Computational Co-Optimization of Design Parameters and Motion Trajectories for Robotic Systems

The International Journal of Robotics Research
XX(X):1–14

DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Sehoon Ha[1] and Stelian Coros[2] and Alexander Alspach[3] and Joohyung Kim[1] and Katsu Yamane[1]

Abstract

We present a novel computational approach to optimizing the morphological design of robots. Our framework takes as input a parameterized robot design as well as a motion plan consisting of trajectories for end-effectors and, optionally, for its body. The algorithm optimizes the design parameters including link lengths and actuator placements while concurrently adjusting motion parameters such as joint trajectories, actuator inputs, and contact forces. Our key insight is that the complex relationship between design and motion parameters can be established via sensitivity analysis if the robot's movements are modeled as spatio-temporal solutions to an optimal control problem. This relationship between form and function allows us to automatically optimize the robot design based on specifications expressed as a function of actuator forces or trajectories. We evaluate our model by computationally optimizing four simulated robots that employ linear actuators, four-bar linkages, or rotary servos. We further validate our framework by optimizing the design of two small quadruped robots and testing their performances using hardware implementations.

Keywords

Legged Robots, Motion Control, Mechanism Design

1 Introduction

In the not-so-distant future, a rich ecosystem of robots for service, search and rescue, personal assistance and education has the potential to improve many aspects of our lives. The process of creating new types of robots, however, is notoriously challenging because their motor capabilities are intimately related to their design. For this reason, creating new robots requires a great deal of experience, and is a largely manual and time-consuming task. This tedious and error-prone approach to creating robots is unfortunately necessitated by the lack of formal models that can predict the complex interactions between the design of a robot and its ability to effectively serve its intended purpose.

Rather than relying on trial-and-error approaches, we seek to develop a computational model with the predictive power required to inform design decisions. To achieve this goal, we must establish a relationship between the form and function of robotic devices. To this end, we model a robot's movements as spatio-temporal solutions to optimal control problems. The sensitivities of these optimal solutions enable a guided exploration on the manifold that relates a robot's morphological design parameters and its motor capabilities (Figure 1).

We test the effectiveness of our approach by optimizing the designs of various robotic devices, including a manipulator with linear actuators, a manipulator with four-bar linkages, a quadruped with linear actuators, and a quadruped with rotary servos, with the goal of minimizing the actuator forces required to realize the desired motions. Further, we fabricated two physical prototypes of small quadruped robots using 3D-printing and validate the simulation results.

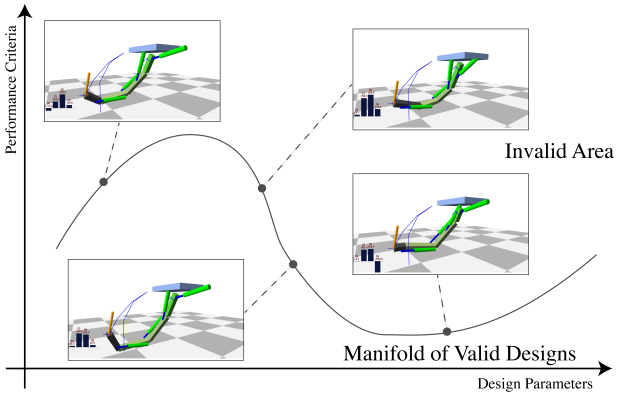

Figure 1. A set of motion and task constraints implicitly define a manifold of valid designs.

[1]Disney Research, USA
[2]ETH Zürich, Switzerland
[3]Toyota Research Institute, USA

Corresponding author:
Sehoon Ha, Disney Research Pittsburgh, 4720 Forbes Avenue, Lower Level, Suite 110 Pittsburgh, PA 15213.
Email: sehoon.ha@disneyresearch.com

2 Related Work

Designing hardware and motion of robots is a complex task that needs to consider numerous parameters and non-intuitive relationships between them. To obtain a good initial guess for this challenging problem, robot designers often get inspiration from creatures in nature. Under this paradigm, many robots have been successfully designed and built by mimicking morphology and locomotion of real-life animals including salamanders (Crespi et al. (2013)), cheetahs (Seok et al. (2014))), kangaroos (Graichen et al. (2015)), and chimpanzees (Kuehn et al. (2014)). Although these animals can be good sources of inspiration, designing robots is still a time-consuming process that requires repetitive design revisions and performance tests. In this work, we focus on providing an intuitive design framework that allows editing with shorter cycles by identifying relationships between design parameters and performance criteria.

To overcome the complex problem of designing robots, there exists prior work to automate the design by optimizing morphology for a given performance criterion, such as locomotion speed or energy consumption. In particular, evolutionary algorithms (EA), such as Genetic Algorithm (GA) or Simulated Annealing (SA), have received considerable attention for solving design optimization due to its capability to handle discrete changes. This approach has been successfully deployed to find the optimal morphology for various types of robots, including virtual creatures (Sims (1994)), manipulators (Leger (1999)), tensegrity robots (Lipson and Pollack (2000)), and soft robots (Cheney et al. (2013)). Recently, Baykal and Alterovitz (2017) proposed a sampling-based approach for optimizing the kinematic design of piecewise cylindrical robots in cluttered environments. Although evolutionary algorithms have been proven to be simple yet effective for exploring various designs, the key limitations are limited guarantee on optimality of the final design and difficulty to reproduce the same results on subsequent trials. Instead of relying on stochastic operations, our framework directly identifies the required changes for optimizing the performance of the robot.

In the robotics and computer graphics community, a number of researchers take the approach of automating the design process by optimizing the morphology of a robot for a given task. In general, this approach solves a large optimization problem that minimizes a performance criterion while satisfying kinematic and dynamic constraints. The task-based design optimization has been widely applied to manipulators (Paredis and Khosla (1991); Ceccarelli and Lanni (2004); Van Henten et al. (2009)), parallel manipulators (Kim and Ryu (2003); Collard et al. (2005); Yun and Li (2011)), and cable-driven mechanisms (Megaro et al. (2017); Li et al. (2017)) for reaching desired workspace and avoiding joint singularities. However, there exist only a few studies on under-actuated robots, such as pipe-cleaning robots (Jung et al. (2011)), stair-climbing mobile robots (Kim et al. (2012)), virtual creatures (Wampler and Popović (2009); Geijtenbeek et al. (2013)), or quadrupeds (Ha et al. (2016)) due to complexity of required models.

Such monolithic design optimization is often not easy to use in practice. One of the main problems is that this approach usually takes long time (a few hours for simple robots) to optimize the design. Furthermore, as the first run of optimization rarely results in a satisfactory design, the user needs to repeat the optimization many times while adjusting the objective function. Instead, our framework provides an iterative editing method that the user can provide the change directions any time during the optimization process.

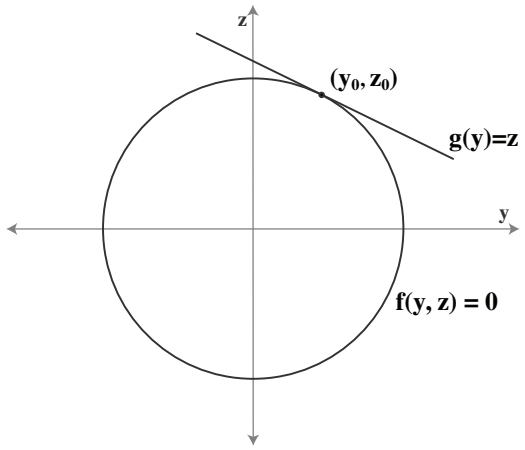

Figure 2. The implicit function theorem converts an implicit relation f into an explicit function g at the given point (y_0, z_0).

Two of the six examples used in this paper utilize linear actuators, motivated by the recent robots empowered by hydraulic or electronic linear actuators to realize large joint torques in manipulators (Hyd), humanoids (Atl), and quadrupeds (Semini et al. (2011); Rong et al. (2012); Spo (a,b)). One of the difficulties of using linear actuators is finding their optimal layout to realize large moment arms while achieving the necessary range of motion, which is not intuitive due to nonlinear relationships between joint positions and forces. The examples demonstrate that our framework can effectively handle linear actuators by including their attachment locations as part of the design parameters.

3 Background: Implicit Function Theorem

The implicit function theorem (Jittorntrum (1978)) is a tool for converting an implicitly defined relationship between two sets of variables to an explicit function. Let the following implicit function $\mathbf{f} : \mathcal{R}^{n+m} \rightarrow \mathcal{R}^m$ be a relationship between two sets of variables, $\mathbf{y} \in \mathcal{R}^n$ and $\mathbf{z} \in \mathcal{R}^m$:

$$\mathbf{f}(\mathbf{y}, \mathbf{z}) = \mathbf{0}. \tag{1}$$

For the given implicit relationships $\mathbf{f}$, our goal is to convert it to an explicit function $\mathbf{g} : \Delta\mathbf{y} \mapsto \Delta\mathbf{z}$ within a small disk around the current point, $(\mathbf{y}_0, \mathbf{z}_0)$ (Figure 2). The Jacobian $D\mathbf{f}$ describes linearized changes around the given point as:

$$(D\mathbf{f})(\mathbf{y}, \mathbf{z}) = \left[\frac{\delta \mathbf{f}}{\delta y_1}, \cdots, \frac{\delta \mathbf{f}}{\delta y_n} \middle| \frac{\delta \mathbf{f}}{\delta z_1}, \cdots, \frac{\delta \mathbf{f}}{\delta z_m}\right] \tag{2}$$

where $y_1, \cdots, y_n$ and $z_1, \cdots, z_m$ are entries of $\mathbf{y}$ and $\mathbf{z}$. When we change $\mathbf{y}_0$ and $\mathbf{z}_0$ by $\Delta\mathbf{y}$ and $\Delta\mathbf{z}$, the change of the function $\Delta\mathbf{f}$ should be zero to remain on the manifold

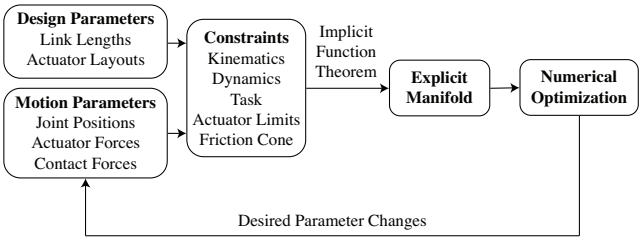

Figure 3. Overview of the proposed framework. Our algorithm co-optimizes the design and motion parameters by relating them with the implicit function theorem.

defined by $\mathbf{f}$:

$$\begin{aligned}\Delta\mathbf{f} = \mathbf{0} &= (D\mathbf{f})(\mathbf{y}_0, \mathbf{z}_0)[\Delta\mathbf{y}; \Delta\mathbf{z}] \\ &= \left[\frac{\delta\mathbf{f}}{\delta y_1}, \cdots, \frac{\delta\mathbf{f}}{\delta y_n}\right]\Delta\mathbf{y} + \left[\frac{\delta\mathbf{f}}{\delta z_1}, \cdots, \frac{\delta\mathbf{f}}{\delta z_m}\right]\Delta\mathbf{z}. \end{aligned} \quad (3)$$

If there are no additional inequality constraints, the explicit function g can be obtained by simply taking the inverse of the Jacobian matrix with respect to $\mathbf{z}$:

$$\Delta\mathbf{z} = -\left[\frac{\delta\mathbf{f}}{\delta z_1}, \cdots, \frac{\delta\mathbf{f}}{\delta z_m}\right]^{-1}\left[\frac{\delta\mathbf{f}}{\delta y_1}, \cdots, \frac{\delta\mathbf{f}}{\delta y_n}\right]\Delta\mathbf{y}. \quad (4)$$

For the general k dimensional function $\mathbf{f} : \mathcal{R}^{n+m} \rightarrow \mathcal{R}^k$, we could take the Moonre-Penrose pseudoinverse or employ numerical optimization to obtain $\Delta\mathbf{z}$ that satisfies Eq. (3) as we describe in the next section.

