KnotSketch: a tool for knot diagram sketching, encoding and re-generation

Gennaro Costagliola[1], Mattia De Rosa[1], Andrew Fish[2], Vittorio Fuccella[1], Rafiq Saleh[3], Sarah Swartwood[2]
[1]Dipartimento di Informatica, University of Salerno, Via Giovanni Paolo II, 84084 Fisciano (SA), Italy
[2] School of Computing, Engineering and Mathematics, University of Brighton, UK
[3]Department of Earth, Ocean and Ecological Sciences, University of Liverpool, UK
{gencos, matderosa, vfuccella}@unisa.it, {andrew.fish,S.Swartwood}@brighton.ac.uk

Abstract

Knots occur in many areas of science and art. The mathematical field of Knot Theory studies an idealised form of knots by viewing them as closed loops in 3-space. They can be formally studied via knot drawings which are well-behaved projections of the knot onto the 2-D plane. Equivalence of knots in 3-space (ambient isotopy) can be encapsulated via sequences of diagram rewriting rules, called Reidemeister moves, but finding such sequences demonstrating isotopy of two knots can be immensely challenging. Whilst there are some sophisticated tools available for some knot theoretic tasks, there is limited (free) tool support for certain knot creation and interaction tasks, which could be useful for lecturers and students within University courses. We present KnotSketch, a tool with multiple functionalities including the ability to: (i) read off a form of Gauss code for a user sketched diagram; (ii) generate a diagram from such a code; (iii) regenerate a knot diagram via a different projection, thereby producing examples of equivalent knot diagrams that may look very different; (iv) interaction capabilities to quickly alter the knot via crossing changes and smooth the curves of the sketched diagram; (v) export facilities to generate svg images of the constructed knots. We evaluate KnotSketch via a case study demonstrating examples of intended usage within an educational setting. Furthermore, we performing a preliminary user study to evaluate the general usability of the tool.
Keywords: *knots, sketching, gauss code, diagram generation.*

1. Introduction

Knots occur within both art and science, and there are many important scientific application domains (e.g. DNA supercoiling [22, 8], quantum wavefunctions [16]). The mathematical field of Knot Theory has been studied extensively, providing a rigorous study of an idealised form of knots (essentially closed loops in 3-space); see [27] for a standard mathematics graduate text book, and [19] for a recent approach aiming to utilise the potential of computers within the field. A standard mathematical approach is to define objects under consideration, provide a formal notion of equivalence, and then to investigate means to try to classify the objects (given two objects, decide if they are equivalent or inequivalent). In this context, two knots are equivalent, called ambient isotopic (isotopic for short), if there is a continuous deformation of the whole of 3-space taking one embedded loop to the other. Knots can be studied thus, making use of knowledge of topological methods.

However, knots can also be studied combinatorially, via knot diagrams (which are regular projections of knots onto a plane, so any crossings are transverse double points). Following Reidemeister [25], isotopy can be realised via diagram transformations: two knots are isotopic if and only if their diagrams (which can be projection onto any suitable plane, so the diagrams can look quite different) differ by a sequence of local diagrammatic transformations (shown later, in Figure 4). Figure 1 shows an instance of the core problem of asking whether diagrams represent the same knot; no answer is provided here, leaving the reader to try to discover an answer for themselves, thereby getting an initial feeling for the challenge via this small example. This question is posed at the beginning of an undergraduate Knot Theory course at the University of Brighton, UK. Since there can be infinitely many diagrams of each knot (considering that different planes of projection can be used, and one could move parts of the knot prior to projection), identifying knot equivalence by comparing diagrams and trying to demonstrate isotopies can be extremely difficult.

Consider the educational context of a knot theory course, with a professor teaching, setting and marking assessments of students, whilst the students are learning, studying, and completing the assignments. The act of drawing of knots can be time consuming and error prone. For example, consider students trying to reproduce the knot shown in Figure 2 in class by hand; this is actually a diagram of the unknot (i.e. it is

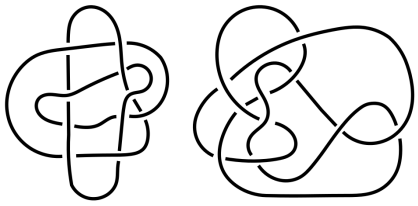


Figure 1: Are these two knots equivalent (isotopic)?

DOI reference number: 10.18293/DMS2016-035

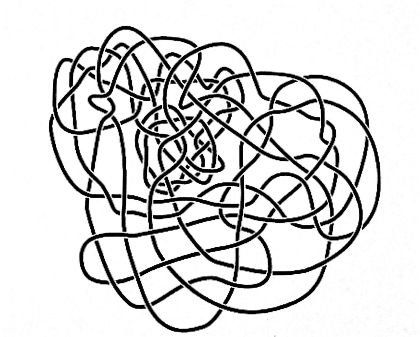


Figure 2: A knot drawing with potential for reproduction errors

equivalent to a knot diagram with no crossings, which can be depicted by a single circle), called the Haken unknot. This motivates the need for a knot drawing tool, with export facilities, and in particular the incorporation of sketch based input facilities to mimic the normal knot drawing method when constructing knots by hand.

