h}(\boldsymbol{q}, \boldsymbol{v}) = \boldsymbol{\tau}$.

In order to enforce the equations of motion of nonlinear systems, we define a set of mathematical equality constraints called *defect* constraints within our framework. Usually, these constraints are defined using a forward dynamics algorithm, but in this paper we argue that using inverse dynamics can be more computationally advantageous.

The standard problem of forward dynamics computes the joint accelerations resultant from commanding torques and applying forces to the robot at a given state, i.e.,

$$\dot{\boldsymbol{v}}_k^* = f^{\mathrm{fd}}(\boldsymbol{q}_k, \boldsymbol{v}_k, \boldsymbol{\tau}_k, \boldsymbol{\lambda}_k), \tag{9}$$

where $f^{\mathrm{fd}}(\cdot)$ is the function that solves forward dynamics, and the asterisk $(\cdot)^*$ denotes a computed intermediate value, whereas terms without an asterisk are NLP variables. Using the *semi-implicit Euler method* as the integration scheme[6] and $h = (t_F - t_I)/N$ as the integration time step, we can compute the state of the robot after h seconds. First, we integrate $\dot{\boldsymbol{v}}_k^*$ to compute the next generalized velocities $\boldsymbol{v}_{k+1}^* = \boldsymbol{v}_k + h\,\dot{\boldsymbol{v}}_k^*$. We can then compute $\dot{\boldsymbol{q}}_{k+1}^*$ from $\boldsymbol{v}_{k+1}^*$, i.e., the time derivative of the generalized coordinates vector for those velocities. In turn, we integrate that time derivative to compute the next coordinates $\boldsymbol{q}_{k+1}^* = \boldsymbol{q}_k + h\,\dot{\boldsymbol{q}}_{k+1}^*$. After these calculations, we end up with two different values of the system's state at mesh point $k+1$: one from the discretized NLP variables, and another computed as a result of the controls applied to the system at mesh point k. To enforce dynamical consistency, we define the defect constraints as

$$\begin{aligned} \boldsymbol{q}_{k+1}^* - \boldsymbol{q}_{k+1} &= \boldsymbol{0} \\ \boldsymbol{v}_{k+1}^* - \boldsymbol{v}_{k+1} &= \boldsymbol{0}. \end{aligned} \tag{10}$$

However, there is an alternative way to enforce dynamical consistency with inverse dynamics. In contrast to (9), inverse dynamics computes the joint torques and forces required to meet desired joint accelerations at a given state, i.e.,

$$\boldsymbol{\tau}_k^* = f^{\mathrm{id}}(\boldsymbol{q}_k, \boldsymbol{v}_k, \dot{\boldsymbol{v}}_k^*, \boldsymbol{\lambda}_k), \tag{11}$$

where $f^{\mathrm{id}}(\cdot)$ is the function that solves the inverse dynamics problem, and the desired joint accelerations can be calculated implicitly with $\dot{\boldsymbol{v}}_k^* = (\boldsymbol{v}_{k+1} - \boldsymbol{v}_k)/h$. Similarly to the forward dynamics case, we compute $\dot{\boldsymbol{q}}_{k+1}^*$ from $\boldsymbol{v}_{k+1}$, and integrate it to compute the next generalized coordinates $\boldsymbol{q}_{k+1}^*$. And finally, we define the dynamics defect constraints as

$$\begin{aligned} \boldsymbol{q}_{k+1}^* - \boldsymbol{q}_{k+1} &= \boldsymbol{0} \\ \boldsymbol{\tau}_k^* - \boldsymbol{\tau}_k &= \boldsymbol{0}. \end{aligned} \tag{12}$$

Notice that the main difference between equations (10) and (12) is that forward dynamics enforces consistency of the generalized velocities whereas inverse dynamics enforces consistency of joint torques commanded to the system.