4 Algorithm

When a robot executes a motion, the robot and motion can be described by a set of design and motion parameters. Our goal here is to develop a tool that can efficiently navigate through the implicitly defined manifold of valid parameters. In this section, we first describe how the implicit function theorem can be applied to interactive robot design, followed by the details of our problem formulation. We then present the details on how to formulate the relationships between various parameters as a linear system using the implicit function theorem, and how to efficiently solve it using the null space of the linear system.

4.1 Overview

A robot design can be described by a set of parameters such as link lengths and actuator attachment points. A robot motion, on the other hand, can be described by joint positions, joint torques, and contact forces at every time frame. These variables collectively form a set of design and motion parameters that defines a robot design and its motion.

The design and motion parameters must satisfy various constraints such as the equation of motion. In some cases, the desired motion may be given as end-effector trajectories that provide additional constraints on the end-effector positions at every frame.

These constraints form the implicitly-defined manifold of valid robot designs and their corresponding motions. We apply the implicit function theorem to derive the relationships among the design and motion parameters. By grouping the parameters in various combinations into $\mathbf{y}$ and $\mathbf{z}$, we can compute how to change a subset of parameters in response to the changes in the other parameters while maintaining the constraints. For example, if we form $\mathbf{y}$ by all the design parameters and include all motion parameters in $\mathbf{z}$, we can compute how the motion should be changed according to a design change.

We can even perform certain types of optimization using this technique (Figure 3). Consider choosing the maximum force of a specific actuator as the only element of $\mathbf{y}$, and giving a small negative value as $\Delta\mathbf{y}$. In this case, Eq. (4) gives how to concurrently change the design and motion to reduce the peak force. Alternatively, we can choose to fix the design and change only the motion by adding the design parameters to $\mathbf{y}$ and specifying zero changes for them.

This formulation provides an entirely new approach to the robot design problem: the user can choose the subset of parameters and their change directions, and the algorithm will automatically compute how the other parameters should be changed for the system to stay on the constraint manifold. In contrast to traditional numerical optimization, the user no longer has to formulate the cost function for each set of optimization variables, or indirectly manipulate the optimization result by adjusting the weights in the cost function.

4.2 Problem Description

4.2.1 Parameters In our problem, we have two sets of parameters: design parameters and motion parameters. Design parameters, such as link lengths $\mathbf{l}$ and attachment points of linear actuators $\boldsymbol{\alpha}$, define the robot's morphology that is constant over time. On the other hand, motion parameters describe the state and control signals of the motion at each of the N sample frames. Motion parameters include joint positions $\mathbf{q}_i$, actuator forces $\boldsymbol{\tau}_i$, and contact forces at the M end-effectors $\mathbf{f}_{i,j}$ $(i = 1, 2, \ldots, N, j = 1, 2, \ldots, M)$. We assume that only the end-effectors make contact with the environment. Finally, we introduced the motion plan scale parameters $\boldsymbol{\kappa} = [\kappa_x, \kappa_y, \kappa_z]^T$ that change the scale of the given motion plan. For instance, the optimization can elongate the step length of a quadruped robot by increasing κ in the forward direction.

By collecting all the design and motion parameters, we define the parameter vector $\mathbf{x}$ as $\mathbf{x} = [\mathbf{l}^T, \boldsymbol{\alpha}^T, \mathbf{q}_1^T, \ldots, \mathbf{q}_N^T, \boldsymbol{\tau}_1^T, \ldots \boldsymbol{\tau}_N^T, \mathbf{f}_{1,1}^T, \cdots, \mathbf{f}_{N,M}^T, \boldsymbol{\kappa}]^T$. We denote the size of $\mathbf{x}$ by P.

4.2.2 Constraints A set of design and motion parameters is valid only when it satisfies the task and physics constraints.

The first constraint we consider is that the motion of the robot must satisfy the equation of motion. We therefore define the objective function associated with the equation of motion as

$$w_i^{EOM}(\mathbf{x}) = |\mathbf{M}\ddot{\mathbf{q}}_i + \mathbf{c} - \mathbf{R}\boldsymbol{\tau}_i - \sum_j^M \mathbf{J}_j^T \mathbf{f}_{i,j}|^2 = 0 \quad (5)$$

where $\mathbf{M}$ is the joint-space inertia matrix, $\mathbf{c}$ is the sum of gravitational, centrifugal and Coriolis forces, $\mathbf{R}$ is a moment arm matrix that maps actuator forces to joint torques, and $\mathbf{J}_j$ is a Jacobian matrix of the position of the jth end-effector with respect to the joint positions. For brevity of

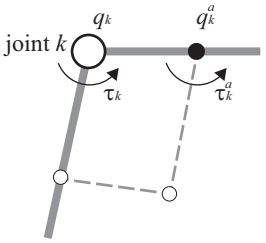

Figure 4. Four-bar linkage example.

representation, we omit the dependency of matrices: for instance, the moment arm matrix $\mathbf{R}$ depends on $\mathbf{l}$, $\boldsymbol{\alpha}$, and $\mathbf{q}_i$. The velocity $\dot{\mathbf{q}}_i$ and acceleration $\ddot{\mathbf{q}}_i$ are computed from finite differences of $\mathbf{q}_i$. When the robot has an unactuated floating base, we set the corresponding values of $\boldsymbol{\tau}_i$ to zero and remove them from the free variables. Because we describe the dynamics of the robot using the full equations of motion, we do not have any specific assumptions such as quasi-static stability.

The moment arm matrix $\mathbf{R}$ is introduced to uniformly represent various actuation mechanisms including rotary servos, four-bar linkages, and linear actuators. When joint k is actuated by a rotary servo directly located at the joint axis, the (k,k) element of $\mathbf{R}$, R_{kk}, is 1. When joint k is driven by a four-bar linkage as shown in Figure 4 where the black joint is actuated and white ones are passive, we simplify the formulation by dividing the mechanism into the main chain (thick gray lines) and supporting structure (thin dashed lines). We treat the supporting structure as a pure torque source and ignore its mass and inertia. Note that some linkage structures, such as the Jansen's linkage, do not have this assumption. In this case, we can represent its dynamics using either the Lagrangian approach (equations of motions) or the Eulerian approach (body dynamics + constraints).

In Eq. (5), the active joint torque $\tau^a_{k,i}$ is included in $\boldsymbol{\tau}_i$ while the equivalent main joint angle $q_{k,i}$ is used in $\boldsymbol{q}_i$. From now, we will omit the frame index i for brevity. The velocities of the main and active joints are related by the ratio h_k as follows:

$$\dot{q}^a_k = h_k(q_k)\dot{q}_k. \tag{6}$$

The principle of virtual work yields

$$\tau_k \delta q_k = \tau^a_k \delta q^a_k. \tag{7}$$

In the four-bar linkage case, therefore, we have $R_{kk} = h_k(q_k)$. For linear actuators, R_{kk} is:

$$R_{kk} = (\mathbf{r}_k \times \mathbf{s}_k)^T \mathbf{a}_k \tag{8}$$

where $\mathbf{s}_k$ is the actuator force direction, $\mathbf{a}_k$ is the direction of the joint axis, and $\mathbf{r}_k$ is a vector pointing from a point on the joint axis to a point on the line of action of the actuator force.

Many tasks can be described by a set of desired end-effector trajectories and its objective function can be defined as:

$$w^{EE}_{i,j}(\mathbf{x}) = |\mathbf{e}_j(\mathbf{l}, \mathbf{q}_i) - \bar{\mathbf{e}}_{i,j}|^2 = 0 \tag{9}$$

where $\mathbf{e}_j(*)$ evaluates the j-th end-effector position and $\bar{\mathbf{e}}_{i,j}$ is the desired position of the j-th end-effector at the i-th frame.

When the robot has linear actuators or four bar linkages, the morphology of the robot has closed kinematic loops. We enforce jth kinematic loop constraint at ith frame as follows:

$$w^{Closed}_{i,j}(\mathbf{x}) = |\mathbf{p}^A_{i,j}(\mathbf{l}, \mathbf{q}_i) - \mathbf{p}^B_{i,j}(\mathbf{l}, \mathbf{q}_i)|^2 = 0. \tag{10}$$

where this constraint indicates that positions of two points $\mathbf{p}^A$ and $\mathbf{p}^B$ should be matched.

In case of cyclic motions such as locomotion, we place an additional constraint that the joint positions for the first and last frames must be identical:

$$w^{Loop}(\mathbf{x}) = |\mathbf{q}_1 - \mathbf{q}_N|^2 = 0 \tag{11}$$

In addition to the equality constraints that represent the implicit relationships, we need to define a set of inequality constraints to limit the range of some of the parameters and enforce contact forces to stay within a friction cone. For brevity, we will omit the details of these straightforward inequality constraints.

