Algorithmic Design of Kinematic Trees Based on CSC Dubins Planning for Link Shapes

Daniel Feshbach[0000−0001−5185−7395], Wei-Hsi Chen[0000−0001−8523−1809], Ling Xu[0009−0002−5343−7911], Emil Schaumburg[0009−0002−0399−551X], Isabella Huang[0009−0003−0813−3908], and Cynthia Sung[0000−0002−8967−1841]

General Robotics, Automation, Sensing & Perception (GRASP) Lab
University of Pennsylvania, 3101 Walnut St, Philadelphia, PA 19104, USA
{feshbach,weicc,xu5,scemil,ihuangg,crsung}@seas.upenn.edu
https://sung.seas.upenn.edu/, https://www.grasp.upenn.edu/

Abstract. Computational tools for robot design require algorithms moving between several layers of abstraction including task, morphology, kinematics, mechanism shapes, and actuation. In this paper we give a linear-time algorithm mapping from kinematics to mechanism shape for tree-structured linkages. Specifically, we take as input a tree whose nodes are axes of motion (lines which joints rotate about or translate along) along with types and sizes for joints on these axes, and a radius r for a tubular bound on the link shapes. Our algorithm outputs the geometry for a kinematic tree instantiating these specifications such that the neutral configuration has no self-intersection. Output designs have linear complexity. The algorithm approach is based on understanding the mechanism design problem as a planning problem for link shapes, and arranging the joints along their axes of motion to be appropriately spaced and oriented such that feasible, non-intersecting paths exist linking them. Since link bending is restricted by its tubular radius, this is a Dubins planning problem, and to prove the correctness of our algorithm we also prove a theorem about Dubins paths: if two point-direction pairs are separated by a plane at least $2r$ from each, and the directions each have non-negative dot product with the plane normal, then they are connected by a radius-r CSC Dubins path with turn angles $\leq \pi$. We implement our design algorithm in code and provide a 3D printed example of a tubular kinematic tree. The results provide an existence proof of tubular-shaped kinematic trees implementing given axes of motion, and could be used as a starting point for further optimization in an automated or algorithm-assisted robot design system.

Keywords: Kinematics, Path Planning, Computational Geometry, Dubins Paths, Computational Robot Design

1 Introduction

When designing linkages for robots, it is necessary to achieve specified poses or motions while avoiding self-intersection. Even with engineering expertise in navigating layers of kinematic and geometric abstractions, this is challenging as the

number of degrees of freedom scales up. Design algorithms mapping kinematic specifications to mechanism shapes could help non-experts design robots and assist experts in exploring the large space of morphologies. A foundational step is characterizing what types of kinematics are possible to geometrically realize with what classes of link and joint shapes. Here, we study kinematics specified by joint axes of motion and linkage shapes abstracted by their tubular bounds.

1.1 Related Works

Our previous work [7] gives a design algorithm for tubular kinematic chains implementing specified axes of motion, along with a catalog of tubular origami patterns implementing links and revolute and prismatic joints. The approach is based on understanding the robot design problem as a path planning problem for link shapes. Since the tubular radius constrains the bending curvature, this is a 3D Dubins planning problem. Since higher-order joint types can be constructed as sequences of revolute and prismatic joints [30], this algorithm can make chains with arbitrary kinematics. However, the chain design algorithm does not generalize to trees because it can involve looping "backwards" around previously-generated parts of the linkage in ways that would make collision avoidance guarantees between different branches elusive.

Computational Design Our work falls into the subfield of computational robot design. The primary objectives of developing robotic design tools are to reduce prerequisite expertise and enhance efficiency and reduce workload for the designer. Current design tools help users create fabricable plans from structural or functional specifications using modular composition [28,22] and optimization [6]. Interactive design [24,2] requires detailed specification of components and placement, often followed by optimization to align with objectives [15,12].

Automated design approaches translate task specifications into robot shapes with methods such as evolutionary algorithms [19,13], search algorithms combining discrete modules [14], or optimizations over parameterized modules [1]. Such methods can give robot designs tailored to the desired goals, but lack formal guarantees of design existence. In contrast, our work takes different input (general kinematics specified by axis of motion, rather than task specifications) and is guaranteed to produce a linkage shape implementing the given kinematics.

Another line of work addresses *kinematic synthesis*, producing axes of motion for a kinematic chain [21], tree [26], or more general linkage [33] achieving given positions or trajectories. Such approaches do not give specific linkage shapes implementing their output kinematics, so they could be combined with our work since we take axes of motion as input and give concrete designs as output.

Dubins Paths Our work is founded on Dubins path planning to find tubular shapes linking joints together. A *Dubins path* is a continuously differentiable path with a minimum turning radius r. Widely applied in motion planning for turning-constrained vehicles [34,20,5,18], theory characterizing Dubins paths has

been studied in 2D and 3D. In 2D, shortest paths have form CSC or CCC (or subsequences thereof), where S is a straight line segment and C is an arc of the minimal radius [10]. For 2D poses $\geq 4r$ apart, shortest paths are CSC [25].

In 3D, shortest paths are either CSC paths, coplanar CCC paths with the middle arc having turn angle $\geq \pi$, or radius-r helicoidal (H) paths [27]. Computing optimal paths is not analytically solved. For CSC paths, shortest paths can be found numerically via a variety of constructions [16,17,9,31,32,3]: for example, [3] analytically reduces the problem to finding zeros of a degree-12 single-variable polynomial and then solves for zeros numerically. (Another line of work extends 2D Dubins paths into a third dimension with its own separate derivative constraint [8,20,23,29], which is a mathematically different problem).

When using Dubins paths to represent linkage shapes, additional constraints are needed to enforce that the linkage will not loop back to locally self-intersect. If position-direction pairs are $4r$ away and separated by a plane whose normal has non-negative dot product with each of the directions, then [7] conjectures that they are connectable by a CSC path with turn angles $\leq \pi$. We prove that conjecture here (Thm. 1) and use it for our tree construction algorithm.

1.2 Contributions and Outline

We study the problem of designing kinematic trees implementing given axes of motion: we prove this is solvable (Thm. 2) with a design of linear complexity by providing a construction algorithm (Alg. 1) giving one such tree structured as a series of bent cylinders. Alg. 1 has essentially linear runtime (it can be made linear by replacing certain unnecessary optimization-based heuristics with procedures selecting any constraint-satisfying solution).

Our design approach involves placing joints along their given axes separated and oriented such that Thm. 1 (conjectured in [7] and proven here) guarantees they are connectable by CSC link paths which (to avoid local self-intersection) have C arcs with turn angles $\leq \pi$. Informally, Thm. 1 states that if two point-direction pairs are separated by a plane at least $2r$ from each, and neither direction is "backwards" with respect to the plane normal, then they are connected by a radius-r CSC Dubins path with turn angles $\leq \pi$. This enables a plane sweep technique for placing joints. We implement Alg. 1 in Python, building on our repository from [11]. To show that the tubular abstraction corresponds to physically realizable shapes, our supplementary materials[1] give a preliminary sketch of a modular generation procedure for 3D printing, and show a printed example of a 3-fingered hand.

Sec. 2 defines terms, states the design problem for tubular kinematic trees from kinematic specifications and motivates our choice of input and output structures. Sec. 3 states our tree design algorithm (Alg. 1) and its runtime complexity. Sec. 4 discusses our implementation of the algorithm in Python, with example results. Sec. 5 proves Thm. 1, and then Sec. 6 uses this to prove Thm. 2 (correctness of Alg. 1). Sec. 7 concludes with directions for future work.

[1] Supplementary materials can be found in the links for this paper's entry under "Related Publications" at https://sung.seas.upenn.edu/research/kinegami/.

2 Definitions and Problem Statement

2.1 General Kinematic Definitions

To define joints, links, and kinematic trees in general, we follow standards in the robotics literature [30]. A *joint* is a connection between two structures constraining their relative motion. The relative position of the structures is described by a *state variable*. A particular state value identified as *neutral* is typically encoded as zero. A *revolute joint* allows rotation about a line called the *axis of motion* (often encoded using Denavit-Hartenberg parameters): the state is a relative angle about the axis. A *prismatic joint* allows translation along an axis of motion: the state is a relative distance along it.

A *link* is a structure connecting joints. A link is *rigid* if its shape can vary only by rigid transformations. A *kinematic tree* is a tree-structured assemblage of links connected by joints. A *configuration* of a kinematic mechanism is a vector of state variables for each joint. A configuration is *valid* if the corresponding geometry has no self-intersection (except where rigidly attached joints and links connect).

2.2 Problem Input and Output

When developing algorithms for mechanism design, choosing appropriate abstractions for inputs and outputs involves balancing expressiveness, fabricability, and computational efficiency concerns.

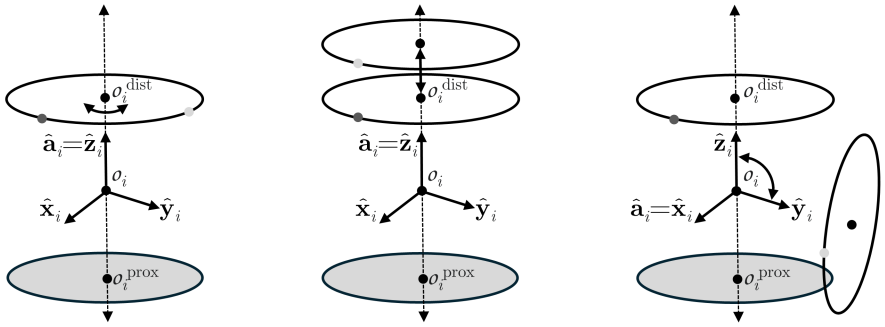
The key input to our algorithm is a tree of axes of motion on which to place joints. We use these because they can exactly express degrees of freedom appropriate for a mechanism's goals while leaving enough design freedom to allow provable guarantees (because there is infinite space on an axis along which to place a joint).

As output, our design approach gives kinematic trees as a series of branching and bending cylinders. This choice, and our overall design strategy, is based on understanding linkage design as a path planning problem. Since any link takes some amount of space it has some constraint on its bending radius, so it can be understood as containing a *Dubins path* related to its shape:

Definition 1 ([27]). *A* (radius-r) Dubins path *is a continuously differentiable curve in* $\mathbb{R}^n$ *with minimum turning radius* r*, i.e., with maximum curvature* $1/r$*. A* C component *of a Dubins path is an arc of radius* r*. An* S component *is a line segment. A component may be* degenerate *(length 0). A* CSC path *is a Dubins path composed of a C component, then an S component, then another C component.*

Then a Dubins path can be understood as characterizing a class of link shapes contained within the extrusion of a circle along the path:

Definition 2. *A* tubular link *is a rigid link bounded by the extrusion of a radius-*r *circle along a radius-*r *Dubins path, called the* centerline path *of the link.*



(a) Revolute joint with attachment type z (b) Prismatic joint with attachment type z (c) Revolute joint with attachment type x

Fig. 1. Notation for tubular joints (Def. 3). The proximal base is in gray. The distal base is in white, and is depicted in both the neutral state and a different state.

