Article

The International Journal of Robotics Research
2018, Vol. 37(2–3) 378–399

DOI: 10.1177/0278364918754676
journals.sagepub.com/home/ijr

Knot grasping, folding, and re-grasping

Weifu Wang[1] and Devin Balkcom[2]

Abstract

This paper analyzes the physical resources necessary and sufficient to tie a knot of given structure. We present the first sufficient bound on the number of fingers required to tie a given knot; the bound is linear in the number of crossings appearing in the knot diagram for a given knot. We also present a lower bound on the required number of fingers, under a particular model of knot tying. We study how many re-grasps are sufficient to tie an arbitrary knot, and present an algorithm that can yield a small sufficient number of re-grasps to tie the given knot. Physical experiments in which different knots are tied and untied by robots, alone and in collaboration with a human, serve as a proof of concept to show the simplicity and correctness of the approach.

Keywords

Manipulation, flexible objects, knot tying

1. Introduction

How hard is knot tying, and what are the fundamental challenges? Different physical resources are used during the execution of any manipulation task. One of the most studied physical resources in robotic manipulation is the number of contact (grasp) points, or fingers, used in the task. From the study of immobilizing rigid bodies (Mishra et al., 1987a; Rimon and Burdick, 1995), to immobilizing chains of rigid bodies (Rimon and van der Stappen, 2012), to constraining the motion of rigid bodies through caging (Rimon and Blake, 1999), to immobilizing cloth (Bell, 2010), a primary measure of difficulty is the number of fingers required.

This paper studies bounds on the number of contacts needed to manipulate string. We also study the number of times the string needs to be released and re-grasped, another measure of the difficulty of the knot-tying process. This paper extends work first presented by Wang and Balkcom (2016).

String can be fully controlled when it is stretched tightly between contacts in a polygonal configuration. If we can find strategies that tie knots while maintaining polygonal structure (possibly while moving fingers to lengthen or shorten segments of string), such strategies give an upper bound on the complexity of tying a given knot, over all models of string. We can also derive some lower bounds that are specific to this particular polygonal model.

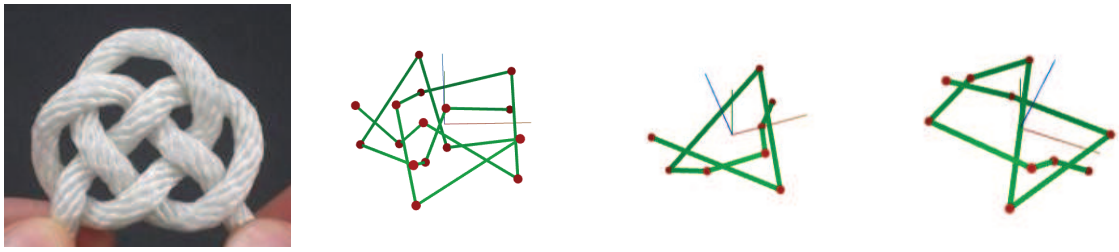
We model fingers as immobilizing specific points on the string. In the mathematical analysis, a major advantage of polygonal knots is that we know the configuration of the string (and, thus, its topology) when the locations of all the contact points are known.

Perhaps the first step in understanding controlled knot manipulation is to determine how many fingers are needed to immobilize string in a configuration that has the topology of a given knot. For an arbitrary knot with k crossings in the *knot diagram* (Section 2), we find that at least $\left\lceil \frac{3+\sqrt{8k+1}}{2} \right\rceil + 1$ and no more than $2k$ contact points are needed to immobilize the knot in a polygonal shape with the right topology. Figure 1 shows an example.

However, the goal is not to immobilize a knot in a goal configuration, but to find continuous motions that manipulate the string from an initial straight configuration to a tied configuration. To find an upper bound on the complexity, we use a follow-the-leader approach to tie the polygonal chain, introducing new joints and vertices as needed. We find that $6k - 2$ fingers are sufficient to tie an arbitrary knot.

[1]Department of Computer Engineering, SUNY, Albany, NY, USA
[2]Department of Computer Science, Dartmouth College, Hanover, NH, USA

Corresponding author:
Weifu Wang, Department of Electrical and Computer Engineering, University at Albany, SUNY, Albany, NY 12222, USA.
Email: wwang8@albany.edu



(a) (b) (c) (d)

Fig. 1. Polygonal grasps of various knots. (a) A double-coin knot. (b) A double-coin knot represented by polygonal arc. (c) An overhand knot represented by polygonal arc. (d) A figure-eight knot represented by a polygonal arc.

If we allow some fingers to slide along the string, or allow fingers to release and re-grasp the string, we may sometimes be able to use fewer fingers to tie the knot, as shown in Section 4.2 and by Theorem 19.

Re-grasping is one of the more practically difficult tasks during much robotic manipulation, since contact is lost and must be re-acquired, possibly using extensive sensing or careful control. We therefore also analyze and attempt to minimize the number of re-grasps, and use the number of re-grasps required to tie a knot as a measure of the difficulty of tying.

We present an algorithm to compute a number of re-grasps sufficient to tie an arbitrary knot. The algorithm recognizes adjacent crossings that can be formed as a sequence of weaving-type motions, and uses these sequences to choose particularly suitable goal configurations for the string, such that several crossings can be achieved by dragging the tip of the string in a straight line. For many knots that we studied, only one re-grasp is required after choosing the knot layout with the provided algorithm.

Table 1 summarizes the different complexity measures for a few knots. Over time, all knots with nine crossings or fewer have been discovered, standardized, and compiled into a table, which is referred as the *standard knot table* (Burde, 1978). Some of the knots used as examples in this paper are described using their knot names from the standard knot table, where the first number represents the number of crossings on the knot, and the subscript is derived from the order in the standard knot table.

The paper is organized as follows. We first introduce the necessary background on knots (Section 2), and analyze the knot-tying process (Section 3.1). We derive the number of contacts necessary and sufficient to grasp arbitrary polygonal knots (Section 3.2) and to fold them (Section 4). We then introduce the knot weaving process (Section 5.1), and identify different types of crossings on knots based on what type of motion can be used to arrange the crossings (Section 5.3). We choose some motion strategies to tie knots with robots alone (Section 5.5), and in collaboration with a human (Section 5.7). We also derive a strategy that can be used to untangle knots (Section 6). Details of a few of the proofs are left to Appendix B.

1.1. Related work

This paper makes use of the language of knot theory (Adams, 2004; Armstrong, 1983; Crowell and Fox, 1977; Livingston, 1993; Manturov, 2004). Although polygonal knots have been studied from the perspective of knot theory, the focus has been on the topology of knots. In contrast, the present work is the first we are aware of that investigates the complexity of polygonal knots, in terms of the number of line segments needed to represent the knot. The problems studied in this paper are also very related to a three-dimensional version of the *Carpenter's Rule Problem* (Biedl et al., 1999), in which the objective is to reconfigure a polygonal chain without self-intersection. However, we allow both the number of joints and joint locations (finger placements) to vary in such a way as to reduce the number of links.

In robotics, knot-tying frequently uses active controls and sensors. Inoue and Inaba (1985) used a $6 + 1$-degree-of-freedom (DOF) robot arm equipped with stereo machine vision to tie knots around a ring. A graph-based language was developed by Hopcroft et al. (1991) to program knot-tying motions, and tested through knot tying with a robot arm.

Untying knots has also been studied recently as a vision and learning problem by Lui and Saxena (2013), where the authors used a vision system to recognize the configuration of the knot, and built a system that follows human demonstration to untie simple overhand knot. The untying task has also been studied as a motion planning problem (Ladd and Kavraki, 2004), where the authors studied the motion needed to untie knots in simulation.

Real and simulated knots were tied using motion planning algorithms developed by Saha, Isto, and Latombe with a string model (Saha and Isto, 2006, 2007; Saha et al., 2006). Sequences of motions were planned and carried out by a pair of robot arms. The approach introduced a hierarchical decomposition of the knot using loops as basic structure. The tree search planner of Wakamatsu et al. (2006) finds sequences of motions (selected from four primitives) to tie or untie a knot. Matsuno and Fukuda (2006) used a vision system to recognize

Table 1. Knots examples and corresponding complexity.

Knot	Example	Number of crossings	Sufficient fingers	Sufficient re-grasps
Overhand		3	16	1
Figure-eight		4	22	1
Knot 6_2		6	34	1
Knot 7_7		7	40	2
Double-Coin (8_{18})		8	66	1
Knot 8_2		8	46	1
Knot 8_3		8	46	1
Knot 8_4		8	46	2
Knot 8_5		8	46	1
Knot 8_{10}		8	46	1
Knot 8_{17}		8	46	2
Knot 9_1		9	52	1
Knot 9_{30}		9	52	1
Knot 9_{31}		9	52	1
Knot 9_{32}		9	52	1
Cloverleaf knot		16	94	6
River knot		25	148	4

crossing points to describe knots and manipulation tasks. Similarly, Takamatsu et al. (2006) used observed crossings and the direction of motion to represent string manipulation tasks.

Recently, we have studied how to tie knots with the assistance of fixtures (Bell, 2010; Bell et al., 2014; Wang et al., 2014,?). The approach resulted in a generalized knot-tying approach, that can tie many different knots effectively using minimum control and no feedback information.

The approach we take in the present paper immobilizes each segment of string at different configurations to reduce uncertainty, just as rigid bodies are immobilized during grasping (Mishra et al., 1987b; Rimon and Blake, 1999; Rimon and Burdick, 1995; Rimon and van der Stappen, 2012).

Starting with a planar representation of the knot, we construct a 3D configuration for the string that preserves the topology implied by the planar diagram while attempting to minimize the number of fingers that must be placed at vertices. Reconstruction of 3D objects from 2D projections has been studied in work going back as far as the 1970s (Huffman, 1971, 1976, 1977; Kanade, 1980; Waltz, 1972). One approach has been to label intersections, and to find contradictions in the labels to determine whether a projection is realizable in three dimensions.

Even though changes of geometry are quite common when manipulating flexible objects, large-scale deformation away from the given goal geometry is often considered an undesirable error, instead of a means to simplify the manipulation process; in this work, we allow such changes in geometry, so as to establish an accurate upper bound. One notable study about changing geometry to simplify manipulation is the work by Demaine et al. (2010) on *kirigami*, in which paper is first folded into a particular shape, so that a single cut can be made such that when the paper is unfolded, a desired pattern or scene is created.

Physical knot theory (Kauffman, 1991) studies the geometry of tight knots formed with thick string (Ashton et al., 2011). The tightness of a knot has been studied in applied mathematics and physics (Baranska et al., 2008; Carlen et al., 2005; Rawdon, 1998).

Stretching string tightly between fingers makes the geometric model of the string studied in this paper reasonable, but we can also envision studies of knot complexity that allow for twisting, bending, and dynamics. Elastic rods have been used to model wires and string for manipulation (Bretl and McCarthy, 2014; Pai, 2002). Friction plays an important role in tightening a knot, or in untying tight string. Analysis of the frictional forces is frequently a key component of the analysis of rigid-body systems (Balkcom et al., Berard et al., 2004; Mason, 2001), and as string wraps around other segments of string with a certain thickness, the friction can be modeled using *capstan equations* (Attaway, 1999). However, in the present work, we consider only string that does not self-contact, so friction can be neglected from the analysis.

2. Mathematical knots and notation

Knots are tied using string or rope that can deform. To categorize and differentiate knots, mathematicians observed that if one connects the open ends of a physical knot, then the structure of the knot is invariant under deformation. Two knotted pieces of string are said to be in the same class if and only if there exists a smooth deformation taking one curve to the other, without separating the ends. Formally, a knot is a smooth embedding of the topological circle into $\mathbb{R}^3$ (Adams, 2004; Armstrong, 1983; Crowell and Fox, 1977; Livingston, 1993; Manturov, 2004); the phrase "topological circle" describes the idea of connecting the open ends.

To describe or illustrate a knot, we usually project the knot onto a plane. When the projection is regular (no three points on the knot project to the same point, and no vertex projects to the same point as any other point on the knot (Livingston, 1993)), the projected diagram is called a *knot diagram*.

On the projected drawing, broken lines are used to indicate where one part of the knot under-crosses the part of the knot that is directly above the broken lines; such locations are called *crossings*. Each crossing is labeled by a unique number, indicating the order of the appearance when tracing along the diagram. Figure 2a shows a shoelace knot diagram. Note that the same number crossing appears twice while tracing the figure; crossing 1 first occurs as an over-crossing, and then later is an under-crossing for the string, as the string is traced from 4 to 2.

