A Versatile Co-Design Approach For Dynamic Legged Robots

Traiko Dinev Carlos Mastalli Vladimir Ivan Steve Tonneau Sethu Vijayakumar

***Abstract*— We present a versatile framework for the computational co-design of legged robots and dynamic maneuvers. Current state-of-the-art approaches are typically based on random sampling or concurrent optimization. We propose a novel bilevel optimization approach that exploits the derivatives of the motion planning sub-problem (i.e., the lower level). These motion-planning derivatives allow us to incorporate arbitrary design constraints and costs in an general-purpose nonlinear program (i.e., the upper level). Our approach allows for the use of any differentiable motion planner in the lower level and also allows for an upper level that captures arbitrary design constraints and costs. It efficiently optimizes the robot's morphology, payload distribution and actuator parameters while considering its full dynamics, joint limits and physical constraints such as friction cones. We demonstrate these capabilities by designing quadruped robots that jump and trot. We show that our method is able to design a more energy-efficient Solo robot for these tasks.**

I. Introduction

To design a robot capable of executing dynamic motions, we need to consider the robot's mechanical design as well as the motion it will execute. A traditional approach is to iterate between mechanical design and motion planning (e.g., [1]). However, it is a challenging process, especially for complex and dynamic robots, as it requires experts both in motion planning and mechanical design. Instead, *concurrent design* (co-design [2]) aims to automate this process by numerically optimizing both the motion and design parameters. As the designer, we first specify a set of design parameters (e.g., morphologies or motor characteristics), constraints (e.g., collision avoidance between robot components), high-level tasks (e.g., a jump) and evaluation metrics (e.g., energy). The algorithm then finds optimal design parameters and motions to more efficiently execute the task.

For the algorithm to find realistic design improvements, it needs to be able to plan feasible motions by considering the robot's full-body dynamics and actuation limits. We can do it efficiently through motion planning frameworks such as Crocoddyl [3], which can run fast enough for predictive control applications [4]. On the other hand, from a designer standpoint, we need to be able to specify arbitrary design constraints and cost functions in order to give the designer tools to fully specify all the parameters of the design.

All authors are with the Edinburgh Centre for Robotics, University of Edinburgh, UK.

Carlos Mastalli is also with the School of Engineering and Physical Sciences, Heriot-Watt University, U.K.

This research was supported by (1) the European Commission under the Horizon 2020 project Memory of Motion (MEMMO, ID: 780684) and (2) the Engineering and Physical Sciences Research Council (EPSRC), and (3) the Alan Turing Institute.

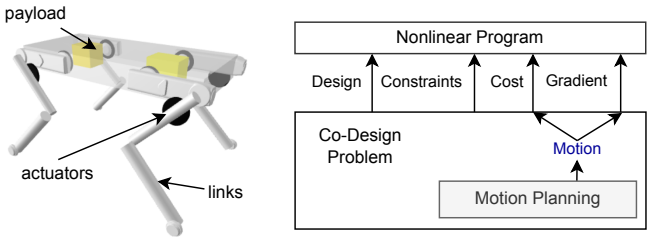


Fig. 1: Illustration of our bilevel optimization approach for robot co-design. In the upper level we use gradient information from the motion planning (lower level) to optimize the design of the robot. Please find the accompanying video at `https://youtu.be/Yxn7K1HXt_I`

Re-implementing motion planning in order to add additional design parameters requires considerable technical work, which is why we seek a modular framework that exploits state-of-the-art motion planners while considering design constraints. With this motivation in mind, we developed a co-design algorithm with the following scope: 1) ability to define arbitrary costs and constraints on continuous design variables, 2) treat the motion planning as a module, and 3) exploit state of the art motion planners that can compute dynamic motion for legged robots which include constraints on the motion parameters. This scope has some subtle differences from other co-design work in the literature.

A. Related Work

In the current literature, a popular approach to co-design is what we call *sampling-based co-design*. These methods are two-staged and exploit variants of Monte-Carlo sampling to find candidate robot designs. The fitness of those candidates is evaluated in a second stage through a motion planner.

The Covariance Matrix Adaptation Evolutionary Strategy (CMA-ES) [5] is a popular sampling approach used in co-design. It uses a Gaussian prior on candidate design parameters and estimates a covariance matrix needed for the following sampling steps. For instance, Wampler et al. [5] used a variant of CMA-ES to co-design various creatures in simulation, and Digumarti et al. [6] co-designed the legs of the quadruped StarlETH to optimize its running speed. Ha et. al. [7] used CMA-ES to optimize design and swing trajectories of planar legged robots. Most recently, Chadwick et al. [8] optimized the legs of quadrupeds and bipeds over uneven terrain for different user-defined co-design metrics, and Fadini et al. [9] computed the actuator properties of a monoped using CMA-ES.

A benefit of the above approaches is that they can use non smooth motion planners in the lower level. However, they do not support hard constraints on the design in the upper level, requiring soft constraints and cost tuning. Moreover, the algorithmic complexity of CMA-ES scales exponentially with respect to the number of design parameters (i.e., decision variables) due to the curse of dimensionality [10], [11]. This limits its application to a reduced number of design parameters and constraints, which in turn limits its scalability, for instance to multiple tasks and environments.

On the other hand, a number of *gradient-based co-design* methods have been proposed in the literature. One approach is to formulate a single nonlinear program that optimizes both motion and design parameters. This approach has been used to co-design legged robots. For instance, Mombaur [12], Buondonno et al. [13] and Spielberg et al. [14] compute the motions, lengths of the robot's limbs and/or actuator parameters in a single nonlinear program. However, the algorithmic complexity of the resulting nonlinear program is its major drawback (e.g. [15]). It also requires to modify the motion planning when including new co-design requirements, making the method non-modular.