The tubular abstraction can express a wide variety of morphologies, but is geometrically simple enough to be computationally practical and to be useful for collision avoidance guarantees. It is also physically realizable, e.g. by 3D printing, as we demonstrate in supplementary materials Sec. A.

A tubular link connects to joints along its circular ends, so the corresponding notion of a *tubular joint* is contained in cylinders swept forward from the circles (notation is illustrated in Fig. 1):

Definition 3. *For a given radius r, a* tubular joint $\mathcal{J}_i$ *is a joint which, in the neutral configuration, has its physical structure bounded by a radius-r cylinder. It connects to incoming links at the* proximal *end of the cylinder and to outgoing links at the* distal *end. The cylinder's central axis direction $\hat{\mathbf{a}}_i$ is the* attachment direction *to the links: the end tangent of the incoming link's centerline path and (in the neutral configuration) the start tangent of the outgoing path.*

A tubular joint's pose *has position $\boldsymbol{o}$ at the cylinder center and orientation $\left(\hat{\mathbf{x}}_i\ \hat{\mathbf{y}}_i\ \hat{\mathbf{z}}_i\right)$ where $\overleftrightarrow{z_i} = \text{line}(\boldsymbol{o}_i, \hat{\mathbf{z}}_i)$ is the axis of motion and $\hat{\mathbf{a}}_i \in \{\hat{\mathbf{x}}_i, \hat{\mathbf{z}}_i\}$. The $\hat{\mathbf{a}}_i \in \{\hat{\mathbf{x}}_i, \hat{\mathbf{z}}_i\}$ condition means that links can connect either along the axis of motion (as in most prismatic joints or in-axis revolute joints) or orthogonally to it (as in elbow-like revolute joints).*

Attachment directions are classified into binary attachment type $a_i \in \{x, z\}$ *(the point of this distinction is that attachment direction is a vector requiring the joint's orientation to be defined, while attachment type can be used as part of the joint specification prior to finding its specific orientation).*

The cylinder's base centers are called the proximal position $\boldsymbol{o}_i^{prox} = \boldsymbol{o}_i - \frac{l_i}{2}\hat{\mathbf{a}}_i$ *and* distal position $\boldsymbol{o}_i^{dist} = \boldsymbol{o}_i + \frac{l_i}{2}\hat{\mathbf{a}}_i$ *respectively, where l_i is the* joint length.

A waypoint *is an empty (i.e., length $l_i = 0$), stateless tubular joint with attachment direction $\hat{\mathbf{a}}_i = \hat{\mathbf{z}}_i$ (useful for adding specifications on link paths).*

We can now define tubular designs and the input specifications for their kinematics:

Definition 4. *A* tubular kinematic tree design $\boldsymbol{\mathcal{T}}$ *is a tree whose nodes are tubular joints at specific poses and whose edges are annotated by tubular links connecting these joints.*

A design has fully specified geometry (joint poses and link paths). Our design problem is to find such poses and paths from the kinematic specification:

Definition 5. *A* tubular joint kinematic specification *is a tuple* $\mathcal{K}_i = (\overleftrightarrow{z_i}, l_i, a_i)$ *consisting of the axis of motion* $\overleftrightarrow{z_i}$*, length* l_i*, and attachment type* a_i *for a tubular joint. A* tubular tree kinematic specification $\boldsymbol{\mathcal{I}}$ *is a tree whose nodes are joint kinematic specifications* $\mathcal{K}_i$ *and whose edges represent connectivity by a link.*

Problem 1 (Tubular Kinematic Tree Design)**.** *Given a joint radius* r *and a tubular tree kinematic specification* $\boldsymbol{\mathcal{I}}$*, find a radius-*r *tubular kinematic tree design* $\boldsymbol{\mathcal{T}}$ *implementing the specification with the neutral configuration valid.*

Thm. 2 proves that Alg. 1 solves Prob. 1in all cases.

3 Tubular Tree Construction Algorithm

Our algorithm (Alg. 1) takes in a tubular radius r and a kinematic tree specification $\boldsymbol{\mathcal{I}}$ (Def. 5), and outputs a tubular kinematic tree design $\boldsymbol{\mathcal{T}}$ (Def. 4) instantiating the specification. The central challenge is to arrange joints along their axes of motion with tubular links connecting them without intersection.

The algorithm builds off the work in [7] that designs tubular kinematic serial chains. That algorithm proceeds along the chain and sequentially places joints along their axes of motion, each one at least $4r$ outwards from a tangent plane to the bounding sphere of the chain structure generated so far. The construction suffices to avoid self-intersection in serial chains, but directly generalizing it to trees can lead to intersections between branches. To avoid this, we propose to place joints with positions strictly increasing in a consistent "forward" direction and sweeping a plane forward in front of each new joint as we add it to the structure. Joint placement order is based on a depth-first traversal of the input specifications; branching paths proceed sideways outside the structure generated so far and then forward to the other side of the plane.

Our algorithm begins by choosing a forward direction $\hat{\mathbf{n}}$ not orthogonal to any joint's axis of motion. It places the root joint arbitrarily on its axis of motion and initializes the bounding cylinder $\mathcal{C}$ about it facing $\hat{\mathbf{n}}$. Then it proceeds to place joints one-by-one in depth-first order, putting each new joint $\mathcal{J}_c$ along its axis of motion such that its whole bounding sphere (radius $b_c = \sqrt{(l_c/2)^2 + r^2}$ about $\boldsymbol{o}_c$) is at least $4r$ in front of the end plane $\mathcal{P}$ of $\mathcal{C}$, as in Fig. 2(a). After placing each new joint, it expands $\mathcal{C}$ to encompass the new joint and links while preserving its direction as $\hat{\mathbf{n}}$, as in Fig. 2(b). Specifically, $\mathcal{C}$ is expanded sideways and forward/backward as needed separately to encompass the bounding sphere of a given joint and of the C components of each link: detailed procedures are in supplementary materials (Alg. B.1 and Alg. B.2).

When adding a new child of a parent that already has children, the algorithm inserts intermediate waypoints to route the path sideways outside of $\mathcal{C}$ and then forward to $\mathcal{P}$ before placing the new joint at least $4r$ in front of $\mathcal{P}$ (Fig. 2(c)). Specifically, it places the first waypoint $4r$ in front of the bounding sphere of

Algorithm 3.1: ConstructKinematicTree($r, \boldsymbol{\mathcal{I}}$)

Input: Tubular radius r, tubular tree kinematic specification $\boldsymbol{\mathcal{I}}$ (each node is a joint's axis of motion $\overleftrightarrow{z_i}$, attachment type $a_i \in \{x, z\}$, and length l_i)
Output: Tubular kinematic tree design $\boldsymbol{\mathcal{T}}$ implementing the specifications with no self-intersection in the neutral configuration

$\hat{\mathbf{n}} \leftarrow$ any direction not orthogonal to any $\overleftrightarrow{z_i}$: as a heuristic, we choose $\arg\max_{\hat{\mathbf{n}} \in S^2} \min_{\mathcal{K}_i \in \boldsymbol{\mathcal{I}}} |\hat{\mathbf{n}} \cdot \text{unitDirection}(\overleftrightarrow{z_i})|$
$\boldsymbol{o}_0 \leftarrow$ any point along $\overleftrightarrow{z_0}$
$\boldsymbol{\mathcal{T}} \leftarrow$ initialize tree to root $\mathcal{J}_0$ with pose $(\text{ForwardOrientation}(\overleftrightarrow{z_0}, \hat{\mathbf{n}}), \boldsymbol{o}_0)$, specified attachment type a_0, and specified length l_0
$\mathcal{C} \leftarrow$ bounding cylinder in direction $\hat{\mathbf{n}}$ of ball $\left(\boldsymbol{o}_0, b_0 = \sqrt{\left(\frac{l_0}{2}\right)^2 + r^2}\right)$
$\mathcal{P}_0 \leftarrow$ current end plane of $\mathcal{C}$
for *index* $c > 0$ *in depth-first traversal of* $\boldsymbol{\mathcal{I}}$ **do**
  $p \leftarrow$ index of parent(c)
  if $\mathcal{J}_p$ *already has children in* $\boldsymbol{\mathcal{T}}$ **then**
    $\boldsymbol{o}_{w_1} \leftarrow$ point closest to $\boldsymbol{o}_p$ in $\mathcal{P}_p + 4r\hat{\mathbf{n}}$ subject to being r out from $\mathcal{C}$
    $\mathcal{J}_{w_1} \leftarrow$ waypoint at $\boldsymbol{o}_{w_1}$ with $\hat{\mathbf{z}}_{w_1} = \hat{\mathbf{n}}$ (other axes chosen arbitrarily)
    $\mathcal{L}_{w_1} \leftarrow$ radius-r extrusion of CSC path from $(\boldsymbol{o}_p^{\text{dist}}, \hat{\mathbf{a}}_p)$ to $(\boldsymbol{o}_{w_1}, \hat{\mathbf{n}})$
    Add $\mathcal{J}_{w_1}$ to $\boldsymbol{\mathcal{T}}$ as a child of $\mathcal{J}_p$ linked by $\mathcal{L}_{w_1}$
    $\mathcal{C} \leftarrow \text{ExpandCylinderToIncludeLink}(\mathcal{C}, \mathcal{L}_{w_1})$ `//Alg. B.2`
    $\mathcal{P}_{w_1} \leftarrow$ current end plane of $\mathcal{C}$
    $\boldsymbol{o}_{w_2} \leftarrow \text{ray}(\boldsymbol{o}_{w_1}, \hat{\mathbf{n}}) \cap \mathcal{P}_{w_1}$
    $\mathcal{L}_{w_2} \leftarrow$ radius-r cylinder in direction $\hat{\mathbf{n}}$ from $\boldsymbol{o}_{w_1}$ to $\boldsymbol{o}_{w_2}$
    $\mathcal{J}_{w_2} \leftarrow$ waypoint at $\boldsymbol{o}_{w_2}$ with $\hat{\mathbf{z}}_{w_2} = \hat{\mathbf{n}}$ (other axes chosen arbitrarily)
    Add $\mathcal{J}_{w_2}$ to $\boldsymbol{\mathcal{T}}$ as a child of $\mathcal{J}_{w_1}$ linked by $\mathcal{L}_{w_2}$
    $\mathcal{C} \leftarrow \text{ExpandCylinderToIncludeLink}(\mathcal{C}, \mathcal{L}_{w_2})$ `//Alg. B.2`
    $\mathcal{P}_{w_2} \leftarrow$ current end plane of $\mathcal{C}$
    $p \leftarrow w_2$
  end
  $b_c \leftarrow \sqrt{\left(\frac{l_c}{2}\right)^2 + r^2}$ `//Joint bounding radius`
  $\boldsymbol{o}_c \leftarrow$ any $\boldsymbol{o}_c \in \overleftrightarrow{z_c}$ with $\text{signedDistance}(\mathcal{P}_p, \boldsymbol{o}_c) \geq 4r + b_c$: as a heuristic, we minimize $||\boldsymbol{o}_c - \boldsymbol{o}_p||$
  $\mathcal{J}_c \leftarrow$ joint with pose $(\text{ForwardOrientation}(\overleftrightarrow{z_c}, \hat{\mathbf{n}}), \boldsymbol{o}_c)$, type a_c, length l_c
  $\hat{\mathbf{a}}_c \leftarrow \hat{\mathbf{x}}_c$ if $a_c = x$, $\hat{\mathbf{z}}_c$ otherwise
  $\mathcal{L}_c \leftarrow$ radius-r extrusion of CSC path from $(\boldsymbol{o}_p^{\text{dist}}, \hat{\mathbf{a}}_p)$ to $(\boldsymbol{o}_c^{\text{prox}}, \hat{\mathbf{a}}_c)$
  Add $\mathcal{J}_c$ to $\boldsymbol{\mathcal{T}}$ as a child of $\mathcal{J}_p$ linked by $\mathcal{L}_c$
  $\mathcal{C} \leftarrow \text{ExpandCylinderToIncludeBall}(\mathcal{C}, \boldsymbol{o}_c, b_c)$ `//Alg. B.1`
  $\mathcal{C} \leftarrow \text{ExpandCylinderToIncludeLink}(\mathcal{C}, \mathcal{L}_c)$ `//Alg. B.2`
  $\mathcal{P}_c \leftarrow$ end plane of $\mathcal{C}$
end