For robots with rotary actuators, the lower and upper limits for joint positions can be simply defined as bounds on the elements of $\mathbf{q}_i$. If the robot includes linear actuators, their joint positions are limited by the stroke ranges. Therefore, we place the following constraint on the positions of each of the K actuators as

$$l^{lo}_k \leq l_{i,k}(\mathbf{x}) \leq l^{hi}_k \tag{12}$$

where l^{lo}_k and l^{hi}_k are the lower and upper bounds of the k-th linear actuator, and $l_{i,k}(*)$ computes its position at the i-th frame.

4.3 Formulation of Linear System

A change in any subset of parameters will likely cause violation of some constraints, and such violation must be corrected by changing the other parameters. We apply the implicit function theorem to obtain the linear relationships between the parameters in order to constrain the parameter changes to stay on the manifold. In the following, we refer to the previously defined H constraints as $w_1, \ldots, w_H$, and their gradients with respect to $\mathbf{x}$ as $\mathbf{C}_1, \cdots, \mathbf{C}_H$.

Defining $\mathbf{f} = (w_1 \ \ldots \ w_H)^T$ may seem to be the most natural choice to apply the implicit function theorem. Unfortunately, we cannot directly use the objective functions because their gradients are always zero for all parameters on the manifold due to their quadratic forms. Instead, we use the gradient $\mathbf{C}_i$ of the constraints as the implicit relationships, where the Jacobian of gradients (i.e. Hessian of W_i) must be equal to zero.

Because the change of the gradient function $\Delta\mathbf{C}_i$ should be zero, we can build a linear system of $\Delta\mathbf{x}$ as:

$$\Delta\mathbf{C}_i = \mathbf{D}_i \Delta\mathbf{x} = \mathbf{0}, \tag{13}$$

where $\mathbf{D}_i$ is the Jacobian of the gradient $[\delta\mathbf{C}_i/\delta\mathbf{x}]$. We apply finite difference to compute $\mathbf{D}_i$ because it is impossible or very difficult to analytically compute them. To improve the accuracy, however, we use analytical gradients for computing $\mathbf{C}_i$ with respect to some of the parameters as detailed in Section 4.6.1.

The constraints on linear actuator positions (Eq. (12)) are also converted to a linear system of $\Delta\mathbf{x}$ using the chain rule:

$$l_k^{lo} \leq l_{i,k}^0 + \mathbf{L}_{i,k}\Delta\mathbf{x} \leq l_k^{hi} \tag{14}$$

where $l_{i,k}^0$ is the initial actuator position $l_{i,k}(\mathbf{x}_0)$ and $\mathbf{L}_k$ is the Jacobian matrix $\delta l_{i,k}/\delta\mathbf{x}$.

By collecting all equality and inequality constraints, the linear system is formulated as follows:

$$\begin{aligned}\mathbf{A}_1\Delta\mathbf{x} &= \mathbf{b}_1\\ \mathbf{A}_2\Delta\mathbf{x} &\geq \mathbf{b}_2,\end{aligned} \tag{15}$$

where

$$\mathbf{A}_1 = \begin{bmatrix} \delta\mathbf{C}_1/\delta x_1 & \delta\mathbf{C}_1/\delta x_2 & \dots & \delta\mathbf{C}_1/\delta x_P \\ \vdots & \vdots & \ddots & \vdots \\ \delta\mathbf{C}_H/\delta x_1 & \delta\mathbf{C}_H/\delta x_2 & \dots & \delta\mathbf{C}_H/\delta x_P \end{bmatrix} \tag{16}$$

$$\mathbf{b}_1 = \begin{bmatrix} 0 & \cdots & 0 \end{bmatrix}^T \tag{17}$$

$$\mathbf{A}_2 = \begin{bmatrix} \mathbf{L}_{1,1} \\ -\mathbf{L}_{1,1} \\ \vdots \\ -\mathbf{L}_{N,K} \end{bmatrix} \tag{18}$$

$$\mathbf{b}_2 = \begin{bmatrix} l_1^{lo} - l_{1,1}^0 & -(l_1^{hi} - l_{1,1}^0) & \cdots & -(l_K^{hi} - l_{N,K}^0) \end{bmatrix}. \tag{19}$$

Note that the matrices of the formulated linear system are very sparse, because the motion parameters only affect the constraints on the corresponding frames.

The final component of the system is the user input specifying the desired changes of a subset of parameters $\Delta\bar{\mathbf{y}}$. We denote the remaining unknown parameters by $\Delta\mathbf{z}$, which we shall determine such that the constraints are maintained after $\Delta\bar{\mathbf{y}}$ is applied. Without losing generality, we can divide the equality and inequality constraints of Eq. (15) as

$$\begin{aligned}\mathbf{A}_{1y}\Delta\bar{\mathbf{y}} + \mathbf{A}_{1z}\Delta\mathbf{z} &= \mathbf{b}_1\\ \mathbf{A}_{2y}\Delta\bar{\mathbf{y}} + \mathbf{A}_{2z}\Delta\mathbf{z} &\geq \mathbf{b}_2.\end{aligned} \tag{20}$$

4.4 Optimization

Instead of using the pseudoinverse as in Eq. (4), we compute $\mathbf{z}$ by applying numerical optimization because 1) we also have inequality constraints, and 2) there may exist multiple solutions or there may be no exact solution, depending on the size of $\bar{\mathbf{y}}$. The optimization problem can be derived from Eq. (20) as

$$\begin{aligned}\min_{\Delta\mathbf{z}} \quad & \Delta\mathbf{z}^T\mathbf{S}\Delta\mathbf{z}\\ \text{s.t.} \quad & \mathbf{A}_{1z}\Delta\mathbf{z} = \mathbf{b}_1 - \mathbf{A}_{1y}\Delta\bar{\mathbf{y}}\\ & \mathbf{A}_{2z}\Delta\mathbf{z} \geq \mathbf{b}_2 - \mathbf{A}_{2y}\Delta\bar{\mathbf{y}},\end{aligned} \tag{21}$$

where $\mathbf{S}$ is a positive definite, diagonal weight matrix, where we use 0.01 for actuator forces $\boldsymbol{\tau}$, 10.0 for contact forces $\mathbf{f}$, and 1.0 for others.

Although this is a standard quadratic programming problem, it is numerically difficult to solve due to a large number of constraints. We therefore represent the feasible solution space with respect to the linear constraints using the null space of the coefficient matrix $\mathbf{A}_{1z}$:

$$\Delta\mathbf{z} = \Delta\mathbf{z}_0 + \mathbf{N}\mathbf{u} \tag{22}$$

where $\Delta\mathbf{z}_0 = \mathbf{A}_{1z}^+(\mathbf{b}_1 - \mathbf{A}_{1y}\bar{\mathbf{y}})$ and $\mathbf{N}$ represents the null space of $\mathbf{A}_{1z}$. Note that both $\mathbf{A}_{1z}^+$ and $\mathbf{N}$ can be efficiently computed from one singular value decomposition. Eq. (21) can now be reduced into

$$\begin{aligned}\min_{\mathbf{u}} \quad & (\Delta\mathbf{z}_0 + \mathbf{N}\mathbf{u})^T\mathbf{S}(\Delta\mathbf{z}_0 + \mathbf{N}\mathbf{u})\\ \text{s.t.} \quad & \mathbf{A}_{2z}(\Delta\mathbf{z}_0 + \mathbf{N}\mathbf{u}) \geq \mathbf{b}_2 - \mathbf{A}_{2y}\Delta\bar{\mathbf{y}}\end{aligned} \tag{23}$$

which can be solved much more efficiently than the original optimization problem. Once we obtain the solution $\mathbf{u}^*$, we can compute the solution of the original problem (Eq. (21)) $\Delta\mathbf{z}^*$ using Eq. (22).

The formulated optimization is solved using Sequential Quadratic Programming. In future, we consider to improve the performance of the solver by exploiting sparsity of the matrix $\mathbf{A}_1$ (Saad (2003)).

4.5 Summary

Algorithm 1 summarizes the optimization algorithm described in this section. It takes the initial parameters $\mathbf{x}_0$ as input, and finds the optimal parameters $\mathbf{x}^*$ that realizes the desired change and also maintain the design and motion on the constraint manifold. In Algorithm 1, $\mathcal{E}$ represents the sum of the objective functions:

$$\mathcal{E} = \sum_i^H w_i(\mathbf{x}). \tag{24}$$

A non-zero $\mathcal{E}$ implies violation of constraints due to numerical errors caused by linear approximation. At each iteration, we attempt to correct this error as described in Section 4.6.2. We run the optimization until it reaches the maximum iteration (1000 in our implementation), or the error $\mathcal{E}$ is greater than a threshold $\bar{\mathcal{E}}$ (10^{-6}).

Algorithm 1 Design Optimization Algorithm

Require: Initial design and motion parameters $\mathbf{x}_0$
1: $\mathbf{x} \leftarrow \mathbf{x}_0$.
2: **while** not reach the maximum iteration **do**
3: determine $\Delta\bar{\mathbf{y}}$ by selecting the desired changes.
4: formulate $\mathbf{A}_1$, $\mathbf{b}_1$ from the equality constraints.
5: formulate $\mathbf{A}_2$, $\mathbf{b}_2$ from the inequality constraints.
6: obtain $\Delta\mathbf{z}$ by solving Eq. (21).
7: $\mathbf{x} \leftarrow \mathbf{x} + (\Delta\bar{\mathbf{y}}^T\Delta\mathbf{z}^T)^T$.
8: correct numerical errors (Section 4.6.2)
9: calculate $\mathcal{E}$ using Eq. (24).
10: **if** $\mathcal{E} > \bar{\mathcal{E}}$ **then**
11: **break.**
12: **end if**
13: $\mathbf{x}^* \leftarrow \mathbf{x}$.
14: **end while**
15: **return** $\mathbf{x}^*$

In line 3 of Algorithm 1, we could also let the user interactively choose $\Delta\bar{\mathbf{y}}$ as demonstrated in Section 5.1. This version of the algorithm provides an intuitive interface for optimizing the design of a robot by specifying desired changes to any of the design or motion parameters. For example, the user could choose to reduce the maximum torque, or adjust the link length depending on the design requirements.

Table 1. The problem parameters.