We can use the labels of the crossings to describe the knot. One such description is the *Gauss code*: a sequence of labels for crossings indicating a walk along the diagram from a given starting point. We use numbers to label crossings, and a superscript "+" or "−" to indicate an over-crossing or an under-crossing. For example, a shoelace unknot with Gauss code $1^+\ 2^-\ 3^-\ 4^+\ 5^-\ 6^-\ 7^+\ 3^+\ 2^+\ 1^-\ 4^-\ 5^+\ 6^+\ 7^-$ has seven crossings, so the length of G is 14 ($|G| = 14$).

The projected knot diagram separates the plane into several disconnected closed *cells*, labeled by capital letters in Figure 2a.

2.1. Polygonal knots

To find an upper bound on manipulation complexity, we study a particular simple strategy, in which the string is always stretched tightly between supporting contacts, making the knot polygonal. Formally, we provide the following definition.

Definition 1 (Armstrong, 1983)**.** *A polygonal knot is a knot whose image in $\mathbb{R}^3$ is the union of a finite set of line segments.*

In this work, we focus on knots that are *tame*, with finitely many crossings.

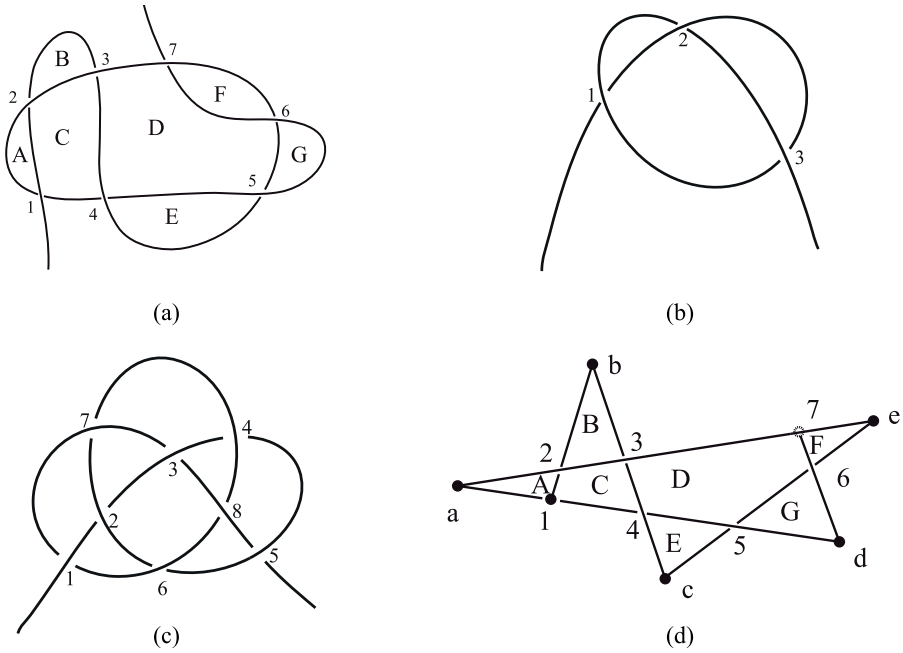


Fig. 2. Shoelace knot diagrams: (a) a shoelace knot, with numbered crossings and cells labeled with letters; (b) an overhand knot, with numbered crossings; (c) a double-coin knot, with numbered crossings; (d) a polygonal (grasp-able) shoelace knot.

A polygonal knot diagram (example in Figure 2d) with k crossings is a 4-connected planar graph $G = (V, E)$ with straight line edges, and $|V| \geq k$. On the diagram, we trim the first and the last links of the string (crossings 1 and 7 in the example figure), so that the first and last crossings become the endpoints. Thus, $k-2$ of the vertices (the crossings) have degree four, and two vertices have degree three; the remainder have degree two.

2.2. Knot cells and knot paths

We will need some vocabulary to describe the anatomy of a knot diagram. We will call the cell that extends to infinity the *exterior cell*, and all other cells *interior cells*. In Figure 2d, the interior cells are labeled *A*–*G*. We refer to the sequence of one or more line segments connecting two labeled crossings on the Gauss code as a *knot path*, which is a subsection of the knot diagram.

We call a knot path that contacts both the exterior cell and at least one interior cell an *exterior knot path* (as an example, see knot paths $(1, 2)$ through a, $(2, 3)$ through b, $(3, 7)$, $(7, 6)$ through e, $(6, 5)$ through d, $(5, 4)$ through c and $(4, 1)$ on Figure 2d). Let an *exterior crossing* be a crossing that is the common endpoint of two adjacent exterior knot paths; all the other crossings are *interior crossings*. If a connection between two crossings is not an exterior knot path, it is called an *interior knot path*. Knot paths are undirected, so (i, j) and (j, i) represent the same knot path. In the remainder of the paper, when we read the Gauss code from different directions, we may use one or the other order.

Each crossing appears twice on the Gauss code for the knot. Define a *knot unit* as a connected subsection of Gauss code where all the labels in the subsection have appeared twice. As an example, consider three overhand knots tied in sequence along a single piece of string, with Gauss code 1^+ 2^- 3^+ 1^- 2^+ 3^- 4^+ 5^- 6^+ 4^- 5^+ 6^- 7^+ 8^- 9^+ 7^- 8^+ 9^-. This structure contains three knot units, $G_1 = 1^+\ 2^-\ 3^+\ 1^-\ 2^+\ 3^-$, $G_2 = 4^+\ 5^-\ 6^+\ 4^-\ 5^+\ 6^-$ and $G_3 = 7^+\ 8^-\ 9^+\ 7^-\ 8^+\ 9^-$. We call a knot path a *bridge* if it connects two knot units.

3. Immobilizing grasps of polygonal knots

How hard is it to grasp a particular knot? Unlike a rigid body, the description of a knot allows many possible geometries; computing the number of fingers needed to grasp the knot entails choosing a suitable geometry as well as a placement of the fingers. This section presents a method for finding a polygonal geometric configuration from a Gauss code; such a configuration of the string may be immobilized simply by grasping each of the vertices. This gives a useful way of computing a measure of the complexity of grasping the knot based on the underlying structure of the knot.

3.1. Properties of polygonal knots

From the Gauss code, we would like to find a polygon that has only a few line segments, that may be simply grasped by fingers at vertices. We present a procedure in the following few subsections to first build a polygonal knot diagram based on the Gauss code (Algorithms 1–3), then project it to three dimensions to find a polygonal configuration for the given knot.

The procedure is recursive, placing all crossings so that the straight line connections between appropriate crossings do not create undesired intersections, which would require the use of additional line segments to correctly represent the knot diagram. We start by placing the crossings on the outside boundaries of the polygonal knot diagram, and proceed inwards.

To derive this procedure and show its correctness, we will need a few properties of polygonal knot diagrams, presented as lemmas. Interested readers may find the proofs of these following lemmas, as well as several others in the paper, in Appendix B.

The first necessary step is to show that the connections between the outward-most crossings form a closed curve, formally, a *Jordan curve*.

Lemma 1. *All the exterior knot paths of a knot unit form a Jordan curve.*

The following lemma is also proven in Appendix B. The basic idea of the proof is based on the fact that the knot diagram is a graph of degree four; we apply Euler's theorem to compute the number of interior cells.

Lemma 2. *A planar polygonal knot diagram with k crossings has k interior cells.*

Lemma 2 forms the basis to prove the relations stated in Lemma 3 between the numbers of interior and exterior cells, crossings, and knot paths. These equations will be used shortly to derive the upper bound on the number of fingers needed to immobilize and fold the given knot.

Lemma 3. *For an arbitrary knot with Gauss code G where $|G| = 2k$, let there be n_{ip} interior knot paths, n_{ep} exterior knot paths, n_{ic} interior crossings and n_{ec} exterior crossing. Then, the following equations and inequalities hold:*

$$n_{ic} = n_{ip} - (k-1) \tag{1}$$
$$n_{ec} = n_{ep} \tag{2}$$
$$n_{ic} + n_{ec} = k \tag{3}$$
$$n_{ip} + n_{ep} = 2k - 1 \tag{4}$$
$$n_{ep} \geq 3 \tag{5}$$
$$k - 1 \leq n_{ip} \leq 2k - 4 \tag{6}$$

3.2. Immobilizing grasps

Given a Gauss code G, we would like to generate an efficient polygonal knot diagram, corresponding to a grasp of the knot. The number of grasp points (fingers) needed is related to the number of crossings, which can be found from the Gauss code.

Lemma 4. *For an arbitrary knot with k crossings, at least $\left\lceil \frac{3+\sqrt{8k+1}}{2} \right\rceil$ sequential line segments are needed to plot the corresponding polygonal knot diagram.*

Although the formal proof of 4 is left to Appendix B, the approach is worth mentioning. The complexity of arbitrary *arrangements* of line segments, described by relationships between the numbers of intersections of those segments and the number of lines, has been well studied in the computational geometry community. Lemma 4 differs only in that the line segments are required to form a single chain, connected in a sequential path. This reduces the number of possible arrangements.

Lemma 4 implies a necessary number of grasp points (fingers) needed to polygonalize a knot. What is a sufficient number? We introduce an algorithm to place all the crossings and extra vertices, starting from exterior crossings and moving inwards. We show that by connecting all the placed crossings (and extra vertices) using line segments, no intersection (except at common end points) is introduced.

For convenience, we sometimes place a negative sign that negates the superscript. For example, an over-crossing can be represented as i^+, or $-i^-$.

We start by showing how the edges are connected to an exterior crossing.

Lemma 5. *Given a crossing with label e that is not the first or the last crossing, if one knot path containing e^a is an exterior knot path, then there must be another exterior knot path containing $-e^a$.*

Lemma 5 shows that even though each crossing label except the first or last appears in four knot paths (either exterior or interior) with each superscript appearing twice, if it is an exterior crossing, then the two exterior knot paths connected to this crossing will contain one of each superscript. This is a critical step in Algorithm 1, which finds a set of exterior crossings on the polygonal knot diagram we intend to build, using only the topological information from the Gauss code as input.

The next lemma indicates what happens if the crossing is either the first or the last crossing on the knot diagram. Without loss of generality, let the first crossing be 1^a and last crossing be l^b. Then, we have the following result.

Lemma 6. *The two knot paths with -1^a as endpoints and two knot paths with $-l^b$ as endpoints are exterior knot paths.*

Now that we have ways to identify exterior knot paths and know two exterior crossings, we can identify all the exterior crossings using the following procedure.

Algorithm 1: Find exterior crossings and knot paths

1. Add two knot paths with -1^a as endpoints (denote as $(p^e, -1^a)$ and $(-1^a, q^f)$) into the exterior knot path list;
2. Add two knot paths with $-l^b$ as endpoints (denote as $(s^c, -l^b)$ and $(-l^b, t^d)$) into the exterior knot path list;
3. Add crossings s^c, t^d, p^e and q^f into a queue Q, and add 1, l, p, q, s and t to the exterior crossings list;

4. Check whether any label appears twice in the exterior knot path list with both superscripts; if yes, delete the both labels from Q;
5. Pop a crossing x^g from the queue, find previous crossing u^h and next crossing v^r of $-x^g$ on the Gauss code. Check whether u and v are in Q (possibly with different superscripts);
6. If u (or v) is in Q, add $(u^h, -x^g)$ (or $(v^r, -x^g)$) to exterior knot path list, add u (or v) to exterior crossing list (note, at most one of u and v can be in the queue);
7. If both u and v are not in Q, add both of them in Q, add both knot paths into exterior knot path list, and add both of them to exterior crossing list;
8. Check whether a subset of exterior knot paths in the list form a cycle; if a cycle is formed, then delete all knot paths not used in the cycle, delete all crossings not used in cycle from Q and exterior crossing list;
9. If Q is not empty, go to step 5;

Let us look at an example. The Gauss code $1^+\ 2^-\ 3^+\ 4^-\ 5^+\ 6^-\ 2^+\ 7^-\ 4^+\ 8^-\ 6^+\ 1^-\ 7^+\ 3^-\ 8^+\ 5^-$ represents a double-coin knot. We use the construction of the knot diagram for the double-coin knot from the Gauss Code as a demonstration of the execution of the Algorithms 1–3; results can be seen in Figure 3, 5, and 6.