To tackle the above-mentioned drawbacks, a few recent pieces of work have proposed a new approach that uses derivative information obtained via sensitivity analysis. Ha et al. [16] proposed to extract the relationship between motion and design by using the implicit function theorem. This allowed them to optimize the design while keeping the motion on the manifold of optimal motions. In a similar fashion, Desai et al. [17] used sensitivity analysis and the adjoint method to determine the same relationship. This latter approach was used in [18] and [19] for human-in-the-loop design optimization of robots. Still, these approaches have limitations. For instance, the method presented in [16] optimizes one target design parameter at a time and requires user input to select that parameter during optimization. The approaches used in [17]–[19] do not impose hard constraints in the motion optimization, but rather use penalty costs. This has the potential of violating the physics constraints. Finally, none of these methods support design constraints, which is a key designer requirement.

B. Our approach

In this paper, we propose a related, but more general solution, where we directly take the derivative of the motion planner and embed it into a nonlinear program. Our approach contains an upper and a lower level optimization for robot design and motion planning, respectively. In the lower level, we use an efficient state-of-the-art constrained motion planner, which is continuously differentiable. In the upper level, we formulate the design constraints and metrics as a nonlinear program, which we solve with a general-purpose nonlinear optimization software that handles arbitrary constraints.

Our approach is modular for differentiable motion planners, similar to genetic algorithms, while also supporting hard constraints on design parameters, which genetic algorithms do not. Since it uses derivative information, it inherently has faster local convergence. Finally, it does not require unconstrained motion planning (as is the case in [18]).

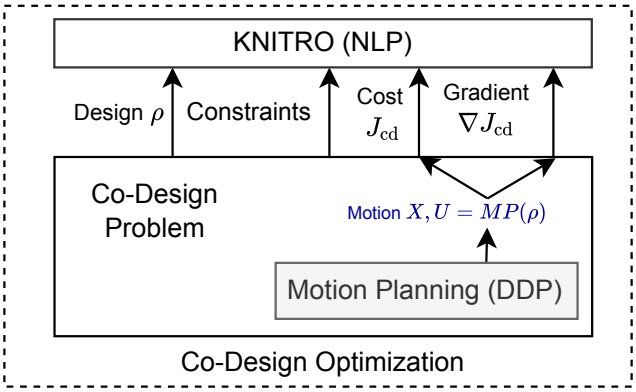


Fig. 2: A schematic of our co-design framework. We optimize the robot's design by differentiating through a motion planner, which is the lower-level problem of our bi-level formulation.

C. Contributions

The main contribution of our work is a novel bilevel optimization approach for robot co-design (Fig. 1). We identify two technical contributions:

i. a modular co-design algorithm that differentiates a motion planner and handles arbitrary co-design constraints and metrics in the upper level;
ii. a complete co-design framework for quadruped robots and dynamic locomotion maneuvers;

Our approach is of practical interest, as it allows for the use of any differentiable motion planner in the lower level without any modification to the motion planning itself. A modular approach like ours can take advantage of the state-of-the-art motion planning algorithms in terms of their convergence via the efficient use of the problem structure, and their ability to solve complex problems involving full robot dynamics and contacts with the environment. We show that gradient information and a bilevel optimization is a feasible approach to co-design for real-world co-design problems.

II. Co-Design Framework

Our co-design framework is illustrated in Fig. 2. First we describe our generic bilevel formulation of the co-design problem. We then describe the lower motion planning level, followed by how we apply our formulation for the design of quadrupeds. Finally, we describe a validation phase of our framework in simulation.

A. Co-design as Bilevel Optimization

We begin by encoding the robot's design into a *design vector* $\boldsymbol{\rho}$. The vector $\boldsymbol{\rho}$ encodes the robot's link lengths and its base shape (width, height, depth), as well as the payload distribution and the actuator parameters – motor mass and

gear ratio. We then formulate the co-design problem over the design vector $\boldsymbol{\rho}$ as a bilevel optimization problem:

$$\begin{aligned}
&\min_{\boldsymbol{\rho},\boldsymbol{X},\boldsymbol{U}} \; J_{\text{cd}}(\boldsymbol{\rho},\boldsymbol{X},\boldsymbol{U}) && \text{(co-design metric)}\\
&\text{s.t. } \boldsymbol{X},\boldsymbol{U} = \text{MP}(\boldsymbol{\rho};\text{TASK}), && \text{(motion planning)}\\
&\quad \underline{\boldsymbol{\rho}} \le \boldsymbol{\rho} \le \bar{\boldsymbol{\rho}}, && \text{(design bounds)}\\
&\quad \boldsymbol{g}(\boldsymbol{\rho}) = 0, && \text{(equality design constraints)}\\
&\quad \boldsymbol{h}(\boldsymbol{\rho}) \le 0, && \text{(inequality design constraints)}
\end{aligned} \tag{1}$$

where $J_{\text{cd}}(\cdot)$ is a user-specified co-design metric that evaluates the goodness of the design through the efficiency of the motion (e.g., the total energy used), $\text{MP}(\cdot)$ is the motion planning function, $\underline{\boldsymbol{\rho}}$ and $\bar{\boldsymbol{\rho}}$ are the lower and upper bounds of the design parameters. $\boldsymbol{g}(\boldsymbol{\rho})$ and $\boldsymbol{h}(\boldsymbol{\rho})$ are general equality and inequality constraints on the design vector (e.g., no collision constraints). We formulate the MP function as a nonlinear optimal control problem, which computes a discrete trajectory of robot states $\boldsymbol{X} = \{\boldsymbol{x}_0, \ldots, \boldsymbol{x}_N\}$ and controls $\boldsymbol{U} = \{\boldsymbol{u}_0, \ldots, \boldsymbol{u}_{N-1}\}$ for a desired task (represented by TASK) such as a trotting or jumping gait. Here N is the planning horizon, which is part of the task description.