Knot codes (like a Gauss code, discussed in Section 2.1) provide a string based representation of the crossings (and their relative orientations) in a knot diagram. In the process of understanding their usage, a student may be asked to produce the code of a given diagram or to try to construct a diagram, given a code, for relatively simple cases. The question "which signed Gauss codes are realisable as (classical) knots?" [17] was open for a long time, but algorithmic solutions have been devised (e.g. [9, 23, 28]). Kauffman [20] extended the class of classical knots (the knots we have described are termed classical knots) to virtual knots, permitting extra crossings without any over/under information, so that every Gauss code is then realisable as a virtual knot. In this paper, we focus on the classical knot case, as would be common in the majority of knot theory courses, but we adopt an underlying approach that can be naturally extended to permit the consideration of virtual knots. The related requirements for a tool are: the automatic reading of the code from a knot drawing, and the automatic generation of a knot diagram from a code.

The codes considered only determine the drawing of a knot diagram drawn on a sphere instead of on a plane; a choice of face (in the shadow of the knot on the sphere, which is the underlying graph given by forgetting the over/under information at the crossings) for stereographic projection is required to provide a drawing in the plane. However, this property can be turned into an interesting benefit. Different choices of (outer) face yield isotopic diagrams (they are, after all just different views of the same knot) but they may look very different. Thus, the alternative choices of outer face give rise to a set of diagrams which are all equivalent but may not look like it, thereby: (i) giving rise to easily constructed examples of equivalence for students to explore; and (ii) helping students to develop their understanding of stereographic projection. Without any tool support, students may find taking a diagram in the plane, considering it as drawn on the sphere and re-projecting using a different outer face, a rather challenging task. It also provides the teacher (or a professional mathematician if the context of intended usage was widened) with a means of examining this somewhat less-familiar relationship between knots that may involve very complex sequences of Reidemiester moves to realize as an isotopy.

We implemented the KnotSketch tool following these elicited requirements. We demonstrate some of its functionalities via a case study, providing a series of worked examples. We follow this up with a preliminary user study to evaluate the general usability of the tool. There are many technicalities to be dealt with in the formal setting of knot theory, but for accessibility to a Computer Science audience we adopted a relatively informal approach, skipping over some technical details (e.g. we presume all knots are tame to rule out pathological examples). An interested reader can refer to graduate level text books on knot theory (e.g. [15, 18]) for more complete details. Discussions about knots (single loops embedded in space) extend to links (disjoint unions of knots) and KnotSketch also supports links.

2. Preliminaries

We describe some facts about knots and their diagrams (informally indicating the content of well known definitions and theorems), and introduce Gauss codes, with examples, providing rationale for the chosen form adopted. A link is a disjoint union of knots, and a link *diagram* is the image of a regular projection (i.e. the only singularities are transverse double points) of the link L with over/under information added at each of the double points, called *crossings*. Every (tame) link has a diagram. Ambient isotopy is an equivalence relation on knots (or links). Each equivalence class of knots is called a *knot type*; equivalent knots have the same knot type. As is common, we abuse language and use the term 'knot' to mean the whole equivalence class (a knot type) or a particular representative. When we say that two knots are different (not equal) we mean that they are inequivalent (i.e they have different knot types). If a knot has the same type as the trivial knot then we say it is *unknotted*. We can orient a knot by nominating one of the two directions along it. If K is an oriented knot then the knot with the opposite orientation, denoted by $-K$, is called the *reverse* of K. The knot $-K^*$ is called the *inverse* of K, where K^* is the mirror image of K (obtainable by switching all crossings in a diagram of K). For oriented links, we can assign to each crossing c of a diagram, its *sign*, which is a value in ± 1, denoted $sign(c)$, as depicted in Figure 3.

Let R_1, R_2 and R_3 denote the diagrammatic moves shown in Figure 4. Two diagrams differ by one of these moves if they are identical outside a small region, and inside the region they differ exactly as shown in the moves. These moves are called the *Reidemeister moves* [25]. We can think of R_3 as moving one of the strands across the crossings of the other two strands. The moves are presumed to preserve orientation. Note that we

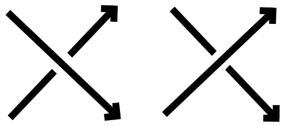


Figure 3: The overpass is the unbroken line for each crossing (on left figure: top left to bottom right; on right figure: bottom left to top right) and the underpass is the broken line. The sign of a crossing is $+$ if traversing along the the underpass, following the orientation, at the crossing the overpass is passing from the left to the right (as in the figure on the left), whilst the sign is $-$ if the overpass is passing from the right to the left (as in the figure on the right).

