Optimal Design of Robotic Character Kinematics

GUIREC MALOISEL*, Disney Research, Switzerland
CHRISTIAN SCHUMACHER*, Disney Research, Switzerland
ESPEN KNOOP*, Disney Research, Switzerland
RUBEN GRANDIA*, Disney Research, Switzerland
MORITZ BÄCHER*, Disney Research, Switzerland

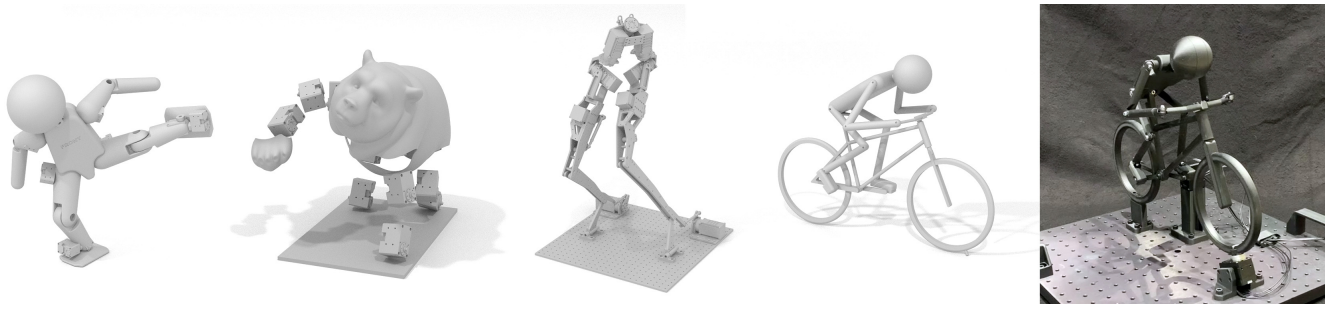

Fig. 1. We present a method for optimizing the kinematics of robot characters, supporting spatial linkages with arbitrary kinematics including loops and overactuation. Our method enables the rapid design of expressive robotic characters, through optimally placing passive and active degrees of freedom.

The kinematic motion of a robotic character is defined by its mechanical joints and actuators that restrict the relative motion of its rigid components. Designing robots that perform a given target motion as closely as possible with a fixed number of actuated degrees of freedom is challenging, especially for robots that form kinematic loops. In this paper, we propose a technique that simultaneously solves for optimal design and control parameters for a robotic character whose design is parameterized with configurable joints. At the technical core of our technique is an efficient solution strategy that uses dynamic programming to solve for optimal state, control, and design parameters, together with a strategy to remove redundant constraints that commonly exist in general robot assemblies with kinematic loops. We demonstrate the efficacy of our approach by either editing the design of an existing robotic character, or by optimizing the design of a new character to perform a desired motion.

CCS Concepts: • **Computing methodologies** → **Physical simulation**; **Computational control theory**.

Additional Key Words and Phrases: optimal design and control, robotic characters

*All authors contributed equally to this research.

Authors' addresses: Guirec Maloisel, guirec.maloisel@disney.com, Disney Research, Switzerland; Christian Schumacher, christian.schumacher@disney.com, Disney Research, Switzerland; Espen Knoop, espen.knoop@disney.com, Disney Research, Switzerland; Ruben Grandia, ruben.grandia@disney.com, Disney Research, Switzerland; Moritz Bächer, moritz.baecher@disney.com, Disney Research, Switzerland.

0730-0301/2023/12-ART $15.00
https://doi.org/10.1145/3618404

ACM Reference Format:
Guirec Maloisel, Christian Schumacher, Espen Knoop, Ruben Grandia, and Moritz Bächer. 2023. Optimal Design of Robotic Character Kinematics. *ACM Trans. Graph.* 42, 6 (December 2023), 15 pages. https://doi.org/10.1145/3618404

1 INTRODUCTION

The design of robotic characters whose motion is driven by complex mechanisms that in turn are driven by several actuators remains an iterative trial-and-error process. Despite recent advances in computational design, we lack techniques that are applicable to complex robot designs with arbitrary kinematic structures. In particular, the optimization of linkages with spatial and closed-loop kinematics remains underexplored.

In this paper, we propose an optimal design technique that takes an initial design of a fully or overactuated robot as input, parameterized with a set of configurable joints. With a set of high-level objectives, we can either edit the design to change a robot's function, or optimize a parameterized version of a new robot to achieve a desired motion as closely as possible.

While it is relatively straightforward to come up with an initial design of a robot's kinematics, it is non-trivial to predict the effect of a local change to the motion of a character, especially for designs that contain kinematic loops and are driven by multiple actuators. Furthermore, optimizing the kinematics for motions with long time horizons results in large, numerically challenging optimization problems. To allow for an iterative design process, it is crucial that a tool produces solutions efficiently and robustly.

Our technique addresses these challenges by enforcing kinematic constraints, efficiently identifying optimal design and control parameters by sequentially solving a quadratic approximation of the constrained design-control problem with dynamic programming. At the technical core of our technique is a reduction of the quadratic approximation to a discrete-time optimal control problem, utilizing

the recursive structure in the problem formulation, together with subspace projection to remove constraints while preserving this structure. Because constraints that enforce the restricted relative motion at mechanical joints, actuators, and configurable joints can lead to a set that contains redundancy, we propose an elimination technique that is agnostic to the kinematics of the specific robot.

As we demonstrate with four examples, our system aids with the iterative design and editing of complex robotic characters where the underlying design space is challenging to optimally navigate with a manual trial-and-error approach.

Succinctly, our contributions are

- a parameterization of a robot's design with configurable joints that enables rapid design iteration.
- a design optimization that eliminates the redundancy in constraints, and is therefore agnostic to the robot kinematics, interfacing with general fully- or overactuated input.
- a reduction of the local approximate design-control to a discrete-time optimal control problem that enables an efficient, scalable, and robust solve using dynamic programming.

2 RELATED WORK

Mechanism Design. Computational mechanism design problems have been studied both within and outside the computer graphics community. Methods for synthesizing and editing linkages, given a periodic animation input have been developed [Bächer et al. 2015; Coros et al. 2013; Thomaszewski et al. 2014]. However, these works focused on single-actuator systems which were primarily planar.

There has also been some work targeting spatial (3D) mechanisms. Zhang et al. [2017] presented a method for retargeting 3D mechanisms into a prescribed bounding volume, while maintaining their function. Cams have also been explored as a method for producing expressive spatial motions [Cheng et al. 2022, 2021; Zhu et al. 2012]. Song et al. [2017] developed a design tool for spatial single-actuator wind-up-toy mechanisms. Huber et al. [2021] consider the discrete design problem of connecting actuators to functions of an animatronic head.

Tools for visualizing and prototyping linkages have been developed by the graphics community [Koo et al. 2014; Mitra et al. 2010]. Liu et al. [2022] analyze the worst-case rigidity of linkage assemblies. There is also work towards recovering the kinematics of a mechanism from a scan [Lin et al. 2017] or from video input [Ceylan et al. 2013]. Related to our parameterization of robot designs, Schulz et al. [2014] developed a tool for the parameterized design of a broad range of objects.

Design tools for mechanisms that exploit compliance have been presented [Megaro et al. 2017a,b; Skouras et al. 2013; Zhang et al. 2021]. There has also been some work targeting the control [Hoshyari et al. 2019] and design [Ebrahimi et al. 2019] optimization for dynamic systems.

Maloisel et al. [2021] solve a design optimization problem for spatial mechanisms, asking for a target region to be reachable and free from singularities. However, this method assumes a non-overactuated mechanism with the same number of objectives as actuators, and their method does not scale well to high-DoF systems.

Optimal Design of Robots. A number of methods have been developed for the optimal control and design of fixed-base and mobile robots with both legs and wheels [Desai et al. 2017; Geilinger et al. 2018; Ha et al. 2018a, 2016, 2017, 2018b]. Interactive design tools for designing legged robots have been proposed [Megaro et al. 2015; Schulz et al. 2017], also for authoring expressive motions [Desai et al. 2019]. Feng et al. [2019] presented a design tool for skinned quadruped robots. More specific design optimization problems have also been considered, including quadruped legs [Fadini et al. 2021] and 7-DoF robot arms [Hwang et al. 2017]. However, all of these works are restricted to articulated body kinematics, requiring a tree-like kinematics structure.

There has been some work targeting the design of closed-loop robot mechanisms. De Vincenti et al [2021] considered the design optimization of a planar hind leg mechanism for a robot quadruped, and Borisov et al. [2021] studied the computational design of underactuated grippers.

Recent work has also considered the co-optimization of robot design and robot controllers [Dinev et al. 2022; Xu et al. 2021a,b; Zhao et al. 2020]. Spielberg et al. [2017] solved a parametric trajectory optimization problem, where parameters and trajectories were simultaneously solved for. Du et al. [2016] solved the design and control problem for multicopters. Learning-based approaches have also been explored [Bjelonic et al. 2023; Schaff et al. 2019].

In contrast, we propose a kinematic design optimization that is applicable to robots with *arbitrarily complex kinematic loops*, recasting the design as a control problem to efficiently co-optimize design and control parameters using dynamic programming, and introducing a technique to remove redundancy in constraints to guarantee well-posedness of the problem for *generic fully- or overactuated input*.

Parallel robots. In the robotics community, there has been significant study devoted to understanding and optimizing parallel robot mechanisms [Merlet 2005, 2006]. These robots generally have a single end effector which is driven by multiple parallel kinematic chains. Methods for optimizing specific robot designs have been developed [Merlet 2002; Pierrot et al. 2009; Wu et al. 2014], including overactuated systems [Leguay-Durand and Reboulet 1997]. Collard et al. [2009] study the optimal design of closed-loop multibody systems in a general setting, but enforce constraints using a soft penalty, and need to tune weights to obtain a good trade-off with the design objectives. There has also been some work developing more general methodology for parallel robot design [Kelaiaia et al. 2023; Lou et al. 2013; Miller 2004; Zanganeh and Angeles 1997].

Our work differs in that we can handle arbitrary robot kinematic structures, and can track an arbitrary number of objectives—with a least-squares solution being returned if the objectives cannot be perfectly tracked.

Optimal control. The problem of simultaneously optimizing design parameters, together with state and control trajectories, can be viewed as a *parameterized optimal control problem*, for which two general solution strategies exist: In a first strategy, an optimal control problem is solved in an inner loop for a fixed set of parameters. By extracting sensitivities of the solution w.r.t. the parameters [Büskens and Maurer 2001], we can then solve for optimal parameter values

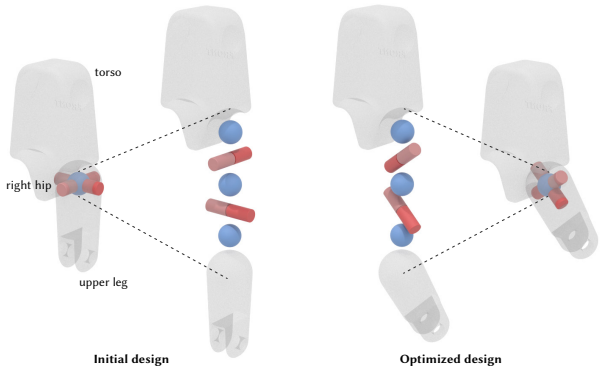

Fig. 2. **Overview.** We parameterize the design of a robot by adding a set of configurable joints, illustrated here with the right hip of Kickbot, a humanoid robot: To parameterize the axes of the two revolute actuators (red cylinders), we add three configurable spherical joints (blue spheres) between the torso, the two actuators, and the upper leg (Initial design). Our technique then outputs optimal design parameters for the three configurable joints and optimal control parameters for the two revolute joints so that the robot achieves a target performance as closely as possible (Optimized design).

in an outer loop [Amos et al. 2018]. However, this strategy requires that the inner optimization has a solution where all constraints are satisfied. For the kinematics design problem we consider here, this cannot be guaranteed as explained below. Alternatively, as done in this work, the parameters can be treated as an integral part of a single, larger optimal control problem by treating them as constant state variables with unknown initial conditions [Kobilarov et al. 2015; Oshin et al. 2022]. This second solution strategy allows constraint violations at intermediate iterations.

One of our core contributions is the reduction of a local quadratic approximation of our nonlinear constrained design problem to a standard discrete-time optimal control problem that we can solve efficiently using differential dynamic programming, exploiting the inherent sparsity along the time horizon [Jacobson and Mayne 1970; Rawlings et al. 2017]. Our solution strategy can be understood as an instance of the second class of strategies to solve parameterized optimal control problems.

3 OVERVIEW

We take as input to our pipeline a robot kinematics model. To allow for the robot design to be optimized, we introduce configurable joints. These are similar to actuators, but their "control parameters" can only be set once for a particular animation. We support configurable versions of all common joint types, allowing for a wide range of design problems to be defined. Our processing is illustrated in Fig. 2.

With our modeling, we target two use cases: editing of an existing robot design, and the design of new robots. Note that in both cases we assume that the desired motion of the robot is known at design time.

For the former, we start with a parameterized robot model performing an animation. We run a kinematics simulation on the existing animation and extract trajectories of points of interest on the robot. By representing these trajectories with spatial spline curves, or applying affine transformations, we can then specify a new motion target that we can achieve by optimizing the design parameters of our configurable joints as also the control parameters of our actuators.

For the latter, a user first creates an initial design of a robot, together with a target motion that can either come from a rigged character or mocap. We then optimize the design and control parameters of the configurable joints and actuators to approximate the target motion as closely as possible. This second use case enables fast iteration to identify a good number of degrees of freedom that can express the artistic intent.

With our configurable joints, we can represent non-mechanical entities for pre-build, and reconfigurable joints of existing robots for post-build design optimizations (see Sec. 8). We can use them to parameterize a component's "length" (e.g., with a parameterized prismatic joint), as well as the position and orientation of a mechanical joint or actuator. We assume our input robots to be fully actuated, meaning that all actuated degrees of freedom can be instantaneously and independently controlled. However, in contrast to other techniques, we support overactuated systems and interface with generic spatial input with kinematic loops.

In the sections that follow, we first discuss how we represent a robot's kinematics (Sec. 4), then introduce our design optimization and fast solution strategy in Secs. 5 and 6. Thereafter, we discuss our set of objectives for our two use cases (Sec. 7), before we discuss results and conclude (Secs. 8 and 9).

4 REPRESENTING A ROBOT'S KINEMATICS

Before we discuss how we optimize a robot's design, we briefly review its representation.

State. A robot consists of a set of rigid components whose time-varying states we represent with 7D vectors that encode their positions $\mathbf{c}$ and orientations $\mathbf{q}$. For orientations, we rely on quaternions and enforce their unit length with constraints of the form $\mathbf{q} \cdot \mathbf{q} = 1$. We use $\mathbf{s}$ to refer to the full state of a robot. Without loss of generality, we assume that all orientations are set to the identity in the character's initial or rest pose. This makes the formulation of kinematic constraints easier.

4.1 Formulating Kinematic Constraints

Joints. Mechanical joints restrict the relative motion between pairs of bodies, A and B. To formulate constraints, we define a frame whose global position, $\mathbf{x}$, coincides with the position of the joint in the robot's initial pose, and whose axes $\mathbf{a}_x$, $\mathbf{a}_y$, and $\mathbf{a}_z$ align with its degrees of freedom. Because we assume initial orientations to be set to the identity, the local frame axes in the body coordinates of A and B equal the global axes, and the local frame positions are $\mathbf{x}_A = \mathbf{x} - \mathbf{c}_A$ and $\mathbf{x}_B = \mathbf{x} - \mathbf{c}_B$. We can then formulate constraints between pairs of components as illustrated in Fig. 3 and summarized in Tab. 1, Joints. This formulation is similar to previous work [Schumacher et al. 2021].