Problem				# Parameters						# Expanded Nodes	
Robot	Actuation	# Actuators	# Frames	Lengths	Actuator Points	Joint Positions	Actuator Forces	Contact Forces	Plan Scale	# Total Parameters	Time per Iteration
Manipulator	Linear	4	7	4	8	28	28	0	0	68	0.8s
Four-bar Arm	Rotary	3	11	5	2	55	55	0	0	117	1.5s
Large Quadruped	Linear	16	13	8	16	208	208	156	0	596	40.2s
Small Quadruped	Rotary	14	13	5	N/A	182	182	120	0	489	32.1s
Tetrabot, Rolling	Rotary	16	8	3	N/A	176	128	81	3	391	4.3s
Tetrabot, Walking	Rotary	16	16	6	N/A	352	256	177	3	794	9.9s
Tetrabot, Both	Rotary	16	8	3	N/A	528	384	258	3	1176	15.1s

4.6 Implementation Note

4.6.1 Computation of Gradients It is desirable to calculate the analytical gradient vector $\mathbf{C}_i$ with respect to all parameters. Unfortunately, most of the constraints are nonlinear and it is difficult to analytically calculate their gradient with respect to some of the parameters, such as the partial derivative of the Coriolis forces with respect to the joint velocities. Instead, we only compute the analytical gradients that can be easily derived. In our experience, this compromise provides enough accuracy while keeping the computational cost reasonable.

For instance, the gradient of end-effector constraints with respect to the link lengths can be simply computed as:

$$\delta w^{EE}/\delta \mathbf{l} = \mathbf{J}_j^T(\mathbf{e}_j(\mathbf{l},\mathbf{q}) - \bar{\mathbf{e}}). \tag{25}$$

Similarly, the analytical gradients of the equation of motion constraint can be easily calculated for the joint forces and contact forces as:

$$\begin{aligned} \delta w^{EOM}/\delta \boldsymbol{\tau} &= \mathbf{R}^T(\mathbf{M}\ddot{\mathbf{q}}_i + \mathbf{c} - \mathbf{R}\boldsymbol{\tau}_i - \sum_j^M \mathbf{J}_j^T \mathbf{f}_{i,j}) \\ \delta w^{EOM}/\delta \mathbf{f} &= \\ & [\mathbf{J}_1 \cdots \mathbf{J}_M]^T(\mathbf{M}\ddot{\mathbf{q}}_i + \mathbf{c} - \mathbf{R}\boldsymbol{\tau}_i - \sum_j^M \mathbf{J}_j^T \mathbf{f}_{i,j}). \end{aligned} \tag{26}$$

4.6.2 Correction of Numerical Errors Because we apply linear approximation to the nonlinear constraints, numerical errors will accumulate and eventually lead to physically invalid solutions. Our framework prevents this numerical deviation by re-optimizing only the motion parameter for the equality constraints.

First, we optimize the joint angles considering only the end-effector constraints (Eq. (9)):

$$\mathbf{q}_i = \underset{\mathbf{q}_i}{\operatorname{argmin}}\, w^{EE}. \tag{27}$$

We then optimize the actuator forces and contact forces to satisfy the equation of motion (Eq. (5)):

$$\boldsymbol{\tau}_i, \mathbf{f}_{i,1}, \cdots, \mathbf{f}_{i,M} = \underset{\boldsymbol{\tau}_i, \mathbf{f}_{i,1}, \cdots, \mathbf{f}_{i,M}}{\operatorname{argmin}} w^{EOM}. \tag{28}$$

Note that the design parameters are not changed in the error correction process.

5 Experiments

In this section, we discuss the simulation and hardware experiments for validating the proposed design optimization algorithm. We tested three sets of examples: simulation of linearly actuated manipulators, simulation of linearly actuated quadrupeds, and hardware of quadrupeds with off-the-shelf rotary servos. The algorithm is implemented in Python with the PyDART (PyDART) and SciPy library (Jones et al. (2001)) on Ubuntu Linux, and the computations are conducted on a single core of 3.40GHz Intel i7 processor. The parameters for all problems are described in Table 1.

Table 2. The results on manipulators with linear actuators.

	Link Lengths(cm)				Actuator Positions (cm)		Maximum Force (N)
Design	Link1	Link2	Link3	Link4	Link2	Link3	Manipulation
Initial	20.0	30.0	30.0	20.0	15.0	5.0	5.89×10^2
Motion Optimized	20.0	30.0	30.0	20.0	15.0	5.0	4.87×10^2
Design Optimized	19.7	24.4	29.5	24.9	14.5	8.3	$\mathbf{3.83\times10^2}$
Length Edited	**24.0**	24.4	29.1	24.9	10.8	8.3	4.80×10^2

5.1 Simulation of Manipulators with Linear Actuators

We first show a proof of concept of our algorithm by optimizing the design of a linearly actuated manipulator. First, we provide an initial manipulator design with link lengths of 20 cm, 30 cm, 30 cm, and 20 cm. The manipulator has four degrees of freedom fully actuated by four actuators, each of which with 25 cm body length and 16 cm stroke. The actuators are monoarticular with 5 cm moment arms except the second biarticular actuator with a 15cm moment arm. The attachment points of actuators can be adjusted vertically, except the points on the base link that can adjusted horizontally. The input task is to move a 5 kg-object along the target trajectory that has the 60cm maximum distance from the base link. Note that the initial design is generated based on simple rules rather than pre-tuned for any performance index.

In this example, we demonstrate two editing modes: force-driven mode for reducing the maximum actuator force, and length-driven mode for adjusting the length of the target link.

We begin with the force-driven editing mode, which provides an intuitive interface for editing the design such that the maximum actuator force is reduced. At each iteration, our algorithm selects the maximum force parameter during the entire motion, and set the desired change as -1 % of its value. Using our algorithm, we are able to find the optimal design that reduces the maximum actuator force from 589 N to 383 N, which happens at the third actuator. The parameters of the resulting design are shown in Table 2, and the visual comparison is provided in Figure 5. In general, the algorithm increases the moment arm for the third actuator by changing its attachment point, and adjusts the second and fourth link lengths to secure the necessary range of motion. Further increasing the moment arm of the third actuator cannot

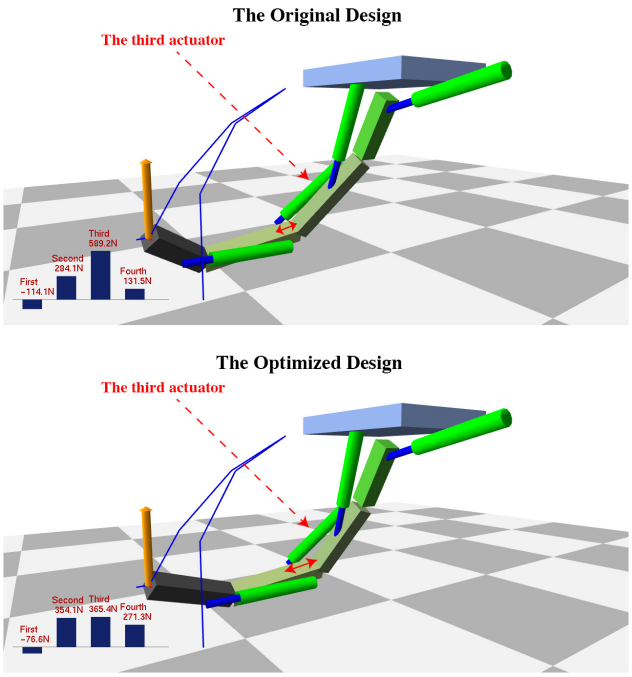

Figure 5. Comparison of the original (top) and optimized (bottom) manipulators. Note the larger moment arm for the third actuator in the optimized manipulator.

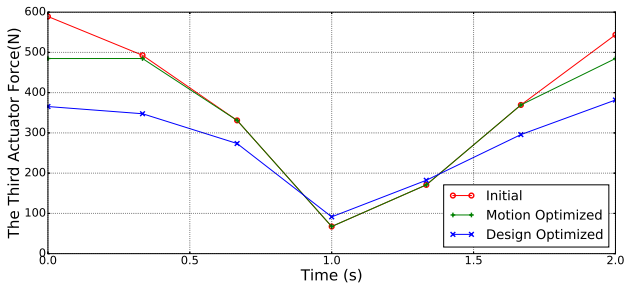

Figure 6. Force profiles of the third actuator in three designs: the initial design and motion (red), the initial design with optimized motion (green), and the concurrently optimized design and motion (blue). The concurrent optimization of the design and motion parameters results in the most efficient actuator force profile.

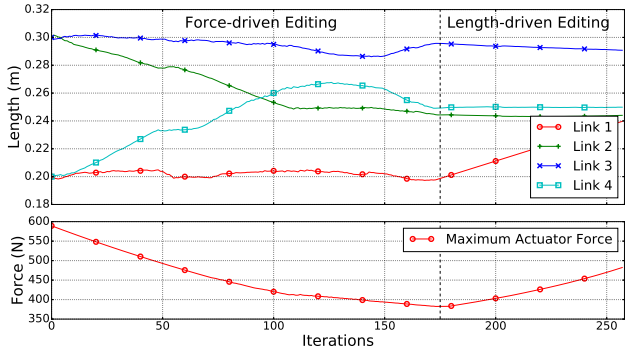

Figure 7. Changes in link lengths and the maximum actuator force the third actuator during the manipulator design optimization process. The first half was edited in the force-driven mode for minimizing the maximum torque, while the second half was edited in the length-driven mode for increasing the length of the first link.

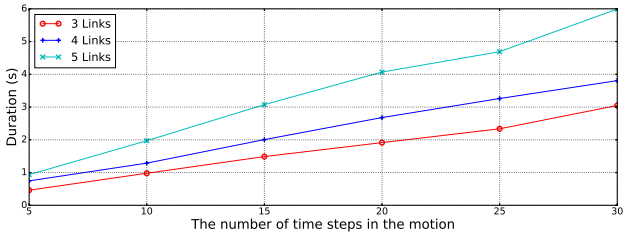

Figure 8. Average time duration for computing a single iteration of Algorithm 1 with various numbers of links and discrete time steps.

Table 3. The results on manipulators with four-bar linkages.

	Link Lengths(cm)			Four bar Lengths (cm)		Maximum Torque (Nm)
Design	**Link1**	**Link2**	**Link3**	**Bar1**	**Bar2**	**Manipulation**
Initial	35.0	30.0	15.0	20.0	20.0	9.81
Motion Optimized	35.0	30.0	15.0	20.0	20.0	9.74
Design Optimized	50.0	**12.4**	17.2	**8.56**	**43.1**	5.59

maintain the range to perform the entire desired motion, which triggers the break condition in line 10 of Algorithm 1.

In our experience, the optimization is not very sensitive to the initial parameters. Optimization trials with different initial layout parameters converge to solutions similar to the one shown in the bottom of Figure 5.