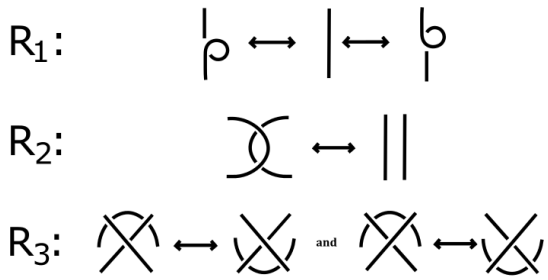


Figure 4: The Reidemeister moves, which are diagrammatic moves encapsulating knot equivalence.

interpret diagrams that differ by homotopy preserving the arc and crossing structure (i.e. the arcs can be moved without changing the underlying crossing structure; e.g. making the knot more "wiggly", or scaling the diagram) as the same.

We say that diagrams D and D' are *isotopic* if D can be obtained from D' by a sequence of moves of type R_i, with $1 \leq i \leq 3$. The diagrams are *regularly isotopic* if R_1 is not used. The following important theorem (see [25]) allows us to study knots and links combinatorially, via diagrams: Suppose that links L_0 and L_1 have diagrams D_0 and D_1, respectively. Then L_0 and L_1 are ambient isotopic if and only if D_0 and D_1 are isotopic.

2.1. Codes

Gauss codes are a means of capturing information in a knot diagram. We adopt a richer form of the codes used in the literature from which others can be recovered. The code of a diagram is given by numbering the crossings, picking an orientation on the curves and then traversing the curves one at a time, writing down the crossings met in order in a complete circuit, noting whether one was on the overpass or underpass (o/u) at each crossing along with an associated sign ($\pm$). It is well defined up to the equivalence relation generated by these choices. Codes for different components of a link are separated by a $/$.

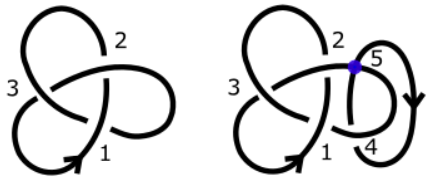


Figure 5: Left: The enhanced Gauss code $o_{1+}\, u_{2-}\, o_{3+}\, u_{1-}\, o_{2+}\, u_{3-}$ indicates the crossings met upon traversal of the knot, together with extra information (over/under, and the sign of the associated immersed curve), realisable as a (classical) knot diagram. Note that the "usual" Gauss code would assign $-$ to every crossing number, yielding: $o_{1-}\, u_{2-}\, o_{3-}\, u_{1-}\, o_{2-}\, u_{3-}$. Right: the code enables the explicit indication of the position of virtual crossings: $o_{1+}\, u_{2-}\, o_{3+}\, u_{1-}\, o_{4-}\, v_{5+}\, o_{2+}\, u_{3-} \,/\, u_{4+}\, v_{5-}$ for the 2-component virtual link. Here, we emphasized the virtual crossing by placing a dot at the crossing (numbered 5).

We describe variations of the code used in the literature. Kauffman, in [20], uses o/u, and the sign of the crossing ($+/-$ described earlier) is attached to each symbol (denoting the number assigned to the crossing), with the same sign occurring for both of the occurrences of this symbol in the code. However, Kurlin [21] includes the same sign of the crossing ($+/-$) but only attaches it to the symbol associated to the undercrossing, thereby removing the need to explicitly indicate o/u's, since the presence of a sign indicates an undercrossing, whilst its absence indicates an overcrossing. Carter [11], considers immersed curves (so the arcs pass through each other at the crossing instead of passing over and under), so there are no o/u's to consider. A variation of the sign convention is adopted, with one $+$ and one $-$ associated to each occurrence of a symbol for a crossing. An intuitive method for calculating this (enhanced) sign convention used is to imagine that the arc under consideration (as we traverse the curve and it passes through a crossing we write down one instance of the symbol for that crossing and decide on its sign) is an undercrossing and calculate the sign as per the earlier method for knots. This leads to one of the two symbols being assigned to a crossing being a $+$ and the other a $-$.

We make use of a variation, which we call the *enhanced Gauss code* (*code* for short) in which we use the sign convention for immersed curves, following Carter, as well as the o/u information (as used by Kauffman) for knots. This generalisation provides us with the ability to deal with curves that do not have any over/under information at the same time as those that do. One application is to explicitly encode virtual crossings in virtual knot diagrams (rather than simply ignoring their presence in the code), permitting the expression of over/under or "through/virtual" at each crossing. The "through/virtual" option can naturally be represented by a v for virtual crossing in the virtual knot diagram setting. Whilst we do not explicitly consider the use of virtual knots within the educational con-

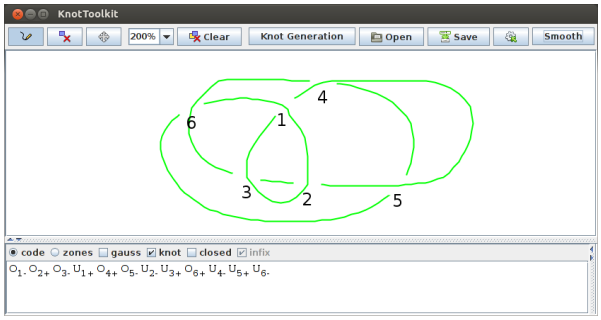


Figure 6: A screenshot of the KnotSketch interface, with *toolbox* at the top, a diagram shown in the *Sketch View* and its automatically produced code in the *Code View*.

text in this paper, we envisage the future usage of this functionality (for advanced postgraduate students or professional mathematicians), and so we decided to adopt a notational convention that would be consistent with this.