We support all common joint types: Cartesian joints have three translational degrees of freedom, and the more commonly used

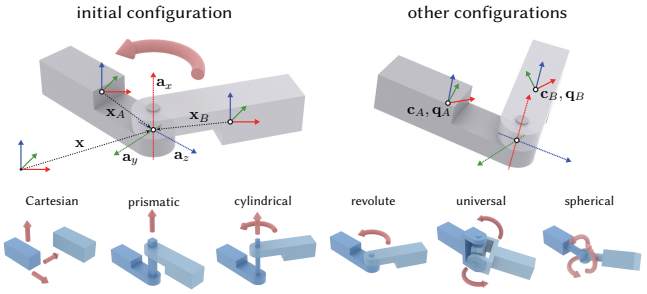

Fig. 3. **Kinematic Constraints.** We assume the orientations, $\mathbf{q}_A$ and $\mathbf{q}_B$, of the two bodies A and B, to be set to the identity in the initial configuration, exemplified here with a revolute joint, actuator, or configurable joint (top). This convention ensures that the three frame axes, $\mathbf{a}_x$, $\mathbf{a}_y$, $\mathbf{a}_z$, are the same in global and local body coordinates for a robot in its initial neutral pose, simplifying the formulation of constraints. We use the global position of the frame, $\mathbf{x}$, to define the positions, $\mathbf{x}_A$ and $\mathbf{x}_B$, in local body coordinates. We support joint, actuator, and configurable joint types with varying translational and rotational degrees of freedom (bottom).

spherical joints have three rotational degrees of freedom. Cylindrical joints and universal joints have two degrees of freedom, and prismatic and revolute joints have a single translational or rotational degree of freedom. We also support a fixed joint that locks the relative motion between a pair of components. This is a useful joint during design exploration, because it can be used to "freeze" degrees of freedom between components, without having to merge them. To prevent a robot from moving in space, we support a ground joint which keeps a single component in its initial position and orientation.

Actuators. For each mechanical joint, we formulate constraints for a corresponding actuator. To this end, we complement the passive constraints with additional constraints, parameterizing them with time-varying control parameters $\mathbf{u}$ (Tab. 1, Actuators). Once we choose values for $\mathbf{u}$, the relative states of the two components that they connect are fully determined. While revolute or prismatic actuators are widely used, spherical actuators exist but are rare. The other actuators are useful to decouple a mechanism at an arbitrary set of mechanical joints, recording their motion, and treating them as actuators with control parameters that we exclude from an optimization to guarantee consistency.

Configurable Joints. To parameterize a robot's kinematics, we introduce an additional joint type that we refer to as configurable joints. They are similar to actuators, but are parameterized with design parameters $\mathbf{p}$ that remain the same throughout an animation, and don't vary with time like control parameters do (Tab. 1, Configurable Joints, $\mathbf{u}$ replaced with $\mathbf{p}$).

Configurable joints provide a useful parameterization interface for two reasons: (1) Mechanical engineers are used to designing robots with mechanical joints and actuators. Configurable joints are therefore a natural extension to create a parameterized robot design using standard CAD modeling tools. (2) Configurable joints allow us to parameterize the design of a robot before or after its built (see Sec. 8).

Table 1. **Joints, Actuators, and Configurable Joints** constrain the relative motion between pairs of components A and B, whose states we represent with 7-vectors $\mathbf{s}_A$ and $\mathbf{s}_B$ that encode the components' positions, $\mathbf{c}_A$ and $\mathbf{c}_B$, and their orientations, $\mathbf{q}_A$ and $\mathbf{q}_B$. We use the Euler-Rodrigues formula to convert a unit-quaternion $\mathbf{q}$ to a rotation matrix $\mathbf{R}(\mathbf{q})$ and $\mathbf{R}(u, \mathbf{a})$ to represent a rotation by u about axis $\mathbf{a}$. $\mathbf{R}_A$ and $\mathbf{R}_B$ abbreviate $\mathbf{R}(\mathbf{q}_A)$ and $\mathbf{R}(\mathbf{q}_B)$. For cylindrical and prismatic joints, we define the difference vector $\mathbf{d} = (\mathbf{R}_A\mathbf{x}_A + \mathbf{c}_A) - (\mathbf{R}_B\mathbf{x}_B + \mathbf{c}_B)$. The Cartesian actuator or configurable joint has three parameters, $\mathbf{u}$, that determine the translations along the three axes, $\mathbf{A} = [\mathbf{a}_x, \mathbf{a}_y, \mathbf{a}_z]$. The spherical actuator or configurable joint is parameterized with a quaternion $\mathbf{u}$ whose length we constrain to 1 during optimizations. The cylindrical and universal actuators or configurable joints are parameterized with two parameters, u_1 and u_2, and the prismatic and revolute actuators or configurable joints with a single parameter u. The fixed joint does not have a corresponding actuator or configurable joint, because it already removes all degrees of freedom. The ground joint keeps a single component fixed in space at its initial position $\mathbf{c}_0$ (and orientation which is set to the identity). $\mathbf{e}_x$, $\mathbf{e}_y$, and $\mathbf{e}_z$ are the three unit vectors.

	Joints	Actuators & Configurable Joints
Cartesian	$(\mathbf{R}_A\,\mathbf{a}_x)\cdot(\mathbf{R}_B\,\mathbf{a}_y)$ $(\mathbf{R}_A\,\mathbf{a}_x)\cdot(\mathbf{R}_B\,\mathbf{a}_z)$ $(\mathbf{R}_A\,\mathbf{a}_y)\cdot(\mathbf{R}_B\,\mathbf{a}_z)$	$(\mathbf{R}_A(\mathbf{x}_A+\mathbf{A}\mathbf{u})+\mathbf{c}_A)-(\mathbf{R}_B\mathbf{x}_B+\mathbf{c}_B)$ $(\mathbf{R}_A\,\mathbf{a}_x)\cdot(\mathbf{R}_B\,\mathbf{a}_y)$ $(\mathbf{R}_A\,\mathbf{a}_x)\cdot(\mathbf{R}_B\,\mathbf{a}_z)$ $(\mathbf{R}_A\,\mathbf{a}_y)\cdot(\mathbf{R}_B\,\mathbf{a}_z)$
spherical	$(\mathbf{R}_A\mathbf{x}_A+\mathbf{c}_A)-(\mathbf{R}_B\mathbf{x}_B+\mathbf{c}_B)$	$(\mathbf{R}_A\mathbf{x}_A+\mathbf{c}_A)-(\mathbf{R}_B\mathbf{x}_B+\mathbf{c}_B)$ $(\mathbf{R}_A\,\mathbf{R}(\mathbf{u})\,\mathbf{a}_x)\cdot(\mathbf{R}_B\,\mathbf{a}_y)$ $(\mathbf{R}_A\,\mathbf{R}(\mathbf{u})\,\mathbf{a}_x)\cdot(\mathbf{R}_B\,\mathbf{a}_z)$ $(\mathbf{R}_A\,\mathbf{R}(\mathbf{u})\,\mathbf{a}_y)\cdot(\mathbf{R}_B\,\mathbf{a}_z)$
cylindrical	$\mathbf{d}\cdot(\mathbf{R}_B\mathbf{a}_y)$ $\mathbf{d}\cdot(\mathbf{R}_B\mathbf{a}_z)$ $(\mathbf{R}_A\,\mathbf{a}_x)\cdot(\mathbf{R}_B\,\mathbf{a}_y)$ $(\mathbf{R}_A\,\mathbf{a}_x)\cdot(\mathbf{R}_B\,\mathbf{a}_z)$	$(\mathbf{R}_A(\mathbf{x}_A+\mathbf{a}_x u_1)+\mathbf{c}_A)-(\mathbf{R}_B\mathbf{x}_B+\mathbf{c}_B)$ $(\mathbf{R}_A\,\mathbf{a}_x)\cdot(\mathbf{R}_B\,\mathbf{a}_y)$ $(\mathbf{R}_A\,\mathbf{a}_x)\cdot(\mathbf{R}_B\,\mathbf{a}_z)$ $(\mathbf{R}_A\,\mathbf{R}(u_2,\mathbf{a}_x)\,\mathbf{a}_y)\cdot(\mathbf{R}_B\,\mathbf{a}_z)$
universal	$(\mathbf{R}_A\mathbf{x}_A+\mathbf{c}_A)-(\mathbf{R}_B\mathbf{x}_B+\mathbf{c}_B)$ $(\mathbf{R}_A\,\mathbf{a}_x)\cdot(\mathbf{R}_B\,\mathbf{a}_y)$	$(\mathbf{R}_A\mathbf{x}_A+\mathbf{c}_A)-(\mathbf{R}_B\mathbf{x}_B+\mathbf{c}_B)$ $(\mathbf{R}_A\,\mathbf{a}_x)\cdot(\mathbf{R}_B\,\mathbf{a}_y)$ $(\mathbf{R}_A\,\mathbf{R}(u_1,\mathbf{a}_x)\,\mathbf{R}(u_2,\mathbf{a}_y)\,\mathbf{a}_x)\cdot(\mathbf{R}_B\,\mathbf{a}_z)$ $(\mathbf{R}_A\,\mathbf{R}(u_1,\mathbf{a}_x)\,\mathbf{R}(u_2,\mathbf{a}_y)\,\mathbf{a}_y)\cdot(\mathbf{R}_B\,\mathbf{a}_z)$
prismatic	$\mathbf{d}\cdot(\mathbf{R}_B\mathbf{a}_y)$ $\mathbf{d}\cdot(\mathbf{R}_B\mathbf{a}_z)$ $(\mathbf{R}_A\,\mathbf{a}_x)\cdot(\mathbf{R}_B\,\mathbf{a}_y)$ $(\mathbf{R}_A\,\mathbf{a}_x)\cdot(\mathbf{R}_B\,\mathbf{a}_z)$ $(\mathbf{R}_A\,\mathbf{a}_y)\cdot(\mathbf{R}_B\,\mathbf{a}_z)$	$(\mathbf{R}_A(\mathbf{x}_A+\mathbf{a}_x u)+\mathbf{c}_A)-(\mathbf{R}_B\mathbf{x}_B+\mathbf{c}_B)$ $(\mathbf{R}_A\,\mathbf{a}_x)\cdot(\mathbf{R}_B\,\mathbf{a}_y)$ $(\mathbf{R}_A\,\mathbf{a}_x)\cdot(\mathbf{R}_B\,\mathbf{a}_z)$ $(\mathbf{R}_A\,\mathbf{a}_y)\cdot(\mathbf{R}_B\,\mathbf{a}_z)$
revolute	$(\mathbf{R}_A\mathbf{x}_A+\mathbf{c}_A)-(\mathbf{R}_B\mathbf{x}_B+\mathbf{c}_B)$ $(\mathbf{R}_A\,\mathbf{a}_x)\cdot(\mathbf{R}_B\,\mathbf{a}_y)$ $(\mathbf{R}_A\,\mathbf{a}_x)\cdot(\mathbf{R}_B\,\mathbf{a}_z)$	$\mathbf{R}(\mathbf{q}^A)\mathbf{x}^A+\mathbf{c}^A-(\mathbf{R}(\mathbf{q}^B)\mathbf{x}^B+\mathbf{c}^B)$ $(\mathbf{R}_A\,\mathbf{a}_x)\cdot(\mathbf{R}_B\,\mathbf{a}_y)$ $(\mathbf{R}_A\,\mathbf{a}_x)\cdot(\mathbf{R}_B\,\mathbf{a}_z)$ $(\mathbf{R}_A\,\mathbf{R}(u,\mathbf{a}_x)\,\mathbf{a}_y)\cdot(\mathbf{R}_B\,\mathbf{a}_z)$
fixed	$(\mathbf{R}_A\mathbf{x}_A+\mathbf{c}_A)-(\mathbf{R}_B\mathbf{x}_B+\mathbf{c}_B)$ $(\mathbf{R}_A\,\mathbf{a}_x)\cdot(\mathbf{R}_B\,\mathbf{a}_y)$ $(\mathbf{R}_A\,\mathbf{a}_x)\cdot(\mathbf{R}_B\,\mathbf{a}_z)$ $(\mathbf{R}_A\,\mathbf{a}_y)\cdot(\mathbf{R}_B\,\mathbf{a}_z)$	
ground	$\mathbf{c}-\mathbf{c}_0$ $(\mathbf{R}\,\mathbf{e}_x)\cdot\mathbf{e}_y$ $(\mathbf{R}\,\mathbf{e}_x)\cdot\mathbf{e}_z$ $(\mathbf{R}\,\mathbf{e}_y)\cdot\mathbf{e}_z$	

Constraints. Given an initial design of a robot in its rest configuration, we create a set of constraints according to Tab. 1

$$\boldsymbol{C}(\mathbf{p}, \mathbf{u}, \mathbf{s}) = \mathbf{0} \tag{1}$$

that represents, together with the state of the components, the kinematics of the robot. The above constraints include a unit length constraint, $\mathbf{q}\cdot\mathbf{q}-1=0$, for each component of the robot. Given a set of design and control parameters, we can use this set of constraints

to solve for the state of the robot, $\mathbf{s}(\mathbf{p}, \mathbf{u})$, and therefore to simulate its kinematic motion.

5 OPTIMIZING A ROBOT'S DESIGN

Our goal is to optimize a character's parameterized joints to achieve a desired motion as closely as possible. Because optimal control parameters change if we make adjustments to design parameters, we need to solve for them simultaneously.

A second important point is that a design parameter change has an effect on the entire motion of a robot, and we therefore need to measure the performance of a particular design for an entire animation to make an optimal choice.

While continuous formulations are elegant, we directly consider a discretized problem statement that shares similarities with a discrete-time optimal control problem. This choice is motivated by our observation that the numerical solution of a discrete-time problem is often better behaved than the solution of its corresponding continuous-time problem. The theory underlying discrete-time problems is also well understood.

To this end, we discretize the target motion into n time intervals Δt and $k = 0, \ldots, n$ time steps, and introduce intermediate objectives, f, that measure a robot's performance with respect to a target animation and ensure that actuator positions and velocities remain within limits. To directly penalize actuator velocities near limits, we introduce time-varying velocity variables $\mathbf{v}$ and set them to $\dot{\mathbf{u}}$. We also introduce a terminal objective F that measures the difference between a robot's terminal state and its user-specified target. We introduce our objectives in Sec. 7.

To minimize the number of optimization variables, we could work with a single set of design parameters $\mathbf{p}$. However, this choice results in a Hessian of the Lagrangian which is no longer a banded matrix due to sparsity along the time dimension. In addition, it would prevent us from applying a fast solution strategy that is based on dynamic programming, requiring a recursive structure and local dependence between consecutive variables. We therefore work with per-time-step design parameters $\mathbf{p}_k$, and enforce equality between them with constraints $\mathbf{p}_{k+1} = \mathbf{p}_k$.

To ensure that orientations in our design and control parameterization are singularity-free, we use quaternions as control and design parameters for spherical and ground actuators and reconfigurable joints (see Tab. 1, right column). To enforce their unit length, we add constraints, $\boldsymbol{\mathcal{P}}(\mathbf{p}_0) = \mathbf{0}$ and $\boldsymbol{\mathcal{U}}(\mathbf{u}_k) = \mathbf{0}$, to the set of constraints. Because we enforce equality between design parameters, we only need to enforce their unit lengths at $k = 0$.

Our discrete-time optimal design problem is therefore

$$\begin{aligned}
\min_{\mathbf{p}_k, \mathbf{u}_k, \mathbf{v}_k, \mathbf{s}_k} \; & \sum_{k=0}^{n-1} f(\mathbf{p}_k, \mathbf{u}_k, \mathbf{v}_k, \mathbf{s}_k) + F(\mathbf{p}_n, \mathbf{u}_n, \mathbf{s}_n) \qquad (2)\\
s.t. \quad & \mathbf{p}_{k+1} - \mathbf{p}_k = \mathbf{0}, \quad k = 0, \ldots, n-1\\
& \frac{\mathbf{u}_{k+1} - \mathbf{u}_k}{\Delta t} - \mathbf{v}_k = \mathbf{0}, \quad k = 0, \ldots, n-1\\
& \boldsymbol{\mathcal{P}}(\mathbf{p}_0) = \mathbf{0}\\
& \boldsymbol{\mathcal{U}}(\mathbf{u}_k) = \mathbf{0}, \quad k = 0, \ldots, n\\
& \boldsymbol{C}(\mathbf{p}_k, \mathbf{u}_k, \mathbf{s}_k) = \mathbf{0}, \quad k = 0, \ldots, n.
\end{aligned}$$

6 SOLVING FOR A ROBOT'S OPTIMAL DESIGN

Our optimal design problem is difficult to solve: It has a design and control parameter set per time step, and the constraints $\boldsymbol{\mathcal{P}}$, $\boldsymbol{\mathcal{U}}$, and $\boldsymbol{C}$, as also the intermediate and terminal objectives, are nonlinear.