Our algorithm also allows optimization of the motion parameters only, which gives another pair of design and motion where the design is the same as the initial one but the force profile is different. Figure 6 compares the torque profile of the third actuator for the three pairs of design and motion. Although optimizing only the motion can also reduce the maximum torque, its final result (487 N) is worse than the result of concurrent design and motion optimization (382 N).

Let us consider the scenario where the user is not satisfied with the proportion of the optimized design and is willing to sacrifice the maximum force to increase the length of the first link. The user can then start the length-driven editing mode and set the desired change of the first link length as $+1$ mm. Given the new input, the algorithm successfully increases the link length while properly adjusting other design and motion parameters as shown in the second half of Figure 7. The maximum achievable length is 24 cm, beyond which the range of motion becomes too small to perform the desired motion. Our algorithm automatically relocates the bottom attachment point of the second actuator closer to the joint in order to obtain the required range of motion. Please refer to Extension 1 for more details.

To demonstrate the scalability of the algorithm, we plot the timing data for computing a single iteration of the algorithm while varying the number of links in the manipulator (design parameters) and the number of time steps in the motion (motion parameters) in Figure 8. The result indicates that the speed of the optimization is slightly more dependent on the number of design variables while showing near-linear relationship to the number of motion parameters.

5.2 *Simulation of Manipulators with Four-bar Linkages*

A four-bar linkage is the simplest closed chain mechanism. It consists of four rigid links that are connected in a closed

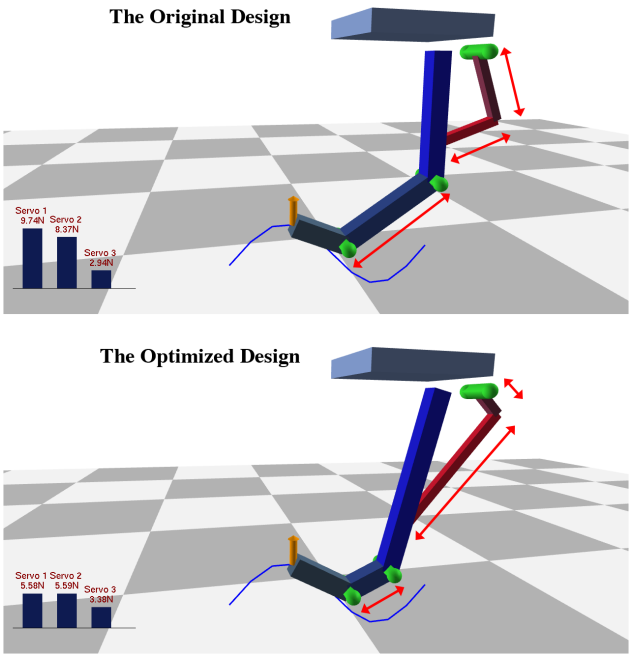

Figure 9. Comparison of the original (top) and optimized (bottom) linkage manipulators. Note that we differentiate the active and passive joints by rendering the servos as green cylinders.

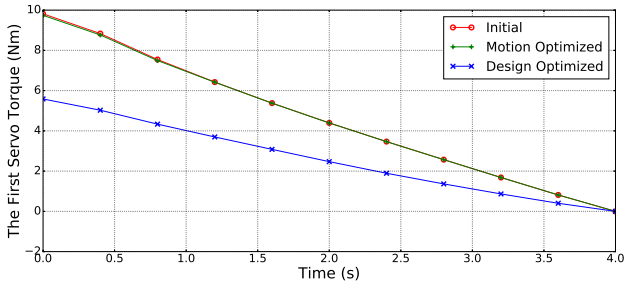

Figure 10. Torque profiles of the first servo in three designs: the initial design and motion (red), the initial design with optimized motion (green), and the concurrently optimized design and motion (blue).

loop by four hinge joints. This simple one degrees of freedom (DoF) mechanism is a great tool for generating complex end-effector trajectories and adjusting torque ratios between input and output joints. However, it is hard to find the proper link lengths due to non-linear relationships between design and motion parameters. In this section, we will demonstrate that the proposed framework can be used for editing robotic systems with four-bar linkages.

As a proof of concept, we employ a planar three DoFs robotic arm with a four-bar linkage as the target platform. We manually provide the initial design parameters of the robotic arm and let the framework to co-optimize the design and motion. The base link and end-effector are connected by three links with 35 cm, 30 cm, and 20 cm lengths. In addition, we create the four bar mechanism at the shoulder joint by attaching two 20 cm links to the base link and the upper arm link. Therefore, the total number of design parameters for this robot arm is seven: five link length parameters, the position of the passive joint on the first link, and the position

Table 4. The results on large quadrupeds with linear actuators.

	Link Lengths(cm)				Actuator Positions (cm)		Maximum Force (N)	
Design	Rear Hip	Rear Thigh	Rear Shin	Rear Foot	Rear Knee	Rear Ankle	Slow Walking	Fast Walking
Initial	15.0	30.0	30.0	20.0	4.0	4.0	1.30×10^3	2.15×10^3
Slow Walking	10.0	35.2	31.9	16.1	9.2	9.0	$\mathbf{0.68\times10^3}$	N/A
Fast Walking	16.1	31.2	32.2	17.1	6.1	3.3	0.89×10^3	$\mathbf{1.29\times10^3}$

of the actuated servo on the base link. The input task is to move the 1 kg-object 60 cm horizontally with the 10 cm sinusoidal height.

The objective of the optimization is to minimize the maximum servo torque, which can potentially allow the robot to execute difficult tasks that are previously infeasible due to torque limits. Similar to the previous experiment on manipulators with linear actuators, we generate two pairs of design and motion by optimizing motion parameters only (*motion optimization mode*) and co-optimizing design and motion parameters (*design optimization mode*).

The resulting design changes are illustrated in Table 3 and Figure 9. In our experiment, the *design optimization mode* successfully reduces the maximum servo torque from 9.81 N to 5.59 N that corresponds to 44 % reduction (Figure 10). Especially, the algorithm increases the joint velocity of the first servo by decreasing the length of the first link (dark red in Figure 9) in the four-bar mechanism. In addition, it also reduces the moment arm of the second servo by shortening the distances between the end-effector and the servo. However, the *motion optimization mode* was not very effective due to limited degrees of freedom, which results in less than 1 % torque reduction. As demonstrated in the results, co-optimization of design and motion achieves much better performances than motion optimization. For more details, please refer to Extension 2.

5.3 Simulation of Large Quadrupeds with Linear Actuators

This set of examples involves a large-size quadruped robot with linear actuators. The initial design of the quadruped has 1.2 m body length and 0.95 m shoulder height at its rest pose with a mass of approximately 90 kg. Each leg has four degrees of freedom digitigrade configuration with four linear actuators. Two biarticular actuators are attached to the hip joints, and two monoarticular actuators are attached to the knee and ankle joints. All the actuators have the same length (25 cm) and stroke range (16 cm) as the actuators for the manipulator example. The design parameters are constrained to be left-right symmetric during the optimization process. We choose a manually-created initial design and generate the input base link and end-effector trajectories by solving space-time optimization using the technique described in Megaro et al. (2015). We treat the motion of the floating base as input.

The goal of this experiment is to optimize the initial design and motion parameters for two different tasks: slow walking with 0.15 m step length and fast walking with 1.0 m step length. In both cases, we optimize the design and motion parameters such that the maximum actuator force is reduced.

Our algorithm successfully derives different designs that reduce the maximum torques by 48 % and 40 % for the

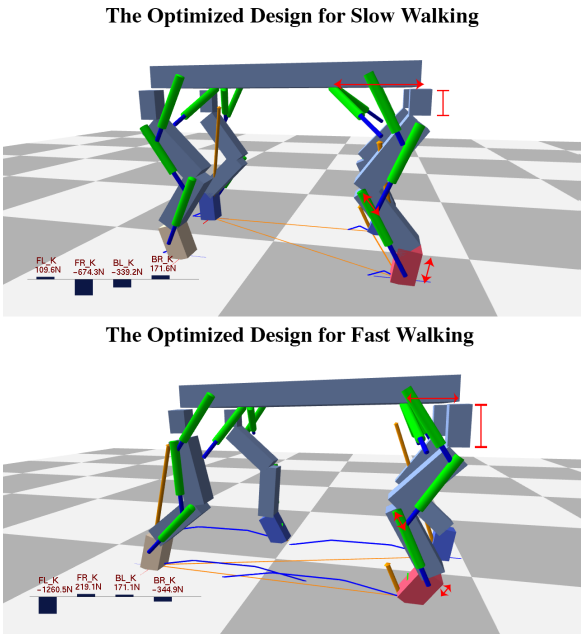

Figure 11. Comparison of the optimized linearly actuated quadrupeds for slow (top) and fast (bottom) walking tasks. We highlighted a few noticeable differences. In general, the design for slow walking has larger moment arms than the design for fast walking.

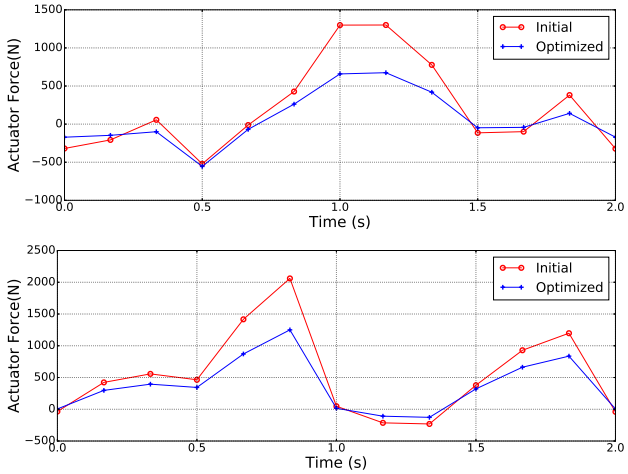

Figure 12. Force profiles of the back right knee in the slow walking optimized (top) and fast walking optimized (bottom) designs.

slow and fast walking tasks respectively (Table 4). Figure 11 compares the optimized design and motion while Figure 12 plots the force profile of the knee actuator that requires the maximum force in the original design. Please refer to Extension 3 for the details of the metamorphosis process. The most notable difference of the two optimized designs is in their knees: the moment arm is increased in the slow walking quadruped while it is kept small in the fast walking optimized design in order to realize larger range of motion. In addition, the algorithm elongated the rear thigh for the slow walking task, while increasing the length of the rear hip for fast walking. For the slow walking task, we have expected straight-knee configuration similar to those of elephants or rhinos, but the algorithm did not result in such design because the enlargement of moment arm has larger effect than the reduction of the joint torque.