Figure 5 shows examples of the (enhanced Gauss) codes of a classical knot diagram (top) and a virtual link diagram (bottom). To enable a comparison with the immersed curve codes, one can consider the shadow of these classical knot diagrams (i.e. make all crossings "through/virtual"); the effect of this on the code is to simply remove the occurrences of o/u, replacing them with v's. This would have necessarily been slightly more complicated if we had used the classical knot sign convention in the code. KnotSketch enables the automatic interpretation of the code of (sketched) link diagrams.

3. KnotSketch

We describe KnotSketch and its main functionalities. KnotSketch is partly based on ink recognition techniques, previously developed for other applications [6, 13, 14].

3.1. User Interface

As shown in Figure 6, the KnotSketch interface is divided into three parts. The upper part is a *toolbox* with buttons to perform the following operations (from left to right):

- change the input mode to *draw*;
- change the input mode to *erase*;
- change the input mode to *move*. At present it is only possible to move whole curves, not parts of a curve (but this would be a useful future additional functionality for interaction);
- change the zoom level;
- clear the current drawing;
- launch the automatic diagram generation facility through a new dialog. The dialog (see Figure 7) contains a text field to enter the code. It is possible to use the clipboard to paste the automatically produced code of a sketched diagram. The code specifies the drawing of the knot on a sphere. By clicking on the OK button, a new dialog is opened (an example is presented in Figure 10 of Section 4). This dialog requires the user to choose a face assigned to be the outside region under stereographic projection from the sphere onto the plane from a list of possible choices (one for each region). A preview pane is shown by the side of the list to assist the user. Moreover, it is possible to personalize the diagram appearance with different visualization options, such as varying the number of spring embedding iterations;
- open a previously saved diagram. If a file describing a diagram drawing (see below) is opened then its content is shown in the *Sketch View* described below, while if a file containing the knot code is opened the diagram generation is launched;
- save the current diagram in a native drawing file format. It is also possible to export a drawing in *svg* format (optionally as an *html* file), whilst the code can be exported in *txt* or *html* format. This facilitates the use of the tool for the production of diagrams that occur in research papers, student submissions, or to include in web-pages;
- open the Options window in order to alter tool settings, such as the size of the gap used in the visualisation of the crossings;
- perform knot beautification using EulerSmooth [30]. Given a drawn diagram, the user can apply a smoothing operation via EulerSmooth to improve the quality of the sketched drawing; EulerSmooth was defined to work with Euler diagrams but it can be heuristically applied to knot diagrams. The smoothing function can be manually controlled and can be applied with various parameter settings.

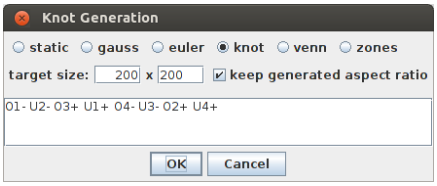


Figure 7: Code to diagram generation box.

In the middle part of the KnotSketch interface the *Sketch View* contains the drawing canvas. The curves are arbitrarily shaped and must be completed with a single pen stroke. Once a stroke has been entered, its endpoints are automatically joined to close the curve. If the user tries to make the curve closed by passing the final part of the curve over the initial part, then any additional crossing created is erased. Upon its completion, a curve is coloured with a colour chosen from a pre-defined list (the predefined colour list can be configured by the user). As soon as a curve is completed, a numeric label is automatically assigned to each crossing point. There are two segments passing through each crossing, and by default the last drawn segment is shown as an over-crossing (as a future option, the user will be able to choose the default behaviour, permitting the automatic provision of alternating

crossings as the knot is traversed, for example); the crossing type (over/under) can be manually changed by the user by clicking on the crossings. The length of the gap used to indicate the crossing can also be configured by the user.

In the lower part of the interface, the *Code View* contains the automatically generated enhanced Gauss code, as described in Section 2.1 (it is possible to select the code and copy it to the clipboard). At the top of this view a toolbox provides a set of options through which the user can select some other forms of codes (that are not required for this particular application).

A user study is discussed in Section 5, and Figures 8-12 show some related screenshots of the tool.

3.2. Back-End

KnotSketch is written as a Java 7 application. Other than knot drawing, the main features of the application are knot interpretation (i.e. computing the code of a drawn knot), knot generation from code and knot beautification. The enhanced Gauss code, displayed in the *Code View*, is incrementally constructed and is updated every time a curve is added or deleted. It is stored in an internal format enabling efficient operation executions. A knot is represented as a closed polyline.