A first solution strategy that comes to mind is sensitivity analysis where we would solve for optimal states for a given set of design and control parameters in the inner loop using Eq. 1, and then for optimal design and control parameters in the outer loop, with a first-order optimality constraint on the inner-loop optimization. This is, however, not an option, because we can easily choose a set of design and control parameters for which there is no inner-loop solution that satisfies all kinematic constraints.

6.1 Formulating a Sequential Quadratic Program

An alternative solution strategy is sequential quadratic programming (SQP) [Nocedal and Wright 2006]. To this end, we introduce Lagrange multipliers $\boldsymbol{\lambda}_k^{\mathcal{D}}$, $\boldsymbol{\lambda}_k^{\mathcal{V}}$, $\boldsymbol{\lambda}_0^{\mathcal{P}}$, $\boldsymbol{\lambda}_k^{\mathcal{U}}$, and $\boldsymbol{\lambda}_k^{\boldsymbol{C}}$ for the five constraint sets and use $\boldsymbol{\lambda}$ to refer to the combined set of multipliers ($\boldsymbol{\mathcal{D}}$: design constraints; $\boldsymbol{\mathcal{V}}$: velocity constraints). We then form the Lagrangian

$$\sum_{k=0}^{n-1} \mathcal{L}^k(\mathbf{p}_k, \mathbf{u}_k, \mathbf{v}_k, \mathbf{s}_k, \boldsymbol{\lambda}) + \mathcal{L}^n(\mathbf{p}_n, \mathbf{u}_n, \mathbf{s}_n, \boldsymbol{\lambda}), \qquad (3)$$

that is partially separable because the design and velocity constraints, that depend on two consecutive time steps, are linear and can therefore be split into two parts.

To perform line search, we need to compute search directions

$$\mathbf{d}_k = \begin{bmatrix} \Delta\mathbf{p}_k \\ \Delta\mathbf{u}_k \\ \Delta\mathbf{v}_k \\ \Delta\mathbf{s}_k \end{bmatrix} \text{ for } k = 0, \ldots, n-1 \text{ and } \mathbf{d}_k = \begin{bmatrix} \Delta\mathbf{p}_k \\ \Delta\mathbf{u}_k \\ \Delta\mathbf{s}_k \end{bmatrix} \text{ for } k = n.$$

by either applying Newton to the Karush–Kuhn–Tucker conditions, or by solving the equivalent quadratic program (QP)

$$\begin{aligned}
\min_{\mathbf{d}_k} \; & \sum_{k=0}^{n} \nabla\mathcal{L}^k \mathbf{d}_k + \frac{1}{2}\mathbf{d}_k^T \nabla^2 \mathcal{L}^k \mathbf{d}_k \qquad (4)\\
s.t. \; & \mathbf{p}_{k+1} - \mathbf{p}_k + \Delta\mathbf{p}_{k+1} - \Delta\mathbf{p}_k = \mathbf{0}, \quad k = 0, \ldots, n-1\\
& \frac{\mathbf{u}_{k+1} - \mathbf{u}_k}{\Delta t} - \mathbf{v}_k + \frac{\Delta\mathbf{u}_{k+1} - \Delta\mathbf{u}_k}{\Delta t} - \Delta\mathbf{v}_k = \mathbf{0}, \quad k = 0, \ldots, n-1\\
& \boldsymbol{\mathcal{P}}^0 + \boldsymbol{\mathcal{P}}_{\mathbf{p}}^0 \Delta\mathbf{p}_0 = \mathbf{0}\\
& \boldsymbol{\mathcal{U}}^k + \boldsymbol{\mathcal{U}}_{\mathbf{u}}^k \Delta\mathbf{u}_k = \mathbf{0}, \quad k = 0, \ldots, n\\
& \boldsymbol{C}^k + \boldsymbol{C}_{\mathbf{p}}^k \Delta\mathbf{p}_k + \boldsymbol{C}_{\mathbf{u}}^k \Delta\mathbf{u}_k + \boldsymbol{C}_{\mathbf{s}}^k \Delta\mathbf{s}_k = \mathbf{0}, \quad k = 0, \ldots, n,
\end{aligned}$$

where we omit arguments for the last three sets of constraints, adding the time step as superscript instead. $\boldsymbol{\mathcal{P}}_{\mathbf{p}}$, $\boldsymbol{\mathcal{U}}_{\mathbf{u}}$, $\boldsymbol{C}_{\mathbf{p}}$, $\boldsymbol{C}_{\mathbf{u}}$, and $\boldsymbol{C}_{\mathbf{s}}$ are constraint Jacobians with respect to design, control, and state variables.

To iteratively find optimal values for these variables, we perform line search with the $L1$ merit function to identify a good step length α [Nocedal and Wright 2006], and update the currently best

estimates

$$\begin{bmatrix}\mathbf{p}_k\\ \mathbf{u}_k\\ \mathbf{v}_k\\ \mathbf{s}_k\end{bmatrix} := \begin{bmatrix}\mathbf{p}_k\\ \mathbf{u}_k\\ \mathbf{v}_k\\ \mathbf{s}_k\end{bmatrix} + \alpha \mathbf{d}_k \quad \text{and} \quad \begin{bmatrix}\mathbf{p}_n\\ \mathbf{u}_n\\ \mathbf{s}_n\end{bmatrix} := \begin{bmatrix}\mathbf{p}_n\\ \mathbf{u}_n\\ \mathbf{s}_n\end{bmatrix} + \alpha \mathbf{d}_n.$$

We also need to update Lagrange multipliers. To do so, we compute an increment $\Delta\boldsymbol{\lambda}$, multiply it with the step length, and use it to update the current best estimate $\boldsymbol{\lambda}$ as explained towards the end of the section.

6.2 Fast Solution with Dynamic Programming

A common approach to compute the search directions for variables and multipliers is to solve the QP by applying a direct sparse linear solver to the equivalent system of linear equations. For large problems, this strategy is limited by its computational cost and the memory that is necessary to assemble the system matrix.

Iterative solvers can circumvent the memory bottleneck by only requiring access to a matrix-vector product operator, and can often be parallelized. However, a careful tuning of tolerances and solver parameters is generally needed. Moreover, QP solvers require the problem to satisfy certain properties, for example positive definiteness of the unconstrained Hessian, which may not hold at a distance from the optimum.

Crucially, an efficient solution strategy must exploit the recursive structure of the problem: the Hessian of the Lagrangian is a banded matrix, more specifically a tridiagonal block matrix because constraints depend on two consecutive time steps only; and the blocks themselves are sparse.

An alternative strategy enabled by this recursive structure is the use of dynamic programming. This strategy is less restrictive when it comes to properties, and provides a direct solution strategy instead of an iterative one, without requiring explicit assembly of the system matrix. Our experiments confirm that this strategy outperforms a sparse solution strategy on the full system in terms of robustness and speed (Sec. 8).

Standard QP and Dynamic Programming. To apply dynamic programming, we first need to bring the above QP into standard form for a *linear* discrete-time optimal control problem [Gros and Diehl 2022]

$$\min_{\tilde{\mathbf{s}}_k, \tilde{\mathbf{u}}_k} \sum_{k=0}^{n-1} \begin{bmatrix}\tilde{\mathbf{s}}_k\\ \tilde{\mathbf{u}}_k\end{bmatrix}^T \begin{bmatrix}\tilde{\mathbf{Q}}_k & \tilde{\mathbf{S}}_k^T\\ \tilde{\mathbf{S}}_k & \tilde{\mathbf{R}}_k\end{bmatrix} \begin{bmatrix}\tilde{\mathbf{s}}_k\\ \tilde{\mathbf{u}}_k\end{bmatrix} + \tilde{\mathbf{s}}_n^T \tilde{\mathbf{P}}_n \tilde{\mathbf{s}}_n \tag{5}$$
$$\text{s.t. } \tilde{\mathbf{s}}_{k+1} = \tilde{\mathbf{A}}_k \tilde{\mathbf{s}}_k + \tilde{\mathbf{B}}_k \tilde{\mathbf{u}}_k, \quad k = 0, \dots, n-1.$$

In this standard form, the "state" and "control" variables are $\tilde{\mathbf{s}}_k$ and $\tilde{\mathbf{u}}_k$. The QP can then be solved with dynamic programming as summarized in Alg. 2.

In the step-by-step derivation that follows, we reduce our QP to this standard form, defining the matrices in the above standard equations.

Definition of State and Control Variables. The linearized design and control constraints in Eq. 4 depend on two consecutive time steps and can easily be brought into standard form. Taking a closer look, we realize that the design constraints depend on $\Delta\mathbf{p}_k$ and $\Delta\mathbf{p}_{k+1}$.

Algorithm 2. Solving a standard QP for a linear discrete-time optimal control problem with dynamic programming. The initial state (or conditions) are assumed to be known.

0. Set $\tilde{\mathbf{s}}_0$ to a constant value.

1. Evaluate $\tilde{\mathbf{P}}_n$.
2. Solve for $\tilde{\mathbf{P}}_k$ backward in time, $k = n-1, \dots, 0$:
$\tilde{\mathbf{P}}_k := \tilde{\mathbf{Q}}_k + \tilde{\mathbf{A}}_k^T \tilde{\mathbf{P}}_{k+1} \tilde{\mathbf{A}}_k$
$\quad - \left(\tilde{\mathbf{S}}_k^T + \tilde{\mathbf{A}}_k^T \tilde{\mathbf{P}}_{k+1} \tilde{\mathbf{B}}_k\right) \left(\tilde{\mathbf{R}}_k + \tilde{\mathbf{B}}_k^T \tilde{\mathbf{P}}_{k+1} \tilde{\mathbf{B}}_k\right)^{-1} \left(\tilde{\mathbf{S}}_k + \tilde{\mathbf{B}}_k^T \tilde{\mathbf{P}}_{k+1} \tilde{\mathbf{A}}_k\right)$
3. Solve for $\tilde{\mathbf{u}}_k$ and $\tilde{\mathbf{s}}_{k+1}$ forward in time, $k = 0, \dots, n-1$:
$\tilde{\mathbf{u}}_k(\tilde{\mathbf{s}}_k) := -\left(\tilde{\mathbf{R}}_k + \tilde{\mathbf{B}}_k^T \tilde{\mathbf{P}}_{k+1} \tilde{\mathbf{B}}_k\right)^{-1} \left(\tilde{\mathbf{S}}_k + \tilde{\mathbf{B}}_k^T \tilde{\mathbf{P}}_{k+1} \tilde{\mathbf{A}}_k\right) \tilde{\mathbf{s}}_k$
$\tilde{\mathbf{s}}_{k+1} = \tilde{\mathbf{A}}_k \tilde{\mathbf{s}}_k + \tilde{\mathbf{B}}_k \tilde{\mathbf{u}}_k(\tilde{\mathbf{s}}_k)$

Analogously, we observe that the velocity constraints depend on the control parameters at k and $k+1$, but only on velocity variables at k. We conclude that the design and control variables must be state variables in the standard form, and the velocity variables take on the role of control variables

$$\tilde{\mathbf{s}}_k := \begin{bmatrix}1\\ \Delta\mathbf{p}_k\\ \Delta\mathbf{u}_k\end{bmatrix} \quad \text{and} \quad \tilde{\mathbf{u}}_k := \Delta\mathbf{v}_k. \tag{6}$$

We added a leading 1 in the definition of states. This trick allows us to combine the gradient and Hessian of the Lagrangian at k into a single quadratic form as required.

Removing Kinematic Constraints. Note that we omit the state variables $\Delta\mathbf{s}_k$ in the above definition of $\tilde{\mathbf{s}}_k$ and $\tilde{\mathbf{u}}_k$. They only appear in the linearized kinematic constraints that uniquely determine their values for a given $\Delta\mathbf{p}_k$ and $\Delta\mathbf{u}_k$

$$\Delta\mathbf{s}_k = \mathbf{c}_k + \mathbf{P}_k \Delta\mathbf{p}_k + \mathbf{U}_k \Delta\mathbf{u}_k \qquad \begin{aligned}\mathbf{c}_k &= -(\boldsymbol{C}_\mathbf{s}^k)^{-1} \boldsymbol{C}^k\\ \mathbf{P}_k &= -(\boldsymbol{C}_\mathbf{s}^k)^{-1} \boldsymbol{C}_\mathbf{p}^k\\ \mathbf{U}_k &= -(\boldsymbol{C}_\mathbf{s}^k)^{-1} \boldsymbol{C}_\mathbf{u}^k,\end{aligned} \tag{7}$$

where we assume that redundant constraints were removed from $\boldsymbol{C}$ and the Jacobian $\boldsymbol{C}_\mathbf{s}^k$ to be a square matrix. By substituting Eq. 7 for $\Delta\mathbf{s}_k$ in the individual Lagrangian terms $\mathcal{L}^k$, we can remove these variables and the kinematic constraints.

Removing Quaternion Unit Length Constraints. It remains to discuss the unit length constraints for design and control variables. Let's first consider the unit length constraints for the design parameters at $k = 0$. By forming a singular value decomposition of the Jacobian $\boldsymbol{\mathcal{P}}_\mathbf{p}^0$, we can represent the solutions that satisfy the constraint with a reduced set of variables $\Delta\bar{\mathbf{p}}_0$

$$\Delta\mathbf{p}_0 = \mathbf{y}_0^{\boldsymbol{\mathcal{P}}} + \mathbf{Z}_0^{\boldsymbol{\mathcal{P}}} \Delta\bar{\mathbf{p}}_0 \text{ with spec. sol. } \mathbf{y}_0^{\boldsymbol{\mathcal{P}}} = -\mathbf{Y}_0^{\boldsymbol{\mathcal{P}}} (\boldsymbol{\mathcal{P}}_\mathbf{p}^0 \mathbf{Y}_0^{\boldsymbol{\mathcal{P}}})^{-1} \boldsymbol{\mathcal{P}}^0, \tag{8}$$

where $[\mathbf{Y}_0^{\boldsymbol{\mathcal{P}}} | \mathbf{Z}_0^{\boldsymbol{\mathcal{P}}}]$ are the right singular vectors, with $\mathbf{Y}_0^{\boldsymbol{\mathcal{P}}}$ corresponding to non-zero singular values. For the control parameters at $k = 0$, we can proceed analogously. The reduced variables are incorporated in the standard algorithm (Alg. 2, Eq. 5) by adding the equation $\tilde{\mathbf{s}}_0 := \tilde{\mathbf{A}}_{-1} \tilde{\mathbf{s}}_{-1}$, with

$$\tilde{\mathbf{A}}_{-1} := \begin{bmatrix}1 & \mathbf{0} & \mathbf{0}\\ \mathbf{y}_0^{\boldsymbol{\mathcal{P}}} & \mathbf{Z}_0^{\boldsymbol{\mathcal{P}}} & \mathbf{0}\\ \mathbf{y}_0^{\boldsymbol{\mathcal{U}}} & \mathbf{0} & \mathbf{Z}_0^{\boldsymbol{\mathcal{U}}}\end{bmatrix} \quad \text{and} \quad \tilde{\mathbf{s}}_{-1} := \begin{bmatrix}1\\ \Delta\bar{\mathbf{p}}_0\\ \Delta\bar{\mathbf{u}}_0\end{bmatrix}, \tag{9}$$

for $k = -1$ to the set of constraints. $\tilde{\mathbf{A}}_{-1}$ represents a mapping from reduced to full space. Note that $\tilde{\mathbf{s}}_{-1}$ represents design and control variables at $k = 0$ in reduced space, while $\tilde{\mathbf{s}}_0$ represents them in full space.

