Designing actuation systems for animatronic figures via globally optimal discrete search

SIMON HUBER, ETH Zürich, Switzerland
ROI PORANNE, University of Haifa, Israel
STELIAN COROS, ETH Zürich, Switzerland

We present an algorithmic approach to designing animatronic figures – expressive robotic characters whose movements are driven by a large number of actuators. The input to our design system provides a high-level specification of the space of motions the character should be able to perform. The output consists of a fully functional mechatronic blueprint. We cast the design task as a search problem in a vast combinatorial space of possible solutions. To find an optimal design in this space, we propose an efficient best-first search algorithm that is guided by an admissible heuristic. The objectives guiding the search process demand that the design remains free of singularities and self-collisions at any point in the high-dimensional space of motions the character is expected to be able to execute. To identify worst-case self-collision scenarios for multi degree-of-freedom closed-loop mechanisms, we additionally develop an elegant technique inspired by the concept of adversarial attacks. We demonstrate the efficacy of our approach by creating designs for several animatronic figures of varying complexity.

CCS Concepts: • **Computing methodologies → Motion processing**; **Collision detection**; **Shape analysis**.

Additional Key Words and Phrases: fabrication, animatronics, multi-motor, many-motor

ACM Reference Format:
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 1 (August 2021), 10 pages. https://doi.org/10.1145/3450626.3459867

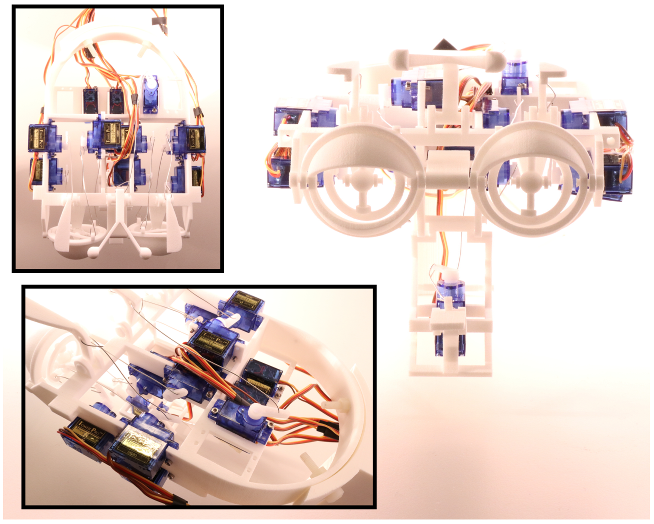

Fig. 1. An animatronic face. Our algorithm assists in finding optimal motor placement that avoid collisions and has efficient force transmission.

1 INTRODUCTION

Animatronic figures are electro-mechanical systems that are meticulously engineered to generate lifelike motions. As can be witnessed in theme parks, museums and special effects studios around the world, these robotic characters can be made to convincingly resemble a vast array of creatures, real or imagined. Needless to say though, the process of designing such marvels of engineering is time-consuming, error-prone, and demands a great deal of experience and domain specific knowledge. In this paper, we therefore tackle the challenge of automating some of the most tedious design tasks that arise while creating animatronic figures.

In its most basic incarnation, as shown in Fig. 2, the bare-bones structure of an animatronic figure consists of an *articulated armature*, a set of *actuators*, and various mechanical structures that act as *force transmission mechanisms*. The articulated armature defines the desired range of motion of the robotic character, and it can be seen as a direct analogue to a traditional animation rig. As such, we model it as a hierarchical arrangement of rigid components that are connected to each other via joints. The individual rigid components in the armature can represent limb segments, miscellaneous body parts such as ears, eyeballs and eyelids, or auxiliary appendages used to drive the motions of a character's lips or eyebrows, for example.

Actuators – commonly just off-the-shelf servomotors – bring animatronic figures to life. The number of actuators embedded in a design defines the set of unique *functions* (i.e. raise/lower left eye-brow, open/close jaw, etc) it has; the more actuators, the richer a space of motions the animatronic figure can generate. The computational design framework we present is specifically developed for animatronic characters that feature a large number of actuated degrees of freedom.

Force transmission mechanisms are used to propagate the movements generated by each actuator to the underlying articulated armature. More specifically, they assign servomotors to individual functions of the animatronic character, both conceptually and physically. In this work, the elements we consider when designing force transmission mechanisms are rigid push/pull *tie rods* and *bellcranks*, though other options – gears, belts, etc. – are also possible. Our

Authors' addresses: Simon Huber, ETH Zürich, Switzerland; Roi Poranne, University of Haifa, Israel; Stelian Coros, ETH Zürich, Switzerland.

0730-0301/2021/8-ART1 $15.00
https://doi.org/10.1145/3450626.3459867

choice is motivated by common practices in animatronic figure design, where, as shown in Fig 2, these two type of elements can be combined to great effect. We further note that tie rods are often endowed with *swivel ends* – spherical bearing joints which enable the design of spatial (i.e. non-planar) mechanisms that are highly versatile and have a large range of motion.

With the anatomy of a typical animatronic figure exposed, we can now define the challenging design problem that we address in this work. The input to our design algorithm consists of an articulated armature. To specify the desired range of motion of the design, each joint of the armature is given the min and max angles it should be able to achieve. Also given as input is a set of candidate actuators; the number of elements in this set must be greater than or equal to the number of functions of the envisioned animatronic character. Our design algorithm decides how to optimally assign actuators from the input set to each target function, and it automatically generates force transmission mechanisms for each assignment. The latter task demands that tie rods are instantiated in the design and physically connected, possibly via bellcranks, to appropriate attachment points on armature and on the horns of the servomotors. The objectives driving the design process demand that the synthesized force transmission mechanisms remain singularity free and do not collide with the armature or with each other at any point in the high-dimensional space of motions the character is expected to be able to execute.

We cast the task of designing animatronic figures as a tree search problem over a vast combinatorial space of possible discrete solutions. To efficiently explore this space in pursuit of an optimal design, we propose a novel best-first search algorithm that is guided by an admissible heuristic. We demonstrate the efficacy of our approach by designing a diverse set of animatronic figures. Our most complex example features 18 functions, and it was automatically designed in under 20 minutes. For comparison, naively evaluating every possible design would take years of compute time.

Succinctly, our technical contributions are:

- Casting the design of high degree-of-freedom animatronic figures as a search problem in a discretized combinatorial space of possible solutions
- An efficient algorithm to find optimal designs in the vast space of possible solutions
- A technique inspired by the concept of adversarial attacks to identify worst-case self-collision scenarios in multi-degree-of-freedom closed-loop mechanisms.

2 RELATED WORK

The past decade has witnessed a significant number of research projects dedicated to the development of computation-driven design methodologies for various classes of physical artifacts [Bermano et al. 2017]. Closest to the problem we focus on here is the body of work addressing the design of mechanical devices that are able to produce choreographed movements. Roughly speaking, this body of work can be divided into techniques that synthesize novel mechanisms [Megaro et al. 2014; Thomaszewski et al. 2014], and methods used to fine-tune or re-purpose existing designs using continuous or stochastic optimization techniques [Bächer et al. 2015; Ha et al. 2018b; Song et al. 2017; Zhang et al. 2017]. Of course, these two concepts can naturally be combined as well [Bharaj et al. 2015; Coros et al. 2013; Zheng et al. 2016; Zhu et al. 2012].