KnotSketch can generate knot drawings from an enhanced Gauss code using a planar graph based construction: the graph contains one node for each crossing, and its combinatorial embedding of the graph can be calculated by using the crossing signs $(+/-)$ and the choice of face assigned to be the outer region. Then an algorithm from the OGDF framework [24] can be used to embed the planar graph in the plane. The user can optionally modify the embedding by selecting a number of iterations of a force directed algorithm [5] to apply. The algorithm theoretically preserves the regions, but approximation due to rounding could cause non-preservation. However, the tool provides a post-check if the initial and final diagram codes are equivalent. Finally, by traversing the graph edges appropriately, the curve for the knot diagram is constructed.

The knot beatification step can be performed by using the Ocotillo java library from EulerSmooth. The drawn knot is treated as if it was an Euler diagram (essentially taking the shadow of the knot, viewing all crossings as if they passed through instead of over/under each other; note that the constraints of non-self intersections for Euler diagrams must be relaxed in this context). The knot diagram is converted to the format used by Ocotillo and the beautification process is performed on it, using the parameters set by the user. After each beautification iteration the diagram is converted back to the original format in order to display the beautification progress to the user. Future enhancements of EulerSmooth in this context should lead to enhancements of the outputs.

4. Case study

We describe a set of activities that can be performed with KnotSketch within an educational context, demonstrating the capabilities of the approach and the tool. We highlight a novel viewpoint that permits an uncommon form of user exploration, which would be challenging for students to perform without tool-based assistance. We discuss general activities that users may undertake along with low level tasks that users may perform along the way.

Consider the task of starting with a code and creating a diagram of a knot with that code. This task can be used to test a users' understanding of the code to diagram process, and to highlight how challenging the construction can be without further assistance. In a simple case, the user can directly sketch the requested diagram. They can check their solution via the automatic production of the code, displayable in the *Code View*. If the knot produced differs from the required solution via crossing changes alone (i.e. switching over and under crossings) then user interaction via clicking on the crossings can be applied to alter the diagram accordingly. If the diagram drawn has the wrong code and the code differs by more than crossing changes then the curve needs to be redrawn. An alternative approach is to use the automatic diagram generation facilities from a code (described in Section 3). For the task of producing a diagram with the correct code, any choice of outer face will do, so the user can select any of the proposed faces. This functionality leads to an interesting possibility for the exploration of knot equivalence (isotopy). Given a code one can generate different views of the knot (one for each face, by using the different choices of outer face in the plane of projection). Since each of the views are diagrams of the same knot viewed from a different perspective, all of these diagrams are equivalent. However, they may look very different and it may be challenging for a user to construct an isotopy (a sequence of Reidemeister moves) from one diagram to the other. Thus, given two knot diagrams that potentially look very different, but which are in fact different projections of the same knot, the user can try to identify this knot equivalence by searching through the options and visually comparing the outputs.

Figure 8-12 show diagrams from the tool similar to those of the case study. In detail, Figure 8 shows a user sketched (and smoothed) drawing of a trefoil knot with 3 crossings, its automatically produced code, along with a more complicated knot with 7 crossings. Figure 9 shows diagrams of a knot K, the knot K' with a single crossing change, and, $m(K)$, the mirror of the knot K, which is shown as K with all crossings changed; the effects on the code are also shown. Figures 7 and 10 show an example of a code being used to generate a knot, demonstrating the user's view at the time of selection of outer face. The code-like information displayed for each choice is a means of specifying the region (details are omitted since they are unimportant for our purposes here); we display drawings arising from all six choices of outer face for this code. Figure 11 shows diagrams (far left and far right) and their (not obvious) equivalence as a sequence of Reidemeister moves (indicated by R_i's). These two diagrams can be seen to be equivalent via projection onto different faces; the left and right diagrams are those shown in Figure 10(f) and Figure 10(e), respectively, up to moving the arcs without changing

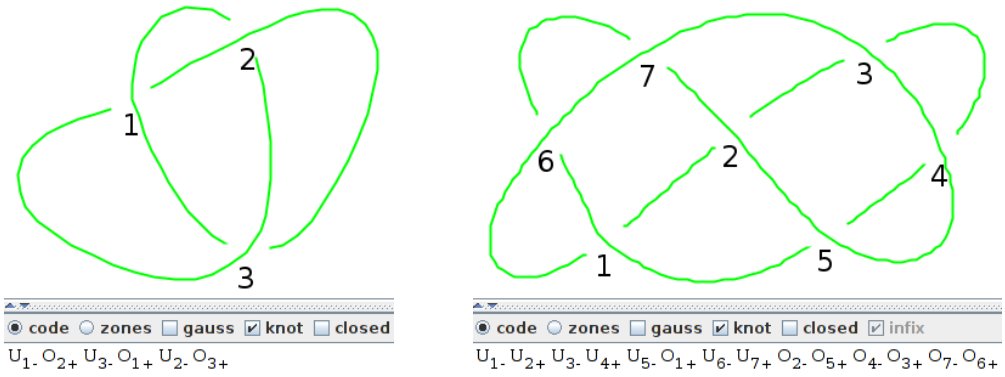


Figure 8: Screenshots of a trefoil knot (left) and a knot with seven crossings (right) and their codes.