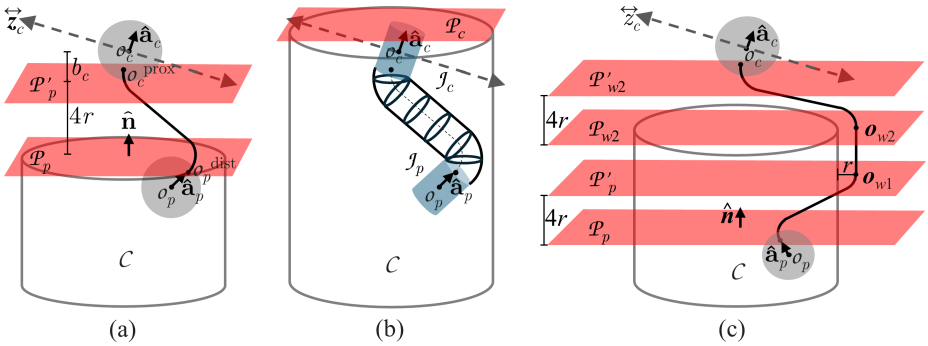


Fig. 2. (a) If $\mathcal{J}_c$ is the first child of $\mathcal{J}_p$, the algorithm places it along its axis of motion $\overleftrightarrow{z_c}$ as close as possible to $\mathcal{J}_p$ such that its entire bounding sphere is beyond $\mathcal{P}'_p = \mathcal{P}_p + 4r\hat{\mathbf{n}}$. (b) After placing $\mathcal{J}_c$ along its axis, it is joint to $\mathcal{J}_p$ with a tubular link tracing a CSC dubins path, and the bounding cylinder $\mathcal{C}$ expands to include the new link and the bounding ball of $\mathcal{J}_c$. (c) If $\mathcal{J}_c$ is not the first child of $\mathcal{J}_p$, it adds intermediate waypoints routing the path outside of and then around $\mathcal{C}$.

the parent joint $\mathcal{J}_p$, and r out from $\mathcal{C}$.Then it places the second waypoint on $\mathcal{P}$ directly in front of the first. Both waypoints are oriented with $\hat{\mathbf{a}}_i = \hat{\mathbf{z}}_i = \hat{\mathbf{n}}$, i.e., they face directly forward.

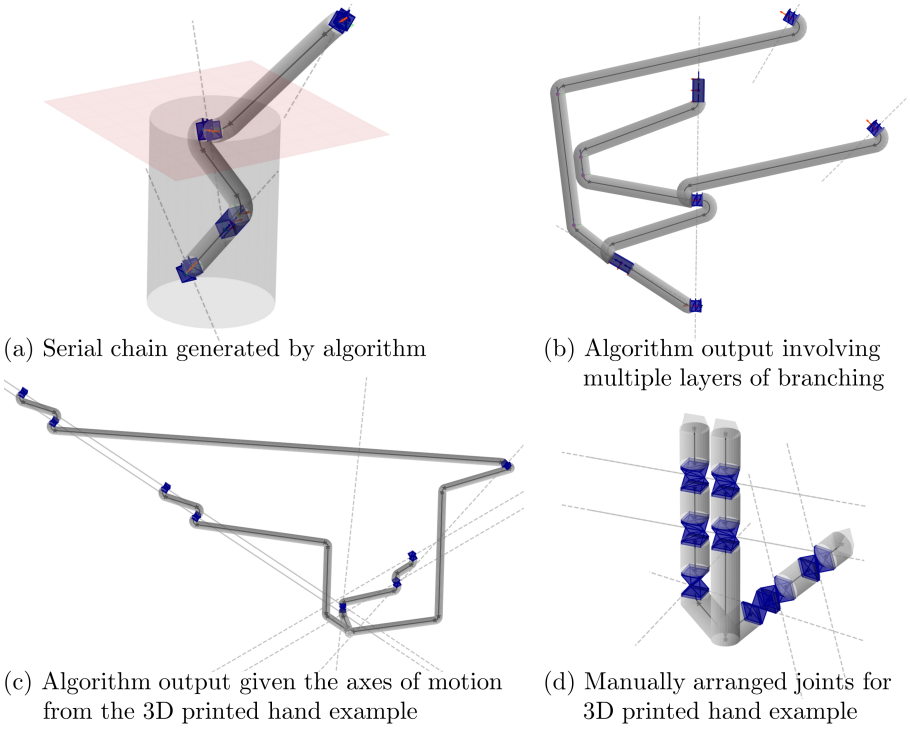
Then to place $\mathcal{J}_c$, the algorithm finds its bounding radius $b_c = \sqrt{\left(\frac{l_c}{2}\right)^2 + r^2}$ and selects joint position $\boldsymbol{o}_c \in \overleftrightarrow{z_c}$ such that its whole bounding sphere ball$(\boldsymbol{o}_c, b_c)$ is at least $4r$ in front of $\mathcal{P}$. Specifically (though arbitrarily), subject to those constraints the algorithm places $\boldsymbol{o}_c$ as close as possible to the immediate predecessor (its parent in $\boldsymbol{\mathcal{I}}$ if this is that parent's first child to be added, the second waypoint if not). The algorithm orients $\mathcal{J}_c$ to ensure that $\hat{\mathbf{a}}_c$ is not facing backwards: it sets $\hat{\mathbf{z}}_c$ to the forward direction of $\overleftrightarrow{z_c}$ and $\hat{\mathbf{x}}_c$ to the direction about $\overleftrightarrow{z_c}$ maximizing $\hat{\mathbf{x}}_c \cdot \hat{\mathbf{n}}$. This operation, called ForwardOrientation$(\overleftrightarrow{z_i}, \hat{\mathbf{n}})$, is written in algorithm form in supplementary materials Alg. B.3.

For each input joint the algorithm insterts the joint itself and at most three links and two waypoints, making the design complexity linear. It can be implemented with linear runtime provided the heuristics on lines 1 and 24 are replaced with procedures seeking any solution satisfying the constraints rather than a heuristically "optimal" one.

4 Implementation and Results

We implement Alg. 1 in Python, building on our code base from [11] for serial chains, and test it on several example inputs[2]. Results are shown in Fig. 3 (a-c). Examples include (a) a serial chain with no branching, (b) a binary tree with multiple layers and some intersecting and coincident joint axes (a "hard

[2]Parts of the code were written in the presence of GitHub Copilot auto-complete.



(a) Serial chain generated by algorithm

(b) Algorithm output involving multiple layers of branching

(c) Algorithm output given the axes of motion from the 3D printed hand example

(d) Manually arranged joints for 3D printed hand example

Fig. 3. Example tubular kinematic tree designs with the links in medium gray, joints in dark blue, and axes of motion as dashed lines. (a) also has the bounding cylinder and its end plane illustrated for the placement of its last joint. (a-c) are outputs of our algorithm, while (d) has joints manually placed along the same specified axes as (c).

case" for self-intersection avoidance), and (c) a tree with three branches from the same source and several joints in each of these branches. All inputs resulted in valid designs implementing the given kinematics without self-intersection in the neutral configuration. These examples are large (long limbed, with joints spaced far apart) due to the algorithm's conservative spacing of joints to guarantee self-intersection avoidance. For example, Fig. 3(d) shows joints manually placed along the same specification as Fig. 3(c). This is the example fabricated in supplementary materials Sec. A.

5 Existence of CSC Paths with Turn Angle $\leq \pi$

Alg. 1 works based on the premise that it is possible to grow an kinematic tree by sweeping a plane in its normal direction $\hat{\mathbf{n}}$ and placing new joints in front of it. Specifically, it ensures that each new joint is at least $4r$ in front of the previous and has attachment direction $\hat{\mathbf{a}}_i$ not facing backwards (i.e., $\hat{\mathbf{a}}_i \cdot \hat{\mathbf{n}} \geq 0$).

To show that this makes them connectable by links without local self-collision, we will require Thm. 1.

To prove Thm. 1, we will break down a CSC path into CS and SC components where the S segments must connect and align along a plane $\mathcal{P}$ between them. Since SC paths are just CS paths in reverse, we begin by proving properties of CS paths ending in $\mathcal{P}$, first in 2D and then in 3D. Fig. 4 illustrates notation.

Lemma 1 (2D CS Paths)**.** *In a plane Q, let $O \subset Q$ be an oriented circle with radius r parameterized by angle θ, and let $\overline{O} = Q - \text{interior}(O)$ be the portion of Q on and outside of O. Define the* proper tangent $\mathbf{q}'$ *of a point $\mathbf{q} \in O$ as $\mathbf{q}' = \frac{1}{r}\frac{dO}{d\theta}(\mathbf{q})$, the unit tangent agreeing with the circle orientation. Define $\mathbf{g} : \overline{O} \to O$ as the function such that the proper tangent from $\mathbf{g}(\mathbf{p})$ points to $\mathbf{p}$, i.e., such that $\mathbf{p} \in \text{ray}(\mathbf{g}(\mathbf{p}), \mathbf{g}(\mathbf{p})')$.*

1. *The function $\mathbf{g}$ is well defined (i.e., for every point $\mathbf{p} \in \overline{O}$, there exists a unique point $\mathbf{g}(\mathbf{p})$ on O whose proper tangent points to $\mathbf{p}$) and smooth.*
2. *Let $\ell \subset \overline{O}$ be a line intersecting O at most at a single point, and $\mathbf{g}_\ell$ be the restriction of $\mathbf{g}$ to ℓ. If ℓ does not intersect O, then $\mathbf{g}_\ell$ is injective.*

Proof. 1. Without loss of generality, say O is centered at the origin. Consider an arbitrary point $\mathbf{q} \in O$ in polar coordinates $\mathbf{q} = (r, \theta_\mathbf{q})$. For any $t \geq 0$, let

$$\mathbf{f}_t(\mathbf{q}) = \mathbf{q} + t\mathbf{q}' \in \overline{O} \tag{1}$$

be the result of travelling a distance t from $\mathbf{q}$ along $\mathbf{q}'$. We can write this map in polar coordinates as $\mathbf{f}_t : O \to \mathbb{R}_+ \times S^1$ given by