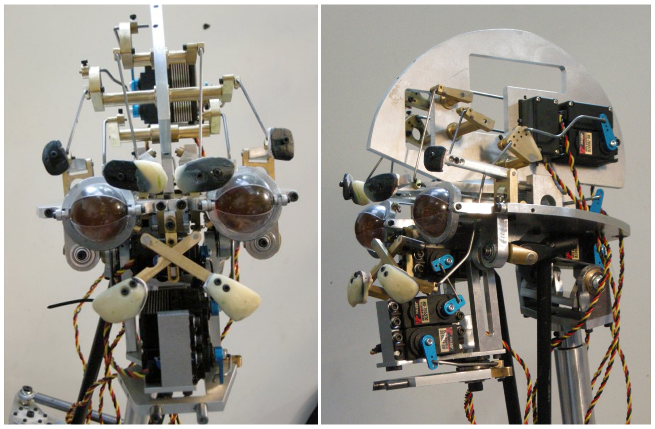
Fig. 2. Internal structure of an animatronic head created by master creature FX artist Gustav Hoegen.

The method we propose synthesizes functional mechanisms from scratch. However, rather than employing a template-based approach where fully-functional mechanisms are used as building blocks [Coros et al. 2013; Zhu et al. 2012], our method operates directly on basic components, tie rods and bellcranks, which are computationally assembled to form complex mechanical structures. From this point of view, our work is closest related to that of Thomaszewski and his colleagues [Thomaszewski et al. 2014]. However, while their method is tailored to designing planar, single degree of freedom structures, our method specifically targets spatial (i.e. non-planar) mechanisms driven by a large number of actuators that must operate in concert. This new problem setting drastically increases the complexity of the design process and therefore necessitates a different algorithmic approach. For example, an appropriate design must ensure that self-collisions are avoided throughout the animatronic figure's targeted range of motion. This is very challenging. For one degree of freedom mechanisms, a simple monotonic search along the actuation parameter suffices to identify configurations where self collisions occur, and simple analytic formulas have also been developed [Zheng et al. 2016]. In the multi-motor setting, we instead propose to formulate the task of identifying self collisions as a numerical optimization problem. To determine if a given design is free of self collisions, we find the values of the motor angles that minimize the distance between selected pairs of mechanical components. This approach is inspired by the concept of adversarial attacks, where the aim is to explicitly find worst-case scenarios that break a given system [Huang et al. 2017].

Prior work has tackled design problems for robotic systems with multiple actuated degrees of freedom, typically in the context of locomotion or flight [Du et al. 2016; Geilinger et al. 2018; Jelisavcic et al. 2017; Leger et al. 1999; Megaro et al. 2015]. Some of these works formulate the design task as tree or graph search problems [Ha et al. 2018a; Zhao et al. 2020], as we do. However, prior methods generally assume tree-like serial structures for the robotic designs they can generate. Designs for animatronic figures, on the other hand, are much more mechanically complex. As they are specifically designed to resemble living creatures, their underlying articulated armatures are slim and it is not possible to place motors directly at the joints. Mechanical force transmission elements are therefore a necessity, and they turn animatronic figures into complex multi-degree-of-freedom closed-loop mechanisms. Our novel design algorithm, which is cast as an efficient search process driven by an admissible heuristic, is specifically developed for these types of mechatronic systems.

We also note that there are several recent projects that are closely related, but complementary to our work. For example, Desai and her colleagues developed a framework for assembly-aware design of electro-mechanical devices [Desai et al. 2018]. The designs they target contain no moving parts, but the techniques they propose could be used to automatically prepare the designs generated with our method for fabrication. Computational issues related to the design of soft skins for animatronic characters have also been investigated [Bickel et al. 2012; Feng et al. 2019]. The animatronic designs generated with our method could also be used to drive the motion of such soft skins, although ensuring that the force transmission mechanisms are designed to be sufficiently strong remains an avenue for future work. We would also like to note that techniques developed to model [Geilinger et al. 2020; Hahn et al. 2019], design [Bern et al. 2017; Ma et al. 2017; Megaro et al. 2017; Tang et al. 2020] and control [Bern et al. 2019; Hoshyari et al. 2019] compliant mechanisms and soft robots could also be used in conjunction with the algorithmic methodology we present in order to create increasingly lifelike animatronic characters that are composed of a mix of rigid and deformable materials.

3 OVERVIEW

Given an animatronic figure with multiple articulated components, our goal is to optimally place, assign and attach motors to actuate these components. Each component can be made out of several mechanical parts. We refer to the motors as *drivers* and to the mechanical parts they are connected to as *followers*. As input to the algorithm, the user specifies a range of motion for each component. In addition, the user defines a region of space where the drivers are allowed to reside. Usually, this will be a plate within the body of the mechanism that the drivers can be mounted on. This region is oversampled with drivers, i.e. more drivers than the necessary degrees of freedom are placed, from which a subset that globally optimizes a certain performance objective is automatically picked, as discussed in sec. 3.1. In other words, we *assign* a driver to each follower. An assignment is physically made by connecting the followers to the drivers with tie rods. This can be done in many different ways, e.g. attachment points on the driver and follower, or via bellcrank mechanisms, all of which are considered during optimization. See Fig. 3 for an illustration.

Our algorithm employs a *Branch-and-Bounds* approach, and the general idea is as follows. A complete assignment is an assignment of all followers to drivers such that all degrees of freedom are accounted for, while a partial assignment only treats some of the followers. Every partial or complete assignment has a certain cost,

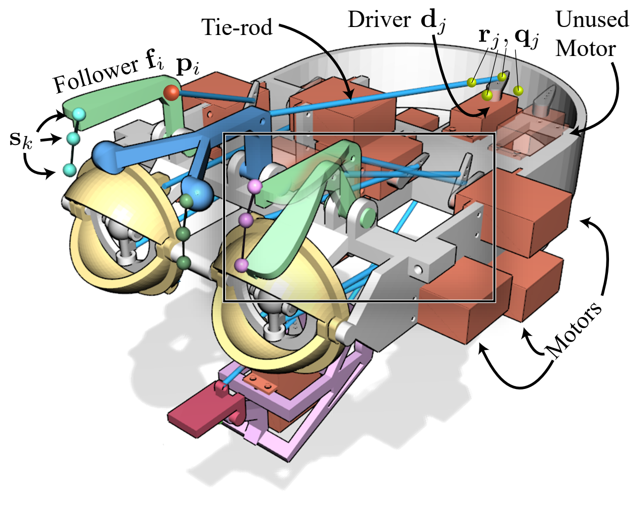
Tie-rod
Driver $\mathbf{d}_j$
$\mathbf{r}_j, \mathbf{q}_j$
Unused Motor
Follower $\mathbf{f}_i$ $\mathbf{p}_i$
$\mathbf{s}_k$
Motors

Fig. 3. Overview of the system and the terminology used. To visualize the motions of the followers, we trace a single points on them (shown as spheres in the figure). The highlighted frame in the middle shows two poses of the same follower. There is a single, unused motor visible, as determined by the algorithm, but there are other unused ones hidden behind the mechanism.

and thus our goal is to search all possible complete assignments for the one with minimal cost. Partial assignments can be organized in a *tree* structure, where children of a node contain the same partial assignment, appended with an additional single assignment (see Fig. 4). The leaves of the tree are the complete assignments, and therefore what we seek is to find the leaf with the minimal cost. Based on this observation, for each partial assignment we propose two heuristics that underestimate the minimal cost of a leaf of the partial assignment's subtree. This can be used, for example, as the basis for the A^* algorithm, a particular instance of Branch-and-Bound, which we discuss in Sec. 4.1.

3.1 Problem statement

Input. The animatronic figure is a mechanism, which consists of a set of rigid bodies or *links* that are connected by joints. In our setting, we distinguish between three types of links: Drivers are the horns, the exposed rotating part, of the motors, and their pose is directly determined by the motors' angles. Followers are the rigid parts of the articulated components, which are intended to be actuated by the motors. Drivers and followers can be attached to *tie-rods*, the third type of link, at various points on their surfaces using ball and socket joints. An assignment of a driver to follower is physically fulfilled by directly connecting them together with a tie-rod. Note that in sec. 4.2 we also discuss assignments utilizing bellcranks, for better force transmission.