We consider the motion planner as a general nonlinear function that maps from design parameters $\boldsymbol{\rho}$ to motions $\mathbf{m} = \{\boldsymbol{X},\boldsymbol{U}\}$. Thus, we can write the derivative of the co-design cost as:

$$\begin{aligned}
\nabla_{\boldsymbol{\rho}} J_{\text{cd}} &= \frac{dJ_{\text{cd}}(\boldsymbol{\rho},\mathbf{m})}{d\boldsymbol{\rho}} = \frac{\partial J_{\text{cd}}}{\partial \mathbf{m}}\frac{\partial \mathbf{m}}{\partial \boldsymbol{\rho}} + \frac{\partial J_{\text{cd}}}{\partial \boldsymbol{\rho}}\\
&= \frac{\partial J_{\text{cd}}}{\partial \mathbf{m}}\frac{\partial \text{MP}(\boldsymbol{\rho};\text{TASK})}{\partial \boldsymbol{\rho}} + \frac{\partial J_{\text{cd}}}{\partial \boldsymbol{\rho}},
\end{aligned} \tag{2}$$

where $\frac{\partial \mathbf{m}}{\partial \boldsymbol{\rho}}$ is the derivative of the motion with respect to the design parameters. This derivative can be computed using sensitivity analysis of the motion planner itself. However, the resulting expression is dependent on the optimization used in the lower level and thus not a modular solution. Computing it is also cumbersome as it involves differentiating through a complex nonlinear program.

Instead, we can directly consider the derivative $\frac{\partial \text{MP}(\boldsymbol{\rho};\text{TASK})}{\partial \boldsymbol{\rho}}$, or even more generally, directly $\nabla_{\boldsymbol{\rho}} J_{\text{cd}}$. This derivative would be difficult to compute analytically, however in practice the dimension of $\boldsymbol{\rho}$ is small, compared to the dimension of the motion. For instance, in our trotting experiment, $\dim(\boldsymbol{\rho}) = 17$, while $\dim(\boldsymbol{m}) = 9163$. Hence we can obtain $\nabla_{\boldsymbol{\rho}} J_{\text{cd}}$ directly through a numerical differentiation procedure that runs in parallel, i.e., we compute the derivative for each component of the $\boldsymbol{\rho}$ using multiprocessing. Using a one-sided finite difference approach, this requires a total of $\dim(\boldsymbol{\rho}) + 1$ calls to the motion planner. For each component of $\boldsymbol{\rho}$, we have:

$$\nabla_{\boldsymbol{\rho}_i} J_{\text{cd}} \approx \frac{J_{\text{cd}}(\boldsymbol{\rho}_i^+, \text{MP}(\boldsymbol{\rho}_i^+;\text{TASK})) - J_{\text{cd}}(\boldsymbol{\rho}, \text{MP}(\boldsymbol{\rho};\text{TASK}))}{\epsilon} \tag{3}$$

where $\boldsymbol{\rho}_i^+$ is the design vector with ϵ added to its i^{th} element. Using the derivative $\nabla_{\boldsymbol{\rho}} J_{\text{cd}}$, we can then optimize the design with gradient-based optimization.

This approach directly considers the motion as a function of the motion planner and does not assume a particular form of motion planning. Thus it allows us to use the full-body dynamics, friction cone constraints, control and state bounds in a nonlinear optimal control formulation (motion planner). This is in contrast to previous fixed-point approaches ([16], [17], [18], [19]) in which (i) the update rule needs to be derived manually for the used motion planner and (ii) arbitrary design constraints (on the vector $\boldsymbol{\rho}$) are not supported.

B. Co-design: Upper Level

We focus our work on improving the design of the 12 Degrees of Freedom (DoFs) SOLO robot [20]. Particularly, we are interested in quadrupedal locomotion gaits such as trotting and jumping. To plan for these gaits, the motion planner takes as parameters the following:

- The task, which is the desired gait, consisting of the contact sequence and timings
- The initial joint configuration $\boldsymbol{q}_0$
- The robot's joint limits

Each of these are computed in the upper level from the design vector $\boldsymbol{\rho}$ and updated each time the optimizer calls the motion planner to compute the optimal trajectory. We compute the initial state of the robot $\boldsymbol{q}_0$ using inverse kinematics so that the angle at the knee joint of the shortest leg is $45°$. We then run forward kinematics to set the foot positions, gait sequence and timings based on the task. We used the library PINOCCHIO [21] for computing the robot's kinematics and dynamics. We also set the lower and upper control bounds $(\underline{\boldsymbol{u}}, \bar{\boldsymbol{u}})$, and finally compute the optimal motion. We present an overview of our algorithm in Algorithm 1. In the upper level, we use the interior-point/direct algorithm provided in KNITRO [22], which requires the derivatives of the motion planner using the parallel scheme described.