The main subject of this paper revolves around the two formulations explained above to enforce the nonlinear system dynamics: *inverse dynamics* vs. *forward dynamics*. We developed our framework with both options in mind, and allow to easily toggle between one approach and the other when specifying trajectory optimization problems. The ability to select either one of the formulations was particularly useful during evaluation, which we explain in the next section.

V. Experiments and Results

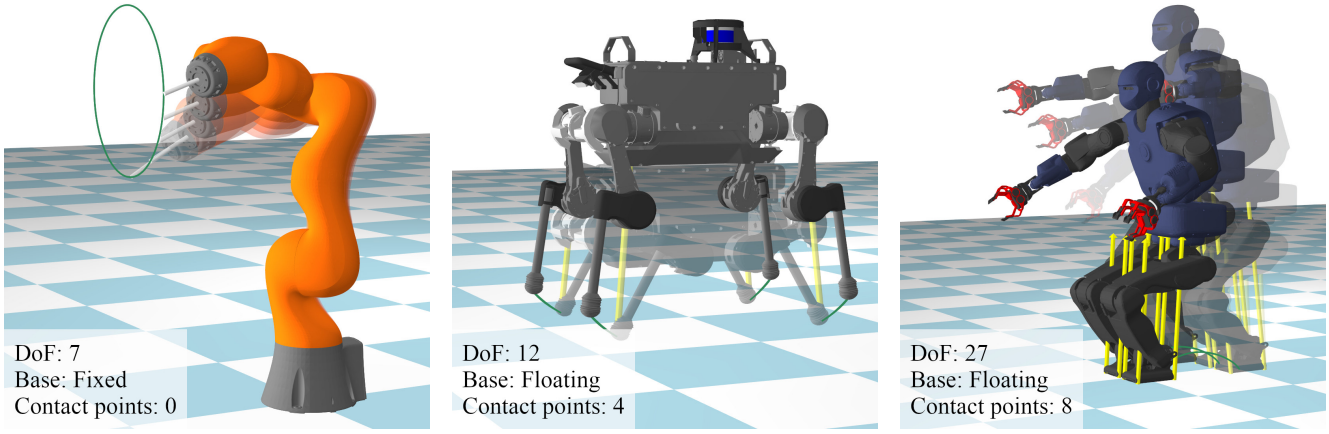
This section is organized into four subsections:

A. Compares the computation time and number of solver iterations required to find locally-optimal solutions;
B. Evaluates the robustness of each approach as problem discretization gets more coarse (larger time steps);
C. Analyzes the performance of each formulation for the minimization of a cost function; and finally,
D. Shows hardware validation of the planned motions.

All the evaluations in this section were carried out in a single-threaded process on an Intel i7-6700K CPU with 4.0 GHz and 32 GB 2133 MHz memory. The framework we propose has been implemented in Julia [21], using the rigid-body dynamics library RBD.jl [9], and the optimization library Knitro [22]. To solve the formulated NLP problems, we used the interior-point method[7] of Waltz *et al.* [23].

[6] Semi-implicit Euler and explicit Euler methods are first-order integrators. However, the semi-implicit Euler method is more accurate because it is a symplectic integrator, i.e., it conserves the system's energy approximately, while the explicit Euler method is known to add potential energy to the system. For simplicity, higher-order integrators were not considered.

[7] Interior-point methods (IPM) are generally preferred for large-scale problems. Compared to active-set methods (ASM), IPM are faster for NLP problems with many inequality constraints; and given a good initialization, IPM usually outperform ASM. [3]



(a) Manipulator tracing a circular path. (b) Quadruped jumping in-place. (c) Humanoid jumping $0.5\,\mathrm{m}$ forward.

Fig. 2: Motion trace of each robot's task. Robots: KUKA's LBR iiwa 14, ANYbotics' ANYmal, and PAL Robotics' TALOS.