The *state* of the mechanism (excluding drivers and tie-rods) is defined by a vector $\mathbf{s}$ containing the poses of the rigid bodies. The space of possible states is determined by the set of constraints induced by the joints. Each articulated component $\mathbf{C}_k$ has multiple degrees of freedom, and once the poses of all of its designated followers parts are fixed, the entire state of the component $\mathbf{s}_k$ can be

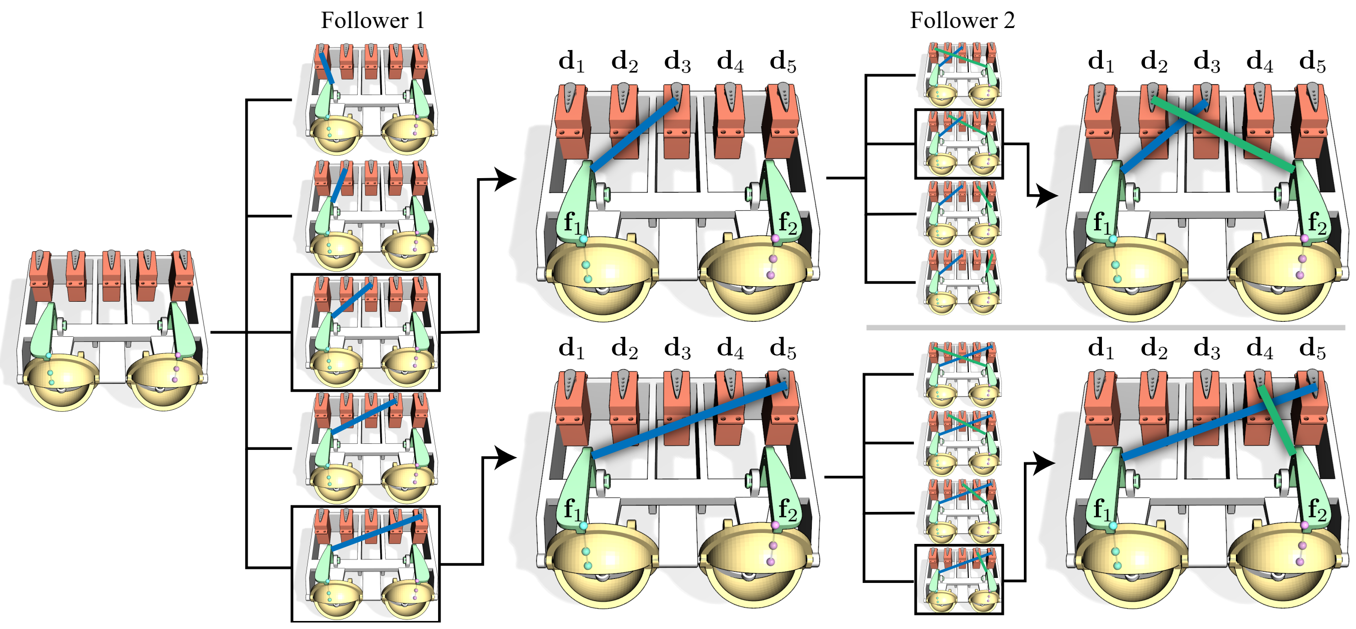
Follower 1
Follower 2
$\mathbf{d}_1$ $\mathbf{d}_2$ $\mathbf{d}_3$ $\mathbf{d}_4$ $\mathbf{d}_5$
$\mathbf{f}_1$
$\mathbf{f}_2$

Fig. 4. A subset of the tree structure we use for assignments. In this example, there are 2 followers and 5 drivers. The first column show all 5 assignments of one follower to all drivers. From these, we expand the tree for two of the assignments. Each of these has 4 other assignment for the second follower, as depicted in the image.

uniquely determined. Fixing a follower is achieved in practice by connecting it to a driver by a tie-rod. This essentially adds more constraints and reduces the number of degrees of freedom. Assuming that all degrees of freedom are accounted for, that is, all followers are connected to drivers, we can uniquely define the state of the entire mechanism by $\mathbf{s}(\mathbf{q})$ where $\mathbf{q}$ is a vector containing the angles of all of the drivers. This is, of course, dependant on the specific assignment as we discuss next. See sec. 4.2 on how we determine the state.

To define the input, we let the user interact with a virtual model of the animatronic figure, and specify a sample of desired possible states for each articulated component. This can also be done by specifying the min/max angles each joint should be able to achieve. We denote the sampled states of the component by $\mathbf{s}_{kl}$ where l is some index. The user can also specify which parts of the mechanism are to be considered as followers, which we denote by $\mathbf{F} = \{f_i\}$, $i = 1, \ldots, m$ and a set of candidate attachment points on the surface of the followers (in local coordinates) $\mathbf{p}_i$ which will be considered for connecting tie-rods. While this can be done automatically, many choices can be ruled out immediately by the informed user, due to considerations such as aesthetics and fabrication limitations. Additionally, our algorithm requires a list of candidate motor poses, denoted by $\mathbf{D} = \{d_j\}$, $i = j, \ldots, n$ which also provides the range of poses for the drivers. Again, this list can be either generated automatically, by sampling a region in space where drivers are allowed to be placed, as provided by the user, or specific locations, based on the user's informed guess. Finally, a set of possible attachment points on the drivers and initial motor angle are to be provided, which we denote by r_j and $\mathbf{q}_j$, respectively.

Local costs. We define a single assignment of a follower to a driver as a pair $\mathbf{a}_{ij} = (f_i, d_j)$. This assignment means that there is a tie-rod connecting a point from $\mathbf{p}_i$ on the follower, to a point from r_j on the driver. In order to decide which two specific points we should connect, we compute a *cost* for each choice, and pick the two points with the lowest cost. We term this cost $\mathcal{L}(\mathbf{a}_{ij})$ as the *local* cost associated with $\mathbf{a}_{ij}$. This cost measures properties that can be inferred from $\mathbf{a}_{ij}$ alone, without considering others pairs. For example, we can measure the ability of reach target poses and the torque required to actuate the follower, and penalize collisions between the tie-rod and the static components of the mechanism. More detail appears in Sec. 4.2.

Assignments. A partial assignment is commonly defined as an injective function, which in our case would be $\mathbf{A} : \tilde{\mathbf{F}} \subset \mathbf{F} \to \mathbf{D}$. Herein, a more useful representation is as a set of pairs. With this, a partial assignment $\mathbf{A}$ is also a set, where each driver and each follower appear at most once. A complete assignment is a partial assignment where *all* followers appear once. Partial assignments can be organized in a *tree* structure, where children of a node $\mathbf{A}$ contain the same partial assignment, appended with an additional single assignment, e.g. $\mathbf{A} \cup \{\mathbf{a}_{ij}\}$. The leaves of the tree correspond to all complete assignments. We discuss a specific structure for this tree and how to utilize it in sec. 4.1.

Global costs. A partial or complete assignment $\mathbf{A}$ has a *global* cost $\mathcal{G}_(\mathbf{A})$ associated with it, that evaluates the performance of the assignment as a whole. Different considerations can be taken into account here, related to the distribution of motors, for example. The global cost we employ aims to eliminate collisions between the tie-rods that connect different driver-follower pairs. We develop an approach inspired by adversarial attacks to find worst-case self-collision configurations, as described in sec. 4.2.