C. Motion Planning: Lower Level

The lower level of our co-design bilevel optimization algorithm computes the motion trajectory $\{\boldsymbol{X},\boldsymbol{U}\}$ given a task and design $\boldsymbol{\rho}$. We formulate this lower level optimization as a hybrid nonlinear optimal control problem with fixed contact sequence and timings (Equation (4)):

$$\begin{aligned}
&\arg\min_{\boldsymbol{X},\boldsymbol{U}} \; \sum_{k=0}^{N-1} ||\boldsymbol{q}_k \ominus \boldsymbol{q}_{\text{ref}}||_{\boldsymbol{Q}}^2 + ||\boldsymbol{v}_k||_{\boldsymbol{N}}^2 + ||\boldsymbol{u}_k||_{\boldsymbol{R}}^2 + ||\boldsymbol{\lambda}_k||_{\boldsymbol{K}}\\
&\text{s.t.}\\
&\quad \text{for each contact phase: } p \in \mathcal{P} = \{1, 2, \cdots, N_p\}\\
&\quad\quad \text{if } \underline{\Delta t}_p \le k \le \bar{\Delta} t_p\text{:}\\
&\quad\quad\quad \boldsymbol{q}_{k+1} = \boldsymbol{q}_k \oplus \int_{t_k}^{t_k+\Delta t_k} \boldsymbol{v}_k \; dt, && \text{(integrator)}\\
&\quad\quad\quad \boldsymbol{v}_{k+1} = \boldsymbol{v}_k + \int_{t_k}^{t_k+\Delta t_k} \dot{\boldsymbol{v}}_k \; dt,\\
&\quad\quad\quad (\dot{\boldsymbol{v}}_k, \boldsymbol{\lambda}_k) = \boldsymbol{f}_p(\boldsymbol{q}_k, \boldsymbol{v}_k, \boldsymbol{u}_k), && \text{(contact dyn.)}\\
&\quad\quad \text{else:}\\
&\quad\quad\quad \boldsymbol{q}_{k+1} = \boldsymbol{q}_k,
\end{aligned}$$

$$
\begin{aligned}
(\boldsymbol{v}_{k+1}, \boldsymbol{\lambda}_k) &= \boldsymbol{\Delta}_p(\boldsymbol{q}_k, \boldsymbol{v}_k), && \text{(impulse dyn.)} \\
\boldsymbol{g}(\boldsymbol{q}_k, \boldsymbol{v}_k, \boldsymbol{u}_k) &= \boldsymbol{0}, && \text{(equality)} \\
\boldsymbol{h}(\boldsymbol{q}_k, \boldsymbol{v}_k, \boldsymbol{u}_k) &\leq \boldsymbol{0}, && \text{(inequality)} \\
\underline{\boldsymbol{x}} \leq \boldsymbol{x}_k &\leq \bar{\boldsymbol{x}}, && \text{(state bounds)} \\
\underline{\boldsymbol{u}} \leq \boldsymbol{u}_k &\leq \bar{\boldsymbol{u}}. && \text{(control bounds)}
\end{aligned} \tag{4}
$$

The state $(\boldsymbol{q}, \boldsymbol{v}) \in X$ lies in a differential manifold formed by the configuration $\boldsymbol{q} \in \mathbb{SE}(3) \times \mathbb{R}^{n_j}$ and its tangent vector $\boldsymbol{v} \in \mathbb{R}^{n_x}$ (with n_x and n_j as the dimension of the state manifold and number of joints, respectively). The control $\boldsymbol{u} \in \mathbb{R}^{n_j}$ is the vector of input torques, $\boldsymbol{\lambda}_k$ is the vector of contact forces, $\ominus$ and $\oplus$ are the *difference* and *integration* operators of the state manifold, respectively. Then $\boldsymbol{q}_{\text{ref}}$ is the reference standing upright robot posture, and $\boldsymbol{f}_p(\cdot)$ represents the contact dynamics under the phase p. To account for effects of discrete contact changes, $\boldsymbol{\Delta}_p(\cdot)$ is used to define an autonomous system that describes the contact-gain transition ([23]). $\boldsymbol{Q}$, $\boldsymbol{N}$, $\boldsymbol{R}$ and $\boldsymbol{K}$ are positive-define weighting matrices, $(\underline{\boldsymbol{x}}, \bar{\boldsymbol{x}})$ and $(\underline{\boldsymbol{u}}, \bar{\boldsymbol{u}})$ are the lower and upper bounds of the system state and control. $\underline{\Delta} t_p$ and $\bar{\Delta} t_p$ defines the timings of the contact phase p. We compute the hybrid dynamics and its derivatives as described in [3].

During contact phases, we use a linearized friction-cone constraint via a $(\boldsymbol{A}\boldsymbol{\lambda}_{\mathcal{C}(k)} \leq \boldsymbol{r})$, where $(\boldsymbol{A}, \boldsymbol{r})$ are computed from a predefined number of edges, and minimum and maximum normal contact forces, respectively. $\mathcal{C}(k)$ describes the set of active contacts. During the swing phases, we also include contact-placement constraints $(\log(\boldsymbol{p}_{\mathcal{G}(k)}^{-1} \circ \boldsymbol{M}_{\mathbf{p}_{\mathcal{G}(\mathbf{k})}}) = \boldsymbol{0})$, where $\log(\cdot)$ describes the log operator used in Lie algebra, $\boldsymbol{p}_{\mathcal{G}(k)}$ and $\boldsymbol{M}_{\mathbf{p}_{\mathcal{G}(\mathbf{k})}}$ are the reference and current placements of the set of swing contacts $\mathcal{G}(k)$.

We solve the motion planning problem (Eq. (4)) with the Feasibility-Driven Control-limited DDP (BOX-FDDP) algorithm [24], a variant of the Differential Dynamic Programming (DDP) algorithm. BOX-FDDP uses direct-indirect hybridization and enforces hard-constraints for the control limits. We employ a soft quadratic barrier to enforce inequality, equality and state constraints defined in Eq. (4). We implemented the algorithm using the open-source library CROCODDYL [3].