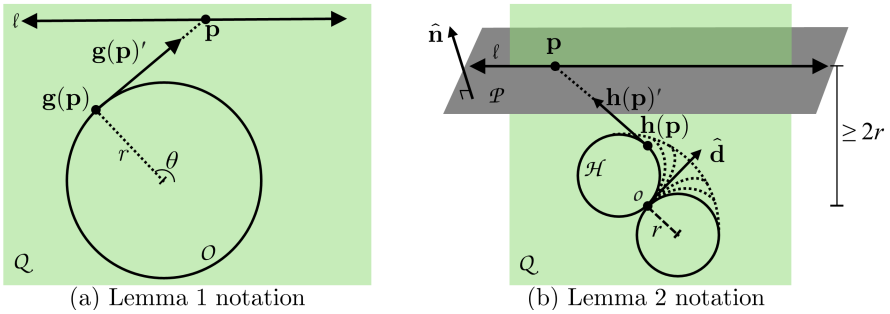
$$\mathbf{f}_t(\mathbf{q}) = (\sqrt{r^2 + t^2}, \arctan(t/r) + \theta_\mathbf{q}) \tag{2}$$

which is clearly injective (since r and t are fixed, it is just a constant scaling and rotation outwards). Its image is the circle about the origin of radius $\sqrt{r^2 + t^2}$, so we can define an inverse $\mathbf{f}_t^{-1}(\mathbf{p})$ from that larger circle to O.
Now, given any point $\mathbf{p} \in \overline{O}$ expressed in polar coordinates $\mathbf{p} = (||\mathbf{p}||, \theta_\mathbf{p})$, let $t = \sqrt{||\mathbf{p}||^2 - r^2}$.This gives $\mathbf{f}_t^{-1}(\mathbf{p}) \in O$ constructed to have proper tangent point to $\mathbf{p}$. This point is unique by injectivity of $\mathbf{f}_t$ and since $t = \sqrt{||\mathbf{p}||^2 - r^2}$ is the only choice for $t \geq 0$ making $||\mathbf{p}|| = \sqrt{r^2 + t^2}$.
This lets us define $\mathbf{g}$ by $\mathbf{g}(\mathbf{p}) = \mathbf{f}_t^{-1}(\mathbf{p})$. In polar coordinates, this is

$$\mathbf{g}(\mathbf{p}) = \mathbf{f}_t^{-1}(\mathbf{p}) = \left(r, \theta_\mathbf{p} - \arctan\frac{t}{r}\right) = \left(r, \theta_\mathbf{p} - \arctan\frac{\sqrt{||\mathbf{p}||^2 - r^2}}{r}\right) \tag{3}$$

which is smooth with respect to both $\theta_\mathbf{p}$ and $||\mathbf{p}||$ (since $||\mathbf{p}|| \geq r$).

2. Let $\ell \subset \overline{O} - O$ be a line not intersecting O, and suppose $\mathbf{p}_1, \mathbf{p}_2 \in \ell$ have $\mathbf{g}(\mathbf{p}_1) = \mathbf{g}(\mathbf{p}_2)$ which we will call $\mathbf{q}$. Then the proper tangent of $\mathbf{q}$ points to both $\mathbf{p}_1$ and $\mathbf{p}_2$, i.e., $\mathbf{p}_1, \mathbf{p}_2 \in \text{ray}(\mathbf{q}, \mathbf{q}') \subseteq \ell \cap \text{ray}(\mathbf{q}, \mathbf{q}')$. Since ℓ does not intersect O, it cannot be colinear with this ray, so it can only intersect it at a single point. Therefore $\mathbf{p}_1 = \mathbf{p}_2$. □



(a) Lemma 1 notation (b) Lemma 2 notation

Fig. 4. Notation for (a) Lem. 1 and (b) Lem. 2.

Next we will consider how CS paths behave in 3D. The radius-r arcs from a point-direction pair $(\mathbf{o}, \hat{\mathbf{d}})$ form a horn torus $\mathcal{H}$ parameterizable by arc turn angle θ and azumith angle φ about line$(\mathbf{o}, \hat{\mathbf{d}})$. For example, in a frame where $\mathbf{o} = \mathbf{0}$ and $\hat{\mathbf{d}} = (0, 0, 1)$, the horn torus is

$$\mathcal{H}(\theta, \varphi) = r \begin{pmatrix} (1 + \cos\theta)\cos\varphi \\ (1 + \cos\theta)\sin\varphi \\ \sin\theta \end{pmatrix}. \tag{4}$$

The following lemma gives smooth maps from $\mathcal{P}$ to points and tangents on $\mathcal{H}$ corresponding to CS paths to $\mathcal{P}$ (see Fig. 4 for notation illustration).

Lemma 2 (3D CS Paths)**.** *Let $(\mathbf{o}, \hat{\mathbf{d}})$ be a 3D Dubins pose and $\mathcal{P}$ be a plane. Suppose $\mathbf{o}$ is at least $2r$ from $\mathcal{P}$. Let $\mathcal{H} : S^1 \times S^1 \to \mathbb{R}^3$ be the horn torus of radius-r arcs from $(\mathbf{o}, \hat{\mathbf{d}})$, parameterized as $\mathcal{H}(\theta, \varphi)$ by turn angle θ and azumith φ. Then:*

1. *There exists a function $\mathbf{h} : \mathcal{P} \to \mathcal{H}$ taking a given $\mathbf{p} \in \mathcal{P}$ and outputting a point $\mathbf{q} = \mathbf{h}(\mathbf{p}) = \mathcal{H}(\theta_\mathbf{q}, \varphi_\mathbf{q})$ on the torus such that:*
   (a) $\mathbf{q}$ is the arc endpoint of a CS path from $(\mathbf{o}, \hat{\mathbf{d}})$ to $\mathbf{p}$, i.e.,

$$\mathbf{p} = \mathbf{q} + t\frac{\partial \mathcal{H}}{\partial \theta}(\theta_\mathbf{q}, \varphi_\mathbf{q}) \tag{5}$$

   for some real $t \geq 0$, and
   (b) $\mathbf{q}$ and $\mathbf{p}$ have the same azumith $\varphi_\mathbf{q} = \varphi_\mathbf{p}$.
2. *Let $\hat{\mathbf{n}}$ be the normal vector to $\mathcal{P}$ pointing towards the opposite side to $\mathbf{o}$. If $\hat{\mathbf{n}} \cdot \hat{\mathbf{d}} \geq 0$, then the C arc in such a path has turn angle $\theta_\mathbf{q} \in [0, \pi]$.*
3. *If $\mathcal{H}$ does not intersect $\mathcal{P}$, then $\mathbf{h}$ is injective.*
4. *If $\hat{\mathbf{n}} \cdot \hat{\mathbf{d}} \geq 0$, then $\mathbf{h}$ is smooth.*
5. *Suppose $\hat{\mathbf{n}} \cdot \hat{\mathbf{d}} \geq 0$, and let $\hat{\mathbf{f}} : \mathcal{P} \to S^2$ be given by*

$$\hat{\mathbf{f}}(\mathbf{p}) = \frac{1}{r}\frac{\partial \mathcal{H}}{\partial \theta}(\mathbf{h}(\mathbf{p})), \tag{6}$$

   the S unit direction of the CS path implicitly defined by $\mathbf{h}$. Then $\hat{\mathbf{f}}$ is smooth.

Proof. Note that since $\mathbf{o}$ is at least $2r$ from $\mathcal{P}$, the only way $\mathcal{H}$ and $\mathcal{P}$ could intersect is if $\mathbf{o}$ is exactly $2r$ from $\mathcal{P}$ and $\mathcal{P}$ is parallel to $\hat{\mathbf{d}}$. In this case, $\mathcal{H}$ and $\mathcal{P}$ intersect at a single point with turn angle $\theta = \pi$.

Given $\mathbf{p} \in \mathcal{P}$, let $\varphi_{\mathbf{p}}$ be its azumith angle about $(\mathbf{o}, \hat{\mathbf{d}})$ (if $\mathbf{p}$ is along line$(\mathbf{o}, \hat{\mathbf{d}})$, the azumith is not well defined but choosing any angle for it works for the following). Let $Q \subset \mathbb{R}^3$ be the plane defined by azumith $\varphi_{\mathbf{p}}$, and let $O = \mathcal{H}(S^1, \varphi)$ be the circle on $\mathcal{H}$ with azumith $\varphi_{\mathbf{p}}$ (there are two torus circles on Q, one in direction $\varphi_{\mathbf{p}}$ and the other in direction $\pi + \varphi_{\mathbf{p}}$: O is the former). Note that $\ell = Q \cap \mathcal{P}$ is a line (the planes must intersect because they both contain $\mathbf{p}$, but cannot coincide completely because Q contains $\mathbf{o} \notin \mathcal{P}$).