The total cost. With the global cost defined, we can finally formulate our optimization problem. Our goal is to find a complete assignment that *globally* minimizes the total cost $C(\mathbf{A})$, that we define as the sum of local costs *and* global cost. Formally,

$$\min_{\mathbf{A}\in\mathcal{A}} C(\mathbf{A}) = \mathcal{G}(\mathbf{A}) + \sum_{\mathbf{a}_{ij}\in\mathbf{A}} \mathcal{L}(\mathbf{a}_{ij}) \tag{1}$$

where $\mathcal{A}$ is the set of all complete assignments. In the next sections we describe the algorithms for optimizing (1), and the various aspects of the costs in more details.

4 METHOD

4.1 Branch-and-bound

The A algorithm.* We begin this section by describing the the optimization approach first, followed by a detailed description of the local and global costs and how we compute them. As mentioned, the main idea is to organize partial assignments in a tree structure. We let the root be the *empty* assignment. Then, each level i of the tree is dedicated to a single follower f_i. In the first level, i.e., all the children of the root, each node represents the assignment of f_1 to one drivers. That is the first level contain the nodes $\mathbf{a}_{1j} = (f_1, d_j)$ for all j's. Let $\mathbf{A}$ be an assignment of the $(i-1)$-th level. We define its children to be all *valid* assignments $\mathbf{A} \cup \{\mathbf{a}_{ij}\}$, that is, for all d_j's that are not already part of $\mathbf{A}$. With this structure, all possible partial assignments appear in the tree exactly once. We note however, that different orderings of the followers will produce different trees.

We can naively traverse the leaves of the tree (all $\frac{m!}{(m-n)!}$ node), but of course this approach is not scalable. Instead, the branch-and-bound approach calls for *bounding* the cost of of each node's subtree. A subtree that has lower bound higher than the current upper bound for the solution will not be traversed, potentially saving considerable amount of resources. To this end, we note the $C(\mathbf{A})$ for a node is already a lower bound on the cost of $\mathbf{A}$'s subtree. Indeed, by appending $\mathbf{A}$ the cost can only increase. Thus, if we do encounter such a node, we prune the node's subtree and proceed with the next node in our traversal order. Furthermore, it is possible to devise a hueristic, which never *overestimate* the cost of the subtree, also known as an *admissible* heuristic. This also informs the preferred order in which the tree should be traversed, which leads to the standard A* algorithm. In the common A* notation, we seek an evaluation function $f(\mathbf{A}) = g(\mathbf{A}) + h(\mathbf{A})$, where $g(\mathbf{A}) = C(\mathbf{A})$ is the cost of the current node, and $h(\mathbf{A})$ is the admissible heuristic.

The heuristic. For the A* algorithm to run efficiently, $h(\mathbf{A})$ should be quick to evaluate, and provide a tight lower bound. A good balance between these criteria is detrimental to the performance of A*. In our case, the definition of $C(\mathbf{A})$ strongly suggests of a particular heuristic: We can quickly and accurately compute the optimal *complementary* assignment to $\mathbf{A}$ (that is, the assignment the completes $\mathbf{A}$) in terms of local costs only. In other words, given $\mathbf{A}$, we propose to define $h(\mathbf{A})$ as

$$h(\mathbf{A}) = \min_{\substack{\mathbf{A}_c\cup\mathbf{A}\in\mathcal{A}\\ \mathbf{A}_c\cap\mathbf{A}=\emptyset}} \sum_{\mathbf{a}_{ij}\in\mathbf{A}_c} \mathcal{L}(\mathbf{a}_{ij}). \tag{2}$$

Eq. (2) is nothing more than a standard assignment problem which can be solved in cubic time using the Hungarian algorithm. To boost the performance, we can precompute all $m\times n$ local costs beforehand, and recall the values when required.

The Hungarian algorithm can still be costly to run for every node visited. As an alternative, we propose a *greedy* approach, which allows multiple assignments to the same driver in (2). We denote the resulting heuristic by $\tilde{h}(\mathbf{A})$. Of course, the optimal assignments predicted $\tilde{h}(\mathbf{A})$ are not legal assignments, but they do serve the purpose of achieving an underestimating heuristic quickly. Clearly, $\tilde{h}(\mathbf{A}) \le h(\mathbf{A})$, and computing $h(\mathbf{A})$ is done in linear time. We summarize the algorithm in Algorithm 1. It is important to note that the choice of heuristic has no influence on the final result but on the number of nodes visited in the tree. See Table 1 in the results Section to see how the two different heuristics influence the number of nodes visited.

A alternatives.* The main drawbacks of the A* algorithm are the fact the it does not produce intermediate results, and that the entire frontier must be stored in a priority queue, leading to high memory consumption. Many alternatives have been proposed in the past (see e.g. [Hansen and Zhou 2007; Sun and Koenig 2007]) We experimented with yet another alternative more tailored to our problem. The approach is to reach candidate solutions early, so subtrees can be pruned more effectively. With this approach, at a node $\mathbf{A}$, we prioritize its children based on h or $\tilde{h}$. In contrast to A*, we fully traverse the first child of $\mathbf{A}$ before proceeding with the second one, and so on. We maintain the current best solution, and use it to prune the tree. That is, if the heuristic for the currently visited node $\mathbf{A}$ is greater than the current best solution, we prune the tree at $\mathbf{A}$. We initialize this algorithm with a solution obtained by the Hungarian algorithm. This approach, which requires $O(mn)$ memory, provides a suboptimal solution quickly, and then finds better solutions over time. We summarize this approach in Algorithm 2, and discuss the practical differences between the two strategies in Sec. 5.

4.2 Cost evaluation

Computing states. As mentioned in Sec. 3.1, the computation of the various states of the mechanism has a central role, both for specifying the input and evaluating the costs. We use two different approaches – symbolic kinematics and Newton's method – for different parts of the algorithm, both with their own strengths and weaknesses. For symoblic kinematics, we follow the approach presented in [Bächer et al. 2015]. With this approach, we can finds closed form expressions the pose of the follower as a function of the angles of the motor alone. We note that, originally, the paper discussed only 2D mechanisms, but the extension to 3D is straightforward. The advantage of this approach is it extremely fast and reliable compared to constraint based evaluation, i.e. Newton's method. The main limitation of this approach is that it cannot treat mechanism with kinematic loops. This is not an issue when computing local costs, since for these, we fix the entire mechanism except for the driver

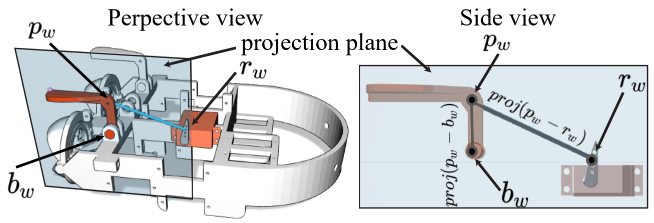

Fig. 5. The mechanism in a specific state $\mathbf{s}_{kl}$, displaying the hinge plane corresponding to the highlighted follower.

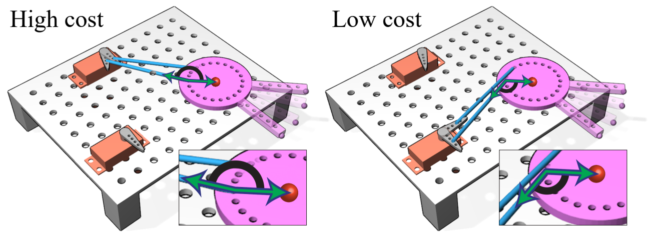

Fig. 6. Different assignment incur different costs. In this example, the assignment on the left has the two vectors almost parallel, which means that the mechanism is at a singular point. That is, the motor will struggle actuating the follower. On the right, they are almost orthogonal, indicating a close to optimal configuration.