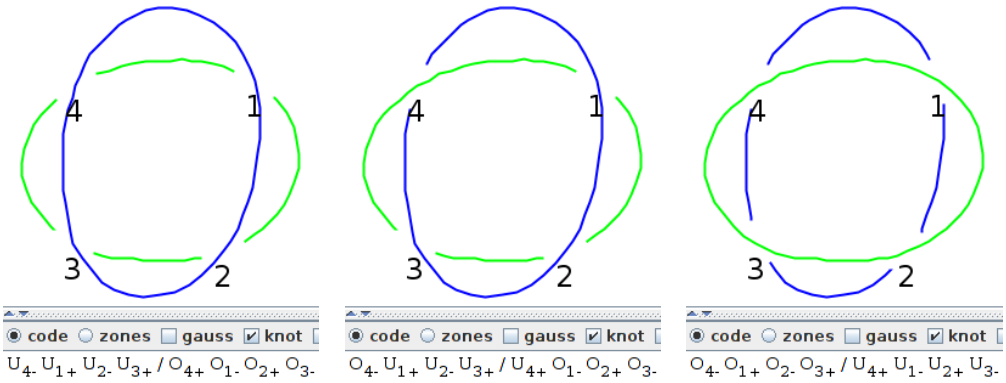


Figure 9: A diagram of a link (left), the diagram obtained by changing crossing 4 (middle), and the mirror of original diagram, given by changing all crossings (right).

the crossing structure. Figure 12 shows an example of generation of a knot diagram from a complicated code, showing the initial creation of a knot, along with an initial improvement using the EulerSmooth functionalities.

5. Experimental setup

We performed a preliminary evaluation on a student population to test the usability of the tool itself. In particular, we measured the perceived usability of the system through a System Usability Scale (SUS) [7] questionnaire. The questionnaire is composed of 10 statements to which participants assign a score indicating their strength of agreement on a 5-point scale. The final SUS score ranges from 0 to 100. Higher scores indicate better perceived usability. We also gathered some participants' free-form comments during and after the experiment.

The demographic mostly consisted of Computer Science students, which may help with familiarity with the use of software. In the future, we will also examine the usage with Mathematics students whom may be more familiar with the mathematical context, but perhaps less proficient in software usage. We recruited a total of 10 participants (3 female). The ages ranged from 21 to 51 (with mean $M = 31$, and standard deviation $SD = 10.4$). All of them were habitual computer users and had previous experience with touch-screens. None of them had any prior knowledge of knot theory.

The set of tasks we considered were as follows.

TASK 1 Given the following knot code, generate a diagram of it using the "generation from knot code" functionality of KnotSketch: $o_{1+}\ o_{2-}\ u_{3+}\ u_{1-}\ u_{4+}\ u_{5-}\ o_{5+}\ o_{4-}\ u_{2+}\ o_{3-}$;

TASK 2 Given the knot drawn on the sheet, reproduce it using KnotSketch and save its knot code in a text file. Moreover, save the drawing in the tool's native file format (this task was repeated three times - TASK 2.1, TASK 2.2, TASK 2.3 - once for each knot in Figure 13);

TASK 3 Open the native format of the drawing previously produced in TASK 2.2 and generate its projection onto a