1. Applying Lem. 1 in plane Q to line ℓ, there exists a point $\mathbf{g}_\ell(\mathbf{p})$ whose proper tangent $\mathbf{g}_\ell(\mathbf{p})'$ points to $\mathbf{p}$. Then the arc along O to $\mathbf{g}_\ell(\mathbf{p})$, followed by the segment in direction $\mathbf{g}_\ell(\mathbf{p})'$ to $\mathbf{p}$, is a CS path from $(\mathbf{o}, \hat{\mathbf{d}})$ to $(\mathbf{p}, \mathbf{g}_\ell(\mathbf{p})')$. So we can define $\mathbf{h}(\mathbf{p}) = \mathbf{g}_\ell(\mathbf{p})$. □
2. Consider a point $\mathbf{q} = \mathcal{H}(\theta_{\mathbf{q}}, \varphi_{\mathbf{q}})$ with turn angle $\theta_{\mathbf{q}} \in (\pi, 2\pi)$, and suppose that its proper tangent points towards $\mathcal{P}$, i.e., ray$(\mathbf{q}, \mathbf{q}')$ intersects $\mathcal{P}$. Let $\mathbf{p}$ be the point of intersection: it must be a single point (rather than the whole ray) because $\theta_{\mathbf{q}} \neq \pi$ so $\mathbf{p} \notin \mathcal{P}$ which means ray$(\mathbf{q}, \mathbf{q}') \not\subset \mathcal{P}$.
   Having turn angle $\theta_{\mathbf{q}} \in (\pi, 2\pi)$ means ray$(\mathbf{q}, \mathbf{q}')$ points "backwards" to intersect the Dubins axis line$(\mathbf{o}, \hat{\mathbf{d}})$ behind $\mathbf{o}$. We will see below that if $\hat{\mathbf{n}} \cdot \hat{\mathbf{d}} \geq 0$ then $\mathbf{q}$ and $\mathbf{p}$ have opposite azumiths (i.e., in the $\varphi = \varphi_{\mathbf{q}}$ plane they are separated by the Dubins axis). Since $\mathbf{h}$ is defined to preserve azumith, this means $\mathbf{q} \neq \mathbf{h}(\mathbf{p})$. But $\mathbf{p}$ is the only point in $\mathcal{P}$ that the proper tangent from $\mathbf{q}$ points to, so $\mathbf{q} \notin \text{Im}(\mathbf{h})$. Since no point with turn angle in $(\pi, 2\pi)$ is in Im$(\mathbf{h})$, all of the CS paths implicitly defined by $\mathbf{h}$ have turn angle in $[0, \pi]$.
   Suppose $\hat{\mathbf{n}} \cdot \hat{\mathbf{d}} = 0$, i.e., $\mathcal{P}$ is parallel to the Dubins axis. Since ray$(\mathbf{q}, \mathbf{q}')$ points backwards towards the Dubins axis and intersects $\mathcal{P}$, and $\mathcal{P}$ is at least $2r$ from $\mathbf{o}$ and never intersects the Dubins axis, ray$(\mathbf{q}, \mathbf{q}')$ must cross the Dubins axis before reaching $\mathcal{P}$, so the azumith of $\mathbf{p}$ is $\varphi_{\mathbf{q}} + \pi$, not $\varphi_{\mathbf{q}}$.
   Alternatively, suppose $\hat{\mathbf{n}} \cdot \hat{\mathbf{d}} > 0$. Since $\mathbf{o}$ is behind $\mathcal{P}$ relative to the $\hat{\mathbf{n}}$ direction by at least $2r$, having $\hat{\mathbf{n}} \cdot \hat{\mathbf{d}} > 0$ means $\mathcal{P}$ intersects the Dubins axis in front of $\mathbf{o}$: call this intersection point $\mathbf{p}_2$. Since ray$(\mathbf{q}, \mathbf{q}')$ intersects the Dubins axis behind $\mathbf{o}$, for $\mathbf{p}$ to occur between $\mathbf{q}$ and the Dubins axis means that it is strictly behind $\mathcal{H}$ relative to $\hat{\mathbf{d}}$, so the line ℓ from $\mathbf{p}$ to $\mathbf{p}_2$ would pass through $\mathcal{H}$. But $\mathbf{p}, \mathbf{p}_2$ are both on $\mathcal{P}$ so $\ell \subset \mathcal{P}$, so this would mean $\mathcal{P}$ intersects $\mathcal{H}$, which we noted above can only happen when $\mathcal{P}$ is parallel to $\hat{\mathbf{d}}$ (which contradicts $\hat{\mathbf{n}} \cdot \hat{\mathbf{d}} > 0$). □
3. Suppose $\mathcal{H}$ does not intersect $\mathcal{P}$, and let $\mathbf{p}_1, \mathbf{p}_2$ be distinct points in $\mathcal{P}$. Since the Dubins axis intersects $\mathcal{P}$ at most once, at least one of them is not at this intersection and thus has well-defined azumith. If their azumiths differ, then $\mathbf{h}(\mathbf{p}_1)$ and $\mathbf{h}(\mathbf{p}_2)$ will have distinct azumiths (since $\mathbf{h}$ preserves azumith) so $\mathbf{h}(\mathbf{p}_1) \neq \mathbf{h}(\mathbf{p}_2)$. Otherwise they have the same azumith, so we can apply Lem. 1 in their shared azumith plane Q with $\ell = \mathcal{P} \cap Q$ and get $\mathbf{h}(\mathbf{p}_1) = \mathbf{g}_\ell(\mathbf{p}_1) \neq \mathbf{g}_\ell(\mathbf{p}_2) = \mathbf{h}(\mathbf{p}_2)$ because $\mathbf{g}_\ell$ is injective (since $\ell \subset \mathcal{P}$ does not intersect $\mathcal{H}$). □

4. Suppose $\hat{\mathbf{n}} \cdot \hat{\mathbf{d}} \geq 0$. Let $\tau \subseteq \mathcal{H}$ be the points on $\mathcal{H}$ whose proper tangents point towards $\mathcal{P}$.First we will deal with the case where $\mathcal{H}$ does not intersect $\mathcal{P}$, and therefore restricting the codomain of $\mathbf{h}$ to τ makes $\mathbf{h}$ bijective. In this case it has inverse $\mathbf{h}^{-1} : \tau \to \mathcal{P}$, and parameterizing τ by turn angle θ and azumith φ gives a bijective parameterization of $\mathcal{P}$ as $\mathbf{h}^{-1}(\theta, \varphi)$. Now to show $\hat{\mathbf{f}}$ is smooth, it suffices to show smoothness with respect to each parameter. First, we will see that $\mathbf{h}$ is smooth with respect to the azumith φ. Note that fixing θ along $\mathcal{H}$ defines a cone about the Dubins axis. The intersection of this cone with $\mathcal{P}$ is a smooth curve (a half-hyperbola if $\hat{\mathbf{n}} \cdot \hat{\mathbf{d}} = 0$, an ellipse if $\hat{\mathbf{n}} \cdot \hat{\mathbf{d}} > 0$), so travelling along this curve in $\mathcal{P}$ represents rotating φ for fixed θ. Since rotating φ for fixed θ defines a circle (which is a smooth curve) along $\mathcal{H}$, $\mathbf{h}$ is smooth with respect to φ.
   Second, we will consider varying θ within a plane Q with fixed $\varphi = \varphi_{\mathbf{p}}$. Within Q, there are two torus circles: $O = \mathcal{H}(S^1, \varphi_{\mathbf{p}})$ is the one facing towards $\mathbf{p}$, but there is also ${}^{+\pi}O = \mathcal{H}(S^1, \varphi_{\mathbf{p}} + \pi)$ on the other side of $\mathbf{o}$. By Lem. 1, we know $\mathbf{g}_\ell$ is smooth, and taking the tangent with respect to θ is smooth, so $\mathbf{h}$ is smooth along the subset of $\ell = Q \cap \mathcal{P}$ with azumith φ (as opposed to $\varphi + \pi$). If $\hat{\mathbf{n}} \cdot \hat{\mathbf{d}} = 0$, this subset is all of ℓ so we have smoothness of $\hat{\mathbf{f}}$ with respect to β. Otherwise, we have to deal with where ℓ crosses over the Dubins axis. Let $\mathbf{s} = \text{ray}(\mathbf{o}, \hat{\mathbf{d}}) \cap \ell$, and note that this is generated by $\theta = 0$. At $\mathbf{s}$ the azumith is undefined, and on each side of it on ℓ the azumiths are φ and $\varphi + \pi$ respectively. Since the respective circles meet at $\mathbf{o}$ which has $\theta = 0$, the map $\mathbf{g}_\ell$ from either circle agrees there, witnessing the continuity of $\mathbf{h}$ at $\mathbf{s}$. Furthermore, along each circle we have $\frac{\partial \mathbf{g}_\ell}{\partial \theta}(0, \varphi) = r\hat{\mathbf{d}}$, so $\frac{\partial \mathbf{h}}{\partial \theta} = r\hat{\mathbf{d}}$ is well-defined. Similarly, the symmetry between the circles (along with a direction flip of θ to account for the fact that travelling along ℓ decreases θ along one circle but then increases it along the other) gives agreement between sides of the subsequent derivatives with respect to θ.
   Finally we return to the case where $\mathcal{H}$ intersects $\mathcal{P}$. This can only happen if $\hat{\mathbf{n}} \cdot \hat{\mathbf{d}} = 0$ and they intersect at a single point $\mathbf{p}_0$. Letting $\ell_0 = \text{ray}(\mathbf{p}_0, \mathbf{p}_0')$, note that change in azumith is orthogonal to ℓ_0, and thus sweeping azumith defines $\mathcal{P}$ by a line sweep. Therefore we can bijectively parameterize $\mathcal{P}$ by azumith φ and distance s along the azumith-constant line. The same analysis as above shows how $\mathbf{h}$ is smooth with respect to φ. For smoothness with respect to s, the only place where this differs from the above analysis for θ is that $\text{ray}(\mathbf{p}_0, \mathbf{p}_0') \subseteq \mathcal{P}$ differs in s but has the same turn angle π. This exception does not break smoothness of $\mathbf{h}$ because $\mathbf{p}_0$ is on the boundary of τ (since increasing θ past $\mathbf{p}_0$ makes the proper tangent not point to $\mathcal{P}$). □
5. We have shown $\mathbf{h}$ is smooth. The partial derivative $\frac{\partial \mathcal{H}}{\partial \theta}$ is also smooth, as is multiplying by $\frac{1}{r}$. So $\hat{\mathbf{f}}$ is the composition of smooth maps, and therefore it is smooth. □

Now we can use the smoothness of the $\hat{\mathbf{f}}$ maps to find a point on a plane where the CS and SC paths on each side connect and align, proving Thm. 1:

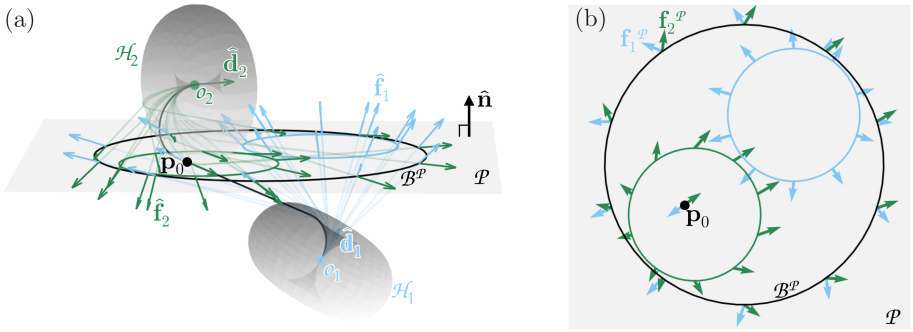


Fig. 5. Notation for the proof of Thm. 1, (a) in 3D and (b) in plane $\mathcal{P}$.