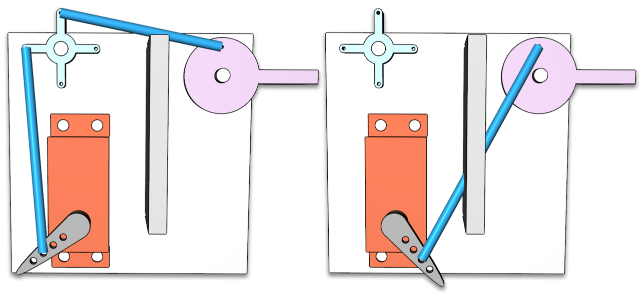

Fig. 7. Using via points in the form of a bellcrank, can assist in overcoming obstacles, obtaining more efficient force transmission and reducing the cost.

and follower in question, as we explain below. In order to compute global costs, we employ Newton's method. With Newton's method, we cast the problem of finding valid states as an optimization problem. This provides us with added flexibility which we leverage as discuss discussed below.

Computing local costs. To determine the local cost of the single assignment $\mathbf{a}_{ij}$ assigning f_i to d_j, we recall that f_i belongs to a specific articulated component $\mathbf{C}_k$, and that the user specified a list of desired states for this component, denoted by $\mathbf{s}_{kl}$, where $l = 1, \dots l_k$. Out of all possible pairs of tie-rod attachment points $\mathbf{p} \in p_i$ and $r \in r_j$ on f_i and d_j, we must find the most suitable ones, in the sense that they can efficiently allow driver to actuate the component and achieve the target poses. To measure efficiency, we examine three points: $\mathbf{p}_w, r_w$ and $\mathbf{b}_w$ which are the point $\mathbf{p}, r$ and the base joint of f_i, in world coordinates. Then, we define two vectors: $S = \mathbf{p}_w - \mathbf{b}_w$, and $T = \mathbf{p}_w - r_w$ and the smooth measure of efficiency as $\left|\hat{S} \cdot pro\hat{j}(T)\right|$, where $proj(T)$ is the projection of T on the plane spanned by the hinge axis at point b_w. This measure calculates the moment arm which tells you if the tie-rod is pushed or pulled how much of this force is applied to the joint axis. See Figure 5 for an illustration of the specific elements of the local cost in a specific state.

The position $\mathbf{p}_w$ is given by the pose of the component, but r_w depends on the motor angle. To fix an angle, we consider the additional degree of freedom of our problem, that is, the *initial* motor angle $\bar{\mathbf{q}} \in \mathbf{q}_j$. To that angle, we can w.l.o.g. assign the first state $\mathbf{s}_{k1}$ to $\bar{\mathbf{q}}$. Given these parameters, e.g. $\mathbf{p}, r$ and $\bar{\mathbf{q}}$, and the pose $\mathbf{s}_{k1}$, which for conciseness we stack in a vector α we can compute the length $\ell = \ell(\alpha)$ of the required tie-rod. Next, we can examine the rest of the poses $\mathbf{s}_{kl}$. Using symbolic kinematics we can compute, for each pose, the unique motor angle $\mathbf{q}_l = \mathbf{q}(\alpha, \ell; \mathbf{s}_{kl})$ for the driver (noting that α and ℓ are fixed), or conclude that one does not exist. In case $\mathbf{q}_l$ exists, we can calculate the efficiency measure as described above. Let it be denoted by $\mathcal{E}(\alpha, \ell; \mathbf{s}_{kl})$. Then, we define the cost of the pair $\mathbf{p}$ and r and the angle $\bar{\mathbf{q}}$ to be

$$\mathcal{E}(\mathbf{p}, r, \bar{\mathbf{q}}) = \sum_{l=1}^{l_k} \mathcal{E}(\alpha, \ell; \mathbf{s}_{kl}) \tag{3}$$

In case $\mathbf{q}_l$ does not exist, we set $\mathcal{E}$ to an arbitrarily high value. Finally, the local cost of $\mathbf{a}_{ij}$ is defined as the minimum over all pairs of attachment and initial angles, namely

$$\mathcal{L}(\mathbf{a}_{ij}) = \min_{\mathbf{p}, r, \bar{\mathbf{q}}} \mathcal{E}(\mathbf{p}, r, \bar{\mathbf{q}}) \tag{4}$$

Force transmission. To allow for greater flexibility, we can extend the mechanism by adding *via points*, which we denote by $\mathbf{v}_t$. These are additional rigid bodies that are connected attached to the mechanism by a hinge joint, and allow for force transmission. Via point are useful in order to avoid collisions, and in order to reduce the local costs in some cases (Fig. 7 and the inset). For every $\mathbf{a}_{ij}$, we additionally check the cost of attaching d_j to f_i via $\mathbf{v}_t$. The cost is simply the sum of the cost of attaching d_j to $\mathbf{v}_t$, treating $\mathbf{v}_t$ as a follower, and the cost of attaching v_t to f_i, treating $\mathbf{v}_t$ as a driver. Adding via point to the algorithm multiplies the number of nodes in the tree. In this paper we only experimented using up to one via point per pair, but allowing for more is a simple extension.

Computing global costs. As mentioned, global costs can have several use cases. In this work we use it to prevent designs that lead to self-collisions between the mechanical elements used to transmit forces from motors to the armature. We identify worst-case collision configurations using an approach inspired by the concept of adversarial attacks. More precisely, we actively drive the mechanism towards configurations where the internal components get as close as possible to each other. We do so using Newton's method, minimizing an objective that measures the distance between individual pairs of tie rods. If a state is found where this distance is less then 7mm between the centers of the two tie-rods, that is, a collision is possible, we set the global cost of the assignment to an arbitrarily high value. Otherwise, the cost of the pair is zero. We sum up all the pairwise costs to obtain the non-smooth global cost itself. We note that the same pair $\mathbf{a}_{i_1 j_1}, \mathbf{a}_{i_2 j_2}$ is likely to appear in several assignments, several times throughout the optimization. To save time, we store the cost of each pair that was encountered and recall it when necessary. Finally, we note that, when only considering pairwise costs, the problem can be cast as a quadratic assignment problem, which is known for its high complexity [Burkard et al. 1998].

Algorithm 1: Assignment A*

```
Input: Desired figure poses
Output: Optimal assignment
priority-queue.push(new RootNode())
while node not leaf do
    for child of node do
        cost = child.localCost() + child.globalCost()
        cost += child.heuristic() (Eq. 2)
        priority-queue.push(child, cost)
    end
    node = priority-queue.pop()
end
return node
```

Algorithm 2: Assignment B&B

```
Input: Desired figure poses
Output: Optimal assignment
/* Initialize current best using the Hungarian
   algorithm                                      */
node = hungarian()
upperBound = node.localCost + node.globalCost
bestNode = node
stack.push(root)
while stack not empty do
    vector = []
    node = stack.pop()
    for child of node do
        cost = child.localCost() + child.globalCost()
        cost += child.heuristic() (Eq. 2)  if
         cost < upperBound then
            if node.isLeaf() then
                upperBound = cost
                bestNode = child
            end
            vector.push(child, cost)
        end
    end
    vector.sort()
    stack.extend(vector)
end
return bestNode
```

5 RESULTS

In this section we evaluate our algorithm and demonstrate different designs. We refer the reader to the accompanying video for further demonstrations, and to the gallery of models is shown in Fig. 11. In all of our experiments we used the A* approach with the greedy heuristic $\tilde{h}(\mathbf{A})$. That is, the heuristic that does not use the Hungarian algorithm. We compare our method of choice with these approaches in this section as well.