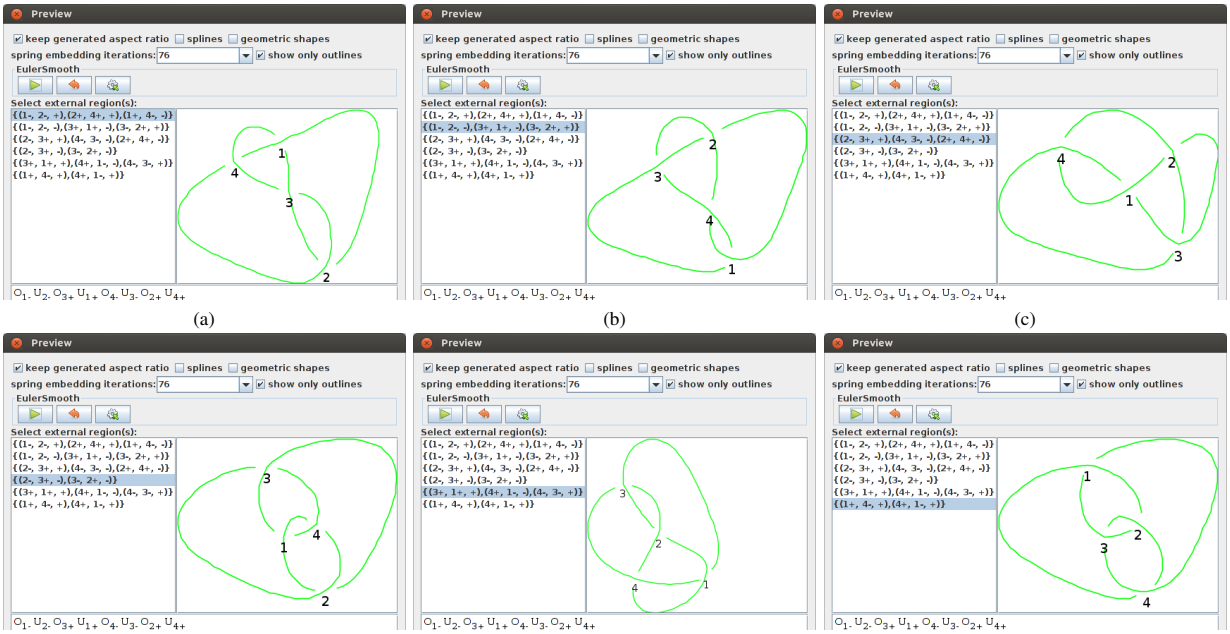


Figure 10: Displaying the preview of the different choices of outer face for the projection, using the code from Figure 7.

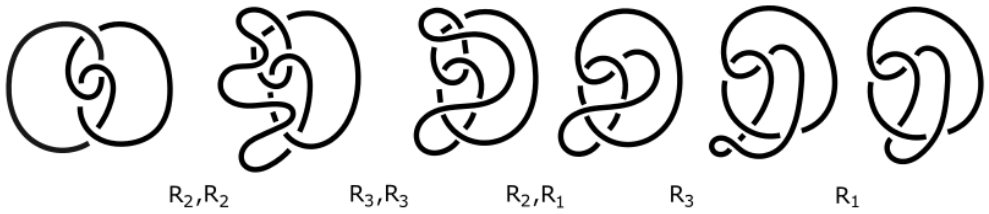


Figure 11: A sequence of Reidemeister moves demonstrating the equivalence between the left diagram (c.f. the diagram in Figure 10(f)) and the right diagram (c.f. the diagram in Figure 10(e)).

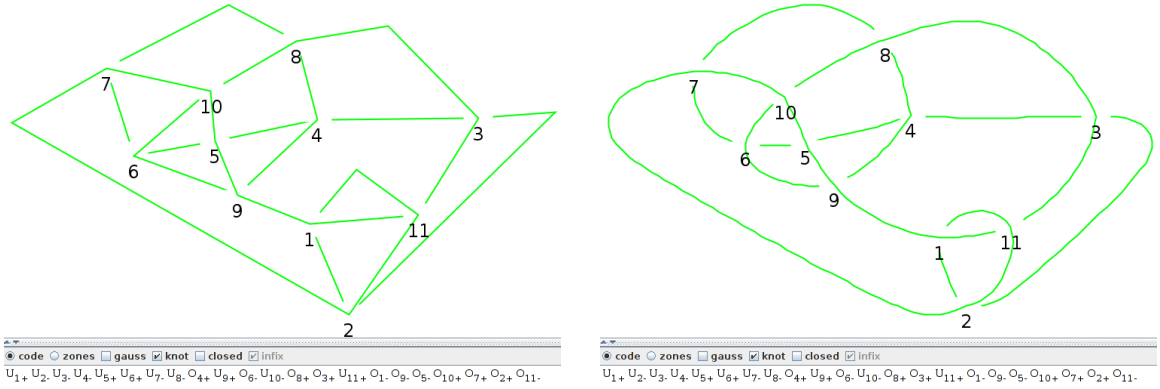


Figure 12: Generation of a knot diagram from a complicated code, with the initial creation of a knot (left), along with an initial improvement using the Eulersmooth functionalities (right).

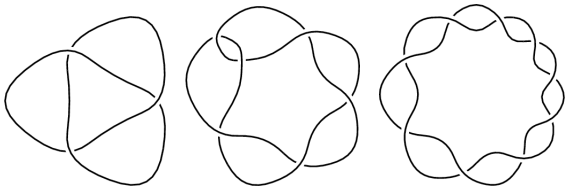
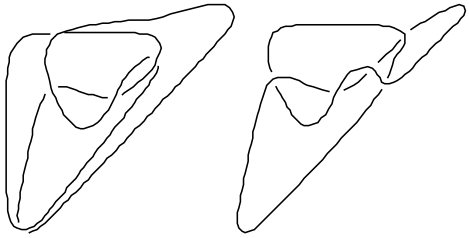
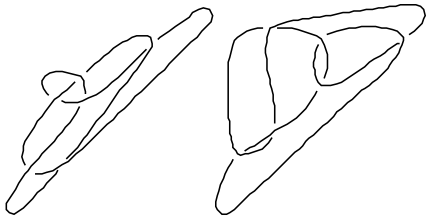


Figure 13: TASK 2 knots.



(a) Pair of knots for TASK 4.1.



(b) Pair of knots for TASK 4.2.

Figure 14: TASK 4 knot pairs.

different outer face;

TASK 4 Given the pairs of knot diagrams on the sheet, indicate if they differ by redrawing via projection onto a different outer face or not (the task was repeated twice - TASK 4.1, TASK 4.2 - once for each knot pair in Figure 14).

The time limit for the completion of each subtask was 5 minutes.