Theorem 1 (3D CSC Paths With Bounded Turns). *Let $\mathcal{P}$ be a plane with normal $\hat{\mathbf{n}}$. Let $\mathbf{o}_1, \mathbf{o}_2 \in \mathbb{R}^3$ be positions and $\hat{\mathbf{d}}_1, \hat{\mathbf{d}}_2 \in S^2$ be 3D unit vectors indicating direction. Suppose $\mathbf{o}_1$ is at least $2r$ behind $\mathcal{P}$, $\mathbf{o}_2$ is at least $2r$ in front of plane, $\mathbf{o}_2$ at least $2r$ in front of $\mathcal{P}$, $\hat{\mathbf{n}} \cdot \hat{\mathbf{d}}_1 \geq 0$, and $\hat{\mathbf{n}} \cdot \hat{\mathbf{d}}_2 \geq 0$. Then there exists a CSC path whose turn angles are $\leq \pi$ from $\mathbf{o}_1$ facing $\hat{\mathbf{d}}_1$ to $\mathbf{o}_1$ facing $\hat{\mathbf{d}}_1$.*

Proof. A valid CSC path is a CS path from $(\mathbf{o}_1, \hat{\mathbf{d}}_1)$ to $(\mathbf{p}, \hat{\mathbf{t}})$ where $\mathbf{p} \in \mathcal{P}$, then an SC path from $(\mathbf{p}, \hat{\mathbf{t}})$ to $(\mathbf{o}_2, \hat{\mathbf{d}}_2)$. Let $\mathcal{H}_1, \mathcal{H}_2$ be the horn tori of radius-r arcs from $(\mathbf{o}_1, \hat{\mathbf{d}}_1)$ and $(\mathbf{o}_2, -\hat{\mathbf{d}}_2)$ respectively. Since $\mathbf{o}_1$ is at least $2r$ behind $\mathcal{P}$ and $\hat{\mathbf{n}} \cdot \hat{\mathbf{d}}_1 \geq 0$, Lem. 2 says there exists a smooth map $\hat{\mathbf{f}}_1 : \mathcal{P} \to S^2$ giving the S direction of a CS path from $(\mathbf{o}_1, \hat{\mathbf{d}}_1)$ to a given $\mathbf{p} \in \mathcal{P}$. Considering the SC part of the path in reverse and $\mathcal{P}$ oriented with $-\hat{\mathbf{n}}$, Lem. 2 similarly gives a smooth map $\hat{\mathbf{f}}_2 : \mathcal{P} \to S^2$ giving the S direction of a CS path from $(\mathbf{o}_2, -\hat{\mathbf{d}}_2)$ to a given $\mathbf{p} \in \mathcal{P}$. We are looking for a point $\mathbf{p} \in \mathcal{P}$ inducing CS and SC paths that align, i.e., a zero of the map $\mathbf{s} = \hat{\mathbf{f}}_1 + \hat{\mathbf{f}}_2$. Note $\mathbf{s}$ is smooth because $\hat{\mathbf{f}}_1$ and $\hat{\mathbf{f}}_2$ are.

Let $\mathbf{f}_1^{\mathcal{P}}, \mathbf{f}_2^{\mathcal{P}}, \mathbf{s}^{\mathcal{P}}$ be the projections of $\hat{\mathbf{f}}_1, \hat{\mathbf{f}}_2, \mathbf{s}$ onto $\mathcal{P}$ (smooth vector fields on $\mathcal{P}$). Let $\mathcal{H}_1^{\mathcal{P}}, \mathcal{H}_2^{\mathcal{P}}$ be the projections of the tori onto $\mathcal{P}$. Let $\mathcal{B}$ be any bounding ball including both $\mathcal{H}_1$ and $\mathcal{H}_2$ strictly in the interior, and let $\mathcal{B}^{\mathcal{P}}$ be its projection (a disc) onto $\mathcal{P}$. Now consider any point $\mathbf{p} \in \mathcal{B}^{\mathcal{P}} \subset \mathcal{P}$. Since $\mathbf{f}_1^{\mathcal{P}}(\mathbf{p})$ is the direction from a point on $\mathcal{H}_1^{\mathcal{P}}$ (in the strict interior of $\mathcal{B}^{\mathbf{P}}$) to $\mathbf{p}$ which is on the boundary, $\mathbf{f}_1^{\mathcal{P}}(\mathbf{p})$ is transverse (non-tangent) to the circle and pointed outwards. The same is true for $\mathbf{f}_2^{\mathcal{P}}(\mathbf{p})$, and thus for $\mathbf{s}^{\mathcal{P}}$. So $\mathbf{s}^{\mathcal{P}}$ is a vector field pointing transversely outwards along the boundary of $\mathcal{B}^{\mathcal{P}}$, so the Poincaré–Hopf Theorem [4] says it must have a zero at some point $\mathbf{p}_0 \in \mathcal{B}^{\mathcal{P}}$. Since $\hat{\mathbf{f}}_1(\mathbf{p}_0)$ and $\hat{\mathbf{f}}_2(\mathbf{p}_0)$ are both unit vectors and they point on opposite sides of $\mathcal{P}$, $\mathbf{f}_1^{\mathcal{P}}(\mathbf{p}_0) + \mathbf{f}_2^{\mathcal{P}}(\mathbf{p}_0) = 0$ implies that $\hat{\mathbf{f}}_1(\mathbf{p}_0) + \hat{\mathbf{f}}_2(\mathbf{p}_0) = \mathbf{s}(\mathbf{p}_0) = 0$ as desired. This forms a CSC path from $(\mathbf{o}_1, \hat{\mathbf{d}}_1)$ to $(\mathbf{o}_2, \hat{\mathbf{d}}_2)$ with S direction $\hat{\mathbf{f}}_1(\mathbf{p}_0) = -\hat{\mathbf{f}}_2(\mathbf{p}_0)$ intersecting $\mathcal{P}$ at $\mathbf{p}_0$. □

6 Proof of Tubular Tree Construction Algorithm

Theorem 1 justifies how we space out the joints to ensure the existence of valid link paths, so we are now ready to prove the correctness of Alg. 1.

Theorem 2 (Tubular Kinematic Tree Construction). *Given a tubular radius r and a tubular tree kinematic specification $\boldsymbol{\mathcal{I}}$, Alg. 1 outputs a tubular kinematic tree design $\boldsymbol{\mathcal{T}}$ composed of radius-r tubes implementing the input specifications with no self-intersection in the neutral configuration.*

Proof. Our algorithm places joints with $\overleftrightarrow{z_i} = \ell(\boldsymbol{o}_i, \hat{\mathbf{z}}_i)$, and links joints based on the input tree structure, ensuring the result satisfies the kinematic specification. So showing correctness means verifying that (1) the construction succeeds (i.e., that all geometric operations are valid) and (2) that the resulting tree has no self-intersection in the neutral configuration.

1. Most of our operations are direct constructions, but three require justification that solutions exist: (a) the choice of $\hat{\mathbf{n}}$ (line 1), (b) the optimization placing $\boldsymbol{o}_c$ (line 24), and (c) the existence of CSC paths for links (lines 11,27).
   (a,b) The choices of $\hat{\mathbf{n}}$ and $\boldsymbol{o}_c$ are closely related: $\boldsymbol{o}_c$ is constrained to be on $\overleftrightarrow{z_c}$ and at least $4r + b_c$ in front of $\mathcal{P}_p$. So a solution is guaranteed to exist if $\overleftrightarrow{z_c}$ crosses $\mathcal{P}_p + (4r + b_c)\hat{\mathbf{n}}$, i.e., if $\overleftrightarrow{z_c}$ is not orthogonal to $\hat{\mathbf{n}}$. Since there are finitely many joint axes, there must exist a direction not orthogonal to any of them (we prove this in supplementary materials Sec. C), i.e., where $\hat{\mathbf{n}} \cdot \hat{\mathbf{z}}_i > 0$ for all axes. Choosing $\hat{\mathbf{n}}$ to maximize the minimum $|\hat{\mathbf{n}} \cdot \hat{\mathbf{z}}_i|$ achieves this.
   (c) To avoid local self-intersection, we need paths with turn angles $\leq \pi$. Since waypoint 2 is placed directly in front of waypoint 1, an S path connects them. All other link cases are constructed such that the child's proximal position is $\geq 4r$ farther (in $\hat{\mathbf{n}}$) than the parent's distal position. The algorithm orients each joint with $\hat{\mathbf{a}}_i \cdot \hat{\mathbf{n}} \geq 0$, so Thm. 1 (Sec. 5) guarantees that there exists a CSC path from the parent to the child with turn angles $\leq \pi$.
2. To show there is no self-intersection, we first need to see that the bounding cylinder correctly includes everything generated so far. It is expanded to encompass the bounding balls of each new joint and of each link C arc: the nontrivial part is to show that this also includes the link S components. Since an S component in a CSC link is the convex combination of its end discs and the end discs are contained in the C components' respective bounding balls, any convex set (including our cylinder) including the C bounding balls also includes the S components.
   Now we show by induction that there is no self-intersection except between links from the same parent (which are allowed to intersect because they are inherently rigidly attached). In the base case, the structure consists only of the root joint so there is nothing else for it to intersect with. The inductive step is to add a new joint $\mathcal{J}_c$ as a child of $\mathcal{J}_p$. Suppose as an inductive hypothesis that the currently-generated structure has no self-intersection

except between links from the same parent. Since the new link(s) and joint we add for $\mathcal{J}_c$ are sequentially connected and move strictly forward, the new structure does not intersect with itself. It remains to show that this does not intersect anything previously generated except the allowed link intersection at the distal end of $\mathcal{J}_p$.
Since links have C arc angles $\leq \pi$ and joints cannot face backwards (i.e., $\hat{\mathbf{a}}_i \cdot \hat{\mathbf{n}} \geq 0$), an outgoing link does not intersect with its parent joint $\mathcal{J}_p$ or the incoming link to $\mathcal{J}_p$. Since the tree is generated depth-first and joints are placed with bounding spheres are strictly increasing along $\hat{\mathbf{n}}$, we know that anything that currently exists past $\mathcal{P}_p$ is a descendent of $\mathcal{J}_p$. Furthermore, since every child of p has bounding sphere beyond $\mathcal{P}'_p = \mathcal{P}_p + 4r\hat{\mathbf{n}}$, anything that already exists between $\mathcal{P}_p$ and $\mathcal{P}'_p$ is part of another link directly from $\mathcal{J}_p$ (which is permissible for the new link to intersect). Then the new link exits $\mathcal{C}$ either in front (if this is the first child of $\mathcal{J}_p$, in which case there is nothing in front to intersect with) or out the side in between $\mathcal{P}_p$ and $\mathcal{P}'_p$. Then since the subsequent waypoints, links, and new joint are outside of $\mathcal{C}$, they cannot intersect anything previously generated. □

7 Conclusion

We prove that any kinematic specification with a tree morphology can be implemented as a tubular structure of a given radius, justifying the validity and generality of abstracting kinematic tree designs by their tubular skeleton. The existence guarantee is based on a design algorithm which has essentially linear runtime and gives linear-complexity designs, making it suitable as a step within a system with more computationally intensive goals helping non-experts design robots and helping experts explore high-degree-of-freedom morphologies.

As future work, we plan to incorporate additional constraints and objectives in a design pipeline giving more practical results. On the input end, this may leverage existing kinematic synthesis techniques [21,26] to turn task specifications into axes of motion which our algorithm then turns into a concrete design. On the design end, this may include post-generation optimization to minimize length, maximize compactness, or satisfy a space constraint. We are also interested in accounting for reachability and maneuverability properties (i.e., ensuring self-collision avoidance in certain non-neutral configurations) and mechanical analysis of actuators and forces. Finally, we plan to incorporate our algorithms into an interactive graphical tool for human-in-the-loop design which outputs fabricable files for origami or 3D printing.

Acknowledgments. Support for this project has been provided in part by NSF Grant No. 2322898 and in part by the Army Research Office under the SLICE Multidisciplinary University Research Initiatives Program award under Grant W911NF1810327. The authors also thank Gabriel Unger for fabrication advice, and thank a reviewer for a much shorter and simpler proof of Lem. 3 (in supplementary materials).