The remaining unit quaternion constraints for $k = 1, \ldots, n$ are less straightforward to remove. To do so, we need to look at the velocity and unit quaternion constraints for control parameters together, rearranging terms to align time steps

$$\Delta\mathbf{u}_{k+1} = \Delta\mathbf{u}_k + \Delta t \Delta\mathbf{v}_k - (\mathbf{u}_{k+1} - \mathbf{u}_k - \Delta t\mathbf{v}_k)$$
$$\boldsymbol{\mathcal{U}}_{\mathbf{u}}^{k+1}\,\Delta\mathbf{u}_{k+1} = -\boldsymbol{\mathcal{U}}^{k+1}, \qquad k = 0, \ldots, n-1.$$

A projection of control parameters onto a reduced set, $\Delta\bar{\mathbf{u}}_{k+1}$, as above for $k = 0$ is not an option, because the matrix $\mathbf{Z}_{k+1}^{\boldsymbol{\mathcal{U}}}$ for a singular value decomposition of the Jacobian $\boldsymbol{\mathcal{U}}_{\mathbf{u}}^{k+1}$ would appear in front of a reduced set of control parameters $\Delta\bar{\mathbf{u}}_{k+1}$, and cannot be brought to the other side because it is not a square matrix and hence not invertible. This violates the recursion requirement for dynamic programming.

An alternative is to work with reduced velocity variables. To this end, we substitute the velocity equations for $\mathbf{u}_{k+1}$ in the second equation

$$\boldsymbol{\mathcal{U}}_{\mathbf{u}}^{k+1}\,\Delta\mathbf{v}_k = -\frac{1}{\Delta t}\boldsymbol{\mathcal{U}}_{\mathbf{u}}^{k+1}\Delta\mathbf{u}_k - \boldsymbol{\mathcal{V}}^k$$
$$\boldsymbol{\mathcal{V}}^k := \frac{1}{\Delta t}\left(\boldsymbol{\mathcal{U}}^{k+1} + \boldsymbol{\mathcal{U}}_{\mathbf{u}}^{k+1}\,(\mathbf{u}_k - \mathbf{u}_{k+1} + \Delta t\mathbf{v}_k)\right),$$

then represent the solutions with a reduced set $\Delta\bar{\mathbf{v}}_k$

$$\Delta\mathbf{v}_k = \mathbf{y}_{k+1}^{\boldsymbol{\mathcal{U}}} + \mathbf{X}_{k+1}^{\boldsymbol{\mathcal{U}}}\Delta\mathbf{u}_k + \mathbf{Z}_{k+1}^{\boldsymbol{\mathcal{U}}}\Delta\bar{\mathbf{v}}_k$$
$$\text{with } \mathbf{X}_{k+1}^{\boldsymbol{\mathcal{U}}} = -\frac{1}{\Delta t}\mathbf{Y}_{k+1}^{\boldsymbol{\mathcal{U}}}(\boldsymbol{\mathcal{U}}_{\mathbf{u}}^{k+1}\mathbf{Y}_{k+1}^{\boldsymbol{\mathcal{U}}})^{-1}\boldsymbol{\mathcal{U}}_{\mathbf{u}}^{k+1}$$
$$\text{and } \mathbf{y}_{k+1}^{\boldsymbol{\mathcal{U}}} = -\mathbf{Y}_{k+1}^{\boldsymbol{\mathcal{U}}}(\boldsymbol{\mathcal{U}}_{\mathbf{u}}^{k+1}\mathbf{Y}_{k+1}^{\boldsymbol{\mathcal{U}}})^{-1}\boldsymbol{\mathcal{V}}^k.$$

The subspace velocity equations then become

$$\Delta\mathbf{u}_{k+1} = \left(\mathbf{I} + \Delta t\,\mathbf{X}_{k+1}^{\boldsymbol{\mathcal{U}}}\right)\Delta\mathbf{u}_k + \left(\Delta t\,\mathbf{Z}_{k+1}^{\boldsymbol{\mathcal{U}}}\right)\Delta\bar{\mathbf{v}}_k \qquad (10)$$
$$- \Delta t\mathbf{y}_{k+1}^{\boldsymbol{\mathcal{U}}} + (\mathbf{u}_{k+1} - \mathbf{u}_k - \Delta t\mathbf{v}_k).$$

Note that we use reduced velocity variables $\Delta\bar{\mathbf{v}}_k$ instead of $\Delta\mathbf{v}_k$ in our control variables $\tilde{\mathbf{u}}_k$ for all time steps k.

Final Dynamic Programming Algorithm. The above derivations result in a final algorithm and uniquely defined matrices $\tilde{\mathbf{Q}}_k$, $\tilde{\mathbf{S}}_k$, $\tilde{\mathbf{R}}_k$, $\tilde{\mathbf{P}}_n$, $\tilde{\mathbf{A}}_k$, and $\tilde{\mathbf{B}}_k$ as summarized in Alg. 3 and Tab. 8 for the reader's convenience. In step 4 in Alg. 3, we solve for the state $\tilde{\mathbf{s}}_{-1}$ by minimizing the objective $\tilde{\mathbf{s}}^T\tilde{\mathbf{P}}_{-1}\tilde{\mathbf{s}}$. This is necessary because the standard algorithm (Alg. 2) assumes a given initial state $\tilde{\mathbf{s}}_0$, whereas our design and control parameters at $k = 0$ (and therefore, their reduced versions at $k = -1$) are *unknowns*. Taking into account the leading 1 in our state representation, the minimization reduces to a linear system of equations

$$\min_{\tilde{\mathbf{s}}} \begin{bmatrix}1\\ \Delta\bar{\mathbf{p}}\\ \Delta\bar{\mathbf{u}}\end{bmatrix}^T \tilde{\mathbf{P}}_{-1}\begin{bmatrix}1\\ \Delta\bar{\mathbf{p}}\\ \Delta\bar{\mathbf{u}}\end{bmatrix} \text{ with } \tilde{\mathbf{P}}_{-1} := \begin{bmatrix}0 & \tilde{\mathbf{p}}_{\mathbf{p}}^T & \tilde{\mathbf{p}}_{\mathbf{u}}^T\\ \tilde{\mathbf{p}}_{\mathbf{p}} & \tilde{\mathbf{P}}_{\mathbf{pp}} & \tilde{\mathbf{P}}_{\mathbf{pu}}\\ \tilde{\mathbf{p}}_{\mathbf{u}} & \tilde{\mathbf{P}}_{\mathbf{pu}}^T & \tilde{\mathbf{P}}_{\mathbf{uu}}\end{bmatrix} \qquad (11)$$

$$\Leftrightarrow \qquad \tilde{\mathbf{s}}_{-1} := \begin{bmatrix}\tilde{\mathbf{P}}_{\mathbf{pp}} & \tilde{\mathbf{P}}_{\mathbf{pu}}\\ \tilde{\mathbf{P}}_{\mathbf{pu}}^T & \tilde{\mathbf{P}}_{\mathbf{uu}}\end{bmatrix}^{-1}\begin{bmatrix}\tilde{\mathbf{p}}_{\mathbf{p}}\\ \tilde{\mathbf{p}}_{\mathbf{u}}\end{bmatrix}.$$

Algorithm 3. Final dynamic programming algorithm.

1. Evaluate $\tilde{\mathbf{P}}_n$.
2. Solve for $\tilde{\mathbf{P}}_k$ backward in time, $k = n-1, \ldots, 0$:
   $\tilde{\mathbf{P}}_k := \tilde{\mathbf{Q}}_k + \tilde{\mathbf{A}}_k^T\tilde{\mathbf{P}}_{k+1}\tilde{\mathbf{A}}_k$
   $\quad - \left(\tilde{\mathbf{S}}_k^T + \tilde{\mathbf{A}}_k^T\tilde{\mathbf{P}}_{k+1}\tilde{\mathbf{B}}_k\right)\left(\tilde{\mathbf{R}}_k + \tilde{\mathbf{B}}_k^T\tilde{\mathbf{P}}_{k+1}\tilde{\mathbf{B}}_k\right)^{-1}\left(\tilde{\mathbf{S}}_k + \tilde{\mathbf{B}}_k^T\tilde{\mathbf{P}}_{k+1}\tilde{\mathbf{A}}_k\right)$
3. Evaluate $\tilde{\mathbf{P}}_{-1} := \tilde{\mathbf{A}}_{-1}^T\tilde{\mathbf{P}}_0\tilde{\mathbf{A}}_{-1}$.
4. Evaluate $\tilde{\mathbf{s}}_{-1} := \arg\min_{\tilde{\mathbf{s}}}\,\tilde{\mathbf{s}}^T\tilde{\mathbf{P}}_{-1}\tilde{\mathbf{s}}$.
5. Evaluate $\tilde{\mathbf{s}}_0 := \tilde{\mathbf{A}}_{-1}\tilde{\mathbf{s}}_{-1}$
3. Solve for $\tilde{\mathbf{s}}_k$ and $\tilde{\mathbf{u}}_k$ forward in time, $k = 0, \ldots, n-1$:
   $\tilde{\mathbf{u}}_k(\tilde{\mathbf{s}}_k) := -\left(\tilde{\mathbf{R}}_k + \tilde{\mathbf{B}}_k^T\tilde{\mathbf{P}}_{k+1}\tilde{\mathbf{B}}_k\right)^{-1}\left(\tilde{\mathbf{S}}_k + \tilde{\mathbf{B}}_k^T\tilde{\mathbf{P}}_{k+1}\tilde{\mathbf{A}}_k\right)\tilde{\mathbf{s}}_k$
   $\tilde{\mathbf{s}}_{k+1} = \tilde{\mathbf{A}}_k\tilde{\mathbf{s}}_k + \tilde{\mathbf{B}}_k\tilde{\mathbf{u}}_k(\tilde{\mathbf{s}}_k)$

Lagrange Multipliers. The output of Alg. 3 are search directions, $\mathbf{d}_k$, for optimization variables, identical to the ones we would obtained by solving the equivalent QP. To compute a corresponding search direction, $\Delta\boldsymbol{\lambda}$, for the Lagrange multipliers, we expand the first equation of the Karush–Kuhn–Tucker system that is equivalent to the QP in Eq. 4, solving for the individual multiplier increments by utilizing the recursive structure as summarized in Alg. 4. We then perform the update of the current best multiplier estimates

$$\begin{bmatrix}\boldsymbol{\lambda}_k^{\mathcal{D}}\\ \boldsymbol{\lambda}_k^{\mathcal{V}}\\ \boldsymbol{\lambda}_k^{\mathcal{U}}\\ \boldsymbol{\lambda}_k^{C}\end{bmatrix} := \begin{bmatrix}\boldsymbol{\lambda}_k^{\mathcal{D}}\\ \boldsymbol{\lambda}_k^{\mathcal{V}}\\ \boldsymbol{\lambda}_k^{\mathcal{U}}\\ \boldsymbol{\lambda}_k^{C}\end{bmatrix} + \alpha\begin{bmatrix}\Delta\boldsymbol{\lambda}_k^{\mathcal{D}}\\ \Delta\boldsymbol{\lambda}_k^{\mathcal{V}}\\ \Delta\boldsymbol{\lambda}_k^{\mathcal{U}}\\ \Delta\boldsymbol{\lambda}_k^{C}\end{bmatrix} \qquad \text{for } k = 1, \ldots, n-1,$$

and

$$\begin{bmatrix}\boldsymbol{\lambda}_0^{\mathcal{D}}\\ \boldsymbol{\lambda}_0^{\mathcal{V}}\\ \boldsymbol{\lambda}_0^{\mathcal{P}}\\ \boldsymbol{\lambda}_0^{\mathcal{U}}\\ \boldsymbol{\lambda}_0^{C}\end{bmatrix} := \begin{bmatrix}\boldsymbol{\lambda}_0^{\mathcal{D}}\\ \boldsymbol{\lambda}_0^{\mathcal{V}}\\ \boldsymbol{\lambda}_0^{\mathcal{P}}\\ \boldsymbol{\lambda}_0^{\mathcal{U}}\\ \boldsymbol{\lambda}_0^{C}\end{bmatrix} + \alpha\begin{bmatrix}\Delta\boldsymbol{\lambda}_0^{\mathcal{D}}\\ \Delta\boldsymbol{\lambda}_0^{\mathcal{V}}\\ \Delta\boldsymbol{\lambda}_0^{\mathcal{P}}\\ \Delta\boldsymbol{\lambda}_0^{\mathcal{U}}\\ \Delta\boldsymbol{\lambda}_0^{C}\end{bmatrix}, \quad \begin{bmatrix}\boldsymbol{\lambda}_n^{\mathcal{U}}\\ \boldsymbol{\lambda}_n^{C}\end{bmatrix} := \begin{bmatrix}\boldsymbol{\lambda}_n^{\mathcal{U}}\\ \boldsymbol{\lambda}_n^{C}\end{bmatrix} + \alpha\begin{bmatrix}\Delta\boldsymbol{\lambda}_n^{\mathcal{U}}\\ \Delta\boldsymbol{\lambda}_n^{C}\end{bmatrix}$$

for the first and last time steps, with the step length α.

Algorithm 4. Solving for Lagrange multiplier increments.

1. Compute $\mathbf{h}_{\mathbf{p}}^k, \mathbf{h}_{\mathbf{u}}^k, \mathbf{h}_{\mathbf{s}}^k, k = 0, \ldots, n$ and $\mathbf{h}_{\mathbf{v}}^k, k = 0, \ldots, n-1$:
   $\mathbf{h}_{\mathbf{p}}^k := \mathcal{L}_{\mathbf{pp}}^k\Delta\mathbf{p}_k + \mathcal{L}_{\mathbf{pu}}^k\Delta\mathbf{u}_k + \mathcal{L}_{\mathbf{pv}}^k\Delta\mathbf{v}_k + \mathcal{L}_{\mathbf{ps}}^k\Delta\mathbf{s}_k + \mathcal{L}_{\mathbf{p}}^k, \quad k < n$
   $\mathbf{h}_{\mathbf{p}}^n := \mathcal{L}_{\mathbf{pp}}^n\Delta\mathbf{p}_n + \mathcal{L}_{\mathbf{pu}}^n\Delta\mathbf{u}_n + \mathcal{L}_{\mathbf{ps}}^n\Delta\mathbf{s}_n + \mathcal{L}_{\mathbf{p}}^n$
   $\mathbf{h}_{\mathbf{u}}^k := \mathcal{L}_{\mathbf{up}}^k\Delta\mathbf{p}_k + \mathcal{L}_{\mathbf{uu}}^k\Delta\mathbf{u}_k + \mathcal{L}_{\mathbf{uv}}^k\Delta\mathbf{v}_k + \mathcal{L}_{\mathbf{us}}^k\Delta\mathbf{s}_k + \mathcal{L}_{\mathbf{u}}^k, \quad k < n$
   $\mathbf{h}_{\mathbf{u}}^n := \mathcal{L}_{\mathbf{up}}^n\Delta\mathbf{p}_n + \mathcal{L}_{\mathbf{uu}}^n\Delta\mathbf{u}_n + \mathcal{L}_{\mathbf{us}}^n\Delta\mathbf{s}_n + \mathcal{L}_{\mathbf{u}}^n$
   $\mathbf{h}_{\mathbf{v}}^k := \mathcal{L}_{\mathbf{vp}}^k\Delta\mathbf{p}_k + \mathcal{L}_{\mathbf{vu}}^k\Delta\mathbf{u}_k + \mathcal{L}_{\mathbf{vv}}^k\Delta\mathbf{v}_k + \mathcal{L}_{\mathbf{vs}}^k\Delta\mathbf{s}_k + \mathcal{L}_{\mathbf{v}}^k, \quad k < n$
   $\mathbf{h}_{\mathbf{s}}^k := \mathcal{L}_{\mathbf{sp}}^k\Delta\mathbf{p}_k + \mathcal{L}_{\mathbf{su}}^k\Delta\mathbf{u}_k + \mathcal{L}_{\mathbf{sv}}^k\Delta\mathbf{v}_k + \mathcal{L}_{\mathbf{ss}}^k\Delta\mathbf{s}_k + \mathcal{L}_{\mathbf{s}}^k, \quad k < n$
   $\mathbf{h}_{\mathbf{s}}^n := \mathcal{L}_{\mathbf{sp}}^n\Delta\mathbf{p}_n + \mathcal{L}_{\mathbf{su}}^n\Delta\mathbf{u}_n + \mathcal{L}_{\mathbf{ss}}^n\Delta\mathbf{s}_n + \mathcal{L}_{\mathbf{s}}^n$
2. Compute $\Delta\boldsymbol{\lambda}_k^{\mathcal{V}} = -\frac{1}{\Delta t}\,\mathbf{h}_{\mathbf{v}}^k$, $k = 0, \ldots, n-1$.
3. Compute $\Delta\boldsymbol{\lambda}_k^{C} = \left(\boldsymbol{C}_{\mathbf{s}}^k\right)^{-T}\mathbf{h}_{\mathbf{s}}^k$, $k = 0, \ldots, n$.
4. Solve for $\Delta\boldsymbol{\lambda}_k^{\mathcal{D}}$ backward in time, $k = n-1, \ldots, 0$:
   $\Delta\boldsymbol{\lambda}_{n-1}^{\mathcal{D}} = \mathbf{h}_{\mathbf{p}}^n - \left(\boldsymbol{C}_{\mathbf{p}}^n\right)^T\Delta\boldsymbol{\lambda}_n^{C}$
   $\Delta\boldsymbol{\lambda}_k^{\mathcal{D}} = \mathbf{h}_{\mathbf{p}}^{k+1} - \left(\boldsymbol{C}_{\mathbf{p}}^{k+1}\right)^T\Delta\boldsymbol{\lambda}_{k+1}^{C} + \Delta\boldsymbol{\lambda}_{k+1}^{\mathcal{D}}$
5. Compute $\Delta\boldsymbol{\lambda}_k^{\mathcal{U}}$, $k = 0, \ldots, n$:
   $\Delta\boldsymbol{\lambda}_k^{\mathcal{U}} = \left(\boldsymbol{\mathcal{U}}_{\mathbf{u}}^k\left(\boldsymbol{\mathcal{U}}_{\mathbf{u}}^k\right)^T\right)^{-1}\boldsymbol{\mathcal{U}}_{\mathbf{u}}^k\left(\mathbf{h}_{\mathbf{u}}^k - \left(\boldsymbol{C}_{\mathbf{u}}^k\right)^T\Delta\boldsymbol{\lambda}_k^{C} + \Delta\boldsymbol{\lambda}_k^{\mathcal{V}} - \Delta\boldsymbol{\lambda}_{k-1}^{\mathcal{V}}\right)$
6. Solve for $\Delta\boldsymbol{\lambda}_0^{\mathcal{P}} = \left(\boldsymbol{\mathcal{P}}_{\mathbf{p}}^0\left(\boldsymbol{\mathcal{P}}_{\mathbf{p}}^0\right)^T\right)^{-1}\boldsymbol{\mathcal{P}}_{\mathbf{p}}^0\left(\mathbf{h}_{\mathbf{p}}^0 - \left(\boldsymbol{C}_{\mathbf{p}}^0\right)^T\Delta\boldsymbol{\lambda}_0^{C} + \Delta\boldsymbol{\lambda}_0^{\mathcal{D}}\right)$