D. Verification in Simulation

We also validated our design improvements in the PYBULLET physics simulator. ([25]). To do so, we execute the motion plan for both the nominal and the optimized designs, and record the percentage improvement in costs ΔJ_{cd} (similar to [26]). We use a proportional-derivative (PD) controller with feed-forward torque to track the planned motion:

$$\boldsymbol{u} = \boldsymbol{u}^* + \boldsymbol{K_p}(\boldsymbol{q}_j^* - \boldsymbol{q}_j) + \boldsymbol{K_d}(\boldsymbol{v}_j^* - \boldsymbol{v}_j),$$

where $\boldsymbol{u}^*$, $\boldsymbol{q}_j^*$ and $\boldsymbol{v}^*$ are the reference feed-forward command, joint positions and velocities computed in Eq. (4), respectively. $\boldsymbol{K_p}$ and $\boldsymbol{K_d}$ are the PD gains. We tune these gains through a grid search procedure. We run the simulator on a 20×20 grid for $\boldsymbol{K_p} \in [1, 20]$ and $\boldsymbol{K_d} \in [0.1, \boldsymbol{K_d}/2]$. Then, we pick the gains that lead to the smallest tracking error for both designs. This procedure allows us to fairly compare and account for different robot dimensions and weights, as larger robots require higher gains and vice-versa.

Algorithm 1 Co-design optimization

procedure MP($\boldsymbol{\rho}$; TASK)
  Compute initial state $\boldsymbol{q}_0$ using inverse kinematics
  Set control bounds $\underline{\boldsymbol{u}}, \bar{\boldsymbol{u}}$ based on actuator parameters
  Run forward kinematics on $\boldsymbol{q}_0$ and set foot positions
  Set gait sequence and timings based on TASK
  Compute $\boldsymbol{m}$, the optimal motion
  return $\boldsymbol{m}$
end procedure
procedure CODESIGN
  Start at a design $\boldsymbol{\rho} = \boldsymbol{\rho}_0$
  while $J_{\text{cd}}(\boldsymbol{\rho}, \text{MP}(\boldsymbol{\rho}; \text{TASK}))$ decreasing **do**
    Compute $\nabla_{\boldsymbol{\rho}} J_{\text{cd}}$ via finite differences in parallel
    Update $\boldsymbol{\rho}$ using one step of the NLP solver
    Save the resulting motion to $\boldsymbol{m}$ and cost J_{cd}
  end while
  return $(\boldsymbol{\rho}, J_{\text{cd}})$ – optimal design and its cost value
end procedure

A designer can use this second stage to validate the correctness of the dynamics model used in motion planning and the improvements in co-design cost.

III. CO-DESIGN FORMULATION – ROBOT MODEL, COST FUNCTION AND CONSTRAINTS

Our design vector $\boldsymbol{\rho}$ consists of the lengths of the lower- and upper-leg limbs, the x-, and z-attachment points of the legs, the trunk shape: width, height and depth. We also model the x-, and z-positions of the two electronics boxes in the base of the robot. We thus implicitly constrain a symmetrical design along the direction of motion (the x-direction).

Next, we use an actuator model and optimize both the gear ratio and motor mass, which are the same for all motors, for simplicity. All these properties are included in the robot model to compute masses and inertias of the relevant links. For the limbs, we scale the volume linearly with the length of the leg as a simple proxy measure for structural integrity.

A. Actuator Model and Cost Function

Following [9] and [27] we model the mass of the motor m_m and parameterize the control limits $\underline{\boldsymbol{u}}$ and $\overline{\boldsymbol{u}}$ using an exponential regression based on m_m. We used the regression values from [9], which were fitted on datasheets from *Antigravity*, *Turnigy*, *MultiStar* and *PropDrive*:

$$\overline{\boldsymbol{u}} = -\underline{\boldsymbol{u}} = 5.48\, m_m^{0.97}. \tag{5}$$

Following [9], the dynamics of the system in the motion planning phase are frictionless and the actuator model is present in the co-design cost function. Given applied controls $\boldsymbol{u}$ at the robot's joints, the total torque at the motor (τ_t) is:

$$\tau_t = \frac{\boldsymbol{u}}{n} + \tau_f, \tag{6}$$

where n is the gear ratio and τ_f is the friction torque. The friction torque itself models the combined Coulomb and viscous friction at the transmission, which the motor needs to overcome. Thus:

$$\tau_f = \tau_\mu \,\text{sign}(\omega_m) + b\,\omega_m, \tag{7}$$

where τ_μ is the Coulomb friction parameter, b is the viscous friction parameter and ω_m is the motor angular speed, which is n times the joint angular speed.

We then consider three power losses – mechanical power, Joule effect from the motor winding resistance, and friction losses from the transmission:

$$P_{\text{mech}} = \tau_f \omega_m, \quad P_{\text{joule}} = \frac{1}{K_m}\tau_f^2, \quad P_{\text{fric}} = \tau_f \omega_m, \tag{8}$$

where $K_m = 0.15 m_m^{1.39}$ is the speed-torque gradient of the motor, again computed using an exponential regression on the motor mass.