Algorithm efficiency. The overall performance of the algorithm and its different stages are detailed in 1. The tree search algorithm highly depends on the problem and how tight the heuristic is. In terms of computation reuse, as mentioned, we compute all of the local costs in a preprocessing step, but pairwise collision for the global costs is computed on-the-fly, and then stored. The reason is that not all pairs of single assignments necessarily appear during the search. As can be seen from the table, the timing is dominated by the computation of bar-bar collisions. This indicates that there is a considerable potential for improvement, by creating a customized procedure for these types of collisions. We also provide in the table the total number of leaves, which are equivalent to complete assignments, vs the number of nodes our algorithm visited. Note that the number of leaves is just a fraction of the total number of nodes in the tree, and that the number of nodes visited is generally many orders of magnitude smaller, indicating that our heuristic is effective

Efficacy. Generally, the designer of a animatronic has a good intuition on regions where motors should reside. The main challenge is to avoid collision that are difficult to predict in a complex system. The typical use-case we envision is for the designer to designate a large set of likely positions, and let the algorithm find the optimal solution. Our experiments indicate that our algorithm can achieve that goal in seconds to minutes for typical sized problems. Nevertheless, the goal we set for ourselves is to find the optimal assignment from a highly redundant set of motors. As a stress test, we optimized a common model (baby animatronic) with 8 followers, and 100 potential drivers. The input and the solution, which was computed in 10 hours, is shown in Fig. 8. As mentioned, we believe the performance can still be vastly improved by developing a dedicated solver for bar-bar collisions. Additionally, we show a moderately sized problem with 21 drivers and 6 followers in Fig. 9.

A vs. Branch-and-Bound.* We compared the performances of A* and branch and bound over some of the designs, and the results are summarized in Table. 1. In general, both algorithms reach the same designs, but A* seem to outperform Branch-and-Bound. However, as mentioned in Sec. 4.1, Branch-and-Bound outputs intermediate results as it finds them. Additionally, the memory footprint of Branch-and-Bound is genenally insignificant in comparison to A*. In Fig. 10 we illustrate the progression of Branch-and-Bound by displaying the cost as a function of time. The vertical broken line represent the time it took for A* to reach the solution. As can be seen, A* does conclude before Branch-and-Bound, but can provide

Table 1. Performance statistics.

Model	Method	Heuristic	Drivers	Followers	Local cost total time	Bar-bar collision time	Total time	# Nodes evaluated	# Leaves total
Face	A*	Greedy	16	14	38 s	662s	827s	33697	10.4×10^{12}
Face	A*	Hungarian	"	""	"	576s	670s	14445	"
Face	B&B	Greedy	"	"	"	822s	1168s	55863	"
Baby (designed)	A*	Greedy	10	10	167s	514s	773s	36024	-
Baby (Oversampled)	A*	Greedy	100	8	300m	9h48m	10h6m	3.6×10^{6}	7.5×10^{15}
Hand	A*	Greedy	6	6	0.8 s	10s	11s	94	720
Hand	A*	Hungarian	"	"	"	9s	9.7s	66	"
Hand	B&B	Greedy	"	"	"	115s	12s	88	"
Hand (Oversampled)	A*	Greedy	21	6	6 s	130s	137s	685	39×10^{6}
Hand (Oversampled)	B&B	Greedy	"	"	"	339.3s	350.5s	2399	"
Hexapod	A*	Greedy	18	14	30s	1412s	1702s	77506	266 "

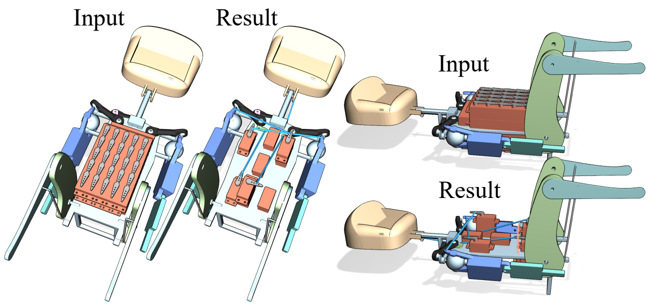

Fig. 8. A stress test result, running our algorithm on a baby animatronic with 100 drivers, and 8 followers.

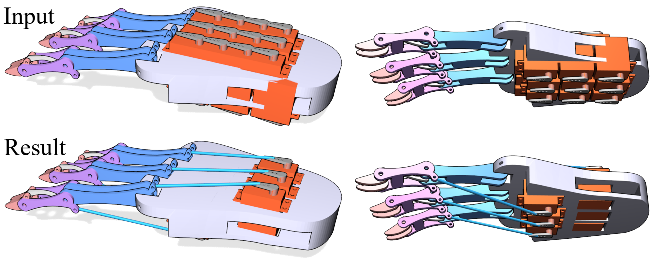

Fig. 9. A hand example, with 21 motors and 6 followers.

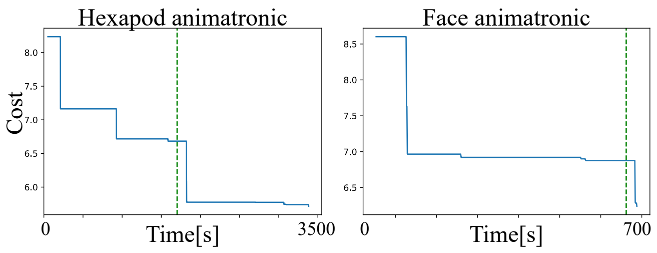

Fig. 10. A convergence plot of Branch-and-Bound, with a comparison to A*. The blue line shows the progression of Branch-and-Bound over time, while the broken green line indicates the point in time where A* returned the global minimum. While Branch-and-Bound gradually improves the candidate solution over time, before finally returning the global minimum, A* outputs *only* the global minimum, as soon as it is found.

arguably acceptable results. The choice between the two depends on the situation. Under a strict time budget, Branch-and-Bound is preferable since it can provide early suboptimal results. Otherwise, use of A* is suggested.

Heuristics comparison. The choice of heuristic is pivotal for the performance of the algorithm. A good heuristic balances between its computation time, and how effective it is in prioritizing nodes. We experimented with the greedy and Hungarian approaches, and the results can be seen in Table. 1. Our current conclusion is that for smaller scale problems, the Hungarian heuristic outperform the greedy one. However, the situation changes for larger problems. The explanation for this could be related to how the Hungarian algorithm scales; while effective for prioritization, the algorithm scales cubically. As a result, the first levels of the tree require an excessive amount of time to run, which is not balanced by the improved prioritization. One future avenue of investigation, is whether we can use results from similar assignments to speed up the Hungarian algorithm's runtime, which could potentially improve its performance over more complex designs.

Local vs Global Costs. The local costs tell if between a specific motor and follower a connection is possible. But just relying on the local costs to guide the search will often result in not desirable designs. See Figure 12 for examples when searching for the global optimum when ignoring vs not ignoring global costs.

Shoulder Mechanism. Figure 13 shows a recreated shoulder mechanism. It needs multiple motors working together and uses two universal joints. It has 4 degrees of freedom with all the motors placed on the rib cage and additionally leverages multiple via - components.