A. *Evaluation of Convergence*

In order to evaluate and compare forward dynamics against inverse dynamics in the context of direct transcription, we used our framework to specify tasks in the form of numerical optimization problems for different types of robots: a manipulator, a quadruped, and a humanoid. Those robots were selected as they allows us to evaluate the formulations for distinct features: fixed- and floating-base systems, single-point and surface contacts, and low and high dimensionality (degrees of freedom). For each task on each robot, we solved the optimization problem twice: first defining the defect constraints with forward dynamics, and then with inverse dynamics. We would like to emphasize that the *only* changing factor in the problem formulation was the toggling between forward and inverse dynamics for the definition of the defect constraints; any and every other aspect of the NLP formulation was kept unchanged.

Performance of general NLP solvers is greatly affected by the linear solver used for solving the linear systems of equations of the problem. For this reason, we tested different state-of-the-art linear solvers[8] exhaustively. For interior-point methods, another important factor that affects performance is the update strategy of the barrier parameter. Therefore, for all of our evaluations, we tested the different strategies available within the Knitro [22] library[9] exhaustively.

In the remainder of this subsection, we present the task specifications for each robot and indicate all the parameters for reproducibility. Then, we present the results we obtained for each task, which evaluated the solver's performance in terms of computation time and number of iterations taken by the solver until a locally-optimal solution was found.

1) Manipulator: We evaluated the different formulations using a fixed-base robot arm with seven DoF (shown in Figure 2a). The task we specified consisted of using the end-effector tool to trace a circular path parameterized by $[0.5, 0.2\cos\theta, 0.8 + 0.2\sin\theta] \forall \theta \in [0, 2\pi]$. The total duration of the motion was specified to $2.0\,\mathrm{s}$ and the trajectory was discretized at $150\,\mathrm{Hz}$, resulting in a total of 301 mesh points. The initial guess passed to the solver was a fixed nominal configuration, zero velocities, and zero torques.

2) Quadruped: The quadruped robot we used is shown in Figure 2b. This system is more complex than the manipulator due to its floating-base, more DoF (three motors per leg), and because it needs to handle contact forces. For this robot, we defined a jumping task by enforcing the contact forces to be zero during a short period of time. The trajectory was discretized at $100\,\mathrm{Hz}$, the total duration of the motion was set to $2.0\,\mathrm{s}$, and the interval specified for the flight-phase was $[1.0, 1.2]$ s. We did not constrain the placement of the feet during the flight-phase, therefore allowing the solver to converge to a solution comprised of the most natural feet swing paths according to the system dynamics. The initial guess passed to the solver was a fixed standing configuration, zero velocities, zero torques, and zero contact forces.

3) Humanoid: Finally, we considered the humanoid robot shown in Figure 2c. This robot is more complex than the quadruped robot because it has 27 DoF[10] (seven per arm, six per leg, and one at the torso), and its feet cannot be simplified to single-point contacts. Similarly to the quadruped, we also defined a jumping task for the humanoid. The trajectory was discretized at $125\,\mathrm{Hz}$, the total duration of the motion was set to $1.2\,\mathrm{s}$, and the interval specified for the flight-phase was $[0.5, 0.8]$ s. The initial guess was a standing configuration, zero velocities, zero torques, and zero contact forces.

The results of the experiments on these robots are shown in Table II, where smaller numbers indicate better performance. The rows of the table are grouped according to robot, dynamics, and linear solver. Each row then contains the time[11] taken to solve the optimization problem for each barrier update strategy, as well as the total number of iterations (inside

[8] MA{27, 57, 86, 97}. Some of these solvers support parallelization, but in order to have a fair comparison we always used a single-core only.

[9] Those strategies are: `monotone`, `adaptive`, `probing`, `dampmpc`, `fullmpc`, and `quality`. Knitro's documentation explains and provides references for each strategy—`artelys.com/docs/knitro/`.

[10] The real robot has more DoF: grippers, neck, and one more DoF at the torso. But for simplicity, we assumed those joints were fixed to zero.