Unlike in [9], we cannot ignore the mechanical power, as the foot start and end positions are dependent on the robot body structure and the total energy is not conserved between designs (and thus not constant). We thus follow [27] and compute the integral of the above terms ignoring power regenerative effects, summed over each of the motors:

$$J_{\text{cd}} = \int_{t_0}^{t_N} \sum_{\text{motor}} P_{\text{elec}} + \max(P_{\text{fric}}, 0)\ \text{dt}, \tag{9}$$

where $P_{\text{elec}} = \max(P_{\text{mech}} + P_{\text{joule}}, 0)$ is the positive electrical power (as defined in [27]). The friction power is separate, as it is due to the transmission. We integrate over the planning horizon and sum the non-negative power of each of the 12 SOLO motors. Thus $J_{\text{cd}}(\cdot)$ is the integral of these power terms, corresponding to the energy used during the motion (the total work). Finally, we note that the SOLO robot's actuators use a custom gearbox, thus making the gear ratio independent from the motor [20]. This allows us to treat them as separate optimization targets.

B. Constraints

We then specify constraints on the design vector $\boldsymbol{\rho}$. Firstly, we add a volumetric collision constraint on the electronics boxes, the Inertial Measurement Unit (IMU) box and the motherboard (MB) box:

$$(x_{\text{mb}} - z_{\text{imu}})^2 + (x_{\text{mb}} - z_{\text{imu}})^2 \leq (r_{\text{mb}} + r_{\text{imu}})^2, \tag{10}$$

where $x_{\text{mb}}, z_{\text{mb}}, x_{\text{imu}}, z_{\text{imu}}$ are the coordinates of the two boxes and $r_{\text{mb}} = 0.0361\text{m}, r_{\text{imu}} = 0.0282\text{m}$ are the radii of the smallest circumscribed sphere around them.

Finally, we specify linear constraints on the positions of the two electronics boxes and the positions of the legs so that they are within the base of the robot:

$$\begin{aligned} -\frac{w_b}{2} \leq & x_{\text{imu}} \leq \frac{w_b}{2}, -\frac{w_b}{2} \leq x_{\text{mb}} \leq \frac{w_b}{2}, -\frac{d_b}{2} \leq z_{\text{imu}} \leq \frac{d_b}{2}, \\ -\frac{d_b}{2} \leq & z_{\text{imu}} \leq \frac{d_b}{2}, -\frac{w_b}{2} \leq x_{\text{fr}} \leq \frac{w_b}{2}, -\frac{w_b}{2} \leq x_{\text{hr}} \leq \frac{w_b}{2}, \\ & -\frac{d_b}{2} \leq z_{\text{fr}} \leq \frac{d_b}{2}, \quad -\frac{d_b}{2} \leq z_{\text{hr}} \leq \frac{d_b}{2} \end{aligned} \tag{11}$$

where w_b and d_b are the width and depth of the base and $x_{\text{fr}}, z_{\text{fr}}$ and $x_{\text{hr}}, z_{\text{hr}}$ are the x- and z-coordinates of the front and hind shoulders. Note these inequalities constraints are defined in the upper level optimization.

C. Task Description

We are interested in optimizing the SOLO robot design for specific tasks. As such, we fix the task description in the lower motion planning level and optimize for the most efficient robot in terms of energy.

For trotting, the high-level motion task is to take two steps forward, each of 0.05m, with a fixed step height of 0.05m. The step height is fixed, as the optimal step height is always 0m. We allocated 22 and 37 knots[1] for the swing and double support phases of the motion, respectively, and used a symplectic Euler integrator with time-step of 10ms.

For jumping, the task is to jump forward 0.1m with a step height of 0.15m. We used the same integrator and time-step as in the trotting case. We defined 20 knots for the flight phase and 40 knots for the take-off and landing phases.

Finally, for both tasks, the initial design parameters $\boldsymbol{\rho}_0$ were matched to the Solo robot design.

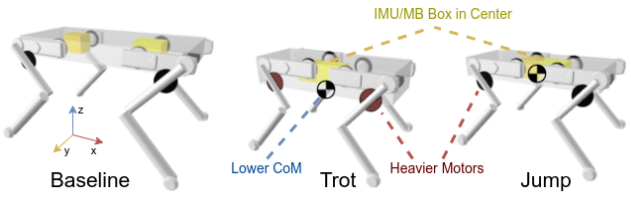
D. Results

The resulting robot designs and cost improvements are in Figures 3a and 3b. For both trotting and jumping, we plotted the energy contributions from the positive electrical power at the motor as P_{elec} versus the friction contribution from the transmission as P_{fric}. The algorithm chooses to minimize the electro-mechanical losses while increasing the friction losses. This is similar to [9], as small motors are much more energy inefficient since the reciprocal of the speed-torque gradient exponentially decreases ($K_m = 0.15 m_m^{1.39}$), increasing the Joule losses.

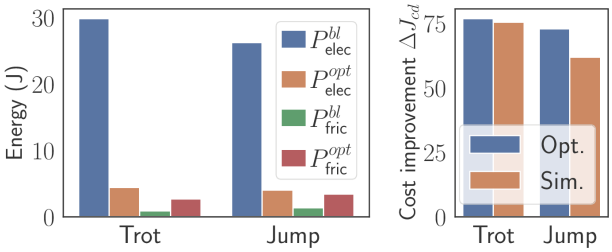
For trotting specifically, the friction losses are smaller, as trotting is a more static motion with smaller motor velocities, and friction is velocity-dependent. Thus the dominating cost is the electro-mechanical energy. This allows for a heavier robot with bigger motors than the optimal design for a jumping task – the optimal motor mass is $m_m = 0.179\text{kg}$ and gear ratio is $N = 16.062$ with a total robot weight of 3.805kg. The initial motor mass and gear ratio for the SOLO robot are $m_m = 0.053\text{kg}$ and $N = 9$ and the robot weighs 2.421kg. With a higher gear ratio the optimizer reduced the electro-mechanical energy further. Furthermore, we see a increase in base depth, which allows for the upper legs to be attached higher to the base of the robot. This allows for a lower center of mass, which can increase stability.