6.3 Removing Redundancy in Kinematic Constraints

As mentioned earlier, we assume a non-redundant set of kinematic constraints, as required for the Jacobian $\boldsymbol{C}_\mathbf{s}$ to be invertible in Eq. 7. Furthermore, sets of constraints with redundancy can yield an infeasible QP when linearized in Eq. 4. Unfortunately, robots with kinematic loops often suffer from such redundancy.

For example, linkages are used to place actuators where there is space, while they provide the source of motion where it is needed. Linkages introduce redundancy. The simplest case to see this is a four bar linkage: because we have four components, we have a total of 28 state variables. With a quaternion unit length constraint per component, we restrict 4 out of the 28 DoFs, leaving 6 DoFs per component unconstrained. The linkage is driven by a revolute actuator (6 constraints), has three revolute joints ($3\times5 = 15$ constraints), and we ground one component (6 constraints). We therefore have a 28 dimensional state and a total of 31 constraints. Even though we work with the minimal number of constraints when we consider the degrees of freedom of individual joints and actuators, we have, in general, more constraints than unknown states for general robotic characters with kinematic loops. Because linkages are only one source of redundancy in $\boldsymbol{C}$, we need a general and automated solution to remove unnecessary constraints.

Our constraint elimination process takes as input a reference state $\mathbf{s}$ of the robot (e.g., its initial pose or first frame of an animation), and automatically selects a non-redundant subset of constraints in $\boldsymbol{C}$ so that this subset contains as many constraints as unknown states in $\mathbf{s}$. Because the behavior in a neighborhood of $\mathbf{s}$ needs to be considered to choose the "right" subset, we rely on the Jacobian $\boldsymbol{C}_\mathbf{s}$.

However, before we compute the Jacobian that we use for analysis, we remove all actuators, replacing them with corresponding passive joints. This is necessary because actuators, for a particular set of control parameters $\mathbf{u}$, hold the robot in the state $\mathbf{s}$. We would therefore not see the "mobility" of the robot in a neighborhood of $\mathbf{s}$ if we analyzed the Jacobian of the actuated system directly. If we analyze the Jacobian corresponding to the passive system, however, we see the mobility of mechanical joints *and* actuators.

Before we analyze the Jacobian of the passive system, we normalize each row. Each of its rows i can be understood as a direction, in the space of states $\mathbf{s}$, along which the kinematic structure is immobile. The mobility of the passive system is only possible along directions orthogonal to every row. The goal is therefore to extract a minimal subset of rows that form an as-orthogonal-as-possible basis of the same initial space. Equivalently, after identifying and removing redundant constraints, directions corresponding to the removed rows must still be spanned by the remaining ones, to prevent the introduction of undesired mobility. This motivates the following selection process:

We first form the singular value decomposition of $\boldsymbol{C}_\mathbf{s}$, and extract the left singular vectors $\mathbf{Z}$ that correspond to zero singular values, such that $\mathbf{Z}^T\boldsymbol{C}_\mathbf{s} = 0$. Each row k of these equations provide a linear combination that evaluates to zero

$$\sum_i z_{ik}(\boldsymbol{C}_\mathbf{s})_i = 0, \tag{12}$$

where $(\boldsymbol{C}_\mathbf{s})_i$ refers to row i of the Jacobian. For any j such that $z_{jk} \neq 0$, we can use equation k in Eq.12 to eliminate constraint j that is already in the span of the other constraints

$$(\boldsymbol{C}_\mathbf{s})_j = -\sum_{i\neq j}\frac{z_{ik}}{z_{jk}}(\boldsymbol{C}_\mathbf{s})_i. \tag{13}$$

To prevent any unwanted mobility, we choose j so that the constraint that is the "least" orthogonal to others, or the one that results in the lowest right-hand-side coefficients in Eq. 12, is removed

$$j = \operatorname*{argmin}_j \min_k \sum_{i\neq j}\frac{z_{ik}^2}{z_{jk}^2}. \tag{14}$$

Note that we normalized the rows of the Jacobian to make the coefficients comparable. After adding j to the set of eliminated constraints, we remove the corresponding equation k, and subtract it from the remaining equations

$$z_{:i} := z_{:i} - \frac{z_{ji}}{z_{jk}}z_{:k}, \tag{15}$$

setting coefficients z_{ji} to zero that correspond to the eliminated constraint j. We iterate the process until we have used all equations from Eq. 12.

The Jacobian of the subset of selected constraints has full row rank, and if we add back the additional constraints that actuators add, we get a square, full rank Jacobian for the actuated system. The only exception is an overactuated robot where we have more actuators than necessary. For overactuated robots, after removing redundancy in the passive Jacobian, we repeat the above process, but using the Jacobian of the actuated system with passive redundancy removed and considering only actuation constraints for elimination.

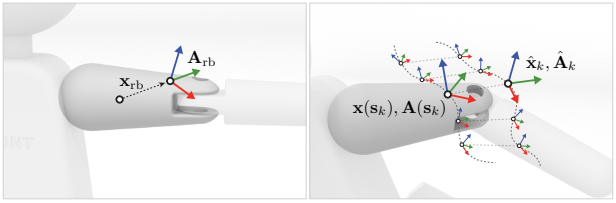

Fig. 4. **Tracking Objectives.** In a local coordinate frame of a rigid body that we want to guide based on a target animation, we define the position $\mathbf{x}_{\mathrm{rb}}$ and/or orientation $\mathbf{A}_{\mathrm{rb}}$ (left). The global motion over time of the position, $\mathbf{x}(\mathbf{s}_k)$, and orientation, $\mathbf{A}(\mathbf{s}_k)$, are then used to define tracking objectives based on the target positions, $\hat{\mathbf{x}}_k$, and orientations, $\hat{\mathbf{A}}_k$ (right).

7 EDITING AND DESIGN WITH OBJECTIVES

When editing the design of an existing robot, we first simulate its kinematic motion and then record trajectories of points of interest. By representing them with spatial cubic Hermite splines, or applying transformations to them, we can then edit the target motion, and therefore the design of the robot. If we design a robot from scratch, a rigged character can serve as a conceptual input, or mocap could serve as a source of motion input.

Independent of the use case, we need to be able to track the difference between the motion of points of interest on the robot and user-provided target motion. To this end, we use tracking objectives similar to the ones described in [Schumacher et al. 2021].

Tracking Objectives. To measure a robot's performance with respect to user-specified targets, we support position and orientation tracking, illustrated in Figure 4. A target trajectory either consists of a target point, $\hat{\mathbf{x}}_k$, or a target orientation, $\hat{\mathbf{R}}_k$, for every time step k, or a combination of the two. We then choose a position, $\mathbf{x}_{\text{rb}}$, and/or orientation, $\mathbf{A}_{\text{rb}}$, in a local coordinate frame of a rigid body whose motion the target trajectory shall guide. During optimization, we transform the local position and orientation to global coordinates using the body's position $\mathbf{c}_k$ and orientation $\mathbf{q}_k$

$$\mathbf{x}(\mathbf{s}_k) = \mathbf{R}(\mathbf{q}_k)\mathbf{x}_{\text{rb}} + \mathbf{c}_k \quad \text{and} \quad \mathbf{A}(\mathbf{s}_k) = \mathbf{R}(\mathbf{q}_k)\mathbf{A}_{\text{rb}},$$

then measure differences with our position and orientation objectives

$$f_{\text{pos}}(\mathbf{s_k}) = \frac{1}{2}\|\mathbf{x}(\mathbf{s}_k) - \hat{\mathbf{x}}\|^2_{\mathbf{W}} \text{ and } f_{\text{ori}}(\mathbf{s_k}) = \frac{1}{2} w_{\text{ori}} \left\|\mathbf{A}(\mathbf{s}_k) - \hat{\mathbf{A}}_k\right\|^2,$$

where we use the weighted norm with $\mathbf{W} = \text{diag}(w_x, w_y, w_z)$ for positions and weigh orientation objectives with w_{ori}. Note that these weights can be set to non-constant values to emphasize preservation of motion either spatially or temporally, or both. For every point of interest, we add position and/or orientation objectives to our intermediate and terminal objectives, f and F.

Kinematic Limits. We support position and velocity limits for actuators, and position limits for reconfigurable joints. We enforce them with a smooth barrier function

$$\beta(x, x_{\max}, \varepsilon) = \begin{cases} -\log\left(\frac{x_{\max} - x}{\varepsilon}\right)^3 & \text{if } x \geq x_{\max} - \varepsilon \\ 0 & \text{otherwise,} \end{cases} \tag{16}$$

that becomes active if a value x is less than an ε from either a user-specified lower or upper limit $x_{\min}$ or $x_{\max}$, resulting in our limits objective

$$f_{\text{lim}}(x) = \beta(-x, -x_{\min}, \varepsilon) + \beta(x, x_{\max}, \varepsilon). \tag{17}$$

For each component x of our control parameters, $\mathbf{u_k}$ and $\mathbf{v_k}$, and our design parameters, $\mathbf{p_k}$, we add a limits objective to our intermediate objective f. For our terminal objective F, we only add position limits.

Regularization. To avoid ill-posed problems, we add regularization terms

$$f^{\boldsymbol{\mathcal{U}}}_{\text{reg}}(\mathbf{u}_k) = \frac{1}{2} w^{\boldsymbol{\mathcal{U}}}_{\text{reg}} \|\mathbf{u}_k - \mathbf{u}^0_k\|^2 \text{ and } f^{\boldsymbol{\mathcal{P}}}_{\text{reg}}(\mathbf{p}_k) = \frac{1}{2} w^{\boldsymbol{\mathcal{P}}}_{\text{reg}} \|\mathbf{p}_k - \mathbf{p}^0\|^2,$$

keeping control parameters close to an initial animation, $\mathbf{u}^0_k$, on the unoptimized design, and design parameters close to their initial values $\mathbf{p}^0$. For some examples, we observe that the nullspace in design parameters can be large, requiring a higher weight for the $w^{\boldsymbol{\mathcal{P}}}_{\text{reg}}$. This can have an effect on the quality of the result. To mitigate its impact, we found that updating $\mathbf{p}^0$ with design parameters from the last iterate in a decreasing frequency, i.e., at iterations 2, 4, 8, 16, etc., is effective, without a noticeable effect on convergence. The regularization terms are added to both, f and F.

8 RESULTS

We demonstrate our method on four different mechanical models, showcasing different applications and workflows that are covered by our approach. We show physical prototypes for two of these demonstrations.

For all demonstrators, we initialize control parameters with a traditional IK solver [Schumacher et al. 2021] and set the Lagrange multipliers to zero. While not safeguarding against convergence to local minima, we observe that this initialization works well, even in cases where the design changes are significant. As an illustration, a comparative study of initial vs. optimized robot performance is provided in Tab. 6.

Kickbot. As a first demonstrator, we consider a small humanoid performing a kicking motion. The input is an animated character rig, with standard spherical "actuators". We seek to create a robot design that is able to track this motion using a reduced set of optimally-placed actuators.

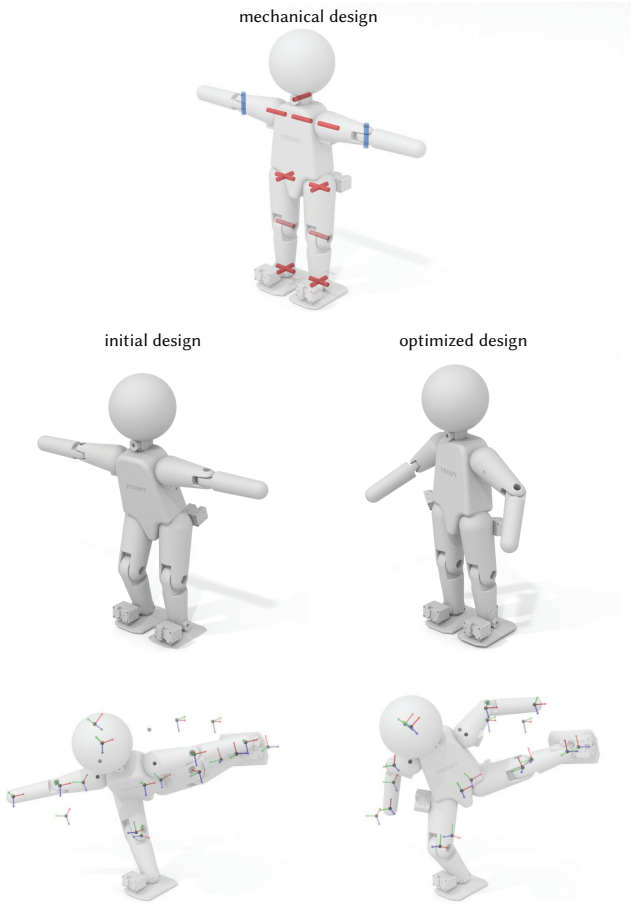

Fig. 5. **Kickbot.** Parameterized mechanical model with 2-DoF ankles, hips, and neck, 1-DoF knees and shoulders, and static arms (top). Design optimization adjusts the parameters of the design (middle) in order to increase the match to the motion targets (bottom). When visualizing the kinematics, we show revolute actuators as red cylinders, and configurable revolute joints in blue. Saturated joints and actuators have been parameterized and can take on any orientation (as described in Fig. 2), while desaturated joints and actuators are not parameterized.

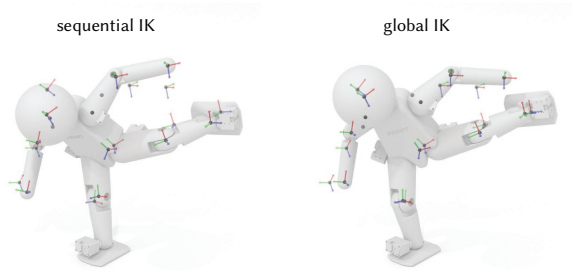

Fig. 6. **Kickbot with Velocity Limits.** When velocity limits are imposed during a sequential IK solve [Schumacher et al. 2021], the resulting motion can lag behind the target animation, and fail to hit targets even when large target weights are used (left). Our global formulation ensures that the tracking error is distributed as well as possible throughout the animation. Important poses can be assigned a large weight, and the optimization will make sure to start moving into the poses early to work around the velocity limits (right).