[11] The times shown in this table (and in future tables) are the minimum value observed over 10 trials. Reporting the minimum time is more reliable than the median or the mean, since all measured noise is positive. [24]

TABLE II: Computation time (in seconds) and number of iterations (inside parenthesis) for each robot. Cases that did not converge have been truncated. The best computation time for each dynamics and each robot is highlighted in bold.

	Dynamics	Linear Solver	Barrier parameter update strategy (`bar_murule`) `monotone`	`adaptive`	`probing`	`dampmpc`	`fullmpc`	`quality`	Average time per iteration (s)
KUKA Manipulator	Forward	MA27	0.999186 (12)	**0.405503 (5)**	0.420237 (5)	0.492032 (6)	0.418245 (5)	0.439486 (5)	0.0837 ± 0.0023
		MA57	0.964917 (12)	0.413397 (5)	0.411683 (5)	0.486253 (6)	0.411253 (5)	0.421502 (5)	0.0822 ± 0.0014
		MA86	1.905320 (12)	0.749751 (5)	0.767782 (5)	0.912592 (6)	0.780835 (5)	0.806475 (5)	0.1553 ± 0.0043
		MA97	1.117970 (12)	0.469281 (5)	0.483895 (5)	0.565918 (6)	0.492413 (5)	0.501238 (5)	0.0961 ± 0.0028
	Inverse	MA27	0.521736 (10)	**0.200804 (4)**	0.217532 (4)	0.268954 (5)	0.262151 (5)	0.234240 (4)	0.0536 ± 0.0028
		MA57	0.576045 (10)	0.224917 (4)	0.237039 (4)	0.282610 (5)	0.296379 (5)	0.243132 (4)	0.0583 ± 0.0018
		MA86	1.164230 (10)	0.469071 (4)	0.502839 (4)	0.600684 (5)	0.628597 (5)	0.527639 (4)	0.1229 ± 0.0060
		MA97	0.692929 (10)	0.279098 (4)	0.276572 (4)	0.338553 (5)	0.355045 (5)	0.297295 (4)	0.0702 ± 0.0023
ANYmal Quadruped	Forward	MA27	6.96923 (25)	2.26414 (7)	3.04293 (10)	2.8056 (9)	—	2.34618 (7)	0.3107 ± 0.0213
		MA57	8.07471 (29)	2.80012 (10)	2.98687 (10)	2.49608 (9)	—	32.1376 (99)	0.2918 ± 0.0203
		MA86	44.6051 (51)	—	—	24.605 (36)	—	6.46898 (10)	0.7350 ± 0.1223
		MA97	6.30551 (21)	**2.21974 (7)**	3.06281 (10)	2.76923 (9)	—	2.26557 (7)	0.3110 ± 0.0093
	Inverse	MA27	3.67818 (16)	2.99674 (13)	2.39058 (10)	2.60960 (11)	—	2.18363 (9)	0.2359 ± 0.0055
		MA57	3.59060 (16)	2.90472 (13)	2.32125 (10)	2.54119 (11)	—	**2.10427 (9)**	0.2290 ± 0.0047
		MA86	6.53682 (16)	5.36693 (13)	4.38130 (10)	4.76909 (11)	—	4.06875 (9)	0.4290 ± 0.0181
		MA97	3.96465 (16)	3.21597 (13)	2.57926 (10)	2.81092 (11)	—	2.37694 (9)	0.2545 ± 0.0071
TALOS Humanoid	Forward	MA27	40.8323 (50)	—	13.4659 (13)	—	—	14.8274 (15)	0.9470 ± 0.1153
		MA57	49.4806 (74)	—	13.3348 (19)	13.4783 (19)	55.3573 (83)	44.236 (66)	0.6834 ± 0.0205
		MA86	56.7882 (45)	41.3767 (32)	26.1488 (19)	—	106.804 (86)	—	1.2933 ± 0.0592
		MA97	36.3671 (49)	13.4293 (18)	13.6355 (18)	**11.4815 (15)**	—	12.9293 (16)	0.7639 ± 0.0264
	Inverse	MA27	14.5289 (27)	8.38412 (15)	8.5284 (14)	8.0221 (13)	—	38.5943 (70)	0.5749 ± 0.0358
		MA57	12.9616 (27)	7.36908 (15)	7.2312 (14)	**6.59744 (13)**	—	36.8871 (73)	0.5001 ± 0.0144
		MA86	17.7234 (25)	11.0047 (15)	11.1492 (14)	9.97684 (13)	—	63.9787 (75)	0.7719 ± 0.0562
		MA97	12.9830 (27)	7.43808 (15)	7.10122 (14)	6.61713 (13)	—	41.8407 (82)	0.5006 ± 0.0125