For jumping, however, a heavy robot is not optimal, as the entire mass of the robot needs to be moved. Thus the optimizer found $m_m = 0.168$ and $N = 17.325$ with a total mass of 3.592kg. The robot is heavier than the baseline, however the legs and the base are smaller. Compared to the optimal trotting design, the motors are lighter, but the gear

[1]Knots are points in time for the discretization of the optimal control problem.



(a) Resulting robot designs for trotting and jumping.



(b) Cost improvements.

Fig. 3: Robot designs and cost improvements on the trotting and jumping tasks. The costs are broken down for electric and friction contributions. We show the optimization and simulation percentage improvement on the bottom right.

ratio for both designs is similar. For both optimal designs, notably the boxes are optimally in the middle of the robot.

Finally, for both optimal designs, we also observed that the cost improvements remain in simulation within 10% of the ones found during optimization.

E. Optimality and Scalability

We compared our gradient-based co-design approach to the CMA-ES genetic algorithm on the trotting task in order to check convergence properties and optimality. We used the open-source CMA-ES library PyCMA [28]. In order to evaluate scalability, we varied the dimensions of the co-design vector by including subsets of the decision variables, namely:

1) $\dim(\boldsymbol{\rho}) = 4$ – leg lengths (front and back)
2) $\dim(\boldsymbol{\rho}) = 6$ – same as 4, and motor mass and gear ratio
3) $\dim(\boldsymbol{\rho}) = 9$ – same as 6, and base shape
4) $\dim(\boldsymbol{\rho}) = 13$ – same as 9, and electronics boxes
5) $\dim(\boldsymbol{\rho}) = 17$ – full model

For CMA-ES we specified a quadratic soft penalty for all constraints. We ran CMA-ES with population sizes $N = [10, 20, 50]$ and selected $N = 50$, which achieved the same or lower costs than our approach on all problems.

Importantly, this corresponds to 50 calls to the motion planner by CMA at each iteration versus $\dim(\boldsymbol{\rho}) + 1$ calls for our approach. Thus we measure time to convergence, as a per-iteration measure would favor our approach heavily. Both approaches used multi-threading with 8 threads and were given the same computational budget.

We then plot the mean and standard deviation for costs and time to 99% convergence over 20 runs at $N = 50$ in Fig. 4. On the trotting task our approach has better scalability

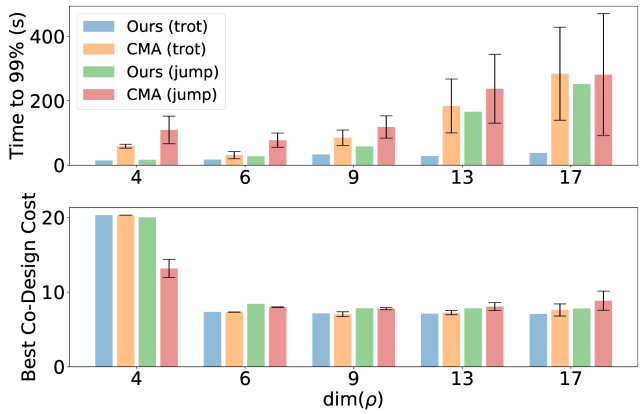


Fig. 4: Scalability results for different problem dimensions.

than CMA-ES, which is expected given the convergence properties of CMA-ES. On the jumping task convergence is slower for both with CMA-ES having a large deviation in convergence time for larger problems. Importantly, CMA-ES is not deterministic and although the average time for the complex jumping task is comparable to our approach, the worst-case time we observed is 600 seconds for CMA versus 252 seconds for our approach (both for the 17-DOF jumping co-design task).

Finally, of interest is that we are able to achieve similar best co-design costs as CMA-ES across problem dimensions for the given co-design problems. This could indicate that our local gradient-based bilevel approach can achieve globally optimal solutions in practice, for problems like the ones studied here.

IV. Conclusion

In this paper we proposed a modular co-design framework for dynamic quadruped locomotion. Our approach is based on bilevel optimization and exploits the derivatives of a hybrid nonlinear optimal control problem (lower level problem that describes the motion planner). Our algorithm allows for the use of complex, state-of-the-art motion planners in the co-design loop together with linear and nonlinear design constraints in the upper level. One advantage of using DDP-style motion planning in our work is the guaranteed physical feasibility of the motion. When using other motion planners, this consistency might not be guaranteed and the resulting gradients might be noisy if the motion constraints are not satisfied. We demonstrated that a coupling between the upper and lower level costs is beneficial. Note that we have a weak coupling, where the lower level has a regularization on the square of the torques and the upper level has the Joule effect cost, also on the square of the torques.

Future work lies in using analytical derivatives instead of using finite differences, which introduce numerical errors when computing the derivative of the motion planner. Furthermore, our approach can use any differentiable motion planner and there are interesting opportunities in using different motion planners and formulations (for instance different

contact models and constraints) which can enable co-design in more complex domains, for instance with sliding or slipping contacts. Additionally, of interest is handling more complex state constraints that come from the environment, for instance for footstep planning – determining the contact locations and timings of footsteps.

REFERENCES

[1] 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," *Journal of Systems and Control Engineering*, vol. 225, 2011.

[2] Q. Li, W. Zhang, and L. Chen, "Design for control-a concurrent engineering approach for mechatronic systems design," vol. 6, 2001.