Table 5. The results on small quadrupeds with rotary actuators.

	Link Lengths(cm)					Maximum Torque	
Design	**Front Thigh**	**Front Shin**	**Rear Thigh**	**Rear Shin**	**Rear Feet**	**Simulation Torque (Nm)**	**Hardware Current (mA)**
Initial	8.00	16.0	8.00	8.00	8.00	2.08	$(7.66\pm0.37)\times10^2$
Optimized	7.87	12.9	9.43	7.28	8.00	**1.42**	$(5.35\pm0.23)\times10^2$

We also cross-validate the optimal designs by re-optimizing the motion parameters with fixed design parameters. As we expected, the fast walking optimized requires 31 % more maximum actuator forces than the slow walking optimized design due to larger moment arms at knees. On the other hand, the slow walking optimized design cannot execute the fast walking task due to its limited range of the motion.

5.4 *Hardware of Small Quadrupeds with Rotary Actuators*

We also validate our algorithm in hardware by implementing a small-size quadruped with off-the-shelf rotary actuators and 3D printed links. The robot has 20 cm body length and 24 cm shoulder height at its rest pose. Each leg has four Dynamixel XM-430 W-210 (Dynamixel (2016)) servos that have 3.7 Nm stall torque at 14.0 V, and each foot is fabricated as rubber-coated hemisphere. For testing purpose, we put four 500 g weights on the top of the base link. We tether the robot to an external computer and replay the motion plan without any balance controller.

We manually choose an initial design of the quadruped and optimize the design such that the maximum joint torque is minimized. Once again, the initial motion plan is generated using the method described in Megaro et al. (2015). Table 5 compares the initial and optimized parameters. Because the initial design requires a maximum torque of 2.08 Nm at its front knees, the algorithm reduces the lengths of the front leg links to create near singular configurations (Extension 4). On the other hand, the algorithm applies minor changes to the rear leg because their torque profiles are predicted to be lower than the front legs. As a result, the algorithm successfully reduces the maximum torque to 1.42 Nm, which is 32% less than the original value (Figure 13).

For hardware validation, we 3D-print new links with the optimized lengths and assembled a new quadruped (Figure 14, Extension 4). After fabrication, we replay the new motion plan that is also concurrently optimized with the design parameters. As the actuators do not have torque sensors, we use the maximum current as the performance criteria. Figure 13 shows the current profiles of two designs at the front right knee for a single cycle of the motion. We choose the cycle with a median peak value among five trials. The maximum current is successfully reduced to 535 mA in the optimized design, which is 30 % less than the original maximum current (766 mA) and roughly matches the reduction ratio in simulation. The predicted torque and actual current profiles do not exactly match but show a large positive Pearson's correlation coefficient of 0.7.

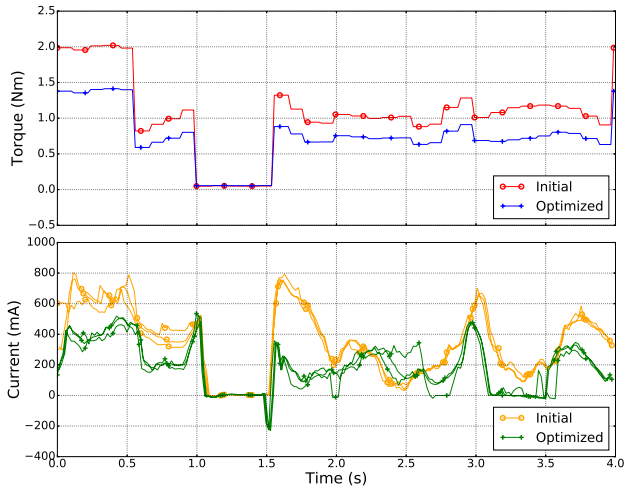

Figure 13. Comparison of the front right knee torques in simulation (top) and on fabricated hardware (bottom).

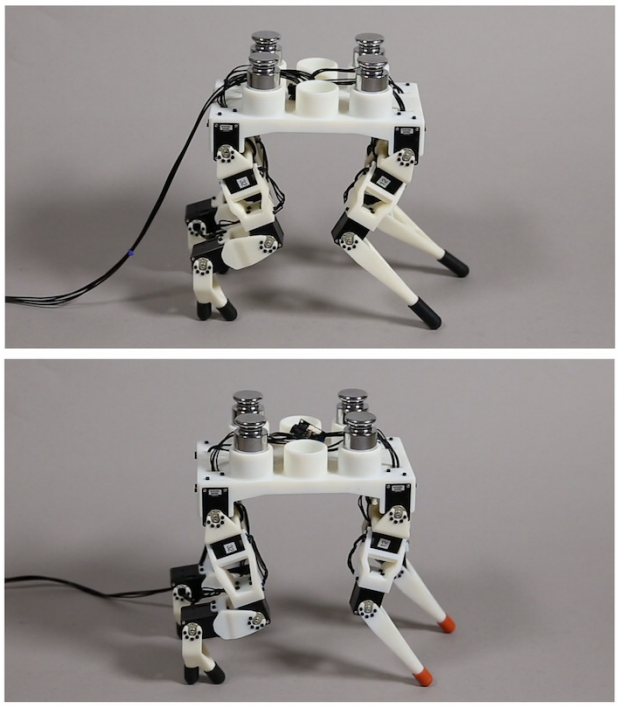

Figure 14. Comparison of the original (top) and optimized (bottom) small quadrupeds with rotary actuators.

5.5 Simulation of Spherically-Symmetric Quadruped Robots

Inspired by the work of Pai et al. (1995), we select the next testing platform as a spherically-symmetric quadruped robot. It is called *Tetrabot* because its shape resembles a tetrahedron (Figure 15). The robot consists of a spherical body and four legs where each leg has a single yaw joint followed by three pitch joints. The radius of the base link is 7 cm and the initial link lengths of legs are set to 6 cm, 12.5 cm, and 19 cm, where these parameters are manually selected by engineers based on prior knowledge. Since all four legs should be exactly the same to maintain the spherically symmetric morphology of Tetrabot, we cannot distinguish the legs within its morphology.

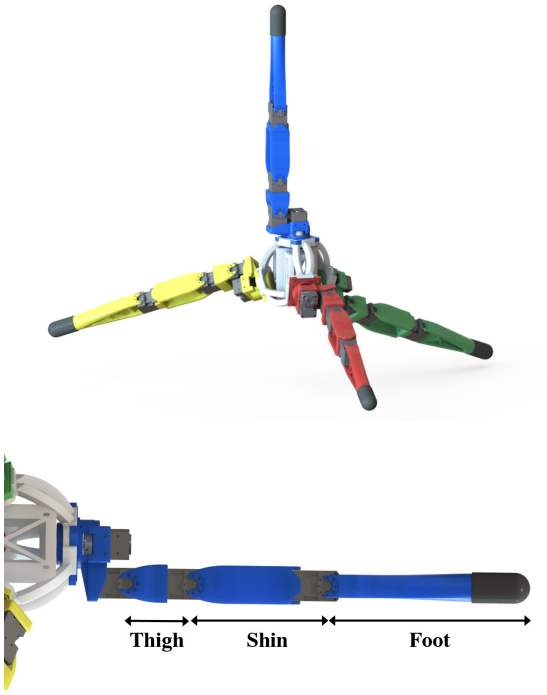

Figure 15. A spherically symmetric quadruped robot, Tetrabot. **(Top)** The zero pose of Tetrabot is similar to a tetrahedron. **(Bottom)** Each leg has four DoFs including one yaw joint and three pitch joints.

In this section, we are interested in optimizing the design parameters of Tetrabot for two motions: the *rolling* gait and the *walking* gait.

The first target motion is the *rolling* gait that walks as tumble, which is also originally introduced in the work of Pai et al. (1995) (The first and second rows of Figure 16). In this motion, Tetrabot begins the gait by gradually pushing its body toward the edge of the support polygon while the "top" leg prepares the landing. When the "top" leg touches the ground, the robot moves its COM to the adjacent equilateral triangle and lifts the "back" legs until it gets back to the initial pose. The advantage of this *rolling* gait is that the robot can easily choose one of the three possible walking directions by rotating the "top" leg. To generate the initial motion plan, we first create the end-effector trajectories from the rolling motion of a tetrahedron. Then we solve inverse kinematics (IK) to match the end-effector positions, while restricting its center of mass position in the corresponding support polygon.

In addition, we design the *walking* gait, which is more similar to regular quadruped locomotion (The third and fourth rows of Figure 16). In this motion, Tetrabot sequentially moves its back, right, left, and front legs while supporting its body with other three legs. The step length of the gait is set to 8 cm with the duration of 4.0 seconds. Once again, the end-effector trajectories and footfall patterns are manually designed by engineers and the joint trajectories are generated by solving IK problems.

The number of design parameters is three if we constrain the design to be spherically symmetric. However, all four legs of the robot do not need to be same for the *walking* gait. Therefore, we select six design parameters for when we optimized the design for the *walking*. The first three are