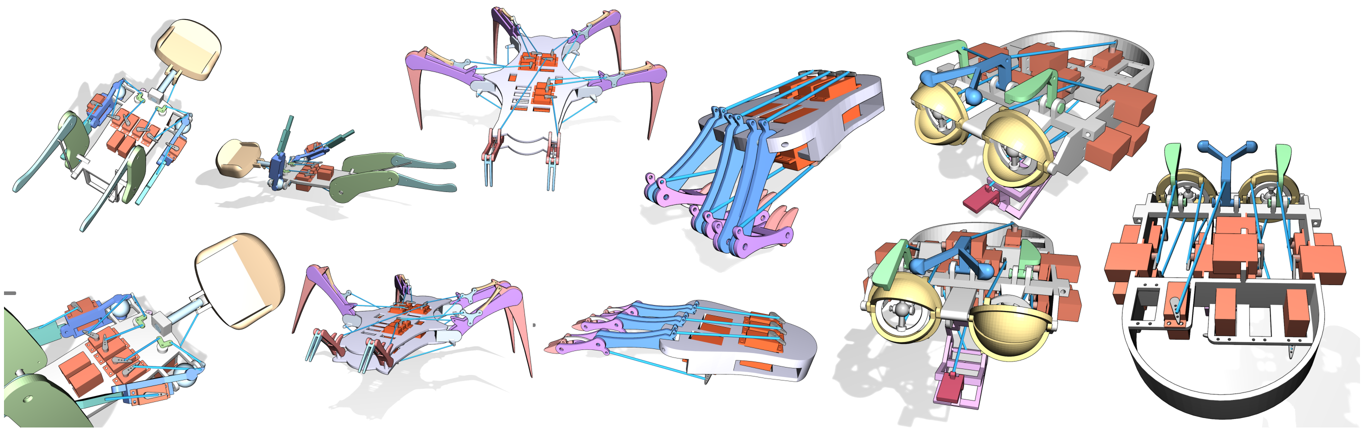

Fig. 11. A gallery showing different results and poses. From left to right, we show a baby model, a hexapod, a hand and a face designs. Note that the motor assignments for the hexapod are asymmetric due to the different ranges of motion specified for each leg.

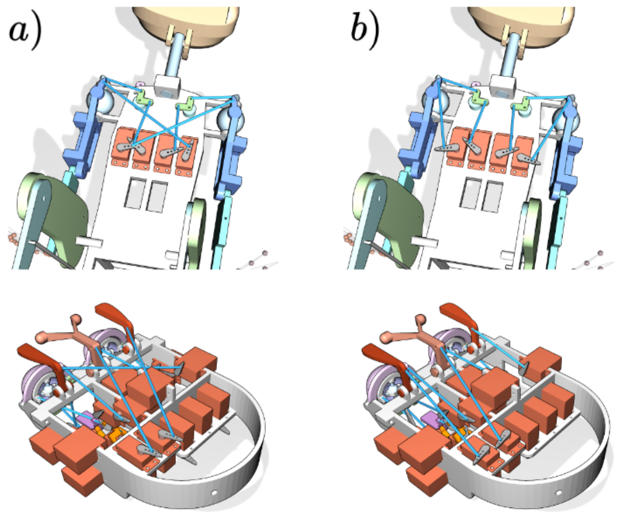

Fig. 12. a) The solution reached when ignoring global costs. b) The solution reached when using global and local costs.

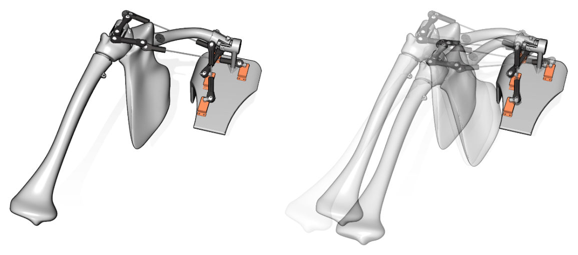

Fig. 13. The left image shows the shoulder mechanism and the right shows possible displacements.

6 CONCLUSIONS AND FUTURE WORK

We presented a computational approach to designing high degree-of-freedom animatronic characters based on range of motion specifications. Given an input set of candidate motors, our algorithm automatically generates an optimal blueprint by reasoning about the following types of design decisions:

- Which of the candidate motors is best suited to drive each function (i.e. movement of the underlying armature)
- In synthesizing force transmission mechanisms that connect the selected actuator to the articulated armature, which attachment point on the motor's horn should be used?
- Which attachment point on the armature should be used?
- Should a bellcrank be used as an intermediate structure between the motor and the armature?

Seen under this light, the design process boils down to a sequence of discrete decisions that must be taken. As such, we cast it as a search problem in the combinatorial space of all possible designs. The objectives driving the search process demand that the synthesized mechanisms remain singularity free and do not collide with the armature or with each other at any point in the high-dimensional space of motions the character is expected to be able to execute.

As evidenced through our results, the best-first search method we propose, which is guided by an admissible heuristic, is able to efficiently find globally optimal designs. We therefore believe algorithms like ours can play an important role in the development of personalized physical embodiments for future generations of intelligent agents. Nevertheless, before this vision can be realized, we see exciting avenues that need to be investigated. We would like, for example, to include in the design process elements that are not rigid. Flexible wires could be used to recreate the motion of lips or eyebrows; spring-damper systems or soft materials could be employed to make the overall designs compliant and safe to interact with; soft skins could give the final result an organic look, if that is desired. For each of these types of elements, the mechanical structure of the animatronic character would have to be designed while properly accounting for interactions with soft materials.

It would also be interesting to increase the diversity of elements used to design force transmissions. It is currently unclear, for example, how our algorithm would scale as we increase the number of ways in which a motor can be connected to the articulated armature. To gracefully handle the growth of the design space, additional heuristics or approximations are likely to be needed. As an alternative, continuous optimization techniques applied to the final designs could drastically reduce the set of discrete choices that must be evaluated using our current approach.

Lastly, we would like to explore ways of making it even more intuitive to specify the desired repertoire of motions an animatronic character should possess. We envision, for example, a system that starts with an animated character (full body, face only, etc) performing a variety of motions. Co-designing internal mechanisms, actuator layouts and soft skins in order to faithfully recreate those motions is particularly exciting as a long term vision.

Acknowledgment. This work has received funding from the European Research Council (ERC) (grant agreement No. 866480).

REFERENCES

Moritz Bächer, Stelian Coros, and Bernhard Thomaszewski. 2015. LinkEdit: interactive linkage editing using symbolic kinematics. *ACM Trans. Graph.* 34, 4 (2015), 99:1–99:8. https://doi.org/10.1145/2766985

Amit H. Bermano, Thomas A. Funkhouser, and Szymon Rusinkiewicz. 2017. State of the Art in Methods and Representations for Fabrication-Aware Design. *Comput. Graph. Forum* 36, 2 (2017), 509–535. https://doi.org/10.1111/cgf.13146

James M. Bern, Pol Banzet, Roi Poranne, and Stelian Coros. 2019. Trajectory Optimization for Cable-Driven Soft Robot Locomotion. In *Robotics: Science and Systems XV, University of Freiburg, Freiburg im Breisgau, Germany, June 22-26, 2019*, Antonio Bicchi, Hadas Kress-Gazit, and Seth Hutchinson (Eds.). https://doi.org/10.15607/RSS.2019.XV.052

James M. Bern, Kai-Hung Chang, and Stelian Coros. 2017. Interactive design of animated plushies. *ACM Trans. Graph.* 36, 4 (2017), 80:1–80:11. https://doi.org/10.1145/3072959.3073700

Gaurav Bharaj, Stelian Coros, Bernhard Thomaszewski, James Tompkin, Bernd Bickel, and Hanspeter Pfister. 2015. Computational design of walking automata. In *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation, SCA 2015, Los Angeles, CA, USA, August 7-9, 2015*, Jernej Barbic and Zhigang Deng (Eds.). ACM, 93–100. https://doi.org/10.1145/2786784.2786803

Bernd Bickel, Peter Kaufmann, Mélina Skouras, Bernhard Thomaszewski, Derek Bradley, Thabo Beeler, Philip Jackson, Steve Marschner, Wojciech Matusik, and Markus H. Gross. 2012. Physical face cloning. *ACM Trans. Graph.* 31, 4 (2012), 118:1–118:10. https://doi.org/10.1145/2185520.2185614