For the hip, we go from 3 DoFs in the input to 2 revolute actuators in the robot, and for the shoulder, we go from a 3-DoF input to a single revolute actuator. At the elbows, we only place a configurable revolute joint, meaning that the elbows must remain fixed over time but their angle can be optimized. For the initial design, the actuators are naively oriented along one of the world axes. We parameterized the orientation of all revolute actuators, with the exception of the knees, by introducing configurable spherical joints on either side of each actuator as illustrated in Fig. 2.

As seen in Fig. 5, and also the supporting video, the initial design tracks the desired motion poorly. The optimized result is able to track the desired motion well, despite the limited number of actuators.

A requirement for robot motions is that they should obey the velocity limits of the actuators. To this end, we add velocity limits to our model and rerun our optimization, only optimizing control parameters. To ensure that the kick motion does not become washed-out, we increase the tracking weight on the kicking foot at the apex of the motion. In contrast to sequential IK [Schumacher et al. 2021], our approach considers jointly the full motion sequence, so it is able to optimally distribute the error along the full sequence, and can therefore hit the kicking pose as desired, shown in Fig. 6.

Legs. Another application of our pipeline is the optimization of reconfigurable robot designs, where components can be reoriented after the robot has been built. We consider a 6-DoF robot leg mechanism, with multiple nested kinematic loops. The mechanism is overactuated (8 actuators) and also over-constrained (6 redundant constraints). The robot has been designed so that one of the hip joint axes is reconfigurable, as shown in Fig. 7. The height of the physical robot is 1150 mm and it weighs 14.8 kg without base plate.

We wish to track an asymmetric twisting motion, which lies beyond the workspace of the initial robot design due to its position limits. Our optimization is able to find a configuration of the reconfigurable joints which allows for the motion to be tracked, as shown in Fig. 7, and also the supporting video. We demonstrate that the resulting motion can be successfully executed on the physical robot, after reconfiguration, and we can also execute other dancing motions on this optimized design.

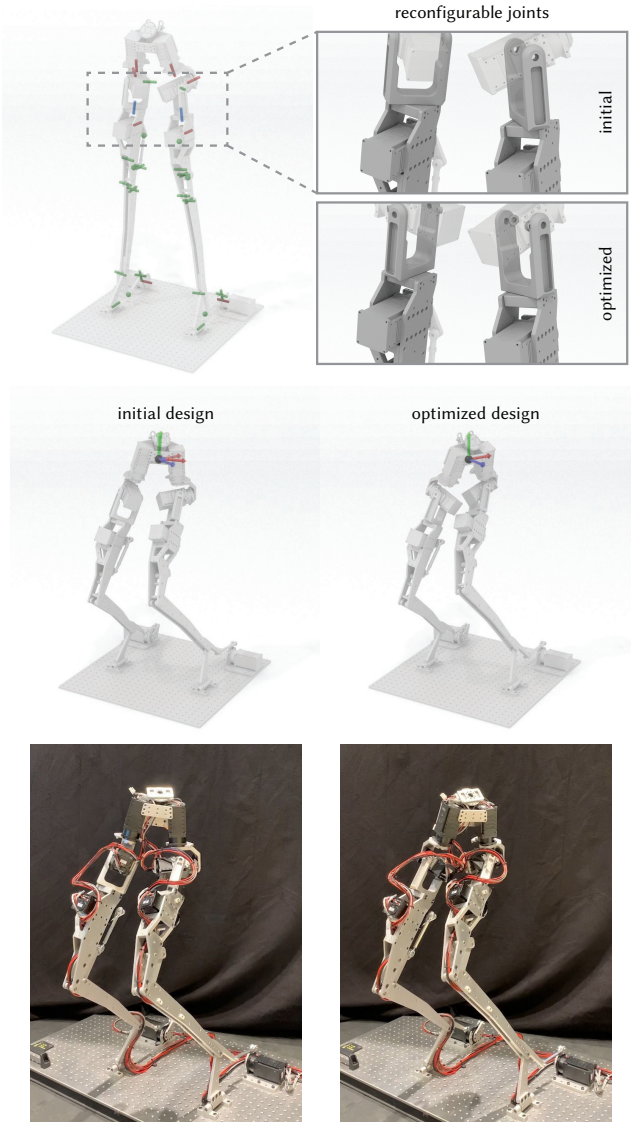

Fig. 7. **Legs.** We consider a robot leg mechanism with closed-loop kinematics (top left) which has reconfigurable joints that can be repositioned after the robot has been built (top right). The robot is overactuated, with 8 actuators and 6 degrees of freedom, and also overconstrained. We ask the pelvis to track a twisting motion, which lies outside the workspace of the initial design (middle left). The optimization reorients the reconfigurable joints so that it can track the motion (middle right). Bottom row shows the physical robot, in both the initial and optimized configurations. See also the supporting video. We follow the same color scheme as Fig. 7 when visualizing the kinematics, with actuators shown as red cylinders, configurable joints shown in blue, and mechanical joints shown in green.

Bear. We also use our pipeline to design an animatronic bear that does a swiping motion. We start with an animation rig and target

motion, and an initial robot design where a single actuator is placed at each rig joint. To give the robot mobility, we also add a single joint at the ankle/wrist and two joints at the hips/shoulders for the 3 limbs which are fixed to the ground. This results in an over-actuated design. We parameterize the orientation of all revolute joints and actuators by introducing configurable spherical joints as illustrated in Fig. 2.

We ask for the body and the free paw to track animation targets. As seen in Fig. 8, the initial design tracks the targets poorly, with significant tracking errors in parts of the motion. The optimization is able to reorient the joints and actuators to give a design with greatly improved tracking of the desired motion. See also the supporting video.

Biker. Finally, we demonstrate the editing and iterative refinement of a spatial 2-DoF biker mechanism, where we parameterize the orientation of all revolute joints as illustrated in Fig. 2. We start from an initial mechanism design created by a mechanical engineer, and prescribe motion profiles for the two actuators. We run forward kinematics on the initial design to extract motion profiles, which we edit and subsequently use as optimization targets. Due to the spatial nature of the design, small changes in the joint orientations lead to significant and counter-intuitive changes in the resultant motion, making this a challenging design space to navigate.

As seen in Fig. 9, the initial design has a limited torso roll, and an asymmetric motion. To remedy this, we ask for a torso roll motion which is symmetric and more pronounced. We also ask for the motion profiles of the actuators to remain unchanged, forcing the optimization to instead make changes to the design. It can be seen that the optimized design is able to track the desired motion closely.

We can then refine our figure design by asking for the torso to lean further forward. We still track a single frame at the torso, but edit the pitch angle of the frame. The optimization is able to find a solution which leans forward while maintaining the desired torso roll motion.

To further refine the motion, we add additional objectives that ask for the elbows to be brought closer to the body, and for the pitching motion of the feet to be reduced. We then rerun our optimization.

Once we're happy with the result, we can fabricate the optimized design as shown in Fig. 9 and also the supporting video. The fabricated robot is 355 mm tall.

Performance. A direct approach to solve the QP in Eq. 4 consists in forming the equivalent linear system, and applying a generic sparse solver. As an alternative, the method introduced in Sec. 6.2 directly exploits the problem structure, and avoids forming the full system matrix, reducing the computational effort both in time and memory. Key statistics for the different demonstrators are summarized in Tab. 5, and a performance evaluation of our technique, in comparison with a sparse direct solver, is provided in Tab. 7. Note that our method and the sparse solver, that we use as a baseline, compute search directions as solutions of the *same* QP, therefore the convergence of the overall SQP approach is the same in both cases.

Our optimizer was implemented in C++, relying on solvers from the Eigen library for linear systems: SimplicialLDLT in Eq. 7; JacobiSVD in Eq. 8; ConjugateGradient in Alg. 3 (with tolerance set to machine precision to prevent error accumulation); and SparseLU for our baseline direct solver. Computations were performed on a machine with an Intel Core i7-7700 processor (4 cores, 4.2 GHz) and 32 GB of RAM.

To estimate how well our method scales with computationally more challenging examples, in particular in comparison to a direct sparse solver approach, we extend the animation of our biker example to 6 and 10 cycles, increasing the number of control variables. We refer to these examples as Biker ext. 1 and 2 in Tab. 7. A sparse direct solver fails to solve the 10 cycle animation, because it runs out of memory. An iterative solver, computing Hessian-vector products on-the-fly, could alleviate this issue, but would unlikely be competitive in time complexity, due to recomputations of derivatives that are no longer stored. Our solution strategy, based on dynamic programming, scales with animation length and complexity of the robot, while a direct solver does not. Moreover, we observe that our solver is more robust, and consistently faster, even for smaller problems.

9 CONCLUSIONS

We have presented a method for optimizing the kinematics of robot characters, given an initial parameterized design and a desired motion. The method supports robots with arbitrary spatial kinematics, including kinematic loops, overactuation, and overconstrained designs, as shown with our demonstrators. By jointly solving for the state, design and control parameters of the robot, we ensure that optimized motions remain kinematically feasible and within the actuator position and velocity limits.

To this end, we have also devised an approach for parameterizing robot designs which is sufficiently general to handle a wide range of practical design problems, including fully-parameterized new robot designs and also the refinement of reconfigurable robot designs. With our demonstrators, we have shown different examples of workflows which are enabled by our tool, including the iterative refinement of spatial linkages, as well as the design of new robot characters. With physical examples, we have verified that our method holds in practice.

9.1 Limitations and Future Work

Although not an issue for the demonstrators seen here, closed-loop mechanisms may exhibit kinematic singularities, which could cause unwanted mobility [Maloisel et al. 2021]. We leave the automated detection and avoidance of singularities as future work.

We also did not implement an automated collision avoidance, which is especially useful for design optimizations of reconfigurable figures for which the component geometry is final. A penalty-based approach is straightforward to integrate into our design optimization.

In this work, we have solved a kinematics problem. A natural extension to the work would be to tackle the dynamics problem. This could open up exciting avenues, such as optimizing for robot designs that exploit passive system dynamics to create more expressive motions [Sun et al. 2023]. A dynamics formulation would also allow for the optimization of underactuated and underconstrained robots.

Table 5. **Key specifications of the robots.**

	Comp.	Joints				Actuators	Config. joints		State vars.	Cts.	Redund. cts.		Control vars.	Design vars.
		Ground	Rev.	Univ.	Spher.	Rev.	Rev.	Spher.			Passive	Active		
Kickbot (1)	37	1	0	0	0	14	2	20	259	259	0	0	14	82
Kickbot (2)	15	1	0	0	0	14	0	0	105	105	0	0	14	0
Legs	26	1	14	4	4	8	2	0	182	190	6	2	8	2
Bear	53	1	9	0	0	12	0	33	371	374	0	3	12	132
Biker	40	1	15	0	3	2	0	23	280	280	0	0	2	92

Comp: number of components; **Joints**: number of joints (ground, revolute, universal or spherical); **actuators**: number of actuators (revolute); **Config. joints**: number of configurable joints (revolute or spherical); **State vars.**: number of state variables; **Cts**: number of constraints; **Redund. cts**: number of redundant constraints (passive and active); **Control vars.**: number of control variables; **Design vars.**: number of design variables.

Table 6. **Performance statistics for key objectives.**

		Pos. error (cm)		Ori. error (°)		Objective
		Max.	Avg.	Max.	Avg.	improv.
Kickbot (1)	Init.	26.3	7.3	126.7	49.5	93.82 %
	Opt.	8.8	2.2	49.4	14.4	
Kickbot (2)	Init.	26.7	3.2	57.0	16.0	67.02 %
	Opt.	13.8	2.3	45.2	13.7	
Legs	Init.	0.41	0.17	29.4	10.0	99.97 %
	Opt.	0.01	0.001	1.3	0.09	
Bear	Init.	7.0	1.1	27.3	5.9	78.31 %
	Opt.	3.7	0.6	18.3	3.7	
Biker	Init.	0.37	0.04	30.2	16.2	99.19 %
	Opt.	1.19	0.34	10.0	3.6	

The max. and avg. errors are computed over all time steps and motion objectives (with uniform weighting). The orientation error measures the angle of the rotation between current and target frames. The objective improvement is the reduction of the overall objective in Eq. 2, relative to its initial value.

Table 7. **Key time performance statistics.**

	Steps	Vars.	Cts.	It.	#It.	Total	Sp. solver
Kickbot (1)	466	171 940	165 354	2.21s	29	1min 5s	10min 24s
Kickbot (2)	523	69 545	62 223	385ms	110	43.6s	3min 25s
Legs	120	23 992	23 030	162ms	15	2.60s	12.4s
Bear	101	53 215	51 904	1.20s	24	29.3s	5min 6s
Biker	120	45 118	44 809	742ms	7	5.29s	36.1s
Biker ext. 1	720	270 718	269 209	3.71s	7	26.3s	5min 36s
Biker ext. 2	1200	451 198	448 729	6.26s	7	44.4s	N/A

Steps: number of time steps in the target animation; **Vars.**: number of optimization variables; **Cts.**: number of optimization constraints; **It.**: average iteration time; **#It.**: number of optimizer iterations; **Total**: total optimization time, with our dynamic programming approach; **Sp. solver**: corresponding optimization time with a sparse solver strategy. The Biker ext. 2 example (10 animation cycles) ran out of memory after 1 iteration in the sparse solver due to the size of the system matrix.

Our approach has tackled the design of robots for which a desired motion is prescribed. However, one could also consider a design problem for which a workspace is specified but the desired motion is not known at design time. One approach here could be to study sensitivities of a robot design in the neighborhood of a motion, but it is also an open question how to best sample such a workspace.

Our work has focused on robots which are fixed to the ground. An interesting avenue for future work would be to handle changing contact states with the environment, which would open up avenues for the optimization of walking robots and also interactive characters.

ACKNOWLEDGMENTS

We wish to thank Maurizio Nitti for artistic input.

REFERENCES

Brandon Amos, Ivan Jimenez, Jacob Sacks, Byron Boots, and J Zico Kolter. 2018. Differentiable MPC for end-to-end planning and control. *Advances in neural information processing systems* 31 (2018).

Moritz Bächer, Stelian Coros, and Bernhard Thomaszewski. 2015. LinkEdit: Interactive Linkage Editing Using Symbolic Kinematics. *ACM Trans. Graph.* 34, 4, Article 99 (jul 2015), 8 pages. https://doi.org/10.1145/2766985

Filip Bjelonic, Joonho Lee, Philip Arm, Dhionis Sako, Davide Tateo, Jan Peters, and Marco Hutter. 2023. Learning-Based Design and Control for Quadrupedal Robots With Parallel-Elastic Actuators. *IEEE Robotics and Automation Letters* 8, 3 (2023), 1611–1618. https://doi.org/10.1109/LRA.2023.3234809

Ivan I Borisov, Evgenii E Khomutov, Sergey A Kolyubin, and Stefano Stramigioli. 2021. Computational design of reconfigurable underactuated linkages for adaptive grippers. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 6117–6123.

Christof Büskens and Helmut Maurer. 2001. Sensitivity analysis and real-time control of parametric optimal control problems using nonlinear programming methods. *Online Optimization of Large Scale Systems* (2001), 57–68.

Duygu Ceylan, Wilmot Li, Niloy J. Mitra, Maneesh Agrawala, and Mark Pauly. 2013. Designing and Fabricating Mechanical Automata from Mocap Sequences. *ACM Transactions on Graphics* 32, 6 (2013), 11 pages.