parenthesis). The last column shows the time spent on each iteration, averaged over the update strategies.

In general, we can see that the computation time mostly depends on the complexity of the system, regardless of linear solver or barrier update strategy; i.e., solving the manipulator task was faster than solving the quadruped task, which in turn was faster than the humanoid task. More importantly, for each robot and given the same choice of linear solver and update strategy, the computation time of inverse dynamics was better than forward dynamics. We can also see that the number of iterations required to solve the problem did not change significantly (apart from a few exceptions). This indicates that the difficulty of the problem itself did not change with the different dynamics defects; it just took longer to solve using forward dynamics—as supported by the information in the last column of the table.[12]

B. Robustness to Coarser Problem Discretizations

In the next experiment, we wanted to compare the ability of each formulation to handle trajectories discretized using fewer mesh points. For that, we defined the same quadruped jumping task repeatedly, but transcribed it using different trajectory resolutions. First, we divided the trajectory into equally spaced segments with a time step of $h = 0.01$; we solved the optimization problem and took the resulting trajectory as our baseline. Then, we incrementally changed h, making the problem more coarse each time, and compared the obtained trajectories against the baseline. The problems were initialized with a nominal configuration repeated for each point, and zero velocities, torques and contact forces. The results of this experiment for the quadruped robot are shown in Figure 3 and Table III.

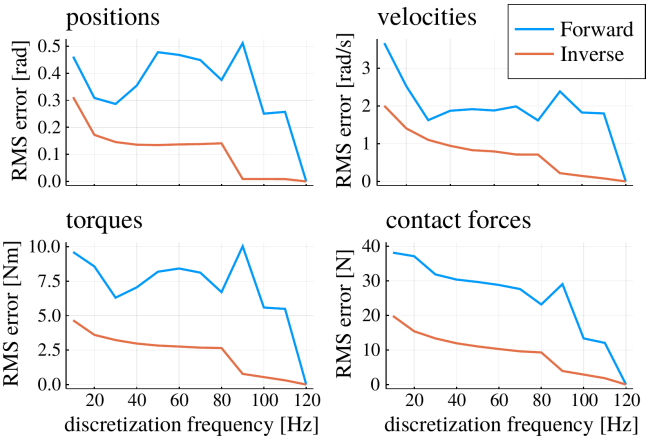


Fig. 3: Root-mean-square error (RMSE) of joint positions, velocities, torques, and contact forces of each formulation for different discretizations, using a baseline of 120 Hz.

The plots in Figure 3 show that the solutions deviate more from the baseline as the number of mesh points used to discretize the problem decreases (in the x-axis, from right to left). But more importantly, the plots reveal that the rate at which deviation occurs is significantly different depending on the formulation. We can see that the root-mean-square error (RMSE) of the formulation using inverse dynamics is significantly lower than that of the forward.

[12]MA86 resulted in longer computation times per iteration because it is designed for multi-core processors, but we only used a single core.

TABLE III: Computation time and number of iterations required for different problem discretizations (quadruped).