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. de Almeida, T.C., Marri, S., Kress-Gazit, H.: Automated synthesis of modular manipulators' structure and control for continuous tasks around obstacles. In: Robotics: Science and Systems (RSS) (2020). https://doi.org/10.15607/RSS.2020.XVI.030
2. Bächer, M., Coros, S., Thomaszewski, B.: Linkedit: Interactive linkage editing using symbolic kinematics. ACM Transactions on Graphics **34**(4), 1–8 (2015). https://doi.org/10.1145/2766985
3. Baez, V.M., Navkar, N., Becker, A.T.: An analytic solution to the 3D CSC Dubins path problem. In: IEEE International Conference on Robotics and Automation (ICRA). pp. 7157–7163 (2024). https://doi.org/10.1109/ICRA57147.2024.10611360
4. Brasselet, J.P., Seade, J., Suwa, T.: Vector fields on Singular Varieties, Lecture Notes in Mathematics, vol. 1987. Springer Berlin, Heidelberg (2009). https://doi.org/10.1007/978-3-642-05205-7
5. Cai, W., Zhang, M., Zheng, Y.R.: Task assignment and path planning for multiple autonomous underwater vehicles using 3D Dubins curves. Sensors **17**(7) (2017). https://doi.org/10.3390/s17071607
6. Censi, A.: A class of co-design problems with cyclic constraints and their solution. IEEE Robotics and Automation Letters **2**(1), 96–103 (2016). https://doi.org/10.1109/LRA.2016.2535127
7. Chen, W.H., Yang, W., Peach, L., Koditschek, D.E., Sung, C.R.: Kinegami: Algorithmic design of compliant kinematic chains from tubular origami. IEEE Transactions on Robotics **39**(2), 1260–1280 (2023). https://doi.org/10.1109/TRO.2022.3206711
8. Chitsaz, H., LaValle, S.M.: Time-optimal paths for a Dubins airplane. In: IEEE Conference on Decision and Control (CDC). pp. 2379–2384 (2007). https://doi.org/10.1109/CDC.2007.4434966
9. Cui, P., Yan, W., Wang, Y., et al.: Reactive path planning approach for docking robots in unknown environment. Journal of Advanced Transportation (2017). https://doi.org/10.1155/2017/6716820
10. Dubins, L.E.: On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. American Journal of Mathematics **79**(3), 497–516 (1957). https://doi.org/10.2307/2372560
11. Feshbach, D., Chen, W.H., Koditschek, D.E., Sung, C.: Kinegami: Open-source software for creating kinematic chains from tubular origami. In: 8th International Meeting on Origami in Science, Mathematics, and Education (8OSME) (2024), https://github.com/SungRoboticsGroup/KinegamiPython
12. Geilinger, M., Poranne, R., Desai, R., Thomaszewski, B., Coros, S.: Skaterbots: optimization-based design and motion synthesis for robotic creatures with legs and wheels. ACM Transactions on Graphics **37**(4), 1–12 (2018). https://doi.org/10.1145/3197517.3201368
13. Gupta, S., Singla, E.: Evolutionary robotics in two decades: a review. Sadhana **40**, 1169–1184 (2015). https://doi.org/10.1007/s12046-015-0357-7
14. Ha, S., Coros, S., Alspach, A., Bern, J.M., Kim, J., Yamane, K.: Computational design of robotic devices from high-level motion specifications. IEEE Transactions on Robotics **34**(5), 1240–1251 (2018). https://doi.org/10.1109/TRO.2018.2830419
15. Ha, S., Coros, S., Alspach, A., Kim, J., Yamane, K.: Joint optimization of robot design and motion parameters using the implicit function theorem. In: Robotics: Science and Systems (RSS) (2017). https://doi.org/10.15607/rss.2017.xiii.003

16. Hota, S., Ghose, D.: Optimal geometrical path in 3D with curvature constraint. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (2010). https://doi.org/10.1109/IROS.2010.5653663
17. Indig, N., Ben-Asher, J.Z., Sigal, E.: Near-optimal minimum-time guidance under spatial angular constraint in atmospheric flight. Journal of Guidance, Control, and Dynamics **39**(7), 1563–1577 (2016). https://doi.org/10.2514/1.G006218
18. Karapetyan, N., Moulton, J., Lewis, J.S., Quattrini Li, A., O'Kane, J.M., Rekleitis, I.: Multi-robot Dubins coverage with autonomous surface vehicles. In: IEEE International Conference on Robotics and Automation (ICRA) (2018). https://doi.org/10.1109/ICRA.2018.8460661
19. Lipson, H., Pollack, J.B.: Automatic design and manufacture of robotic lifeforms. Nature **406**(6799), 974–978 (2000). https://doi.org/10.1038/35023115
20. Lugo-Cárdenas, I., Flores, G., Salazar, S., Lozano, R.: Dubins path generation for a fixed wing UAV. In: International Conference on Unmanned Aircraft Systems (ICUAS). pp. 339–346 (2014). https://doi.org/10.1109/ICUAS.2014.6842272
21. McCarthy, J.M., Soh, G.S.: Geometric design of linkages, Interdisciplinary Applied Mathematics, vol. 11. Springer New York, NY (2010). https://doi.org/10.1007/978-1-4419-7892-9
22. Mehta, A., DelPreto, J., Rus, D.: Integrated codesign of printable robots. Journal of Mechanisms and Robotics **7**(2), 021015 (2015). https://doi.org/10.1115/1.4029496
23. Owen, M., Beard, R.W., McLain, T.W.: Implementing Dubins airplane paths on fixed-wing UAVs. In: Valavanis, K.P., Vachtsevanos, G.J. (eds.) Handbook of Unmanned Aerial Vehicles, pp. 1677–1701. Springer, Dordrecht (2015). https://doi.org/10.1007/978-90-481-9707-1_120
24. Schulz, A., Sung, C., Spielberg, A., Zhao, W., Cheng, R., Grinspun, E., Rus, D., Matusik, W.: Interactive Robogami: An end-to-end system for design of robots with ground locomotion. The International Journal of Robotics Research **36**(10), 1131–1147 (2017). https://doi.org/10.1177/0278364917723465
25. Shkel, A.M., Lumelsky, V.: Classification of the Dubins set. Robotics and Autonomous Systems **34**(4), 179–202 (2001). https://doi.org/10.1016/S0921-8890(00)00127-5
26. Simo-Serra, E., Perez-Gracia, A.: Kinematic synthesis using tree topologies. Mechanism and Machine Theory **72**, 94–113 (2014). https://doi.org/10.1016/j.mechmachtheory.2013.10.004
27. Sussmann, H.: Shortest 3-dimensional paths with a prescribed curvature bound. In: IEEE Conference on Decision and Control (CDC) (1995). https://doi.org/10.1109/CDC.1995.478997
28. Tosun, T., Jing, G., Kress-Gazit, H., Yim, M.: Computer-aided compositional design and verification for modular robots. In: Robotics Research: Volume 1, pp. 237–252. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-51532-8_15
29. Váňa, P., Alves Neto, A., Faigl, J., Macharet, D.G.: Minimal 3D Dubins path with bounded curvature and pitch angle. In: IEEE International Conference on Robotics and Automation (ICRA). pp. 8497–8503 (2020). https://doi.org/10.1109/ICRA40945.2020.9197084
30. Waldron, K.J., Schmiedeler, J.: Kinematics. In: Siciliano, B., Khatib, O. (eds.) Springer Handbook of Robotics, pp. 11–36. Springer (2016). https://doi.org/10.1007/978-3-540-30301-5_2
31. Wang, W., Li, P.: Towards finding the shortest-paths for 3D rigid bodies. In: Robotics: Science and Systems (RSS) (2021). https://doi.org/10.15607/rss.2021.xvii.085

32. Xu, L., Baryshnikov, Y., Sung, C.: Reparametrization of 3D CSC Dubins paths enabling 2D search. In: International Workshop on the Algorithmic Foundations of Robotics (WAFR) (2024)
33. Yim, N.H., Ryu, J., Kim, Y.Y.: Big data approach for synthesizing a spatial linkage mechanism. In: IEEE International Conference on Robotics and Automation (ICRA). pp. 7433–7439. IEEE (2023). https://doi.org/10.1109/ICRA48891.2023.10161300
34. Yong, C., Barth, E.J.: Real-time dynamic path planning for Dubins' nonholonomic robot. In: IEEE Conference on Decision and Control (CDC) (2006). https://doi.org/10.1109/CDC.2006.377829

Supplementary Materials: Algorithmic Design of Kinematic Trees Based on CSC Dubins Planning for Link Shapes

Daniel Feshbach[0000−0001−5185−7395], Wei-Hsi Chen[0000−0001−8523−1809], Ling Xu[0009−0002−5343−7911], Emil Schaumburg[0009−0002−0399−551X], Isabella Huang[0009−0003−0813−3908], and Cynthia Sung[0000−0002−8967−1841]

General Robotics, Automation, Sensing & Perception (GRASP) Lab
University of Pennsylvania, 3101 Walnut St, Philadelphia, PA 19104, USA
{feshbach,weicc,xu5,scemil,ihuangg,crsung}@seas.upenn.edu
https://sung.seas.upenn.edu/, https://www.grasp.upenn.edu/

A Modular Fabrication of Tubular Kinematic Trees via 3D Printing

To show that the tubular abstraction is physically realizable, we sketch a modular fabrication procedure based on 3D printing and construct an example 3-finger hand. The core idea is to break down a tubular kinematic tree design into its component joints and links, and generate STL files of sequentially connectable modules for each.

A.1 Modular Procedure

We have a set of parameterized module designs (Fig. A) for tubular links, string-actuated revolute joints with link attachment direction $\hat{\mathbf{a}} = \hat{\mathbf{x}}$, and hemispherical caps (example end effectors).

Links are generated as hollow circular extrusions along specified CSC Dubins paths. Where a parent joint has multiple children (and thus multiple outgoing connections), its outgoing paths are joined together where they connect to the parent, generating a single branching link structure. Fig. A(a) shows an example branching link, and Fig. A(c) shows a link for a non-branching link on a straight line segment.

Revolute joints with link attachment direction $\hat{\mathbf{a}} = \hat{\mathbf{x}}$ (i.e., orthogonal to the axis of motion) are constructed as a proximal piece, a distal piece, and a cylindrical pin connecting them, as shown in Fig. A(b). Each end has a hollow cylinder on the inside. The distal side has a pair of holes on opposite ends to which to tie strings to actuate the joint via antagonistic pulling. The proximal side has holes to pass the strings back into the tube, so they can route inside the previous links and joints to reach the root and be pulled externally. The string attachment is depicted in Fig. B.