Yingjie Cheng, Peng Song, Yukun Lu, Wen Jie Jeremy Chew, and Ligang Liu. 2022. Exact 3D Path Generation via 3D Cam-Linkage Mechanisms. *ACM Trans. Graph.* 41, 6, Article 225 (nov 2022), 13 pages. https://doi.org/10.1145/3550454.3555431

Yingjie Cheng, Yucheng Sun, Peng Song, and Ligang Liu. 2021. Spatial-Temporal Motion Control via Composite Cam-Follower Mechanisms. *ACM Trans. Graph.* 40, 6, Article 270 (dec 2021), 15 pages. https://doi.org/10.1145/3478513.3480477

Jean-François Collard, Pierre Duysinx, and Paul Fisette. 2009. *Kinematical Optimization of Closed-Loop Multibody Systems*. Springer Netherlands, Dordrecht, 159–179. https://doi.org/10.1007/978-1-4020-8829-2_9

Stelian Coros, Bernhard Thomaszewski, Gioacchino Noris, Shinjiro Sueda, Moira Forberg, Robert W. Sumner, Wojciech Matusik, and Bernd Bickel. 2013. Computational Design of Mechanical Characters. *ACM Trans. Graph.* 32, 4, Article 83 (jul 2013), 12 pages. https://doi.org/10.1145/2461912.2461953

Flavio De Vincenti, Dongho Kang, and Stelian Coros. 2021. Control-Aware Design Optimization for Bio-Inspired Quadruped Robots. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 1354–1361. https://doi.org/10.1109/IROS51168.2021.9636415

Ruta Desai, Fraser Anderson, Justin Matejka, Stelian Coros, James McCann, George Fitzmaurice, and Tovi Grossman. 2019. Geppetto: Enabling Semantic Design of Expressive Robot Behaviors. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–14. https://doi.org/10.1145/3290605.3300599

Ruta Desai, Ye Yuan, and Stelian Coros. 2017. Computational abstractions for interactive design of robotic devices. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 1196–1203. https://doi.org/10.1109/ICRA.2017.7989143

Traiko Dinev, Carlos Mastalli, Vladimir Ivan, Steve Tonneau, and Sethu Vijayakumar. 2022. A versatile co-design approach for dynamic legged robots. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 10343–10349.

Tao Du, Adriana Schulz, Bo Zhu, Bernd Bickel, and Wojciech Matusik. 2016. Computational Multicopter Design. *ACM Trans. Graph.* 35, 6, Article 227 (nov 2016), 10 pages. https://doi.org/10.1145/2980179.2982427

Mehran Ebrahimi, Adrian Butscher, Hyunmin Cheong, and Francesco Iorio. 2019. Design optimization of dynamic flexible multibody systems using the discrete adjoint variable method. *Computers & Structures* 213 (2019), 82–99. https://doi.org/10.1016/j.compstruc.2018.12.007

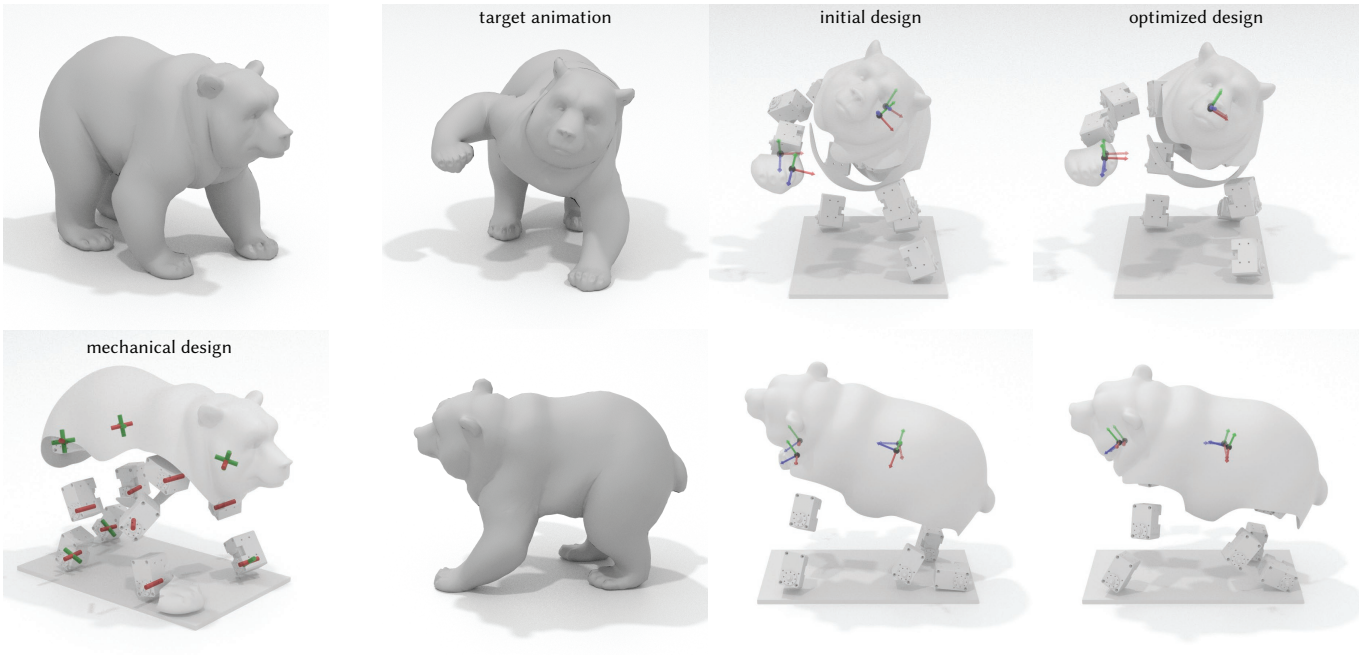

Fig. 8. **Bear.** We start with an initial robot design, which is overactuated and has 3 legs fixed to the ground, and a target animation. We ask to track the motion of the swiping paw and the body. We parameterize the axes of all revolute joints and actuators. The initial motion tracks the motion poorly, and the optimization is able to refine the kinematics so that the tracking is greatly improved. We follow the same color scheme for kinematics, and visualize actuators in red and joints in green.

Gabriele Fadini, Thomas Flayols, Andrea Del Prete, Nicolas Mansard, and Philippe Souères. 2021. Computational design of energy-efficient legged robots: Optimizing for size and actuators. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 9898–9904.

Xudong Feng, Jiafeng Liu, Huamin Wang, Yin Yang, Hujun Bao, Bernd Bickel, and Weiwei Xu. 2019. Computational Design of Skinned Quad-Robots. *IEEE Transactions on Visualization and Computer Graphics* 27, 6 (2019), 2881–2895.

Moritz Geilinger, Roi Poranne, Ruta Desai, Bernhard Thomaszewski, and Stelian Coros. 2018. Skaterbots: Optimization-Based Design and Motion Synthesis for Robotic Creatures with Legs and Wheels. *ACM Trans. Graph.* 37, 4, Article 160 (jul 2018), 12 pages. https://doi.org/10.1145/3197517.3201368

Sébastien Gros and Moritz Diehl. 2022. Numerical Optimal Control. (2022). unpublished.

Sehoon Ha, Stelian Coros, Alexander Alspach, James M. Bern, Joohyung Kim, and Katsu Yamane. 2018a. Computational Design of Robotic Devices From High-Level Motion Specifications. *IEEE Transactions on Robotics* 34, 5 (2018), 1240–1251. https://doi.org/10.1109/TRO.2018.2830419

Sehoon Ha, Stelian Coros, Alexander Alspach, Joohyung Kim, and Katsu Yamane. 2016. Task-based limb optimization for legged robots. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2062–2068. https://doi.org/10.1109/IROS.2016.7759324

Sehoon Ha, Stelian Coros, Alexander Alspach, Joohyung Kim, and Katsu Yamane. 2017. Joint Optimization of Robot Design and Motion Parameters using the Implicit Function Theorem.. In *Robotics: Science and systems*, Vol. 8.

Sehoon Ha, Stelian Coros, Alexander Alspach, Joohyung Kim, and Katsu Yamane. 2018b. Computational co-optimization of design parameters and motion trajectories for robotic systems. *The International Journal of Robotics Research* 37, 13-14 (2018), 1521–1536. https://doi.org/10.1177/0278364918771172 arXiv:https://doi.org/10.1177/0278364918771172

Shayan Hoshyari, Hongyi Xu, Espen Knoop, Stelian Coros, and Moritz Bächer. 2019. Vibration-minimizing motion retargeting for robotic characters. *ACM Transactions on Graphics (TOG)* 38, 4 (2019), 1–14.

Simon Huber, Roi Poranne, and Stelian Coros. 2021. Designing Actuation Systems for Animatronic Figures via Globally Optimal Discrete Search. *ACM Trans. Graph.* 40, 4, Article 174 (jul 2021), 10 pages. https://doi.org/10.1145/3450626.3459867

Soonwoong Hwang, Hyeonguk Kim, Younsung Choi, Kyoosik Shin, and Changsoo Han. 2017. Design optimization method for 7 DOF robot manipulator using performance indices. *International Journal of Precision Engineering and Manufacturing* 18 (2017), 293–299.

David H. Jacobson and David Q. Mayne. 1970. *Differential Dynamic Programming*. American Elsevier Publishing Company.

Ridha Kelaiaia, Ahmed Chemori, Allaoua Brahmia, Adlen Kerboua, Abdelouahab Zaatri, and Olivier Company. 2023. Optimal dimensional design of parallel manipulators with an illustrative case study: A review. *Mechanism and Machine Theory* 188 (2023), 105390.

Marin Kobilarov, Duy-Nguyen Ta, and Frank Dellaert. 2015. Differential dynamic programming for optimal estimation. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*.

Bongjin Koo, Wilmot Li, JiaXian Yao, Maneesh Agrawala, and Niloy J. Mitra. 2014. Creating Works-like Prototypes of Mechanical Objects. *ACM Trans. Graph.* 33, 6, Article 217 (nov 2014), 9 pages. https://doi.org/10.1145/2661229.2661289

Sylvie Leguay-Durand and Claude Reboulet. 1997. Optimal design of a redundant spherical parallel manipulator. *Robotica* 15, 4 (1997), 399–405.

Minmin Lin, Tianjia Shao, Youyi Zheng, Niloy Jyoti Mitra, and Kun Zhou. 2017. Recovering functional mechanical assemblies from raw scans. *IEEE transactions on visualization and computer graphics* 24, 3 (2017), 1354–1367.

Zhenyuan Liu, Jingyu Hu, Hao Xu, Peng Song, Ran Zhang, Bernd Bickel, and Chi-Wing Fu. 2022. Worst-Case Rigidity Analysis and Optimization for Assemblies with Mechanical Joints. *Computer Graphics Forum* (2022). https://doi.org/10.1111/cgf.14490

Yunjiang Lou, Yongsheng Zhang, Ruining Huang, Xin Chen, and Zexiang Li. 2013. Optimization algorithms for kinematically optimal design of parallel manipulators. *IEEE Transactions on Automation Science and Engineering* 11, 2 (2013), 574–584.

Guirec Maloisel, Espen Knoop, Bernhard Thomaszewski, Moritz Bächer, and Stelian Coros. 2021. Singularity-Aware Design Optimization for Multi-Degree-of-Freedom Spatial Linkages. *IEEE Robotics and Automation Letters* 6, 4 (2021), 6585–6592. https://doi.org/10.1109/LRA.2021.3095043

Vittorio Megaro, Espen Knoop, Andrew Spielberg, David I.W. Levin, Wojciech Matusik, Markus Gross, Bernhard Thomaszewski, and Moritz Bächer. 2017a. Designing Cable-Driven Actuation Networks for Kinematic Chains and Trees. In *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation*, Bernhard Thomaszewski, KangKang Yin, and Rahul Narain (Eds.). ACM. https://doi.org/10.1145/3099564.3099576

Vittorio Megaro, Bernhard Thomaszewski, Maurizio Nitti, Otmar Hilliges, Markus Gross, and Stelian Coros. 2015. Interactive Design of 3D-Printable Robotic Creatures. *ACM Trans. Graph.* 34, 6, Article 216 (nov 2015), 9 pages. https://doi.org/10.1145/

2816795.2818137
Vittorio Megaro, Jonas Zehnder, Moritz Bächer, Stelian Coros, Markus Gross, and Bernhard Thomaszewski. 2017b. A Computational Design Tool for Compliant Mechanisms. *ACM Trans. Graph.* 36, 4, Article 82 (jul 2017), 12 pages. https://doi.org/10.1145/3072959.3073636
J-P Merlet. 2002. Optimal design for the micro parallel robot MIPS. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, Vol. 2. IEEE, 1149–1154.
Jean-Pierre Merlet. 2005. Optimal design of robots. In *Robotics: Science and systems*.
Jean-Pierre Merlet. 2006. *Parallel robots*. Vol. 128. Springer Science & Business Media.
Karol Miller. 2004. Optimal design and modeling of spatial parallel manipulators. *The International Journal of Robotics Research* 23, 2 (2004), 127–140.
Niloy J. Mitra, Yong-Liang Yang, Dong-Ming Yan, Wilmot Li, and Maneesh Agrawala. 2010. Illustrating How Mechanical Assemblies Work. *ACM Transactions on Graphics* 29, 3, Article 58 (2010), 12 pages.
Jorge Nocedal and Stephen J. Wright. 2006. *Numerical Optimization*. Springer, New York, NY, USA.
Alex Oshin, Matthew D. Houghton, Michael J. Acheson, Irene M. Gregory, and Evangelos A. Theodorou. 2022. Parameterized Differential Dynamic Programming.
Francois Pierrot, Vincent Nabat, Olivier Company, Sebastien Krut, and Philippe Poignet. 2009. Optimal design of a 4-DOF parallel manipulator: From academia to industry. *IEEE Transactions on Robotics* 25, 2 (2009), 213–224.
James Blake Rawlings, David Q Mayne, and Moritz Diehl. 2017. *Model predictive control: theory, computation, and design*. Vol. 2. Nob Hill Publishing Madison, WI.
Charles Schaff, David Yunis, Ayan Chakrabarti, and Matthew R Walter. 2019. Jointly learning to construct and control agents using deep reinforcement learning. In *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 9798–9805.
Adriana Schulz, Ariel Shamir, David I. W. Levin, Pitchaya Sitthi-amorn, and Wojciech Matusik. 2014. Design and Fabrication by Example. *ACM Trans. Graph.* 33, 4, Article 62 (jul 2014), 11 pages. https://doi.org/10.1145/2601097.2601127
Adriana Schulz, Cynthia Sung, Andrew Spielberg, Wei Zhao, Robin Cheng, Eitan Grinspun, Daniela Rus, and Wojciech Matusik. 2017. Interactive robogami: An end-to-end system for design of robots with ground locomotion. *The International Journal of Robotics Research* 36, 10 (2017), 1131–1147. https://doi.org/10.1177/0278364917723465 arXiv:https://doi.org/10.1177/0278364917723465
Christian Schumacher, Espen Knoop, and Moritz Bächer. 2021. A Versatile Inverse Kinematics Formulation for Retargeting Motions Onto Robots With Kinematic Loops. 6, 2 (2021), 943–950. https://doi.org/10.1109/LRA.2021.3056030
Mélina Skouras, Bernhard Thomaszewski, Stelian Coros, Bernd Bickel, and Markus Gross. 2013. Computational Design of Actuated Deformable Characters. *ACM Trans. Graph.* 32, 4, Article 82 (jul 2013), 10 pages. https://doi.org/10.1145/2461912.2461979
Peng Song, Xiaofei Wang, Xiao Tang, Chi-Wing Fu, Hongfei Xu, Ligang Liu, and Niloy J. Mitra. 2017. Computational Design of Wind-up Toys. *ACM Trans. Graph.* 36, 6, Article 238 (nov 2017), 13 pages. https://doi.org/10.1145/3130800.3130808
Andrew Spielberg, Brandon Araki, Cynthia Sung, Russ Tedrake, and Daniela Rus. 2017. Functional co-optimization of articulated robots. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 5035–5042. https://doi.org/10.1109/ICRA.2017.7989587
Yilun Sun, Chujun Zong, Felix Pancheri, Tong Chen, and Tim C Lueth. 2023. Design of topology optimized compliant legs for bio-inspired quadruped robots. *Scientific Reports* 13, 1 (2023), 4875.
Bernhard Thomaszewski, Stelian Coros, Damien Gauge, Vittorio Megaro, Eitan Grinspun, and Markus Gross. 2014. Computational Design of Linkage-Based Characters. *ACM Trans. Graph.* 33, 4, Article 64 (jul 2014), 9 pages. https://doi.org/10.1145/2601097.2601143
Guanglei Wu, Stéphane Caro, Shaoping Bai, and Jørgen Kepler. 2014. Dynamic modeling and design optimization of a 3-DOF spherical parallel manipulator. *Robotics and Autonomous Systems* 62, 10 (2014), 1377–1386.
Jie Xu, Tao Chen, Lara Zlokapa, Michael Foshey, Wojciech Matusik, Shinjiro Sueda, and Pulkit Agrawal. 2021a. An End-to-End Differentiable Framework for Contact-Aware Robot Design. In *Proceedings of Robotics: Science and Systems*. Virtual. https://doi.org/10.15607/RSS.2021.XVII.008
Jie Xu, Andrew Speilberg, Allan Zhao, Daniela Rus, and Wojciech Matusik. 2021b. Multi-Objective Graph Heuristic Search for Terrestrial Robot Design. In *2021 International conference on robotics and automation (ICRA)*. IEEE.
Kourosh E Zanganeh and Jorge Angeles. 1997. Kinematic isotropy and the optimum design of parallel manipulators. *The International Journal of Robotics Research* 16, 2 (1997), 185–197.
Ran Zhang, Thomas Auzinger, and Bernd Bickel. 2021. Computational Design of Planar Multistable Compliant Structures. *ACM Transactions on Graphics (TOG)* 40, 5 (2021), 1–16.
Ran Zhang, Thomas Auzinger, Duygu Ceylan, Wilmot Li, and Bernd Bickel. 2017. Functionality-aware Retargeting of Mechanisms to 3D Shapes. *ACM Transactions on Graphics (SIGGRAPH 2017)* 36, 4 (2017).
Allan Zhao, Jie Xu, Mina Konaković-Luković, Josephine Hughes, Andrew Spielberg, Daniela Rus, and Wojciech Matusik. 2020. RoboGrammar: Graph Grammar for Terrain-Optimized Robot Design. *ACM Trans. Graph.* 39, 6, Article 188 (nov 2020), 16 pages. https://doi.org/10.1145/3414685.3417831
Lifeng Zhu, Weiwei Xu, John Snyder, Yang Liu, Guoping Wang, and Baining Guo. 2012. Motion-guided mechanical toy modeling. *ACM Transactions on Graphics (TOG)* 31, 6 (2012), 1–10.