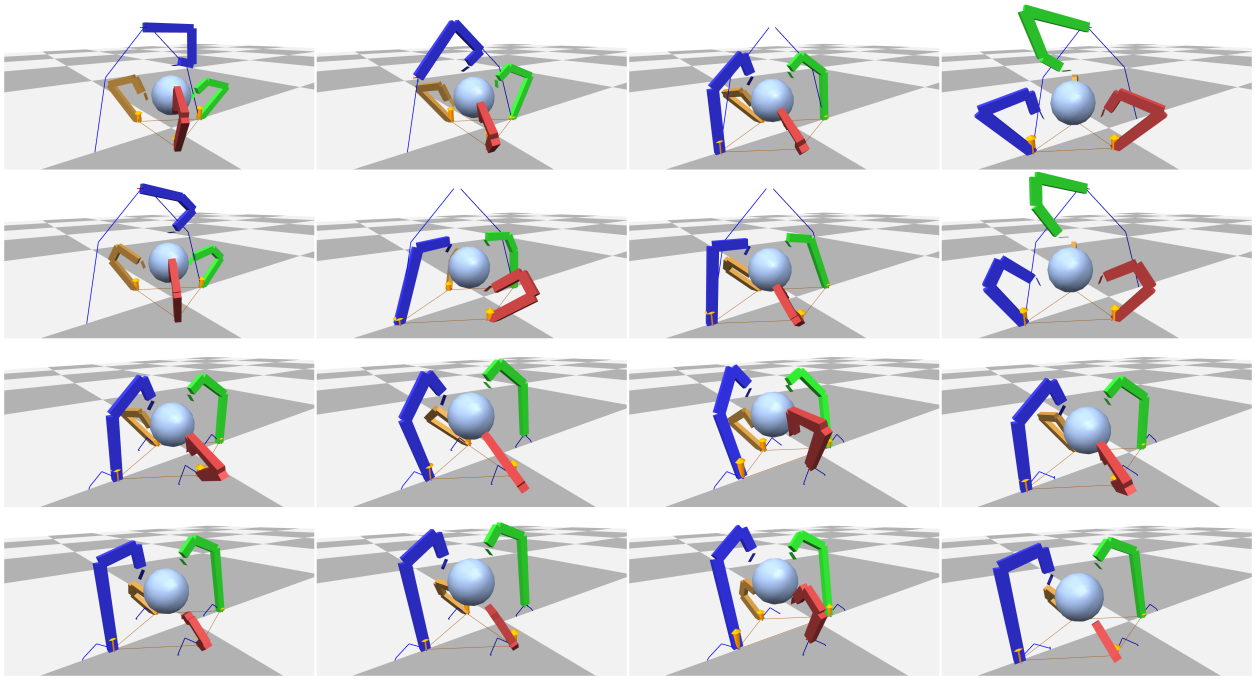

Figure 16. The optimized designs and motions of Tetrabot. For the *rolling* gait, the optimized design (the second row) has shorter link lengths than the initial design (the first row) to avoid large torques at the red and yellow legs. For the *walking* gait, the optimized design (the fourth row) has the shorter red and yellow legs but has the longer blue and green legs than the original design (the third row).

Table 6. The results on Tetrabot for the *rolling* gait.

Rolling	**Link Lengths(cm)**			**Maximum**
Design	**Thigh**	**Shin**	**Foot**	**Torque (Nm)**
Initial	6.00	12.5	19.0	2.30
Motion Optimized	6.00	12.5	19.0	1.88
Design Optimized For Rolling	9.70	7.09	15.1	1.21
Design Optimized For Both	7.15	10.2	16.6	1.58

Table 7. The results on Tetrabot for the *walking* gait.

Walking	**Link Lengths(cm)**						**Maximum**
	Front & Back			**Left & Right**			**Torque (Nm)**
Design	**Thigh**	**Shin**	**Foot**	**Thigh**	**Shin**	**Foot**	
Initial	6.00	12.5	19.0	same			2.49
Motion Optimized	6.00	12.5	19.0	same			1.55
Design Optimized For Walking	5.90	10.4	25.0	5.99	6.49	11.7	0.96
Design Optimize For Both	7.15	10.2	16.6	same			1.58

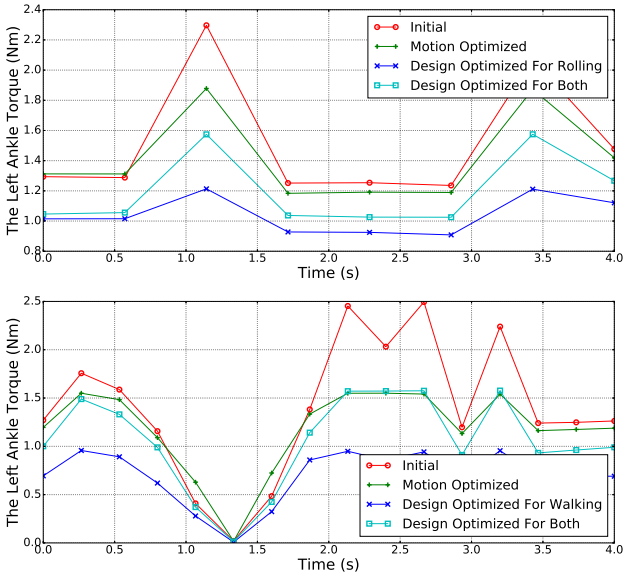

Figure 17. Torque profiles of Tetrabot for the *rolling* gait (Top) and *walking* gait (Bottom).

for the front and back legs, and the latter three are for the left and right legs. We do not optimize the radius of the base link because it has the specialized internal structure to equip a on-board computer and a battery. Finally, we allow the optimization to change the scale of the motion plan by introducing the motion plan scale parameter κ (Section 4.2.1).

One of the biggest differences between the implementation of Pai et al. (1995) and ours is the constraints on joint angles. Because our robot is larger and heavier, we need more supporting structure that limits the range of joint angles. Therefore, we set the limits for hip, knee, and ankle joint position variables as 105°, 120°, 130°, which have been also considered in the optimization.

We summarize the results of the design optimization in Table 6 and Table 7. For the *rolling* gait, the optimal design requires only a maximum torque of 1.21 Nm, which is much less than 2.30 Nm of the initial parameters and 1.88 Nm of the motion-optimized scenario (Top graph in Figure 17). In our observation, the optimization decreases the torques at the left and right legs by shortening the shin and foot while increasing the thigh to secure the desired range of the motion. In the case of the *walking* gait, the *design optimization mode* also reduces the maximum servo torque by 61.4 %, which is much larger than 37.8 % of the *motion optimization mode*

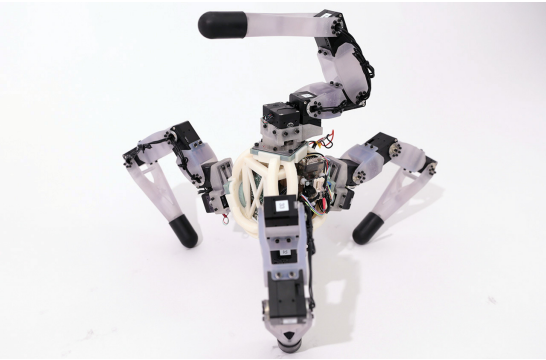

Figure 18. Physical prototype of Tetrabot.

(Bottom graph in Figure 17). The resulting design has much shorter left and right legs because they are heavily loaded, while it has longer lengths for the front and back legs. Also note that the algorithm automatically changes the update direction of the design parameters a few times when some parameters are near limits. Another interesting observation is that all the optimization tends to increase the scale of the plan from 5 % to 8 %, which results in one to two centimeter larger support polygons. Please refer to Extension 5 and Figure 16 for more details of the optimized designs and motions.

In addition, we optimize the design of Tetrabot for both *rolling* and *walking* gaits by creating a new motion plan by concatenating the two motions. We assume that all four legs have the same link length parameters because the motion includes the *rolling* gait. The algorithm finds the optimal design that decreases the maximum torque from 2.49 Nm to 1.58 Nm, which is 36.5 % reduction. Naturally, the optimized maximum torque is larger than the specialized designs for each individual task because the robot needs to execute multiple tasks. For the *walking* task, it is even larger than that of the motion optimization. This is because this multi-gait optimization reaches the lower bound for the *rolling* gait first and does not continue to optimize the maximum torque for the *walking* gait (Figure 17).

5.6 Hardware of Spherically-Symmetric Quadruped Robots

We demonstrate the validity of the optimized design and motion in Section 5.5 by fabricating a physical prototype using 3D printing and off-the-shelf servos. We select the length parameters that are optimized for both *rolling* and *walking* gaits: 7.15 cm , 10.2 cm, and 16.6 cm for thigh, shin, and foot links, respectively. We print the body frame and legs using Stratasys Objet260 Connex (Stratasys) with VeroWhitePlus and VeroClearPlus materials. An Intel UpBoard 1.92 GHz controller (Intel) and a 16.8 V, 4000 mA lithium-ion battery are enclosed in the body. Each leg is actuated by four Dynamixel XM-430 W-350 servos whose stall torque is 4.5 Nm at 14.0 V. The controller software is written in Python using the official Dynamixel SDK (Dynamixel (2016)) and runs on Ubuntu 16.04.

In our experiments, the fabricated hardware can successfully execute the given motion plans even though there exist some errors due to many reasons, such as differences between simulation and real hardware or lack of the feed forward term in PD controller. For the *rolling* motion, the robot safely lands with its top leg without making huge impacts. The robot slips slighly during the *walking* motion, resulting in a decreased step length of 6.7 cm compared to the theoretical value of 8.0 cm. We believe that the accuracy of the motions can be improved by applying system identification or model learning (Khalil and Dombre (2004); Abbeel et al. (2006)).

6 Discussion and Future Work

We presented a novel algorithm for optimizing robot design parameters, such as link lengths and actuator layouts, and the associated motion parameters including joint positions, actuator forces, and contact forces at every frame. To guide the optimization, our algorithm first linearizes the local manifold of valid designs implicitly defined by a set of constraints. It then changes the design parameters in the direction of the locally defined gradient. Our problem formulation can also take additional inequality constraints, such as position limits of linear actuators or friction cones of contact forces. We demonstrated that our algorithm is general enough to optimize designs of various types of robots such as manipulators and legged robots, with both linear and rotary actuators. We also validated the optimized designs in simulation as well as on fabricated hardware using off-the-shelf actuators and 3D-printed links.

There are a few possible directions for future work.

The presented design optimization algorithm selects one target parameter and specifies its change, and determines the changes in other parameters required to stay on the constraint manifold. We implemented this simple update rule because it only needs to select a single parameter for each time step. Although we demonstrated that the proposed scheme is sufficient in various optimization scenarios, it may require many iterations to reach the desired design or even fall into sub-optimal solutions. Therefore, it would be interesting to incorporate more complex optimization algorithms into our framework for selecting the target parameters and their desired changes.

Our algorithm can only consider continuous parameters, such as link lengths or actuator forces. For future research, we would like to include discrete variables, such as the number of actuators, footfall patterns, or the number of joints actuated by a single actuator (e.g. monoarticular or biarticular), because they are also critical to robot performance. However, current implementation of the framework cannot handle discrete variables because it requires the first and second derivatives of the constraints with respect to the parameters. For instance, the moment arm of a linear actuator discontinuously changes when the actuator switches between monoarticular and biarticular configurations.