Both links and the proximal ends of joints have a grid of holes at the proximal base to facilitate string routing and avoid entanglement. Module generation also

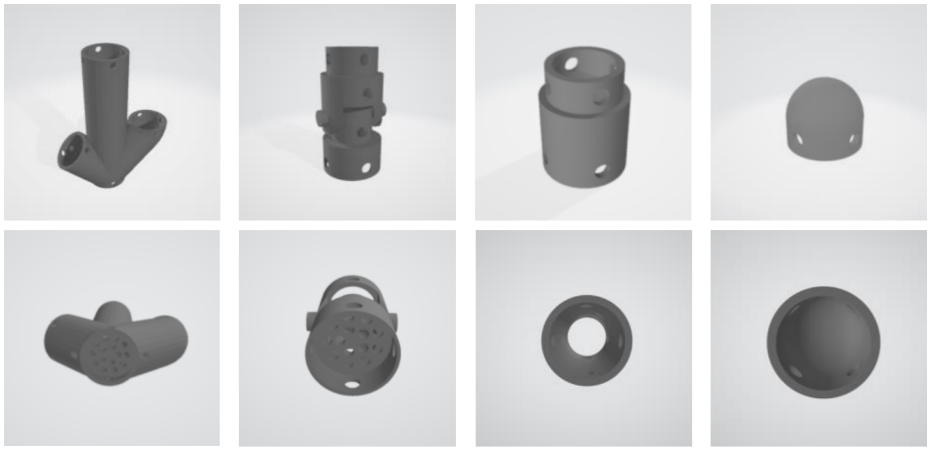


Fig. A. CAD for (a) branch link, (b) revolute, (c) straight link, and (d) cap modules.

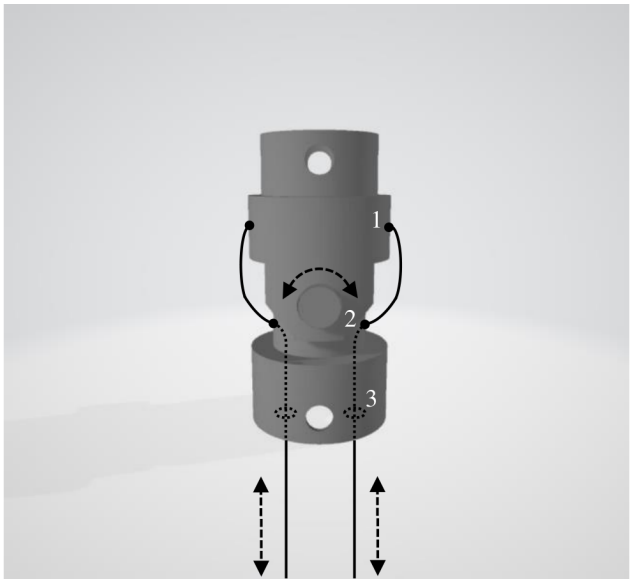
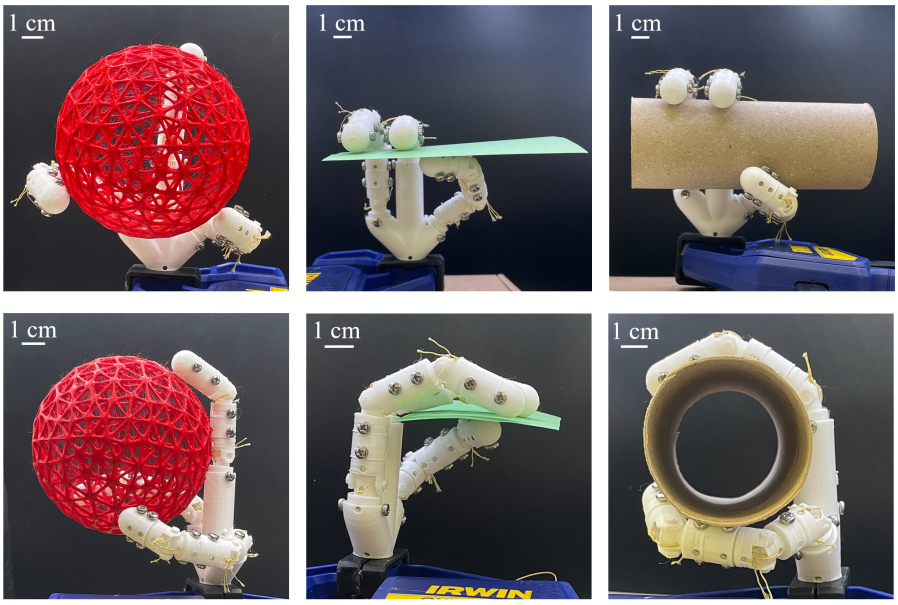


Fig. B. Strings are tied to the revolute joint module at (1), loop around the outside, are fed back in at (2), and are threaded through a grid hole at (3).

incorporates screw size, wall thickness, and other parameters ensuring they can attach together sequentially.

A.2 Hand Example

As an example of the fabrication system, we build a 3-fingered hand. The joints are arranged manually, but the link shapes are generated algorithmically from this arrangement (i.e., by finding CSC paths from the proximal pose of the parent to the distal pose of the child). We choose axes of motion and joint arrangements to enable the hand to have three distinct grasp types: spherical, pinch, and cylindrical. The hand is fabricated from polylactide (PLA) on a Prusa MINI 3D printer. The result is shown in Fig. C. It successfully forms the expected configurations to hold a ball, a piece of paper, and a cylinder. However, the joint design and string routing could use further refinement to improve grip strength and stability.



(a) Spherical grasp (b) Pinch grasp (c) Cylindrical grasp

Fig. C. A 3D printed tubular hand performing (a) spherical, (b) pinch, and (c) cylindrical grasps. The clamp at the root fixes the actuating strings in each configuration.

B Helper Operations for Tree Construction Algorithm

Alg. 1 maintains a bounding cylinder $\mathcal{C}$ of the structure generated so far, keeping constant the direction $\hat{\mathbf{n}}$ of its central axis. Alg. B.1 and Alg. B.2 specify how to do expand the cylinder to include newly added structures while preserving the direction as $\hat{\mathbf{n}}$.

Meanwhile, Alg. B.3 defines how each joint is oriented about its axis of motion, ensuring that $\hat{\mathbf{z}}_i$ is along $\overleftrightarrow{z_i}$ and that neither $\hat{\mathbf{z}}_i$ nor $\hat{\mathbf{x}}_i$ faces backwards (relative to $\hat{\mathbf{n}}$).

Algorithm B.1: ExpandCylinderToIncludeBall($\mathcal{C}, \boldsymbol{o}, b$)

Input: Cylinder $\mathcal{C}$, point $\boldsymbol{o}$, radius b
Output: Cylinder $\mathcal{C}'$ in the same direction as $\mathcal{C}$, bounding $\mathcal{C}$ and ball($\boldsymbol{o}, b$)

$(\mathbf{c}_s, \mathbf{c}_e) \leftarrow$ start and end points of $\mathcal{C}$ central segment
$\hat{\mathbf{n}} \leftarrow (\mathbf{c}_e - \mathbf{c}_s)/||\mathbf{c}_e - \mathbf{c}_s||$
$\mathcal{S} \leftarrow \text{plane}(\mathbf{c}_s, \hat{\mathbf{n}})$
$\mathbf{c}'_s \leftarrow \mathbf{c}_s + \min(\text{signedDistance}(\mathcal{S}, \boldsymbol{o}) - \rho, 0)\hat{\mathbf{n}}$
$\mathcal{P} \leftarrow \text{plane}(\mathbf{c}_e, \hat{\mathbf{n}})$
$\mathbf{c}'_e \leftarrow \mathbf{c}_e + \min(\text{signedDistance}(\mathcal{P}, \boldsymbol{o}) + \rho, 0)\hat{\mathbf{n}}$
$\mathcal{S}' \leftarrow \text{plane}(\mathbf{c}'_s, \hat{\mathbf{n}})$
$O \leftarrow$ minimal circle in $\mathcal{S}'$ bounding $\text{proj}(\mathcal{C}, \mathcal{S}')$ and $\text{circle}(\text{proj}(\boldsymbol{o}, \mathcal{S}'), b)$
$\mathcal{C}' \leftarrow$ cylinder from O in direction $\hat{\mathbf{n}}$ and length $||\mathbf{c}'_e - \mathbf{c}'_s||$

Algorithm B.2: ExpandCylinderToIncludeLink($\mathcal{C}, \mathcal{L}, r$)

Input: Cylinder $\mathcal{C}$, CSC link $\mathcal{L}$, tubular radius r
Output: Cylinder $\mathcal{C}'$ in the same direction as $\mathcal{C}$, bounding $\mathcal{C}$ and $\mathcal{L}$

$\mathbf{e}_1, \mathbf{e}_2 \leftarrow$ end points of S segment in $\mathcal{L}$ centerline path
if $\mathcal{L}$ *starting C arc is non-empty* **then**
  $\mathbf{c}_1 \leftarrow$ center point of starting C arc in $\mathcal{L}$ centerline path
  $\mathcal{C}' \leftarrow \text{ExpandCylinderToIncludeBall}(\mathcal{C}, \mathbf{c}_1, 2r)$
else
  $\mathcal{C}' \leftarrow \text{ExpandCylinderToIncludeBall}(\mathcal{C}, \mathbf{e}_1, r)$
end
if $\mathcal{L}$ *ending C arc is non-empty* **then**
  $\mathbf{c}_2 \leftarrow$ center point of ending C arc in $\mathcal{L}$ centerline path
  $\mathcal{C}' \leftarrow \text{ExpandCylinderToIncludeBall}(\mathcal{C}, \mathbf{c}_2, 2r)$
else
  $\mathcal{C}' \leftarrow \text{ExpandCylinderToIncludeBall}(\mathcal{C}, \mathbf{e}_2, r)$
end

Algorithm B.3: ForwardOrientation($\overleftrightarrow{z}, \hat{\mathbf{n}}$)

Input: Axis $\overleftrightarrow{z}$, unit vector $\hat{\mathbf{n}}$ not parallel to $\overleftrightarrow{z}$
Output: Orientation $(\hat{\mathbf{x}}, \hat{\mathbf{y}}, \hat{\mathbf{z}})$ where $\hat{\mathbf{z}}$ points along $\overleftrightarrow{z}$, $\hat{\mathbf{x}} \cdot \hat{\mathbf{n}} > 0$, and $\hat{\mathbf{z}} \cdot \hat{\mathbf{n}} > 0$

$\hat{\mathbf{z}} \leftarrow$ unit direction along $\overleftrightarrow{z}$ signed such that $\hat{\mathbf{z}} \cdot \hat{\mathbf{n}} \geq 0$
$\hat{\mathbf{x}} \leftarrow$ unit direction orthogonal to $\overleftrightarrow{z}$ maximizing $\hat{\mathbf{x}}_0 \cdot \hat{\mathbf{n}}$
$\hat{\mathbf{y}} \leftarrow \hat{\mathbf{z}} \times \hat{\mathbf{x}}$

C Proof That There Exists a Suitable Direction for $\hat{\mathbf{n}}$

Lemma 3 (Direction Existence)**.** *Given k lines $\ell_1, \ldots, \ell_k$ in $\mathbb{R}^n$, there exists a unit vector $\hat{\mathbf{n}} \in \mathbb{R}^n$ not orthogonal to any of these lines.*

Proof. Since orthogonality is position-independent, we can suppose without loss of generality that all the lines pass through the origin. The unit vectors orthogonal to a line ℓ_i define a unit circle c_i centered at the origin, which is a great circle of the unit sphere centered at the origin. Since finitely many circles $c_1, \ldots, c_k$ cannot cover a sphere, there must exist a direction not orthogonal to any of these lines. □

We thank a reviewer for this proof, which replaces a long inductive argument that was mostly tedious notational book-keeping.