First, we find that 1^- and 5^+ are on exterior knot paths, and so add $(4^-, 5^+)$, $(5^+, 6^-)$, $(6^+, 1^-)$, and $(1^-, 7^+)$ to the exterior knot path list, and put 4^-, 6^+, 6^-, and 7^- in the queue. Since 6 has appeared twice, delete it from the queue. Now start with crossing 4. Note that 4^+ has previous crossing 7^- and next crossing 8^- in the Gauss code. We find that 7 is already in the queue; add knot path $(7^-, 4^+)$ to the exterior knot path list, delete 7 from the queue, and notice the queue is empty; terminate.

So far, we have knot paths $(4^-, 5^+)$, $(5^+, 6^-)$, $(6^+, 1^-)$, $(1^-, 7^+)$, and $(7^-, 4^+)$ in the exterior knot path list for the double-coin knot, and exterior crossings include 1, 4, 5, 6, and 7; the labels form a cycle. We then place the exterior crossings on a plane as the outer boundary for the polygonal knot diagram, using Algorithm 2.

Algorithm 2: Place all exterior crossings

1. Place crossing 1 at $(-1, 0)$ and denote the origin $o = (0, 0)$;
2. In the exterior knot paths list, we find an knot path $(1^a, b)$ that has crossing 1 as an endpoint, and delete it from the list;
3. Place crossing b at $(\cos(\pi - t), \sin(\pi - t))$ where $t = 2\pi/j$;
4. Label the crossing just placed as p, find an exterior knot path (p, q) in the list that has p as an end point, and delete it from the exterior knot path list; if op has angle α, then place q at $(\cos(\alpha - t), \sin(\alpha - t))$;
5. If exterior knot path list has more than 1 knot path, go to step 4;

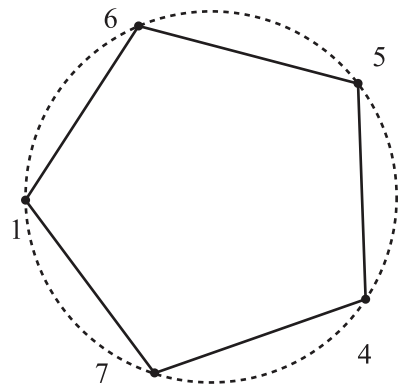


Fig. 3. Placing all exterior crossings for a double-coin knot.

As the goal is to form a polygonal knot diagram, we show below that using a line segment connecting the exterior crossings placed using Algorithm 2 does not create intersection.

Lemma 7. *If all exterior crossings are placed using Algorithm 2, then line segments connecting exterior crossings that are adjacent on the Gauss code do not intersect except at common endpoints.*

So far, we have placed all of the exterior crossings and connected them if they are adjacent on the Gauss code. An example is shown in Figure 3. Now we will work inwards, starting by placing all the crossings that are directly connected to the exterior crossings.

Algorithm 3: Place interior crossings

1. Find all crossings V that directly connect to exterior crossings on the Gauss code;
2. For a $v \in V$, find all adjacent exterior crossings E_v, and delete v from V;
3. Find the smallest circular sector that contains all the crossings in E_v, denote it as aob; let oa and ob intersect a circle centered at o with radius $1/2$ at a' and b', place v at the midpoint of $a'b'$ within aob;
4. If V is not empty, go to step 2;

Figure 4 demonstrates step three in Algorithm 3. Since v connects to a, b and c (or even d), the wedge with oa and ob as boundary contains all the crossings v connects to. Place v on the smaller circle with radius $1/2$. This step, as we state formally below, also does not introduce undesired intersection, if we introduce an ϵ perturbation.

Lemma 8. *If $v \in V$ is placed using Algorithm 3, and all $v \in V$ are connected to their corresponding exterior crossings using line segments, there exists an $\epsilon \geq 0$ perturbation of each v along ov away from o such that these knot paths do not intersect with each other or with previously connected knot paths.*

An example of connecting all crossings in V to exterior crossings for the double-coin knot is shown in Figure 5.

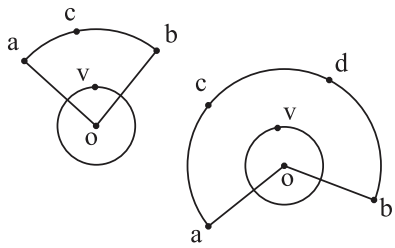


Fig. 4. Finding a placement for v based on the exterior crossings it connects to.

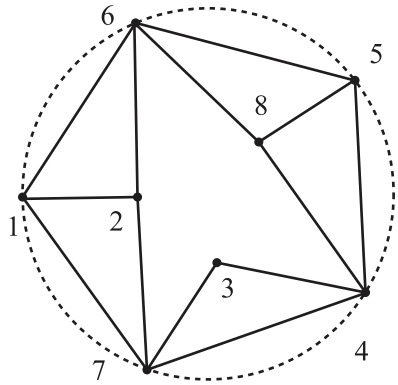


Fig. 5. Connecting crossings in V with the exterior crossings based on the Gauss code.

We now connect the crossings within V. We show that these knot paths do not create intersections.

Let us consider $v_i \in V$ for $i = \{1, 2, 3, 4\}$ and the case where v_3 and v_4 belong to different pieces inside of P, with respect to a polygonal arc (a, v_1, v_2, b) connecting two exterior crossings a and b and passing through v_1 and v_2. Since v_3 and v_4 are both connected to exterior crossings as well (otherwise they will not be in V), then consider a polygonal arc (c, v_3, v_4, d) connecting exterior crossings c and d which passes through v_3 and v_4. These two exterior crossings c and d must also belong to two different pieces with respect to (a, v_1, v_2, b) because all v are placed adjacent to the exterior crossings they are connected to. Then, the only way to avoid intersection between (a, v_1, v_2, b) and (c, v_3, v_4, d) is that one of the two polygonal arcs is outside P, and neither is. Therefore, v_3 and v_4 must belong to the same piece with respect to (a, v_1, v_2, b).

If there are still unplaced crossings, recursively find crossings V_i that directly connect to crossings in V_{i-1} where $V_0 = V$, and place them on the circle with radius $1/2^{n+1}$, perturbing slightly if needed. Analogously to the previous argument, these connections will not intersect with each other.

The final laid-out polygonal knot diagram for the double-coin knot is shown in Figure 6. No extra intersections are introduced, and 15 line segments are used.

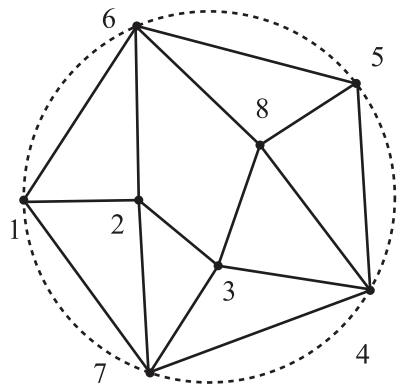


Fig. 6. Connecting crossings in V to for the double-coin knot.

3.3. Grasp complexity

To obtain the sufficient number of fingers to immobilize arbitrary knot into a polygonal shape, we may bound the line segments generated by the above algorithms. The following lemma directly leads to one of the main result of this work: a sufficient number of fingers needed to immobilize any given knot.

Lemma 9. *For an arbitrary tame knot unit with Gauss code G, where $|G| = 2k$, there always exists a polygonal knot diagram of the knot with no more than $2k-1$ line segments.*

Proof. In this proof, we first show a naive and looser bound, and then show that we can improve the bound through perturbation of the line segments.

First, place all the crossings using Algorithms 2 and 3, connecting all of the crossings with line segments. Each of the k crossings is visited twice, so $2k-1$ line segments are used.

However, two crossings a and b may be adjacent twice in the Gauss code, so that the two line segments connecting them overlap. For example, an overhand knot with Gauss code $1^+\,2^-\,3^+\,1^-\,2^+\,3^-$, crossings 1 and 2 appear adjacent to each other twice in G.

For the second time the two crossings a and b are adjacent in the Gauss code, use two line segments instead of one to connect them. Denote the midpoint of ab as $m_{a,b}$. For the second time a connects to b, if it is an exterior knot path, place p at $(1+\epsilon)\,o\vec{m}_{a,b}$, otherwise, place p at $(1-\epsilon)\,o\vec{m}_{a,b}$, where $\epsilon > 0$.

With k crossings, overlap may occur at most $k-1$ times. Therefore, at most $2k-1+k-1 = 3k-2$ line segments are necessary to draw any planar polygonal knot diagram.

Now, let us try to find a alternate representation of the polygonal knot diagram using fewer line segments, by perturbing the original representation using $3k-2$ line segments. Compared with $2k-1$, the extra $k-1$ line segments are due to the overlapping line segments (repeated adjacent crossings on the Gauss code). We originally added one vertex near the center of the connection, to reroute the second connection to avoid overlapping, as shown in Figure 7.

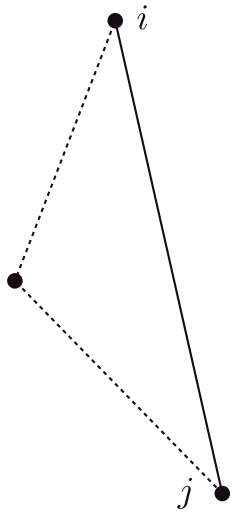


Fig. 7. The method to avoid overlapping in Lemma 9.

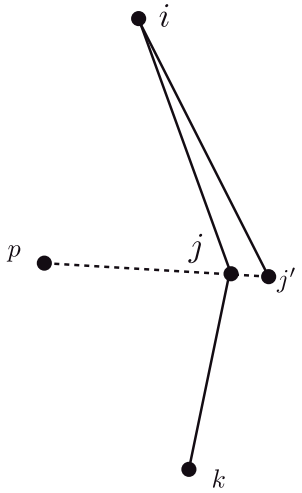


Fig. 8. The method to avoid overlapping in the first case.

Consider two crossings i and j that appear twice in the Gauss code. First consider the case where ij is adjacent to k the first time, and adjacent to p the second time on the Gauss code, and k and p are different crossings. Add a vertex j' on the extension of pj, such that j and j' are not coincident. When the second time i is adjacent to j on the Gauss code, let i connects to j' instead of j, then the crossing j still happens at j, but no collision will happen, as shown in Figure 8.

Now let us consider the case where i and j are connected to k both time on the Gauss code. Choose points j' and j'', where j' is on the extension of ij, and j'' is on the extension of kj, neither coincident with j. The first time ij appears adjacent on the Gauss code, let i connect to j' instead of j, and let i connect to j'' the second time they appear adjacent on the Gauss code, as shown in Figure 9. The crossing j is still at the location of j, but j is a projected intersection instead of an end point of a line segment.

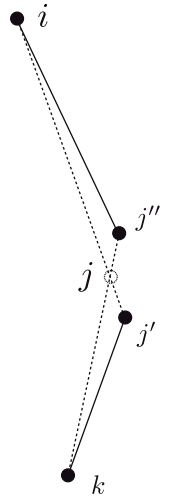


Fig. 9. The method to avoid overlapping in the second case.

By re-arranging the crossings following the principles stated earlier, only $2k-1$ line segments are needed (because we avoided using $k-1$ extra line segments to avoid collision). □

Combining Lemmas 4 and 9, we have the following result.

Theorem 10. *For an arbitrary knot with Gauss code G where $|G| = 2k$, to draw a polygonal knot diagram of this knot, at least $\left\lceil \frac{3+\sqrt{8k+1}}{2} \right\rceil$ and no more than $2k-1$ sequential line segments are required.*

Then, for the polygonal knot itself, we have the following result.

Theorem 11. *For arbitrary polygonal knot with k crossings, at least $\left\lceil \frac{3+\sqrt{8k+1}}{2} \right\rceil$ and no more than $2k-1$ sequential line segments are required to represent the polygonal knot.*

Proof. We construct the 3D polygonal arc using the loose bound of $3k-2$ line segments, and then show that the perturbation does not introduce intersection.

To project to n line segments on the plane, at least n line segments are needed in three dimensions. In the polygonal knot diagram constructed in the proof of Lemma 9 with $3k-2$ line segments, all the crossing are the vertices of the projected planar graph. This, combined with fact that the knot diagram is a regular projection, implies that the same number of projected line segments is sufficient to represent the 3D polygonal knot.