APPENDIX

In Tab. 8, we provide expressions for the matrices in Alg. 3.

Table 8. Definition of $\tilde{\mathbf{Q}}_k$, $\tilde{\mathbf{S}}_k$, $\tilde{\mathbf{R}}_k$, $\tilde{\mathbf{P}}_n$, $\tilde{\mathbf{A}}_k$, and $\tilde{\mathbf{B}}_k$ matrices. To keep the notation concise, we omit the index k in derivatives of the Lagrangian and for matrices $\mathbf{P}$, $\mathbf{U}$, and $\mathbf{c}$. For matrices $\mathbf{X}$, $\mathbf{Z}$, $\mathbf{y}$, we omit the index $k+1$ and the superscript $\boldsymbol{\mathcal{U}}$.

matrices

$$\tilde{\mathbf{Q}}_k = \begin{bmatrix} 0 & \tilde{\mathbf{q}}_\mathrm{p}^T & \tilde{\mathbf{q}}_\mathrm{u}^T \\ \tilde{\mathbf{q}}_\mathrm{p} & \tilde{\mathbf{Q}}_{\mathrm{pp}} & \tilde{\mathbf{Q}}_{\mathrm{pu}} \\ \tilde{\mathbf{q}}_\mathrm{u} & \tilde{\mathbf{Q}}_{\mathrm{pu}}^T & \tilde{\mathbf{Q}}_{\mathrm{uu}} \end{bmatrix} \quad \tilde{\mathbf{S}}_k = \begin{bmatrix} \tilde{\mathbf{s}}_\mathrm{v} & \tilde{\mathbf{S}}_\mathrm{pv}^T & \tilde{\mathbf{S}}_\mathrm{pu}^T \end{bmatrix} \quad \tilde{\mathbf{R}}_k$$

$$\tilde{\mathbf{P}}_n = \begin{bmatrix} 0 & \tilde{\mathbf{p}}_\mathrm{p}^T & \tilde{\mathbf{p}}_\mathrm{u}^T \\ \tilde{\mathbf{p}}_\mathrm{p} & \tilde{\mathbf{P}}_{\mathrm{pp}} & \tilde{\mathbf{P}}_{\mathrm{pu}} \\ \tilde{\mathbf{p}}_\mathrm{u} & \tilde{\mathbf{P}}_{\mathrm{pu}}^T & \tilde{\mathbf{P}}_{\mathrm{uu}} \end{bmatrix} \quad \tilde{\mathbf{A}}_k = \begin{bmatrix} 1 & \mathbf{0} & \mathbf{0} \\ \tilde{\mathbf{a}}_\mathrm{p} & \tilde{\mathbf{A}}_{\mathrm{pp}} & \mathbf{0} \\ \tilde{\mathbf{a}}_\mathrm{u} & \mathbf{0} & \tilde{\mathbf{A}}_{\mathrm{uu}} \end{bmatrix} \quad \tilde{\mathbf{B}}_k = \begin{bmatrix} 0 \\ \mathbf{0} \\ \tilde{\mathbf{b}}_\mathrm{u} \end{bmatrix}$$

matrix entries

$$\begin{aligned}
\tilde{\mathbf{q}}_\mathrm{p} &\coloneqq (\mathcal{L}_{\mathrm{ps}} + \mathbf{P}^T\mathcal{L}_{\mathrm{ss}})\mathbf{c} + (\mathcal{L}_{\mathrm{pv}} + \mathbf{P}^T\mathcal{L}_{\mathrm{sv}})\mathbf{y} + \mathcal{L}_\mathrm{p} + \mathbf{P}^T\mathcal{L}_\mathrm{s} \\
\tilde{\mathbf{q}}_\mathrm{u} &\coloneqq (\mathcal{L}_{\mathrm{us}} + \mathbf{U}^T\mathcal{L}_{\mathrm{ss}})\mathbf{c} + (\mathcal{L}_{\mathrm{uv}} + \mathbf{U}^T\mathcal{L}_{\mathrm{sv}})\mathbf{y} + \mathbf{X}^T(\mathcal{L}_{\mathrm{vs}}\mathbf{c} + \mathcal{L}_{\mathrm{vv}}\mathbf{y}) + \mathcal{L}_\mathrm{u} + \mathbf{U}^T\mathcal{L}_\mathrm{s} \\
&\quad + \mathbf{X}^T\mathcal{L}_\mathrm{v} \\
\tilde{\mathbf{Q}}_{\mathrm{pp}} &\coloneqq \mathcal{L}_{\mathrm{pp}} + \mathcal{L}_{\mathrm{ps}}\mathbf{P} + \mathbf{P}^T\mathcal{L}_{\mathrm{sp}} + \mathbf{P}^T\mathcal{L}_{\mathrm{ss}}\mathbf{P} \\
\tilde{\mathbf{Q}}_{\mathrm{pu}} &\coloneqq \mathcal{L}_{\mathrm{pu}} + \mathcal{L}_{\mathrm{ps}}\mathbf{U} + \mathbf{P}^T\mathcal{L}_{\mathrm{su}} + \mathbf{P}^T\mathcal{L}_{\mathrm{ss}}\mathbf{U} + (\mathcal{L}_{\mathrm{pv}} + \mathbf{P}^T\mathcal{L}_{\mathrm{sv}})\mathbf{X} \\
\tilde{\mathbf{Q}}_{\mathrm{uu}} &\coloneqq \mathcal{L}_{\mathrm{uu}} + \mathcal{L}_{\mathrm{us}}\mathbf{U} + \mathbf{U}^T\mathcal{L}_{\mathrm{su}} + \mathbf{U}^T\mathcal{L}_{\mathrm{ss}}\mathbf{U} + (\mathcal{L}_{\mathrm{uv}} + \mathbf{U}^T\mathcal{L}_{\mathrm{sv}})\mathbf{X} \\
&\quad + \mathbf{X}^T(\mathcal{L}_{\mathrm{vu}} + \mathcal{L}_{\mathrm{vs}}\mathbf{U}) + \mathbf{X}^T\mathcal{L}_{\mathrm{vv}}\mathbf{X}
\end{aligned}$$

$$\begin{aligned}
\tilde{\mathbf{s}}_\mathrm{v} &\coloneqq \mathbf{Z}^T(\mathcal{L}_{\mathrm{vs}}\mathbf{c} + \mathcal{L}_{\mathrm{vv}}\mathbf{y} + \mathcal{L}_\mathrm{v}) \\
\tilde{\mathbf{S}}_{\mathrm{pv}} &\coloneqq (\mathcal{L}_{\mathrm{pv}} + \mathbf{P}^T\mathcal{L}_{\mathrm{sv}})\mathbf{Z} \\
\tilde{\mathbf{S}}_{\mathrm{uv}} &\coloneqq (\mathcal{L}_{\mathrm{uv}} + \mathbf{U}^T\mathcal{L}_{\mathrm{sv}} + \mathbf{X}^T\mathcal{L}_{\mathrm{vv}})\mathbf{Z}
\end{aligned}$$

$$\tilde{\mathbf{R}}_k \coloneqq \mathbf{Z}^T\mathcal{L}_{\mathrm{vv}}\mathbf{Z}$$

$$\begin{aligned}
\tilde{\mathbf{p}}_\mathrm{p} &\coloneqq (\mathcal{L}_{\mathrm{ps}} + \mathbf{P}^T\mathcal{L}_{\mathrm{ss}})\mathbf{c} + \mathcal{L}_\mathrm{p} + \mathbf{P}^T\mathcal{L}_\mathrm{s} \\
\tilde{\mathbf{p}}_\mathrm{u} &\coloneqq (\mathcal{L}_{\mathrm{us}} + \mathbf{U}^T\mathcal{L}_{\mathrm{ss}})\mathbf{c} + \mathcal{L}_\mathrm{u} + \mathbf{U}^T\mathcal{L}_\mathrm{s} \\
\tilde{\mathbf{P}}_{\mathrm{pp}} &\coloneqq \mathcal{L}_{\mathrm{pp}} + \mathcal{L}_{\mathrm{ps}}\mathbf{P} + \mathbf{P}^T\mathcal{L}_{\mathrm{sp}} + \mathbf{P}^T\mathcal{L}_{\mathrm{ss}}\mathbf{P} \\
\tilde{\mathbf{P}}_{\mathrm{pu}} &\coloneqq \mathcal{L}_{\mathrm{pu}} + \mathcal{L}_{\mathrm{ps}}\mathbf{U} + \mathbf{P}^T\mathcal{L}_{\mathrm{su}} + \mathbf{P}^T\mathcal{L}_{\mathrm{ss}}\mathbf{U} \\
\tilde{\mathbf{P}}_{\mathrm{uu}} &\coloneqq \mathcal{L}_{\mathrm{uu}} + \mathcal{L}_{\mathrm{us}}\mathbf{U} + \mathbf{U}^T\mathcal{L}_{\mathrm{su}} + \mathbf{U}^T\mathcal{L}_{\mathrm{ss}}\mathbf{U}
\end{aligned}$$

$$\begin{aligned}
\tilde{\mathbf{a}}_\mathrm{p} &\coloneqq \mathbf{p}_k - \mathbf{p}_{k+1} \\
\tilde{\mathbf{a}}_\mathrm{u} &\coloneqq \Delta t\mathbf{y} + (\mathbf{u}_k + \Delta t\mathbf{v}_k - \mathbf{u}_{k+1}) \\
\tilde{\mathbf{A}}_{\mathrm{pp}} &\coloneqq \mathbf{I} \\
\tilde{\mathbf{A}}_{\mathrm{uu}} &\coloneqq \mathbf{I} + \Delta t\,\mathbf{X}
\end{aligned}$$

$$\tilde{\mathbf{b}}_\mathrm{u} \coloneqq \Delta t\,\mathbf{Z}$$

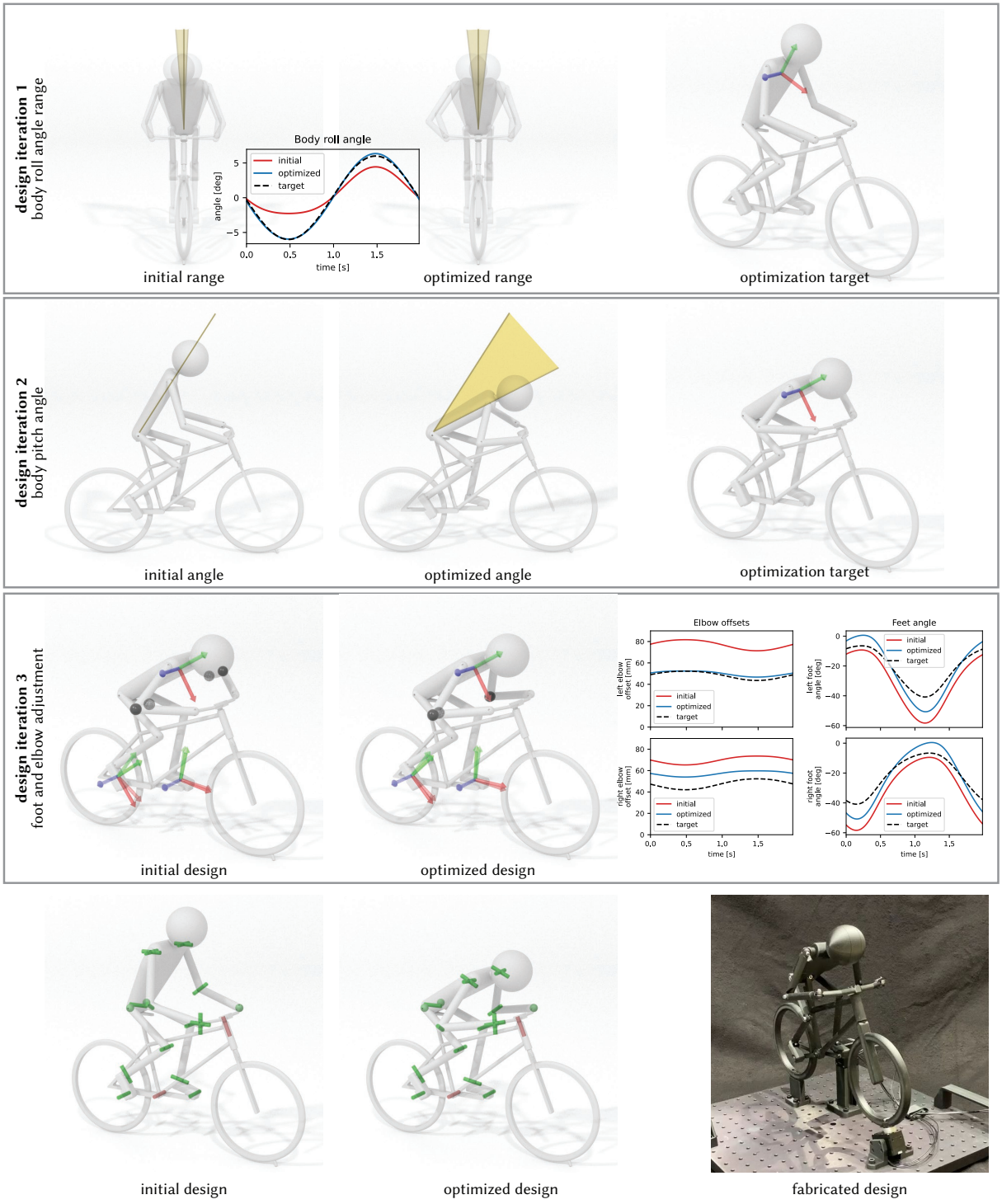

Fig. 9. **Biker.** We start with an initial mechanism design, bottom left, together with prescribed motion profiles for the actuators. We then iteratively refine the design, yielding the optimized result shown bottom center. We first consider the body roll (top row), which for the initial design has a limited range and is asymmetric. We ask for a target roll motion which is symmetric and larger in amplitude, and also ask for the control input to remain unchanged. The optimization is able to match the desired motion by changing the mechanism design. In a second design iteration (second row), we ask for the torso to lean further forward. In a final iteration, we add additional tracking markers to the feet, where we ask for a reduced foot pitching motion, and the elbows, which we ask to bring closer to the body. The optimized design can then be fabricated (bottom right). For visualizing the kinematics, we follow the same color scheme as in the previous figures.