Although we demonstrated one example of motion planning formulation, our approach is agnostic to specific instances of parameterization and problem formulation. As long as we can compute gradients of constraints with respect to variables, our idea of co-optimizing design and motion parameters using the implicit function theorem can be applied to the given scenario. For instance, we can easily adopt a new parameterization method such as (Wampler and

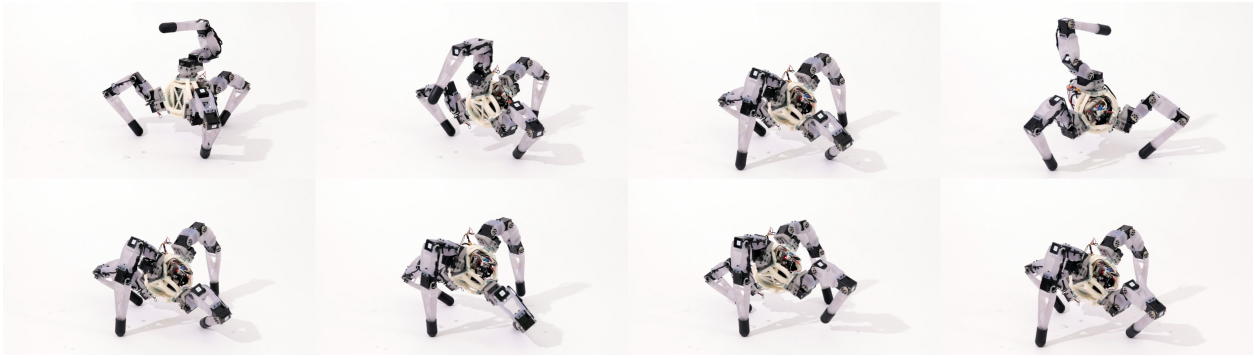

Figure 19. We fabricated the prototype of Tetrabot, which is optimized for both the *rolling* (Top) and *walking* (Bottom) gaits.

Popović (2009)), or add a new constraint such as collision avoidance.

In our implementation, we only used box-shaped links due to their simplicity for parameterization. One interesting direction of future work is to include additional design parameters for editing link shapes. By incorporating the existing shape optimization techniques for adjusting inertial properties (Prévost et al. (2013)) or structural strength (Musialski et al. (2016)), our algorithm will be able to generate more energy efficient and robust robot designs.

In all examples, our input motion plans are quite simple consisting of only a single instance of manipulation or locomotion, resulting in designs specialized for the given motion plan. Although we did a preliminary experiment on two motion plans, it would be also interesting to optimize the design for a family of parameterized motion plans, such as walking with continuous turning angles. We could also consider tasks that require more complex interactions with the environment, such as climbing up stairs.

Appendix A: Index to Multimedia Extensions

Table 8. Index to Multimedia Extensions

Extension	Media Type	Description
1	Video	Results on Manipulators with Linear Actuators
2	Video	Results on Manipulators with Four-bar Linkages
3	Video	Results on Large Quadrupeds with Linear Actuators
4	Video	Results on Small Quadrupeds with Rotary Actuators
5	Video	Results on Spherically-Symmetric Quadruped Robot

Acknowledgements

We would like to show our gratitude to Kevin Gim at Disney Research for his help on hardware experiments.

References

Atlas. URL http://www.bostondynamics.com/robot_Atlas.html.

HydroLek. URL http://www.hydro-lek.com/.

Spot, a. URL https://youtu.be/M8YjvHYbZ9w.

SpotMini, b. URL https://youtu.be/tf7IEVTDjng.

Pieter Abbeel, Morgan Quigley, and Andrew Y Ng. Using inaccurate models in reinforcement learning. In *Proceedings of the 23rd international conference on Machine learning*, pages 1–8. ACM, 2006.

Cenk Baykal and Ron Alterovitz. Asymptotically optimal design of piecewise cylindrical robots using motion planning. In *Robotics: Science and Systems*, 2017.

Marco Ceccarelli and Chiara Lanni. A multi-objective optimum design of general 3R manipulators for prescribed workspace limits. *Mechanism and Machine Theory*, 39(2):119–132, 2004.

Nick Cheney, Robert MacCurdy, Jeff Clune, and Hod Lipson. Unshackling Evolution: Evolving Soft Robots with Multiple Materials and a Powerful Generative Encoding. *Proceeding of the Fifteenth Annual Conference on Genetic and Evolutionary Computation*, 2013.

Jean-François Collard, P. Fisette, and P. Duysinx. Contribution to the Optimization of Closed-Loop Multibody Systems: Application to Parallel Manipulators. *Multibody System Dynamics*, 13(1):69–84, 2005.

Alessandro Crespi, Konstantinos Karakasiliotis, Andre Guignard, and Auke Jan Ijspeert. Salamandra Robotica II: An amphibious robot to study salamander-like swimming and walking gaits. *IEEE Transactions on Robotics*, 29(2):308–320, 2013.

Dynamixel. 2016. URL http://robotis.com.

Thomas Geijtenbeek, Michiel van de Panne, and a. Frank van der Stappen. Flexible muscle-based locomotion for bipedal creatures. *ACM Transactions on Graphics*, 2013.

Knut Graichen, Sebastian Hentzelt, Alexander Hildebrandt, Nadine Kärcher, Nina GaiBert, and Elias Knubben. Control design for a bionic kangaroo. *Control Engineering Practice*, 2015.

Sehoon Ha, Stelian Coros, Alexander Alspach, Joohyung Kim, and Katsu Yamane. Task-based Limb Optimization for Legged Robots. *International Conference on Intelligent Robots and Systems*, 2016.

Intel. *UpBoard*. URL http://www.up-board.org.

K Jittorntrum. An implicit function theorem. *Journal of Optimization Theory and Applications*, 1978.

Eric Jones, Travis Oliphant, and Pearu Peterson. SciPy: Open source scientific tools for Python, 2001. URL http://www.scipy.org/.

Chang Doo Jung, Won Jee Chung, Jin Su Ahn, Myung Sik Kim, Gi Soo Shin, and Soon Jea Kwon. Optimal mechanism design of in-pipe cleaning robot. *IEEE International Conference on Mechatronics and Automation*, pages 1327–1332, 2011.

Wisama Khalil and Etienne Dombre. *Modeling, identification and control of robots*. Butterworth-Heinemann, 2004.

Dongmok Kim, Heeseung Hong, Hwa Soo Kim, and Jongwon Kim. Optimal design and kinetic analysis of a stair-climbing mobile robot with rocker-bogie mechanism. *Mechanism and Machine*

Theory, 50:90–108, 2012.

Sung Gaun Kim and Jeha Ryu. New dimensionally homogeneous Jacobian matrix formulation by three end-effector points for optimal design of parallel manipulators. *IEEE Transactions on Robotics and Automation*, (4), 2003.

Daniel Kuehn, Felix Bernhard, Armin Burchardt, Moritz Schilling, Tobias Stark, Martin Zenzes, and Frank Kirchner. Distributed computation in a quadrupedal robotic system. *International Journal of Advanced Robotic Systems*, 11(1), 2014. ISSN 17298814. doi: 10.5772/58733.

Chris Leger. Automated Synthesis and Optimization of Robot Configurations : An Evolutionary Approach. *Design Engineering*, pages 1–234, 1999.

Jian Li, Sheldon Andrews, Krisztian G Birkas, and Paul G Kry. Task-based design of cable-driven articulated mechanisms. In *Proceedings of the 1st Annual ACM Symposium on Computational Fabrication*, page 6. ACM, 2017.

H. Lipson and J.B. Pollack. Towards continuously reconfigurable self-designing robotics. *IEEE International Conference on Robotics and Automation. Symposia Proceedings*, 2(April): 1761–1766, 2000.

Vittorio Megaro, Bernhard Thomaszewski, Maurizio Nitti, Otmar Hilliges, Markus Gross, and Stelian Coros. Interactive Design of 3D-Printable Robotic Creatures. *ACM Transactions on Graphics*, 2015.

Vittorio Megaro, Espen Knoop, Andrew Spielberg, David IW Levin, Wojciech Matusik, Markus Gross, Bernhard Thomaszewski, and Moritz Bächer. Designing cable-driven actuation networks for kinematic chains and trees. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, page 15. ACM, 2017.

Przemyslaw Musialski, Christian Hafner, Florian Rist, Michael Birsak, Michael Wimmer, and Leif Kobbelt. Non-linear shape optimization using local subspace projections. *ACM Transactions on Graphics (TOG)*, 35(4):87, 2016.

Dinesh K Pai, Roderick A Barman, and Scott K Ralph. Platonic beasts: Spherically symmetric multilimbed robots. *Autonomous Robots*, 2(3):191–201, 1995.

Christiaan J Paredis and Pradeep K Khosla. An Approach for Mapping Kinematic Task Specifications into a Manipulator Design. *International Conference on Advanced Robotics*, 1, 1991.

Romain Prévost, Emily Whiting, Sylvain Lefebvre, and Olga Sorkine-Hornung. Make it stand: balancing shapes for 3d fabrication. *ACM Transactions on Graphics (TOG)*, 32(4):81, 2013.

PyDART. *A Python Binding of Dynamic Animation and Robotics Toolkit*. URL http://pydart2.readthedocs.io.

Xuewen Rong, Yibin Li, Jiuhong Ruan, and Bin Li. Design and simulation for a hydraulic actuated quadruped robot . *Journal of mechanical science and technology*, 2012.

Yousef Saad. *Iterative methods for sparse linear systems*, volume 82. siam, 2003.

Claudio Semini, Nikos G Tsagarakis, Emanuele Guglielmino, Ferdinando Cannella, and Darwin G Caldwell. Design of HyQ a Hydraulically and Electrically Actuated Quadruped Robot. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, 2011.

Sangok Seok, Albert Wang, Meng Yee Chuah, Dong Jin Hyun, Jongwoo Lee, David M. Otten, Jeffrey H. Lang, and Sangbae Kim. Design Principles for Energy-Efficient Legged Locomotion and Implementation on the MIT Cheetah Robot. *IEEE/ASME Transactions on Mechatronics*, 20(3), 2014.

Karl Sims. Evolving Virtual Creatures. *ACM Transactions on Graphics*, (July):15–22, 1994.

Stratasys. *Connex Objet260*. URL http://www.stratasys.com.

E.J. Van Henten, D.A. Vant Slot, C.W.J. Hol, and L.G. Van Willigenburg. Optimal manipulator design for a cucumber harvesting robot. *Computers and Electronics in Agriculture*, 65(2):247–257, 2009.

Kevin Wampler and Zoran Popović. Optimal gait and form for animal locomotion. *ACM Transactions on Graphics*, 28:1, 2009.

Yuan Yun and Yangmin Li. Optimal design of a 3-PUPU parallel robot with compliant hinges for micromanipulation in a cubic workspace. *Robotics and Computer-Integrated Manufacturing*, (6), 2011.