Place a crossing with coordinates (x_i, y_i) in the knot diagram at $(x_i, y_i, 1)$ if it is an over-crossing, and at $(x_i, y_i, -1)$ if it is an under-crossing. If a vertex on the polygonal knot diagram has degree two, place it at $(x_i, y_i, 0)$. Since the projected line segments do not intersect, the original 3D line segments also do not intersect. Therefore, no more than $3k-2$ line segments are needed to describe an arbitrary polygonal knot with k crossings.

When projecting the perturbed polygonal diagram back to three dimensions, in the first case, j and j' have different z coordinates, so the two line segments will not intersect. In the second case, because j' and j'' also have different z coordinates, the two connecting line segments will not intersect. Therefore, only $2k-1$ line segments are needed to represent the polygonal knot. □

Since each polygonal arc with n links has $n+1$ end points, we have the following corollary, which is the main result of the paper regarding the complexity of immobilizing a knot.

Corollary 1. *At least* $\left\lceil \frac{3+\sqrt{8k+1}}{2} \right\rceil + 1$ *and no more than* $2k$ *grasp points are needed to grasp an arbitrary knot with k crossings in a polygonal knot configuration.*

4. Continuous folding of knots

In this section, we investigate the problem of folding a polygonal arc from a straight line configuration to a knotted configuration, yielding formulas for computing a sufficient number of fingers needed to tie a knot. We may expect that achieving continuous folding motion will require at least as many fingers as the immobilizing grasps of the final configuration studied in the previous section.

We will need some terms. Given an arbitrary x–y plane, we say a polygonal knot K_p is *approximated* by a polygonal arc P if the projection of P on this x–y plane is a polygonal knot diagram of K_p. We use the word approximate as a reminder that a knot is a 3D structure, and we are only concerned with the relative z coordinates at each crossing, rather than finding a polygonal arc that overlaps every vertex on the polygonal knot. A link l on K_p is approximated by a link l' on P if the projection of l' overlaps with the projection of l on the x–y plane. We do, however, need to choose the x–y plane carefully to ensure that the projection of a given K_p is a regular, i.e. no three points on K_p projects onto the same point on this x–y plane.

Theorem 12. *An arbitrary knot with Gauss code G where* $|G| = 2k$ *can always be folded from a polygonal arc of no more than* $6k - 3$ *links, starting from a straight configuration.*

The proof is fairly lengthy; the basic approach is to fold an approximation of the computed polygonal arc by threading a slightly different polygonal arc of string along it, using a follow-the-leader approach. Additional fingers and bends are added as needed to allow threading without self-intersection.

Proof. We approximate as follows. For two vertices v_i and v_j on the knot, if $i < j$, then let v_i appear earlier than v_j when traversing along the knot from endpoint v_0. Once the links up to l_j ($p_{j-1}p_j$) on the arc P have approximated the links up to v_iv_{i+1} on the knot, then all vertices before p_j on the arc are fixed. When approximating a link v_iv_{i+1} using links from vertex p_a to p_b on the arc ($a < b$), it is possible to ensure that p_a and v_i, p_b, and v_{i+1} have the same coordinates; we will do so. Initially, for the ith link (l_i) on the arc between vertex p_{i-1} and p_i for which $1 \le i \le n$, let the length of the link $d(p_{i-1}, p_i)$ ($d(l_i)$) equal the distance between v_{i-1} and v_i, the $(i-1)$th and ith vertex on the knot, $d(v_{i-1}, v_i)$.

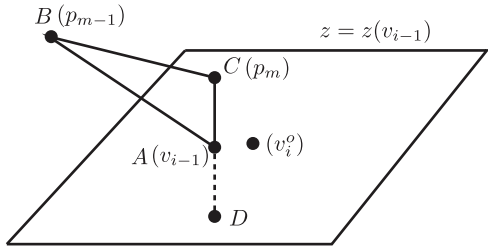


Fig. 10. Folding links given arbitrary small clearance.

Now fold, starting with the first link of the polygonal knot, v_0v_1. Without loss of generality, let v_1 be an over-crossing. We use the link p_0p_1 on the arc to approximate the first link of the knot, while keeping all the links from p_1 to p_m ($m \ge n$) above the plane $z = z(v_1)$. Then approximate v_1v_2 using link p_1p_2, by rotating all the links following p_1p_2 around p_1 until p_2 has the same coordinates as v_2 and all other vertices are below plane $z = z(v_2)$. Then, repeat this process if such a rotation around p_i does not collide with the fixed links (we will consider the collision case next). After approximating an over-crossing (under-crossing) v_i using links before vertex p_j on the arc, all vertices after p_j will be above (under) the plane $z = z(v_i)$.

When approximating a vertex v_i (where all vertices before p_j are used to approximate vertices up to v_{i-1}, without loss of generality, let v_i be an under-crossing and v_{i-1} be an over-crossing). Since all the previous links are fixed in space, it is possible that their projection forms a loop on x–y plane such that the previous attempted rotation around p_j will collide with the fixed links. Let v'_{i-1} and v'_i denote the projected points on x–y plane, and the crossing over v_i (denote as v_i^o) has been placed, we have $d(p_j, p_{j+1}) \ge d(v'_{i-1}, v'_i)$.

In this situation, insert two additional links l^1_{j+1} and l^2_{j+1} before link l_{j+2} and after link l_{j+1}, pointing from p_j to p_{j+1} (recall that links l_1 to l_i are fixed. We need to move l_{i+1} and all links after it to let p_{j+1} occupy the same location as v_i), and change the length of l_{i+1}. Let $d(l^1_{j+1}) = d(l_{j+1}) \ge \max(l^a_b)$ where $b \ge j+2$ and $a = \{0, 1, 2\}$ ($l^0_i = l_i$), and let $l^2_{j+1} = \sqrt{d(v_i, v_i^o)^2 + \epsilon^2}$. For arbitrary small $d(v'_{i-1}, v'_i)$, every link l^a_b and its previous link can rotate around one of the two endpoints with the smaller index such that all vertices after p_a (including p_a) can move continuously along a line $L^{v_{i-1}}$ that is perpendicular to the x–y plane and pass through v_{i-1}, either up or down depending on whether v_i is an over-crossing or an under-crossing, respectively. During the folding, since all vertices move along $L^{v_{i-1}}$, no link will collide with the fixed links.

An example of the folding is shown in Figure 10. During the folding, we combine links to form AB, so as to maintain the constraint that length $l(AB) \geq l(BC)$.

After the folding sequence moves link l^1_{j+1} $(p_{j+1}p^1_{j+1})$ to be perpendicular to the x–y plane where p_{j+1} is directly above p^1_{j+1}, move p^1_{j+1} on the plane $z = z(v_{i-1})$ along $\vec{v'_{i-1}v'_i}$ and stop when $d(p^1_{j+1}, v^o_i) = \epsilon > 0$. Then move link l^2_{j+1} to let p^2_{j+1} occupy the location of v_i. Repeat this process to approximate the knot using the arc.

Since there are at most $2k - 1$ links on original layout, we may at most need to add two links every time we approximate a new link. Therefore, we need at most $6k - 3$ links. □

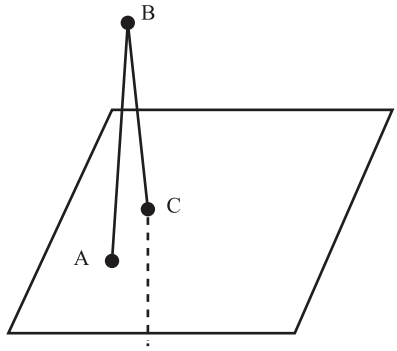


Fig. 11. Sliding fingers allow the string to bend and thread through small openings.

4.1. Better bounds for specific knots

Although the preceding theorem gives a sufficient number of links, analysis of particular knots may allow tighter bounds. For example, to fold the configuration calculated using the algorithms from Section 3.2, seven links are sufficient to fold an overhand knot (Theorem 11 requires seven line segments), and 18 links are sufficient for the double-coin knot (Theorem 11 requires 15 line segments). In the following, we show that even for the simplest knot, overhand knot, the upper bound we derived above is not tight. This suggests a very interesting challenge for future work.

Theorem 13. *At least five links are necessary to fold an overhand polygonal knot.*

Proof. Four links are necessary to create three crossings. However, consider the plane P formed by the first three endpoints of the links v_1, v_2, and v_3. To form an overhand knot, the projection of v_4 must be outside the triangle formed by v_1, v_2, and v_3. Without loss of generality, let v_4 be above P. Then, to form an overhand knot, the last link must penetrate P, with the last vertex v_n above P and projects outside triangle formed by v_1, v_2, and v_3 (this can be verified using the Gauss code). Then, v_4 (above P) must connect to v_{n-1}, which is below P. Therefore, $n - 1 = 5$, and five links are necessary to form an overhand knot. □

Theorem 14. *There exist* 5*-link polygonal arcs that cannot fold into any overhand polygonal knot.*

Proof. Consider a 5-link polygonal arc with lengths (100, 1, 1, 1, 100). Folding the overhand knot involves an operation to insert one of the links on the end through a triangle formed by other links, which in this case has a limited size. Two of the edges have length 1, and the other edge is shorter than 2. What is more, one of the endpoints for the outer link has to be no further than 1 unit away from the triangle. Therefore, such motion cannot be achieved. □

4.2. Knot tying with sliding fingers

In the previous analysis, we did not allow the fingers to move along the string or release contact. Owing to these constraints, the number of fingers needed to tie a knot is quite large.

In fact, even if we only allow fingers to slide along the string, the number of fingers needed to tie an arbitrary knot with k crossings can be reduced to just $2k$, twice the number of crossings. Let us use the polygonal configuration computed in previous section and the same follow-the-leader motion sequence to tie the knot. In the previous approach, we may need to add fingers during each threading motion along the z axis, because the polygonal links cannot bend at arbitrary point. Additional fingers were used to form the triangle to allow threading through arbitrary small opening.

Theorem 15. *If a contact is allowed to slide along the string, then an arbitrary knot with k crossings can be folded by using* 2*k fingers.*

Proof. If we allow the fingers to slide along the string, no triangle needs to be formed to thread the string along z axis, following the approach used in Theorem 12. As shown in Figure 11, finger at point B can continuously slide along the string, so the distance to the finger at A is reduced, while the finger at C is threading further down along the Z axis. Therefore, only $2k$ fingers are sufficient to tie an arbitrary knot. □

We have just shown that a polygonal overhand knot requires six fingers to be immobilized. Since an overhand knot has only three crossings, the sufficient number of fingers $2k$ is tight both for immobilizing the knot and tying the knot.

5. Knot weaving

The number of fingers proposed in the previous section is still too large to be provided by a traditional pair of robot arms. To physically tie knots with robots, we need to find alternative strategies to the follow-the-leader approach described above. One strategy to tie knots is to simulate human approaches to knot tying, in which some sections of string are laid out and immobilized together, and then other sections of string are threaded through the loops formed. This approach requires some release and re-grasping of the

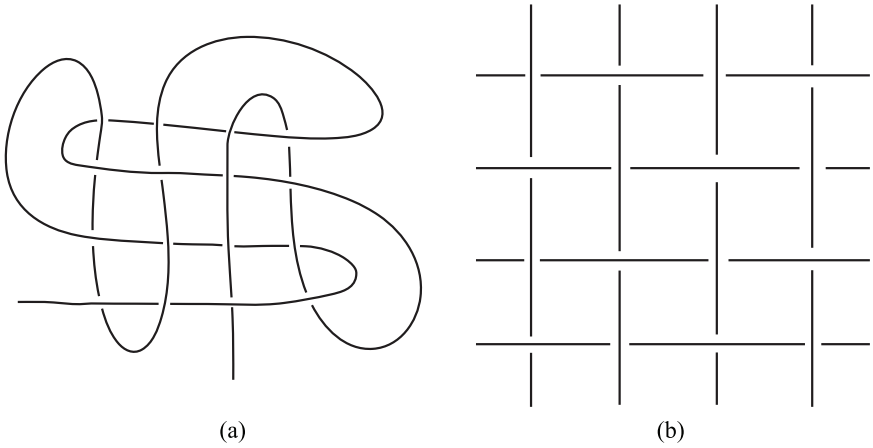


Fig. 12. The comparison between (a) a cloverleaf knot and (b) a weaving pattern.

string, if executed with a robot arm that is itself too large to pass through loops of the string. Since re-grasping can be difficult, we attempt to reduce the required number of re-grasps needed.