Frequency	Forward Dyn.		Inverse Dyn.	
	Time (s)	Iter.	Time (s)	Iter.
120 Hz	5.737	15	3.950	13
110 Hz	3.224	9	3.634	13
100 Hz	3.289	10	3.295	13
90 Hz	3.916	14	2.774	13
80 Hz	5.227	21	1.821	9
70 Hz	4.062	19	1.344	8
60 Hz	1.518	9	1.193	8
50 Hz	2.226	16	0.983	8
40 Hz	1.099	9	0.785	8
30 Hz	0.731	8	0.534	7
20 Hz	0.433	7	0.354	7
10 Hz	0.382	13	0.189	8

Table III shows the computation time (in seconds) and the number of iterations required to solve the quadruped task using different discretizations. We can see that the time required to solve the problem using inverse dynamics follows a clear pattern: it decreases as the problem gets more coarse; and the same goes for the number of iterations. On the other hand, for the formulation using forward dynamics, a pattern does not seem to exist.

The results shown in Figure 3 and Table III provide strong evidence that defining the defect constraints with inverse dynamics is the more robust approach to different problem discretizations, both in terms of deviation from realistic solutions and in terms of computation performance.

C. Optimization with an Objective Function

Thus far we have analyzed the trajectory optimization performance for feasibility problems, i.e., finding a set of variables $\boldsymbol{\xi}$ satisfying all the constraints defined in Table I. However, in optimization it is common to define a cost function to be minimized (or a value function to be maximized). In this next experiment, we evaluate the performance of our formulation when a cost function is considered. We are interested in minimizing actuator torques, as well as the ground-reaction contact forces involved. To achieve this, we define the following optimization objective:

$$\min_{\boldsymbol{\xi}} \quad \sum_{k=1}^{M-1} \frac{\boldsymbol{\tau}_k^\top \boldsymbol{\tau}_k + \boldsymbol{\lambda}_k^\top \boldsymbol{\lambda}_k}{M-1}. \tag{13}$$

We tested the above cost function for the jumping task on the quadruped, using the MA57 linear solver and the `adaptive` barrier parameter update strategy. Both formulations converged to very similar solutions: the RMSE between the two trajectories was 0.038. The final objective value of each solution was $3.567\,801 \times 10^4$ and $3.567\,804 \times 10^4$ for forward and inverse dynamics, respectively. Despite converging to similar solutions, the formulation employing inverse dynamics finished in $6.208\,\mathrm{s}$, showing better performance than the formulation using forward dynamics which took $14.570\,\mathrm{s}$. The time in seconds corresponds to the minimum value measured over a total of 10 samples.

Figure 4 shows the evolution of the cost and feasibility error throughout the optimization. The star-shaped marker denotes the point at which the local minimum of the problem was found. In the left plot, we can see that inverse dynamics reached values close to the optimal cost much earlier than forward dynamics. And in the right plot, we can also see that inverse dynamics required less iterations than forward dynamics to cross the faint-green line, which marks the point at which the error becomes acceptable to be considered feasible. Inverse dynamics converged in 26 iterations, and forward dynamics converged in 43 iterations. Inadvertently, one advantage of the forward formulation was that its final feasibility error was smaller than that of inverse dynamics.

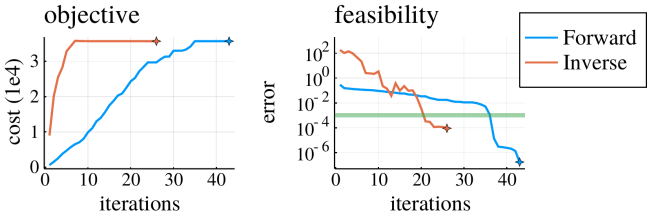


Fig. 4: Evolution of the cost and the feasibility error during convergence. The faint-green line at $y = 10^{-3}$ denotes the tolerance under which we consider a problem to be feasible.