Rainer E. Burkard, Eranda Çela, Panos M. Pardalos, and Leonidas S. Pitsoulis. 1998. *The Quadratic Assignment Problem.* Springer US, Boston, MA, 1713–1809. https://doi.org/10.1007/978-1-4613-0303-9_27

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 (2013), 83:1–83:12. https://doi.org/10.1145/2461912.2461953

Ruta Desai, James McCann, and Stelian Coros. 2018. Assembly-aware Design of Printable Electromechanical Devices. In *The 31st Annual ACM Symposium on User Interface Software and Technology, UIST 2018, Berlin, Germany, October 14-17, 2018*, Patrick Baudisch, Albrecht Schmidt, and Andy Wilson (Eds.). ACM, 457–472. https://doi.org/10.1145/3242587.3242655

Tao Du, Adriana Schulz, Bo Zhu, Bernd Bickel, and Wojciech Matusik. 2016. Computational multicopter design. *ACM Trans. Graph.* 35, 6 (2016), 227:1–227:10. http://dl.acm.org/citation.cfm?id=2982427

X. Feng, J. Liu, Y. Yang, H. Wang, H. Bao, B. Bickel, and W. Xu. 2019. Computational Design of Skinned Quad-Robots. *IEEE Transactions on Visualization Computer Graphics* 01 (dec 2019), 1–1. https://doi.org/10.1109/TVCG.2019.2957218

Moritz Geilinger, David Hahn, Jonas Zehnder, Moritz Bächer, Bernhard Thomaszewski, and Stelian Coros. 2020. ADD: analytically differentiable dynamics for multi-body systems with frictional contact. *ACM Trans. Graph.* 39, 6 (2020), 190:1–190:15. https://doi.org/10.1145/3414685.3417766

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 (2018), 160:1–160:12. https://doi.org/10.1145/3197517.3201368

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.

Sehoon Ha, Stelian Coros, Alexander Alspach, Joohyung Kim, and Katsu Yamane. 2018b. Computational co-optimization of design parameters and motion trajectories for robotic systems. *Int. J. Robotics Res.* 37, 13-14 (2018). https://doi.org/10.1177/0278364918771172

David Hahn, Pol Banzet, James M. Bern, and Stelian Coros. 2019. Real2Sim: visco-elastic parameter estimation from dynamic motion. *ACM Trans. Graph.* 38, 6 (2019), 236:1–236:13. https://doi.org/10.1145/3355089.3356548

Eric A. Hansen and Rong Zhou. 2007. Anytime Heuristic Search. *J. Artif. Int. Res.* 28, 1 (March 2007), 267–297.

Shayan Hoshyari, Hongyi Xu, Espen Knoop, Stelian Coros, and Moritz Bächer. 2019. Vibration-minimizing motion retargeting for robotic characters. *ACM Trans. Graph.* 38, 4 (2019), 102:1–102:14. https://doi.org/10.1145/3306346.3323034

Sandy H. Huang, Nicolas Papernot, Ian J. Goodfellow, Yan Duan, and Pieter Abbeel. 2017. Adversarial Attacks on Neural Network Policies. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Workshop Track Proceedings.* OpenReview.net. https://openreview.net/forum?id=ryvlRyBKl

Milan Jelisavcic, Matteo de Carlo, Elte Hupkes, Panagiotis Eustratiadis, Jakub Orlowski, Evert Haasdijk, Joshua E. Auerbach, and A. E. Eiben. 2017. Real-World Evolution of Robot Morphologies: A Proof of Concept. *Artificial Life* 23, 2 (2017), 206–235. https://doi.org/10.1162/ARTL_a_00231 PMID: 28513201.

Chris Leger et al. 1999. *Automated synthesis and optimization of robot configurations: an evolutionary approach.* Carnegie Mellon University USA.

Li-Ke Ma, Yizhonc Zhang, Yang Liu, Kun Zhou, and Xin Tong. 2017. Computational design and fabrication of soft pneumatic objects with desired deformations. *ACM Trans. Graph.* 36, 6 (2017), 239:1–239:12. https://doi.org/10.1145/3130800.3130850

Vittorio Megaro, Bernhard Thomaszewski, Damien Gauge, Eitan Grinspun, Stelian Coros, and Markus H. Gross. 2014. ChaCra: An Interactive Design System for Rapid Character Crafting. In *The Eurographics / ACM SIGGRAPH Symposium on Computer Animation, SCA 2014, Copenhagen, Denmark, 2014*, Vladlen Koltun and Eftychios Sifakis (Eds.). Eurographics Association, 123–130. https://doi.org/10.2312/sca.20141130

Vittorio Megaro, Bernhard Thomaszewski, Maurizio Nitti, Otmar Hilliges, Markus H. Gross, and Stelian Coros. 2015. Interactive design of 3D-printable robotic creatures. *ACM Trans. Graph.* 34, 6 (2015), 216:1–216:9. https://doi.org/10.1145/2816795.2818137

Vittorio Megaro, Jonas Zehnder, Moritz Bächer, Stelian Coros, Markus H. Gross, and Bernhard Thomaszewski. 2017. A computational design tool for compliant mechanisms. *ACM Trans. Graph.* 36, 4 (2017), 82:1–82:12. https://doi.org/10.1145/3072959.3073636

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 (2017), 238:1–238:13. https://doi.org/10.1145/3130800.3130808

Xiaoxun Sun and Sven Koenig. 2007. The Fringe-Saving A* Search Algorithm: A Feasibility Study. In *Proceedings of the 20th International Joint Conference on Artifical Intelligence* (Hyderabad, India) *(IJCAI'07).* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2391–2397.

Pengbin Tang, Jonas Zehnder, Stelian Coros, and Bernhard Thomaszewski. 2020. A harmonic balance approach for designing compliant mechanical systems with non-linear periodic motions. *ACM Trans. Graph.* 39, 6 (2020), 191:1–191:14. https://doi.org/10.1145/3414685.3417765

Bernhard Thomaszewski, Stelian Coros, Damien Gauge, Vittorio Megaro, Eitan Grinspun, and Markus H. Gross. 2014. Computational design of linkage-based characters. *ACM Trans. Graph.* 33, 4 (2014), 64:1–64:9. https://doi.org/10.1145/2601097.2601143

Ran Zhang, Thomas Auzinger, Duygu Ceylan, Wilmot Li, and Bernd Bickel. 2017. Functionality-aware retargeting of mechanisms to 3D shapes. *ACM Trans. Graph.* 36, 4 (2017), 81:1–81:13. https://doi.org/10.1145/3072959.3073710

Allan Zhao, Jie Xu, Mina Konakovic-Lukovic, Josephine Hughes, Andrew Spielberg, Daniela Rus, and Wojciech Matusik. 2020. RoboGrammar: graph grammar for terrain-optimized robot design. *ACM Trans. Graph.* 39, 6 (2020), 188:1–188:16. https://doi.org/10.1145/3414685.3417831

Changxi Zheng, Timothy Sun, and Xiang Chen. 2016. Deployable 3D linkages with collision avoidance. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation, Zurich, Switzerland, July 11-13, 2016*, Barbara Solenthaler, Matthias Teschner, Ladislav Kavan, and Chris Wojtan (Eds.). Eurographics Association / ACM, 179–188. http://dl.acm.org/citation.cfm?id=2982843

Lifeng Zhu, Weiwei Xu, John Snyder, Yang Liu, Guoping Wang, and Baining Guo. 2012. Motion-guided mechanical toy modeling. *ACM Trans. Graph.* 31, 6 (2012), 127:1–127:10. https://doi.org/10.1145/2366145.2366146