Many knots have alternating over- and under-crossing patterns when traced along the string. This pattern also shows up in weaving. One difference is that in weaving, the crossings that have alternating over- and under-crossing pattern (a *weaving sequence*) are geometrically adjacent to each other. Through a comparison between knot tying and weaving, we show many crossings can be arranged together using simple motions, which in turn reduces the total number of re-grasps used.

Let us consider the example of a weaving and a cloverleaf knot (both with 16 crossings) shown in Figure 12. Even though all the crossings may appear in the same geometrical locations, when traced along the string, the over/under patterns are different.

5.1. Knots and weaving

In a weaving loom, the warp is the set of strings that form the basic structure around which the weft (the string pulled by the shuttle) is woven. The approach we take to knot tying is to find a simple substructure of the knot in such a way that the under-crossing always appears before its over-crossing; this spiral-like structure is somewhat analogous to the warp. The rest of the knot, like the weft, is then arranged with respect to this warp to construct the more challenging crossings.

The approach we present will form weft crossings in iterations, and form necessary warp crossing between iterations. Therefore, in this approach, we sometimes immobilize segments of string that are manipulated as weft in previous iterations, and consider them as (immobilized) warp in the following iterations.

Let us consider the following example. For a double-coin knot with Gauss code $1^+\ 2^-\ 3^+\ 4^-\ 5^+\ 6^-\ 2^+\ 7^-\ 4^+\ 8^-\ 6^+\ 1^-\ 7^+\ 3^-\ 8^+\ 5^-$, the last five crossings counting from the right open end are 5, 8, 3, 7, and 1. These five crossings can be formed by dragging the right open end along a straight line like the motion of a shuttle on a loom, provided that the other segments of string are arranged appropriately. An example of the rearranged polygonal configuration of a double-coin knot is shown in Figure 13. We implemented the knot arrangement using the proposed layout with a Da Vinci robot arm. Figure 14 shows the results of the implementation.

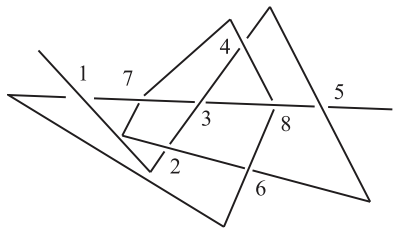
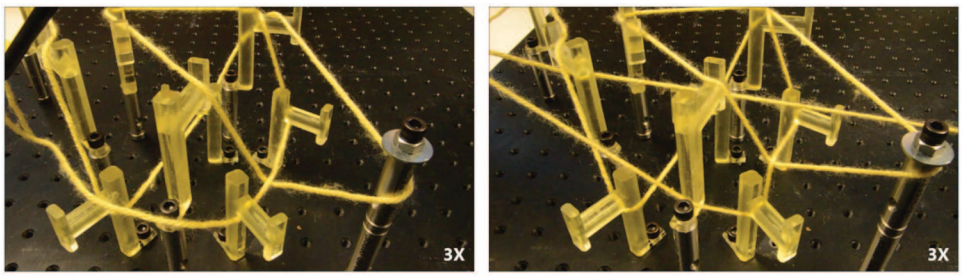


Fig. 13. A configuration of a double-coin knot with the last five crossings on a straight line.

5.2. Division into warp and weft stages

In this section, we introduce a method to change the geometry of a knot to allow the division of knot arrangement into warp and weft stages. In traditional weaving, the warp does not cross itself, but knots may have a more complex structure. We allow crossings in the warp, as long as those crossings can be achieved without regrasps, and then immobilized. We refer to all crossings formed by the warp stage as *warp crossings*, and all crossings formed in the weft stage as *weft crossings*. We will see that a crossing i is a weft crossing if and only if after all the crossings to the right (or all of the crossings to the left) of i^a have been removed, the crossing to the left (right) of i^a has a different sign than a.



(a) (b)

Fig. 14. Arranging a double-coin with a Da Vinci robot arm. (a) Arranging the warp crossings of a double-coin knot. (b) Completing the arrangement of a double-coin knot with one re-grasp.

5.3. Forming or removing crossings based on weaving sequence

We define a *minimal Gauss code* as a Gauss code that cannot be simplified by performing Reidemeister moves: three local motion primitives that create or remove crossings (Reidemeister, 1927). Adding or removing of a crossing from a structure with minimal Gauss code through physical manipulation of the string can only be achieved if one of the two appearances of the crossing number is at the beginning or the end of the Gauss code.

What happens if we remove crossings one-by-one from the open ends? Intuitively, we know that after a certain number of removals, the remaining crossing pattern is no longer knotted, because eventually the knot is untied if we remove all crossings. The knotting and unknotting process are symmetric. It is easier to see the pattern when removing crossings from an existing sequence of crossings, so we choose the unknotting process for analysis.

We define a *weaving sequence* as a sequence of alternating over- and under-crossings that have to be formed or deleted in the given order indicated by the Gauss code. Consider the example of unknotting a double-coin knot, the Gauss code of which is $1^+\ 2^-\ 3^+\ 4^-\ 5^+\ 6^-\ 2^+\ 7^-\ 4^+\ 8^-\ 6^+\ 1^-\ 7^+\ 3^-\ 8^+\ 5^-$. Let us start from the right end. Crossing 5^- is adjacent to 8^+ on the Gauss code, and the crossings have *different* superscript signs, and are therefore weft crossings. The untying process we employ removes weft crossings one by one, and in this case, we will remove 5^-.

We continue to remove crossings that are part of the same weaving sequence from the right end of the remaining Gauss code, including crossings 8, 3, 7, and 1, in given order. After we remove the last five crossings on the Gauss code, we have $2^-\ 4^-\ 6^-\ 2^+\ 4^+\ 6^+$. Now, the next two crossings from the right have the same superscripts, so they are no longer part of a weaving sequence. We know that the five deleted crossings can be formed by a single weft (weaving) motion, dragging the string along a straight line. We continue searching for weaving sequences from right to left. In this example, there are none, and the remaining structure consists only of warp crossings.

Algorithm 4, which takes the Gauss code of the knot as input, finds a weaving sequence for the given knot.

Algorithm 4: WEAVE

1. Select either left or right end of the Gauss code;
2. Delete crossings from the selected end;
3. If the crossing to be removed has a different sign from the next crossing to removed, then the two crossings belong to the same weaving sequence. Register a new weaving sequence if the current crossing to be removed is not already on a weaving sequence. If the crossing to be removed has the same sign as the next crossing to be removed, then terminate the current weaving sequence; if the crossing to be removed has the same label as the next crossing to be removed (for example, $i^-j^-j^+$ where j^- and j^+ have the same label), then compare the sign with the first crossing with a different label (compare j^+ with i^- in the given example);
4. Repeat steps 2 and 3 until only one crossing is left, and attach the last crossing to the on-going pattern;

With a single pass through the Gauss code, the algorithm outputs a sufficient number of m weaving sequences that can be used to form the knot; m is also a sufficient number of re-grasps to tie the knot with a fixed-base arm.

For a Gauss code with k crossings where $|G| = 2k$, we can find $O(k^2)$ different sequences of labels that are the results of removing the crossings at the beginning or the end of the Gauss code. However, since a weaving sequence can only be formed by weaving with one end of the string, we only need to check the sequence of labels that are the results of removing crossings from solely the left or right end. The total length of such a sequence of labels is $2k$.

The algorithm only checks whether the current crossing has a different sign from the adjacent one. This approach may consider some structures that are unknotted part of a weaving sequence, because the sequence contains adjacent labels with different superscripts, such as $1^+\ 2^-\ 3^+\ 4^-\ 4^+\ 3^-\ 2^+\ 1^-$. This structure is unknotted, but still contains one weaving sequence.

Knots such as the overhand knot contains only one weaving sequence associated with the last two crossings, while the first crossing can be formed by a type I Reidemeister move. Similarly, the *figure-eight knot* with Gauss code

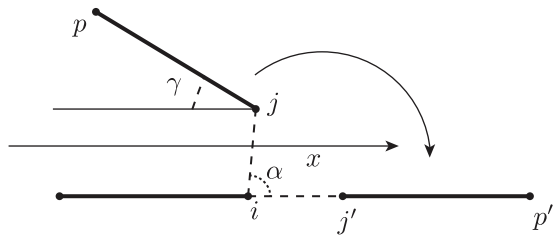


Fig. 15. Rotating extreme segments to align all weaving crossings on a straight line.

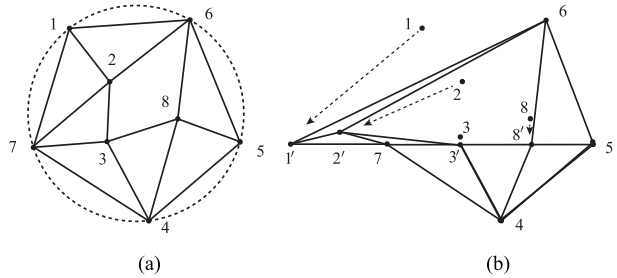


Fig. 16. Reconfiguration of a double-coin knot to align weft crossings onto a straight line. (a) A polygonal knot diagram for a double-coin knot. (b) The rearranged configuration for the double-coin knot, by rotating extreme segments using the proposed method indicated in Algorithm 5.

$1^+\ 2^-\ 3^+\ 4^-\ 2^+\ 1^-\ 4^+\ 3^-$ also contains only one weaving sequence associated the last three crossings where the first crossing is achieved by a type I Reidemeister move. The double-coin knot shown earlier contains one weaving sequence, with the crossings 2, 4 and 6 forming the initial unknotted structure.

Lemma 16. *If a knot can be arranged by following a single weaving sequence, then a motion that removes all weft crossings unties the knot.*

5.4. Aligning crossings on a straight line for simple manipulation

A straight-line motion is easy to achieve even for simple robotic devices. This section will show that weft crossings can always be aligned on a single straight line, without changing the knot topology.

Lemma 17. *In a weaving sequence, each crossing label appears only once.*

Since no crossing appears twice on a weaving sequence, we can move the crossings so that they appear on a straight line. Therefore, a single straight line motion of the string can form multiple crossings at once. Define an *extreme segment* of the weaving sequence as the segment between two exterior crossings.

Algorithm 5 shows how to arrange extreme segments on a straight line if they are on the same weaving sequence. The input to the algorithm is the locations of all the crossings, which may be computed from the Gauss code using Algorithms 1–3.

Algorithm 5: ALIGN

1. For each given extreme segment, connect a line between the two exterior crossings (or an interior to an exterior crossing), and move each of the crossings on the extreme segment to its projection on the connected line;
2. Let crossing i $((x_i, y_i))$ and crossing j $((x_j, y_j))$ be the two adjacent exterior crossings on two adjacent extreme segments, with k connections between them; without loss of generality, let $y_j < y_i$; let p be the other end point on the extreme segment with j as an end point;
3. Rotate all the points above extreme segment between p and j (because $y_j < y_i$) around j then around i in the stated order, so that (x'_j, y'_j) is the new location of crossing j where $y'_j = y_i$, as shown in Figure 15; the angle rotated around crossing i can be calculated as the acute angle α between the x axis and the vector $\vec{ij}$, and the rotation angle around crossing j is $\beta = \pi - \alpha - \gamma$, where γ is the angle between pj and x axis;
4. Along the line $x = (x_i + x'_i)/2$, find k points above (or under) the line $y = y_i$ if $x'_i \geq x_i$ ($x'_i < x_i$) with equal distance. For the endpoints of the k pair of connection between two extreme segments, connect to k points along the line $x = (x_i + x'_i)/2$ in order based on the distance of the end point on segment pj to the exterior crossing j, such that no additional intersection is introduced;
5. Adjust the z coordinates of all crossings so that the crossings on the rearranged weaving sequence lie on a straight line in three dimensions;

The result of applying the process to a double-coin knot is shown in Figure 16.

In previous sections, we have shown that an arbitrary (polygonal) knot with k crossings can be laid out based on its Gauss code using no more than $3k - 2$ line segments. We use this projected configuration to compute where each crossing should be placed so the crossings on a weaving sequence is on a straight line in our experiments.