D. Hardware Validation

We conducted real-world experiments with ANYmal [25] and TALOS [2] to validate the trajectories computed with our framework. The motion planning is performed offline and then the trajectories are sent to the controller for playback. To execute the whole-body motions, we commanded each joint with feedforward torque and feedback on joint position and velocity. For the quadruped, we updated the references for each joint's position, velocity, and torque at $400\,\mathrm{Hz}$. The decentralized motor controller at every joint closes the loop compensating for friction effects. On the humanoid, we updated the references at $2\,\mathrm{kHz}$, and a centralized controller compensates for the motor dynamics and friction.

Figure 1 and Figure 5 contain snapshots of the jumps realized with the humanoid and with the quadruped, respectively. These experiments can be seen in our supplementary video.[13] Jumping motion is challenging to execute in real hardware because it includes a severely underactuated phase when the robot is fully off the ground. Nonetheless, our controller is able to execute our planned trajectories reliably, attesting the dynamical consistency of our formulation.

VI. Discussion

The results of this work indicate that direct transcription implementations relying on forward dynamics to define the defect constraints can be reformulated with inverse dynamics to see an increase in performance, for both feasibility or minimization problems, and without sacrificing the feasibility of the solutions to the optimization problem. An additional reason to prefer inverse dynamics is robustness to coarser discretizations, both in terms of computation efficiency and faithfulness of solutions with respect to finer discretizations.

[13] Supplementary video: `https://youtu.be/HZPKyQcwTPU`

(a) Initial configuration (b) Take-off (c) Full-flight phase (d) Landing (e) Final configuration

Fig. 5: Snapshots of ANYmal [25] performing a $0.5\,\mathrm{m}$-long jump. The length of the black tape on the ground is $0.5\,\mathrm{m}$.

When minimizing a cost function, the locally-optimal solutions computed with either formulation are essentially the same. However, when an objective function is not considered, the formulations may diverge to different solutions. Experimentally, we have observed that the solutions computed with inverse dynamics are easier to perform in real hardware. The reasons behind this divergence are not yet clear to us, and this is something we plan to investigate in future work.

Erez and Todorov [13] observed a striking feature in their results: an emergent coordination between legs and opposite arms during a running gait. In this work, for the humanoid jumping task, we also observed such emerging behavior: the resulting motions swing the arms upwards to build-up energy before the take-off instant. Both in [13] and our work, these features originated without any explicit modeling—reaffirming the power of dynamic trajectory optimization.

VII. Future Work

Robustness to Disturbances: In recent work [26], we took into account uncertainty and robustness to disturbances using direct transcription. Considering uncertainty usually incurs additional computational cost due to more complex problem formulations. With the findings from this paper, we plan to redefine the dynamics defect constraints in that work with inverse dynamics, improving the performance of our robustness framework and making it more competitive.

Parallelization: Simultaneous methods discretize states and controls over time as decision variables. Representing the entire trajectory in this way is amenable for parallelization. In this work, we used a single thread for all computations, but in the future we want to use multiple threads to compute the dynamics defects simultaneously. This could impart some overhead on small problems, but on bigger problems (where system dynamics need to be computed hundreds of times per iteration) it should greatly improve performance.

References

[1] J. T. Betts, *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*, 2nd ed. SIAM, 2010.

[2] O. Stasse, T. Flayols *et al.*, "TALOS: A new humanoid research platform targeted for industrial applications," in *IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, 2017.

[3] D. Pardo, L. Möller *et al.*, "Evaluating Direct Transcription and Nonlinear Optimization Methods for Robot Motion Planning," *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 946–953, 2016.

[4] T. Koolen and R. Deits, "Julia for robotics: simulation and real-time control in a high-level programming language," in *International Conference on Robotics and Automation (ICRA)*, 2019.

[5] S. M. Neuman, T. Koolen *et al.*, "Benchmarking and Workload Analysis of Robot Dynamics Algorithms," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019.

[6] J. Carpentier, G. Saurel *et al.*, "The Pinocchio C++ library – A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives," in *IEEE International Symposium on System Integrations (SII)*, 2019.

[7] N. Schweighofer, M. A. Arbib, and M. Kawato, "Role of the cerebellum in reaching movements in humans. i. distributed inverse dynamics control," *European Journal of Neuroscience*, 1998.