[3] C. Mastalli, R. Budhiraja, W. Merkt, G. Saurel, B. Hammoud, M. Naveau, J. Carpentier, L. Righetti, S. Vijayakumar, and N. Mansard, "Crocoddyl: An Efficient and Versatile Framework for Multi-Contact Optimal Control," in *IEEE ICRA*, 2020.

[4] C. Mastalli, W. Merkt, G. Xin, J. Shim, M. Mistry, I. Havoutis, and S. Vijayakumar, "Agile Maneuvers in Legged Robots: a Predictive Control Approach," 2022.

[5] K. Wampler and Z. Popović, "Optimal gait and form for animal locomotion," *ACM Transactions on Graphics*, vol. 28, 2009.

[6] K. M. Digumarti, C. Gehring, S. Coros, J. Hwangbo, and R. Siegwart, "Concurrent optimization of mechanical design and locomotion control of a legged robot," in *Mobile Service Robotics*. World Scientific, 2014.

[7] S. Ha, S. Coros, A. Alspach, J. Kim, and K. Yamane, "Task-based limb optimization for legged robots," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 2062–2068.

[8] M. Chadwick, H. Kolvenbach, F. Dubois, H. F. Lau, and M. Hutter, "Vitruvio: An Open-Source Leg Design Optimization Toolbox for Walking Robots," *IEEE Robot. Automat. Lett. (RA-L)*, vol. 5, 2020.

[9] G. Fadini, T. Flayols, A. del Prete, N. Mansard, and P. Souères, "Computational design of energy-efficient legged robots: Optimizing for size and actuators," in *ICRA2021*, 2020.

[10] M. N. Omidvar and X. Li, "A comparative study of cma-es on large scale global optimisation," in *Australasian Joint Conference on Artificial Intelligence*. Springer, 2010, pp. 303–312.

[11] N. Hansen, "Benchmarking a bi-population cma-es on the bbob-2009 function testbed," in *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, 2009, pp. 2389–2396.

[12] K. Mombaur, "Using optimization to create self-stable human-like running," *Robotica*, vol. 27, 2009.

[13] G. Buondonno, J. Carpentier, G. Saurel, N. Mansard, A. De Luca, and J.-P. Laumond, "Actuator design of compliant walkers via optimal control," in *IEEE/RSJ IROS*, 2017.

[14] A. Spielberg, B. Araki, C. Sung, R. Tedrake, and D. Rus, "Functional co-optimization of articulated robots," in *IEEE ICRA*, 2017.

[15] W. Sun, G. Tang, and K. Hauser, "Fast uav trajectory optimization using bilevel optimization with analytical gradients," in *2020 American Control Conference (ACC)*. IEEE, 2020, pp. 82–87.

[16] S. Ha, S. Coros, A. Alspach, J. Kim, and K. Yamane, "Computational co-optimization of design parameters and motion trajectories for robotic systems," *The International Journal of Robotics Research*, vol. 37, 2018.

[17] R. Desai, B. Li, Y. Yuan, and S. Coros, "Interactive Co-Design of Form and Function for Legged Robots using the Adjoint Method," *arXiv:1801.00385 [cs]*, Apr. 2018, arXiv: 1801.00385. [Online]. Available: http://arxiv.org/abs/1801.00385

[18] M. Geilinger, R. Poranne, R. Desai, B. Thomaszewski, and S. Coros, "Skaterbots: optimization-based design and motion synthesis for robotic creatures with legs and wheels," *ACM Transactions on Graphics*, vol. 37, 2018.

[19] M. Geilinger, S. Winberg, and S. Coros, "A Computational Framework for Designing Skilled Legged-Wheeled Robots," *IEEE Robot. Automat. Lett. (RA-L)*, vol. 5, 2020.

[20] F. Grimminger, A. Meduri, M. Khadiv, J. Viereck, M. Wüthrich, M. Naveau, V. Berenz, S. Heim, F. Widmaier, T. Flayols, J. Fiene, A. Badri-Spröwitz, and L. Righetti, "An Open Torque-Controlled Modular Robot Architecture for Legged Locomotion Research," *IEEE Robot. Automat. Lett. (RA-L)*, vol. 5, 2020.

[21] J. Carpentier, G. Saurel, G. Buondonno, J. Mirabel, F. Lamiraux, O. Stasse, and N. Mansard, "The pinocchio c++ library: A fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives," in *2019 IEEE/SICE International Symposium on System Integration (SII)*. IEEE, 2019, pp. 614–619.

[22] R. H. Byrd, J. Nocedal, and R. A. Waltz, "K nitro: An integrated package for nonlinear optimization," in *Large-scale nonlinear optimization*. Springer, 2006, pp. 35–59.

[23] R. Featherstone, *Rigid Body Dynamics Algorithms*. Berlin, Heidelberg: Springer-Verlag, 2007.

[24] C. Mastalli, W. Merkt, J. Marti-Saumell, H. Ferrolho, J. Sola, N. Mansard, and S. Vijayakumar, "A Direct-Indirect Hybridization Approach to Control-Limited DDP," 2021.

[25] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning," *GitHub repository*, 2016.

[26] L. Pecyna, A. Cangelosi, and A. Di Nuovo, "A Deep Neural Network for Finger Counting and Numerosity Estimation," in *IEEE Symposium Series on Computational Intelligence (SSCI)*, 2019.

[27] Y. Yesilevskiy, Z. Gan, and C. David Remy, "Energy-optimal hopping in parallel and series elastic one-dimensional monopeds," *Journal of Mechanisms and Robotics*, vol. 10, no. 3, p. 031008, 2018.

[28] N. Hansen, Y. Akimoto, and P. Baudis, "CMA-ES/pycma on Github," Zenodo, DOI:10.5281/zenodo.2559634, 2019.