Lemma 18. *The reconfigured polygonal knot configuration following Algorithm 5 uses no more than* $2k - 1$ *line segments.*

Given that we can use fewer line segments to represent the same knot, we can easily show that by rearranging the knot, we can use fewer fingers to grasp and fold a given knot, even without using re-grasping.

Theorem 19. *Given a knot and the configuration computed in Algorithm 5, fewer than* $6k - 2$ *fingers are needed to tie the knot.*

Proof. In the reconfigured configuration in Algorithm 5, instead of k possible threading operations, the threading

only happens when arranging aligned crossings. For a given knot, let m piercing motions be sufficient to arrange the knot. Since we have shown at most two extra line segments are needed at each threading, and $m \leq k$, we only need $2k - 1 + 2m$ line segments, less than $6k - 3$ line segments. Therefore, we need less than $6k - 2$ fingers to arrange the knot. □

5.5. Re-grasping and weaving

This section analyzes the number of re-grasps needed to arrange each of the two types of crossings. If we arrange a given knot by simply following the Gauss code without any change of geometry, the number of re-grasps needed to arrange a knot maybe be as large as the number of weft crossings, plus one re-grasp for each sequence of warp crossings.

We show that the output of Algorithm 4 also gives a sufficient bound for the number of re-grasps needed to arrange a given knot, if each sequence of weft crossings are aligned on a straight line following Algorithm 5.

We can use a fixed-base robot arm to lay out the warp crossings without re-grasping.

Lemma 20. *Let i be a crossing label. If on a sequence of crossings, i^- always appears before i^+, then the crossings can be laid out by a robot arm without re-grasping.*

Proof. The structure is unknotted and belongs to two layers. If for each layer, we trace along the configuration with a robot arm, then the two layers form the corresponding crossings. Therefore, no re-grasping is needed. □

An elephant-trunk arm with infinite degrees of freedom can arrange any knot without re-grasping. Sometimes, changing the geometry of the knot can reduce the number of degrees of freedom that a fixed-base arm must have to tie the knot using a particular number of re-grasps.

The next lemma shows that changing the geometry of the knot is, in fact, sometimes necessary to optimize this trade-off between degrees of freedom of the arm and the number of re-grasps required: some knot geometries are quite difficult for any finite-DOF arm. For example, a sequence of crossings of the form i^a, j^b, k^a, where a and b have opposite signs. A 4-DOF robot arm needs at least one re-grasp to arrange this sequence of the crossings, if j^a has already been arranged. Such sequence of crossings are the basic structures of the weaving sequence.

A weaving sequence contains a sequence of consecutive over- and under-crossings. However, if we imagine the robot end-effector as the shuttle on the loom, it does not need to re-grasp every time the string switch between over- and under-crossing, if all the crossings on this weaving sequence is on a straight line. However, for a weaving sequence, one re-grasp is still needed.

Lemma 21. *To arrange a weaving sequence with a fixed-base robot arm grasping the ends of the string, at least one re-grasp is needed.*

The number m output by Algorithm 4 gives a sufficient number of re-grasps needed to tie a given knot. For many knots, including the double-coin knot, the number is 1. Since these knots are in a different topology class from a topological loop when one of their end points are grasped by a robot arm and the other end point is attached to the ground, therefore at least one re-grasp is needed. In these cases, the number output by Algorithm 4 is also a lower bound.

5.6. Knot weaving with Da Vinci

We conducted experiments with a Da Vinci surgical robot, which has two symmetric high-precision arms. We computed knot layouts and built fixtures to support string arranged at various heights, allowing weaving to be implemented with a single translation.

Figure 14a shows the layout of the warp crossings of a double-coin knot, arranged without re-grasps. After layout, the effector of the second arm grasps the tip of the string and uses a pure translation to complete the knot, as shown in Figure 14b and in the multimedia attachments.

The preliminary arrangement of the string requires many support structures laid out in the workspace of the robot. For simplicity, we programmed the robot to just arrange the warp crossings of the string at the same heights around simple fixtures. Figure 17 shows the arrangement of a 7_1 knot. Figure 18 shows the arrangement of a double-coin knot.

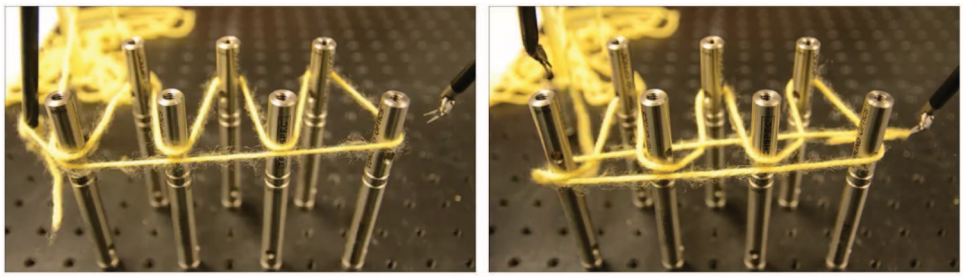
5.7. Robot–human collaboration

When we arrange the segments of string that are not part of a weaving sequence at the same height, the robot arm weaves around arranged segments of string. Even though the locations of the string segments are known, the motion still may not be easy to perform for a robot. Humans, however, can arrange the weaving sequence easily with re-grasps.

For example, a double-coin knot can be tied using the robot to lay out the structure, and allowing the human to finish the weft crossings. We used this technique to tie figure-eight knots, and knots 7_1, 8_2, 8_5, 8_{10}, 9_{31}, and 9_{32} from the standard knot table. Applying Algorithm 4, all the listed knots can be tied with one re-grasp. Figures 19, 20, and 21 show the examples of a human collaborating with an Adept Cobra industrial arm to tie knot 8_{10}, knot 9_{31}, and knot 9_{32}, respectively.

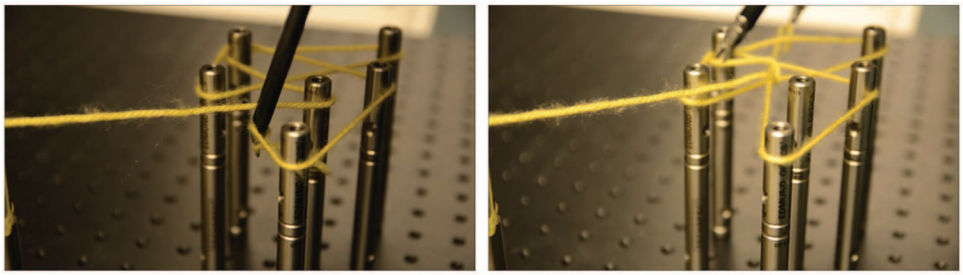
6. Untangling knots

In the previous approach, we changed geometry of knots to simplify the knot arrangement, based on the identification



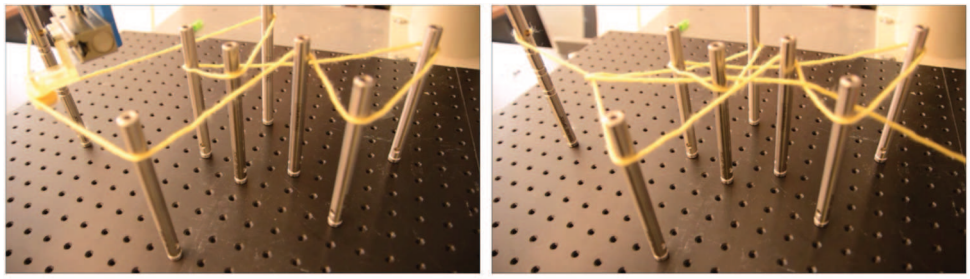
(a) (b)

Fig. 17. Arranging a 7_1 knot with a Da Vinci robot arm: (a) without re-grasping; (b) with one re-grasp.



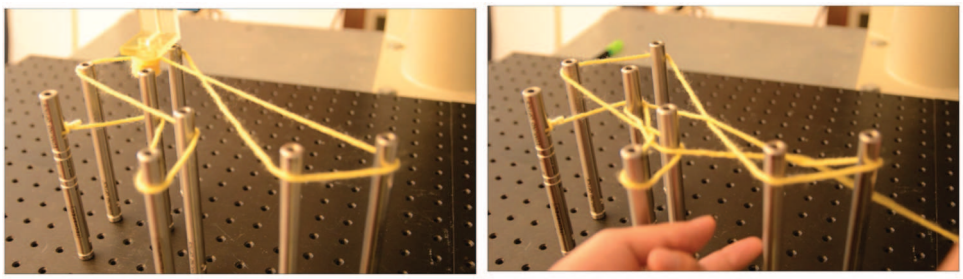
(a) (b)

Fig. 18. (a) Arranging a double-coin by laying out the warp crossings on the same height. (b) Completing the arrangement of a double-coin knot with a single re-grasp.



(a) (b)

Fig. 19. (a) Arranging a 8_{10} knot by robot and human collaborating together. (b) Completing the arrangement of a 8_{10} knot by a human weaving the string.

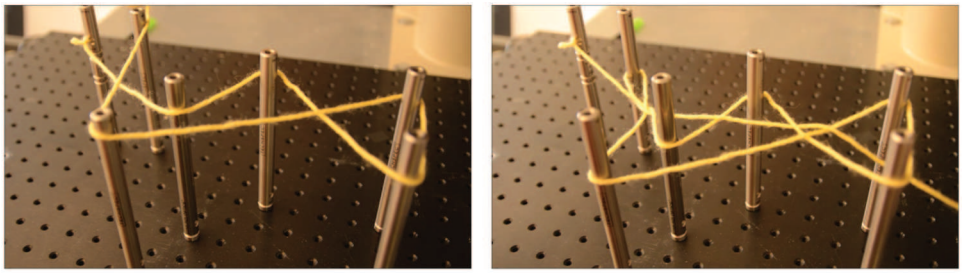


(a) (b)

Fig. 20. (a) Arranging a 9_{31} knot by robot and human collaborating together. (b) Completing the arrangement of a 9_{31} knot by a human weaving the string.

of type 1 and weft crossings on the knots, and we identified the weft crossings by processing the Gauss code.

The same change of geometry can also be used to untangle knots. In this work, we focus on untangling knots from



(a) (b)

Fig. 21. (a) Arranging a 9_{32} knot by robot and human collaborating together. (b) Completing the arrangement of a 9_{32} knot by a human weaving the string.

Fig. 22. Friction prevents the untying of a double-coin knot.

a loose configuration rather than from a tight configuration. We untangle the knot by changing the geometry of the knot and pulling the string several times along straight lines. With each pull of the string, we remove all consecutive weft crossings that extend to the current end of the string. However, non-consecutive weft crossings may not be able to be removed in a single motion.

Given two different sequences of weft crossings, in order for them to be aligned, other crossings need to be relocated. Since each crossing can only appear once on a weaving sequence (weft crossing sequence), two different sequences of weft crossings may not be disjoint. Therefore, they cannot be aligned on the same line. Therefore, without knowing which specific knot we are trying to untangle and detailed analysis of the specific knot, the best we can do with each grasp is to align a sequence of consecutive weft crossings, and remove them by moving the string along a straight line.

Even when a knot is in a loose configuration, the untangling of knots usually have to overcome friction. After we have identified all the weft crossings, and attempt to use a single motion to untangle them, friction may prevent the untangling, as shown in Figure 22 where we attempt to remove the last five crossings by pulling an endpoint of the string in a straight line. Therefore, a pulling motion of string that involves least friction is desirable. It appears that pulling string along a straight line can keep friction relatively low.

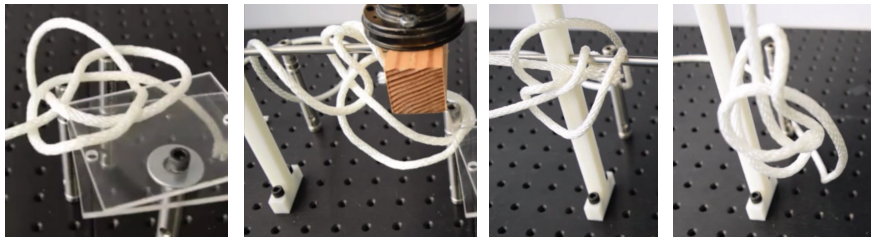
The process of manipulating the string geometry can be described as follows. We first choose one end of the string to untangle the knot. Along the chosen end of the string, we determine a side that is closer to the boundary, left or right. Starting from the chosen end of the string, identify all the cells on the chosen side to the string in sequence, until the last consecutive weft crossing, and find the largest inscribed circle in each cell. Using the same algorithm we presented in the previous section, we will delete crossings from the chosen end, and record how the crossings change, either from under-crossing to over-crossing, or from over-crossing to under-crossing.