[8] B. Mehta and S. Schaal, "Forward Models in Visuomotor Control," *Journal of Neurophysiology*, vol. 88, no. 2, pp. 942–953, 2002.

[9] T. Koolen and contributors, "Rigidbodydynamics.jl," 2016. [Online]. Available: https://github.com/JuliaRobotics/RigidBodyDynamics.jl

[10] M. L. Felis, "RBDL: an efficient rigid-body dynamics library using recursive algorithms," *Autonomous Robots*, pp. 1–17, 2016.

[11] M. Frigerio, J. Buchli *et al.*, "RobCoGen: a code generator for efficient kinematics and dynamics of articulated robots, based on Domain Specific Languages," *Journal of Software Engineering for Robotics (JOSER)*, vol. 7, no. 1, pp. 36–54, 2016.

[12] Sung-Hee Lee, Junggon Kim *et al.*, "Newton-type algorithms for dynamics-based robot movement optimization," *IEEE Transactions on Robotics (T-RO)*, vol. 21, no. 4, pp. 657–667, 2005.

[13] T. Erez and E. Todorov, "Trajectory optimization for domains with contacts using inverse dynamics," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2012, pp. 4914–4919.

[14] M. Posa, C. Cantu, and R. Tedrake, "A direct method for trajectory optimization of rigid bodies through contact," *The International Journal of Robotics Research (IJRR)*, vol. 33, no. 1, pp. 69–81, 2014.

[15] P. G. Gormley, "Stereographic Projection and the Linear Fractional Group of Transformations of Quaternions," *Proceedings of the Royal Irish Academy. Mathematical and Physical Sciences*, vol. 51, 1945.

[16] G. Terzakis, M. Lourakis, and D. Ait-Boudaoud, "Modified Rodrigues Parameters: An Efficient Representation of Orientation in 3D Vision and Graphics," *Journal of Mathematical Imaging and Vision*, 2018.

[17] A. W. Winkler, D. C. Bellicoso *et al.*, "Gait and Trajectory Optimization for Legged Systems through Phase-based End-Effector Parameterization," *IEEE Robotics and Automation Letters*, 2018.

[18] S. Tonneau, D. Song *et al.*, "SL1M: Sparse L1-norm Minimization for contact planning on uneven terrain," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[19] T. Stouraitis, I. Chatzinikolaidis *et al.*, "Online Hybrid Motion Planning for Dyadic Collaborative Manipulation via Bilevel Optimization," *IEEE Transactions on Robotics (T-RO)*, vol. 36, no. 5, 2020.

[20] S. Caron, Q. Pham, and Y. Nakamura, "Leveraging Cone Double Description for Multi-contact Stability of Humanoids with Applications to Statics and Dynamics," in *Robotics: Science and System*, 2015.

[21] J. Bezanson, A. Edelman *et al.*, "Julia: A Fresh Approach to Numerical Computing," *SIAM Review*, vol. 59, 2017.

[22] R. H. Byrd, J. Nocedal, and R. A. Waltz, *Knitro: An Integrated Package for Nonlinear Optimization*. Springer US, 2006.

[23] R. A. Waltz, J. L. Morales *et al.*, "An interior algorithm for nonlinear optimization that combines line search and trust region steps," *Mathematical Programming*, vol. 107, no. 3, pp. 391–408, 2006.

[24] J. Chen, J. Revels, and A. Edelman, "Robust benchmarking in noisy environments," in *Proceedings of the 20th IEEE High Performance Extreme Computing Conference*, Waltham, USA, 2016.

[25] M. Hutter, C. Gehring *et al.*, "ANYmal - toward legged robots for harsh environments," *Advanced Robotics*, 2017.

[26] H. Ferrolho, W. Merkt *et al.*, "Optimizing Dynamic Trajectories for Robustness to Disturbances Using Polytopic Projections," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2020.