Given the x–y plane on which the knot diagram is projected, we place a vector parallel to the z axis at each center of the largest inscribed circle we have identified. The direction of the vector is positive if the crossings associated with the cell change from under-crossing to over-crossing, otherwise negative. We then use a robot arm manipulating a rod to follow these vectors parallel to the z axis to the points above and below the $z = 0$ plane, and connect between these points by following linear motions with the end effector. After tracing all the vectors, we have aligned all the weft crossings. Figures 23 and 24 show the change of the geometry.

We then identify the last weft crossing we have aligned, and the warp crossing adjacent to it, and let the robot grasp any point between the two crossings. The robot arm then pulls the string along the direction parallel to the vector point along the rod we used to align the crossings. After all the consecutive weft crossings are removed and the rod is removed, the knot will be untangled. Even though we have only demonstrated the untangling of loose knots, the principle can be applied to tight knots, if we can identify the crossings and thread a needle through those enclosed cells.

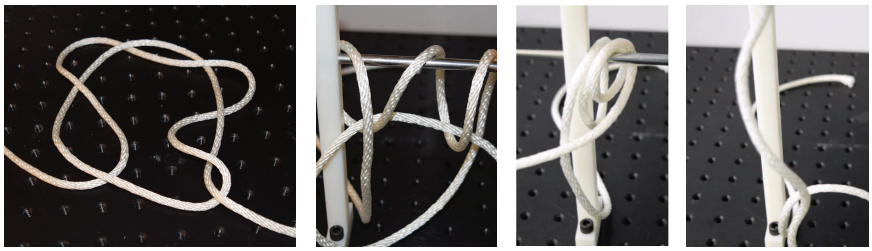
7. Conclusions and future work

In this work, we have studied the physical resources needed to tie arbitrary knots. First bounds on the number of fingers necessary and sufficient to tie knots without re-grasping were derived. We have also considered an alternative knot-tying approaches where re-grasping is allowed. In the



(a) (b) (c) (d)

Fig. 23. Untangling a double-coin knot. (a) The initial configuration of double-coin knot before untangling. (b) Aligning several crossings on a straight line for untangling. (c) Pulling string along straight line to untangle. (d) Knot is fully untangled after removing the rod.



(a) (b) (c) (d)

Fig. 24. Untangling knot 7_1. (a) The initial configuration of knot 7_1 before untangling. (b) Aligning several crossings on a straight line for untangling. (c) Pulling the string endpoint along a straight line to untangle. (d) Knot is fully untangled after removing the rod.

accompanying Extension 1, we have demonstrated how to use our approach to immobilize and arrange any given knot, and also demonstrate how rearranging of the geometry can greatly simplify tying and untying of knots.

For future work, we would like to better understand how motions can be designed to mechanically simplify tying knots and untying even tightened knots. We are also interested to know whether by changing the geometry of the goal, we can manipulate other flexible objects, such as cloth, using simple motions.

We are particularly interested in knots such as the shoelace and sheepshank; humans tie these knots by pulling loops through loops. We can identify these structures from the Gauss code, as type II Reidemeister moves: for each adjacent appearance, crossings *i* and *j* have the same sign, and the sign is different from the crossings adjacent to the *ij* (or *ji*) sequence. Therefore, our approach applies to unknots such as the shoelace or sheepshank, with small modification to the crossings removal process.

Acknowledgements

We would like to thank Dmitry Berenson and Gregory Fisher for use of the Da Vinci robot for some of the experiments.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship and/or publication of this article: This work was supported by the NSF (grant number IIS-1217447).

References

Adams C (2004) *The knot book: an elementary introduction to the mathematical theory of knots*. Providence, RI: American Mathematical Society.

Agarwal PK and Sharir M (1998) Arrangements and their applications. In: *Handbook of Computational Geometry*. Amsterdam: Elsevier, pp. 49–119.

Armstrong MA (1983) *Basic Topology*. New York: Springer-Verlag.

Ashton T, Cantarella J, Piatek M and Rawdon E (2011) Knot tightening by constrained gradient descent. *Experimental Mathematics* 20(1): 57–90.

Attaway SW (1999) The mechanics of friction in rope rescue. In *International Technical Rescue Symposium*.

Balkcom DJ, Trinkle JC and Gottlieb EJ (2002) Computing wrench cones for planar contact tasks. In: *ICRA*. Piscataway, NJ: IEEE, pp. 869–875.

Baranska J, Przybyl S and Pieranski P (2008) Curvature and torsion of the tight closed trefoil knot. *The European Physical Journal B - Condensed Matter and Complex Systems* 66(4): 547–556.

Bell MP (2010) Flexible object manipulation. Technical Report TR2010-663, Dartmouth College, Computer Science, Hanover, NH. Available at: http://www.cs.dartmouth.edu/reports/TR2010-663.pdf (accessed: 6 January 2018).

Bell MP, Wang W, Kunzika J and Balkcom D. Knot-tying with four-piece fixtures. *The International Journal of Robotics Research* 33: 1481–1489.

Berard S, Egan K and Trinkle JC (2004) Contact modes and complementary cones. In *Proceedings of ICRA*, pp. 5280–5286.

Biedl TC, Demaine ED, Demaine ML, et al. (1999) Locked and unlocked polygonal chains in 3D. In: *Proceedings of the Tenth*

Annual ACM-SIAM Symposium on Discrete Algorithms, 17–19 January 1999, Baltimore, MD, pp. 866–867.

Bretl T and McCarthy Z (2014) Quasi-static manipulation of a Kirchhoff elastic rod based on a geometric analysis of equilibrium configurations. *The International Journal of Robotics Research* 33: 48–68.

Burde G (1978) Knoten. *Jahrbuch Ueberblicke Mathematik* 1978: 131–147.

Carlen M, Laurie B, Maddocks JH and Smutny J (2005) Biarcs, global radius of curvature, and the computation of ideal knot shapes. *Physical and Numerical Models in Knot Theory* (Series on Knots and Everything, vol. 36). Singapore: World Scientific, pp. 75–108.

Crowell RH and Fox RH (1977) *Introduction to knot theory*. Berlin: Springer-Verlag.

Demaine ED, Demaine ML, Hawksley A, et al. (2010) Making polygons by simple folds and one straight cut. In: *Revised Papers from the China–Japan Joint Conference on Computational Geometry, Graphs and Applications (CGGA 2010)*, Dalian, China (*Lecture Notes in Computer Science*, vol. 7033). Berlin: Springer, pp. 27–43.

Hopcroft JE, Kearney JK and Krafft DB (1991) A case study of flexible object manipulation. *International Journal of Robotics Research* 10(1): 41–50.

Huffman DA (1971) Impossible objects as nonsense sentences. *Machine Intelligence* 6(1971): 295–323.

Huffman DA (1976) Curvature and Creases: A primer on paper. *IEEE Transactions on Computers* 25(10): 1010–1019.

Huffman DA (1977) Realizable configurations of lines in pictures of polyhedra. *Machine Intelligence* 8(1971): 493–509.

Inoue H and Inaba M (1985) Hand-eye coordination in rope handling. In *Robotics Research: The First International Symposium*, pp. 163–174.

Kanade T (1980) A theory of origami world. Artificial Intelligence 13(3): 279–311.

Kauffman L (1991) *Knots and Physics* (*K & E Series on Knots and Everything*). Singapore: World Scientific.

Ladd AM and Kavraki LE (2004) Using motion planning for knot untangling. *The International Journal of Robotics Research* 23(7–8): 797–808.

Livingston C (1993) *Knot theory* (*The Carus Mathematical Monographs*, vol. 24). Washington DC: The Mathematical Association of America.

Lui WH and Saxena A (2013) Tangled: Learning to untangle ropes with RGB-D perception. In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Tokyo, Japan, 3–7 November 2013, pp. 837–844.

Mason MT (2001) *Mechanics of Robotic Manipulation*. Cambridge, MA: MIT Press.

Manturov V (2004) *Knot Theory*. Boca Raton, FL: CRC Press.

Matsuno T and Fukuda T (2006) Manipulation of flexible rope using topological model based on sensor information. In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2006)*, Beijing, China, 9–15 October 2006, pp. 2638–2643.

Mishra B, Schwartz JT and Sharir M (1987a) On the existence and synthesis of multifinger positive grips. *Algorithmica* 2(1): 541–558.

Mishra B, Schwartz JT and Sharir M (1987b) On the existence and synthesis of multifinger positive grips. *Algorithmica* 2: 541–558.

Pai DK (2002) Strands: Interactive simulation of thin solids using Cosserat models. *Computer Graphics Forum* 21(3): 347–352.

Rawdon EJ (1998) Approximating the thickness of a knot. *Ideal knots* (*Series on Knots and Everything*, vol. 19). Singapore: World Scientific, pp. 143–150.

Reidemeister K (1927) Elementare begründung der knotentheorie. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg* 5(1): 24–32.

Rimon E and Blake A (1999) Caging planar bodies by one-parameter two-fingered gripping systems. *The International Journal of Robotics Reseseach* 18(3): 299–318.

Rimon E and Burdick J (1995) New bounds on the number of frictionless fingers required to immobilize 2D objects. In: *IEEE International conference on Robotics and Automation*, Nagoya, Aichi, Japan, pp. 751–757.

Rimon E and van der Stappen AF (2012) Immobilizing 2-d serial chains in form-closure grasps. *IEEE Transactions on Robotics* 28(1): 32–43.

Saha M and Isto P (2006) Motion planning for robotic manipulation of deformable linear objects. In: *Proceedings IEEE International Conference on Robotics and Automation*, pp. 2478–2484.

Saha M and Isto P (2007) Manipulation planning for deformable linear objects. *IEEE Transaction on Robotics* 23(6): 1141–1150.

Saha M, Isto P and Latombe J.-C. (2006) Motion planning for robotic knot tying. In *Proceedings of the International Symposium on Experimental Robotics*, July 2006.

Takamatsu J, Morita T, Ogawara K, Kimura H and Ikeuchi K (2006) Representation for knot-tying tasks. In: *Proceedings IEEE International Conference on Robotics and Automation*, vol. 22, pp. 65–78.

Wakamatsu H, Arai E and Hirai S (2006) Knotting/unknotting manipulation of deformable linear objects. *The International Journal of Robotics Research* 25: 371–395.

Waltz DL (1972) *Generating Semantic Descriptions From Drawings of Scenes With Shadows*, Massachusetts Institute of Technology, Cambridge, MA, USA. Available at: http://www.ncstrl.org:8900/ncstrl/servlet/search?formname=detail&id=oai%3Ancstrlh%3Amitai%3AMIT-AILab%2F%2FAITR-271.

Wang W and Balkcom D (2016) Grasping and folding knots. In: *IEEE International Conference on Intelligent Robots and Systems (ICRA)*, pp. 3647–3654.

Wang W, Bell MP and Balkcom DJ. Towards arranging and tightening knots and unknots with fixtures. *International Workshop on the Algorithmic Foundations of Robotics (WAFR 2014)*, pp. 677–694.

Wang W, Berenson D and Balkcom DJ. An online method for tight-tolerance insertion tasks for string and rope. In *IEEE International Conference on Robotics and Automation, (ICRA 2015)*, Seattle, WA, 26–30 May 2015, pp. 2488–2495.

Appendix A. Index to multimedia extensions

Archives of IJRR multimedia extensions published prior to 2014 can be found at http://www.ijrr.org, after 2014 all videos are available on the IJRR YouTube channel at http://www.youtube.com/user/ijrrmultimedia

Table of Multimedia Extension

Extension	Media type	Description
1	Video	The media extension demonstrate how to reconstruct the 3D configuration of any given knot based on the input Gauss Code; and the video also demonstrate how the rearranging of the knots can simplify the tying and untying of many knots.

Appendix B. proofs of certain lemmas

Complete proofs of certain of the lemmas appear only in this appendix.

Proof of Lemma 1. On a knot unit, consider the first crossing and last crossing, and ignore the links before first and after last crossing. Then, these two crossing vertices have degree 3. Denote these vertices as v_0 and v_n, corresponding to the first and the last crossing on the Gauss code. Since this is a section of a knot diagram, the graph contains a Eulerian path starting from v_0 to v_n.

If a knot path is only adjacent to the exterior face, it must be a bridge. Since the knot unit does not contain a bridge, no knot path on the projected polygonal arc of the knot unit will adjacent only to the exterior face. Therefore, there exists a set of connected knot paths ∂E that are adjacent to both an exterior face and to an interior face. So, these exterior knot paths are closed.

Since all the three- or four-degree vertices are crossings (intersections of edges on the projected plane), and all such intersections are represented by these vertices, ∂E does not self-intersect.

Since all the knot paths in ∂E are closed and have no self-intersections, they form a Jordan curve. □

Proof of Lemma 2. Let there be $m + k$ vertices in the planar polygonal drawing of the knot, such that there are k crossings, with degree four or three, and m other vertices with degree two. By counting edges, we find that there are $m+2k-1$ edges. Let the number of faces be $|f|$. By Euler's theorem, $|v|+|f| = |e|+2$, and $|f| = m+2k-1+2-(m+k) = k+1$ faces, k of which are interior. □

Proof of Lemma 3. First, there are a total of k crossings, so $m + n = k$. For k crossings, where each crossings appears twice in the Gauss code, there are a total of $2k - 1$ knot paths between crossings, so $i + j = 2k - 1$.

Each exterior knot path is on the boundary of one interior face. Each interior knot path is on the boundary of two interior faces. When we consider the crossings. For the first and the last crossing, they can only be adjacent to two interior faces. For the rest of the exterior crossings, each of them is adjacent to three interior faces (four degrees cuts the plane into four pieces, one of which for the exterior vertex is the exterior face). For an interior vertex, each vertex is adjacent to four interior faces. Therefore, because for all the closed interior faces, the number of boundary crossings equals the number of boundary knot paths,

$$2i + j = 4m + 3(n-2) + 4 \tag{7}$$
$$\Rightarrow i + (2k-1) = m + 3k - 2 \tag{8}$$
$$\Rightarrow i - (k-1) = m. \tag{9}$$

We always have $m \geq 0$, so $i \geq k - 1$. In addition,

$$m + n = k \tag{10}$$
$$\Rightarrow i - (k-1) + n = k \tag{11}$$
$$\Rightarrow i + n = 2k - 1 \tag{12}$$
$$\Rightarrow j = n \tag{13}$$

For an arbitrary knot, we know that the boundary of the planar drawing is closed; therefore, all the exterior edges form a cycle, which involves at least three crossings.

Let us assume only two crossings 1 and 2 are exterior crossings, and crossing 1 is both the first and the last label on the Gauss code. Then this crossing is formed by a type I Reidemeister move, so it can be removed, which leads to a different knot diagram and Gauss code. Then, the only possibility is if 1 is the first label and 2 is the last label on the Gauss code. It can be easily checked that such sequence will result in more crossings on the exterior knot paths. Therefore, $n = j \geq 3$. □

Proof of Lemma 4. Agarwal and Sharir (1998) indicated that n independent line segments (not sequential line segments) can create at most $n(n-1)/2$ intersections. However, for a sequence of n line segments (polygonal arc), there is one common endpoint between each pair of adjacent line segments, which results in $n - 1$ common endpoints. Therefore, the maximum number of intersections for a sequence of n line segments is $n(n-1)/2 - n - 1$ or $(n-1)(n-2)/2$.

Given n sequential line segments, at most $(n-1)(n-2)/2$ intersections (crossings) can be created. To make it a knot diagram, we need to have $(n-1)(n-2)/2 \geq k$. Therefore, $n \geq \left\lceil \frac{3+\sqrt{8k+1}}{2} \right\rceil$. □

Proof of Lemma 5. We know that the on the knot diagram, there exists an Eulerian path from the first crossing to the last crossing. Each of the crossings is visited twice. A crossing e that is not the first or last crossing has degree four, which means it is on a segment of polygonal curve that projects to e^+, and also on a segment of polygonal curve that projects to e^-. Therefore, on the polygonal knot diagram, e has two incoming edges and two outgoing edges. One pair of incoming and outgoing edge is associated with e^+, and the other pair is associated with e^-.

Without loss of generality, denote the incoming edges as $(a,\ e^s)$ and $(c,\ -e^s)$, and outgoing edges as $(e^s,\ b)$ and

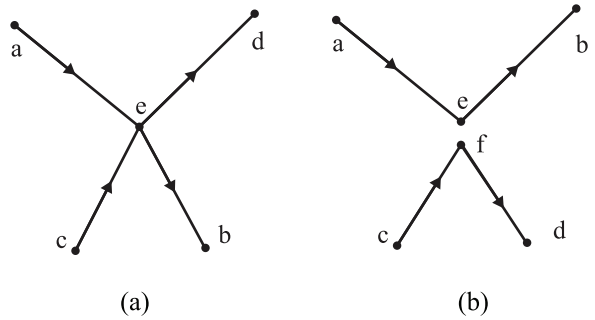


Fig. 25. (a) An example of the edges related to a crossing. (b) If the edges are not structured correctly, the vertex can be separated into two.

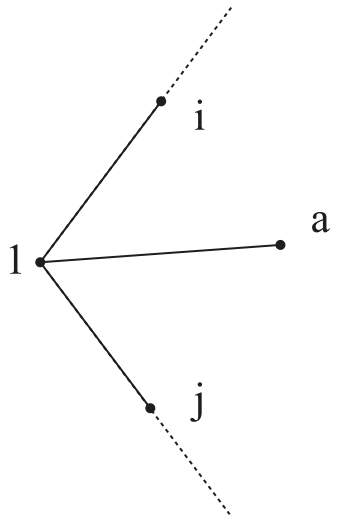


Fig. 26. The line segments that pass through the first crossing.

$(-e^s,\ d)$. Construct two rays starting from e and passing through c and d, respectively; denote them as R_{ec} and R_{ed}. These two rays separate the plane into two disconnected regions, and a and b must belong to different regions, as shown in Figure 25a.

If a and b belong to the same region, then the line-segment sequence (a, e, b) projects an over-crossing (or under-crossing) over (under) line segment (c, e, d). Then, either these two line segment sequences form a type II Reidemeister move (as shown in Figure 25b) where e can be redrawn as two independent vertices or more than two points projects at e, which is a violation of the regular projection property.

Without loss of generality, let us assume knot path $(a,\ e^s)$ is an exterior knot path. Then the knot path contacting the same exterior face will be another exterior knot path, which is either $(c,\ -e^s)$ or $(-e^s,\ d)$ since a and b belong to two different region separated by rays R_{ec} and R_{ed}. □

Proof of Lemma 6. The polygonal arc starts from the first crossing and ends at the last crossing, and the polygonal knot diagram is achieved by trimming the first and last link, making the first and last crossing the new end points of the polygonal arc. These crossings must be adjacent to the exterior face, making them exterior crossings.

Let us consider the first crossing. Without loss of generality, let the first crossing be 1^s. On the polygonal knot diagram, the first crossing has only degree three, with only one outgoing edge associated with 1^s, denoted as $(1^s, a)$, and two edges associated with -1^s, denoted as $(i, -1^s)$ and $(-1^s, j)$, respectively.

If we draw two rays starting from -1^s and passing through i and j, then the two rays separate the plane into two regions with one of them containing edge $(1^s, a)$. Therefore, $(1^s, a)$ cannot be connected to the exterior face, so it cannot be on the boundary. The same argument can be made for the last crossing. □

Proof of Lemma 7. All exterior knot paths form a regular convex polygon P. Because no chord of P will intersect the boundary except at endpoints, no exterior knot path will intersect with non-exterior knot paths between exterior crossings.

Now we will show that chords formed by connecting exterior crossings that are adjacent in the Gauss code do not intersect except at common endpoints. Consider four exterior crossings a, b, c, d, such that a and c are adjacent in the Gauss code, and b and d are adjacent in the Gauss code. Line segment ac cuts P into two pieces. If b and d belong to different pieces of P with respect to ac, then the only way to avoid intersection between ac and bd using any arbitrary curve is if b and d are connected outside P. However, knot path $(b,\ d)$ is not an exterior knot path; therefore to be consistent with the Gauss code, b and d must be in the same piece. □

Proof of Lemma 8. The polygonal arc starts from the first crossing, and ends at the last crossing, and the polygonal knot diagram is achieved by trimming the first and last link, making the first and last crossing the new end points of the polygonal arc. These crossings must be adjacent to the exterior face, making them exterior crossings.

Let us consider the first crossing. Without loss of generality, let the first crossing be 1^s. On the polygonal knot diagram, the first crossing has only degree three, with only one outgoing edge associated with 1^s, denoted as $(1^s, a)$, and two edges associated with -1^s, denoted as $(i, -1^s)$ and $(-1^s, j)$, respectively.

If we draw two rays starting from -1^s and passing through i and j, then the two rays separate the plane into two regions with one of them containing edge $(1^s, a)$. Therefore, $(1^s, a)$ cannot be connected to the exterior face, so it cannot be on the boundary. The same argument can be made for the last crossing. □

Proof of Lemma 16. If we remove all the crossings on the weaving sequence and there is only one weaving sequence, the remaining crossings are all warp crossings by definition. Then, if we continue to remove crossings, every crossing they remove will have the same sign as the next crossing to

remove. Without loss of generality, let us assume the first crossing we will remove is an over-crossing. Then, since all remaining crossings are warp crossings, whenever we are trying to remove a crossing i^a, a is $+$ until all crossings are removed. Then, all these warp crossings can be arranged on two layers. One plane contains only the over-crossings, while the other plane contains all the under-crossings, with finitely many vertical line segments connecting two planes. This structure has the topology of a circle when the ends of the string are connected to each other, an *unknot*. □

Proof of Lemma 17. A weaving sequence contains an alternating over- and under-crossing pattern, and all the crossings on the weaving sequence are adjacent to each other on the Gauss code. If the same label j appears twice, let crossings i^- and k^- be the two crossings on the Gauss code adjacent to j^+, and let s^+ and t^+ be the two crossings adjacent to j^-. The crossings i, k, s and t have the corresponding signs because they are adjacent crossings to j, and they are on the same weaving sequence. Without loss of generality, let j^- be closer to the open end. After the deletion of the crossing j^-, crossings i and k are now adjacent on the Gauss code, and they have the same sign, so they cannot be on the same weaving sequence. □

Proof of Lemma 18. Let us consider the case where, following Algorithm 5, three or more crossings are aligned. For any two adjacent crossings, since the configuration is already polygonal, they must be on the same line.

First, let us consider where only three consecutive crossings are aligned. When three crossings are aligned, the number of line segments used to connect them in sequence changed from three to two. However, if the first and the last crossing was originally connected with a single line segment, one extra line segment is needed to connect the aligned crossings, avoiding overlapping. Therefore, no extra line segment (finger) is needed.

In the process of changing the configuration proposed in Algorithm 5, extra line segments (or fingers) are used only when originally non-intersecting straight line segment was used to connect two crossings that are being aligned. For each one of such connection, one extra line segment will be used.

Let us arrange $m \geq 4$ crossings on a straight line. One line segment is used to connect all crossings in sequence, instead of $m-1$ line segments, reducing $m-2$ line segments. Originally, there could be at most $m-2$ non-intersecting line segments connecting them. Since each crossing can have at most 4 connections, and we are only considering the connection among the m crossings, there can at most be $m-2$ non-intersecting connections. Thus, at most $m-2$ extra line segments can be used.

Since the original representation uses $2k-1$ line segments, the reconfigured configuration uses no more than $2k-1$ line segments. □

Proof of Lemma 21. Let the two ends of a knot be S_1 and S_2, and let S_1 be fixed to ground. Let the base of the robot arm be B and let the end effector be E. The robot arm needs to grasp S_2 with E during the arrangement of the knot. Let us assume no re-grasp is performed during the arrangement of a weaving sequence w. At the end of the arrangement of the weaving sequence, let the configuration of the string be fixed in space. Let curve c_1 be the current configuration of the robot arm, connecting from B to E. Let curve c_2 be the curve of a different robot arm configuration connecting from B to E, where the entire robot arm is outside the convex hull of the knot. In both configurations, the E is attached to S_2. Then, the two curves belong to two different homotopy classes with respect to the string. The curve connecting c_2 to S_2 and then to S_1 has the correct knot topology. Therefore, at least one re-grasp is needed to arrange a weaving sequence. □