The tasks were executed on a *Dell Precision T5400* workstation equipped with an *Intel Xeon* CPU at 2.50 GHz running *Microsoft Windows* operating system and the Java Run-Time Environment 8. The device used for the experiment was a Sympodium ID250 Interactive Pen Display, attached through both USB and RGB cables to the work station.

5.1. Results

The tasks have been successfully completed by almost all participants. Only two tasks were not completed successfully by all participants: TASK 2.3 and TASK 4.1, with two errors each. The mean task completion times are reported, along with standard deviations, in Table 1. As expected, the subtasks of TASK 2 showed increasing difficulties; in particular for TASK 2.3, some users needed to attempt the drawing several times. However, the error on this task was simply an incorrect sign on a single crossing. TASK 4 presented some difficulties for most participants, with relatively high execution times and two errors for TASK 4.1.

Task	Avg time (sec.)	S.D.	User errors
TASK 1	64.7	16.6	0
TASK 2.1	45.3	14.1	0
TASK 2.2	60.2	13.3	0
TASK 2.3	137.6	72.8	2
TASK 3	20.0	11.8	0
TASK 4.1	135.0	48.8	2
TASK 4.2	152.6	41.6	0

Table 1: Task completion times.

Username	Score
participant 1	77.5
participant 2	85
participant 3	62.5
participant 4	75
participant 5	87.5
participant 6	75
participant 7	85
participant 8	77.5
participant 9	92.5
participant 10	85
Average	80.25

Table 2: SUS-like questionnaire scores for user satisfaction.

The scores of the questionnaire calculated from the responses of the participants are shown in Table 2, indicating scores ranging from 75 to 92.5, with an average value of 80.25. This value indicates a good level of satisfaction [4]. An analysis of the questionnaire scores showed that the participants judged the tool very useful for its purpose, but they did not find all the features simple to use and some participants needed some learning time to be able to master them. For instance, some participant lamented the lack of feedback on the presence of errors in a written code; others had problems in finding out how to generate the projection of a drawing onto a different outer face.

Similar considerations emerged after a brief interview/discussion with the participants involved in the experiment to understand what aspects of the tool were of help in the execution of the tasks: probably due to unfamiliarity with Knot Theory, some participants expressed initial doubts about the comprehension of the tool's purpose. Furthermore, some of them had difficulty in completing TASK 4 even with the help of the tool, with one participant stating: "I would have preferred more support from the tool to perform TASK 4". Nevertheless, he declared "I would never be able to complete that task without the help of the tool".

Tool	KnotSketch	KnotPlot [26]	Knot ID [2]
Hand sketch	Yes	Yes (but requires license)	No (point and click/drag)
3D visualisation	No	Yes	Yes
Link diagrams	Yes	Yes	No
Save/Load/Export diagram	• Supports saving and loading diagrams • Diagrams can be exported in SVG and HTML format	• Supports saving and loading diagrams • Diagrams can be exported in EPS format	• Load list of coordinates • Load gauss code (no visualisation of the knot in this case)
Edit diagram	• Supports adding, deleting and moving individual components	• Supports adding, deleting and moving individual components or a segment of any component	• Supports adding, deleting or modifying individual segments of any component
Diagram generation via code	Supports Gauss Code	Supports Dowker code and Braid word	No generation (but identification available from Gauss Code)
Diagram reprojection via outer face	Yes	No (but rotation of 3D visualisation is possible)	No
Diagram smoothing	Yes	Yes	No

Table 3: Visualisation features of KnotSketch, KnotPlot and KnotID

6. Related Work

Existing knot tools have varied functionalities. Table 3 provides comparison of visualisation aspects of KnotSketch with KnotPlot [26] and Knot ID [2]. We provide some further details and briefly discuss other notable tools below. To the best of our knowledge, the re-projection facility onto different outer faces offered by KnotSketch is novel, as is the capability to deal with (sketching, interpreting) virtual knots or links (note that the study was restricted to classical knots and links).

KnotPlot [26] is a widely used program for visualising and interacting with knots that has theoretical underpinnings as described in [29]. A knot can be sketched by hand (in the upgraded version requiring a license) or constructed from code using a tangle notation system developed by Conway [12], or from a braid word, or a Dowker-Thistlethwaite code (which does not uniquely specify composite knots [26, page 98]). A knot can be refined into a smoother configuration using a 3D technique, which is computationally expensive, especially for large knot diagrams with more than 50 crossings [26, page 105]. KnotPlot has a built-in database containing a wide range of knots and links which can be viewed, modified and saved, along with other features such as: computing the Alexander and HOMFLY polynomials, the writhe, the average crossing number, the thickness and the Dowker code of a knot; searching for minimal stick conformations and interesting random knots; generation of arbitrary braids; enabling the consideration of open knots or links; interactive construction of knotted and linked spheres in four dimensions.

KnotID [2] is a web application that allows the viewing of topological information about knots. The application can be used: (1) to import three-dimensional curves (as a list of 3D coordinates or by generating torus knots from two coprime numbers); (2) to draw two-dimensional curves; (3) to directly input an enhanced version of the Gauss code similar to those described in Section 2.1. In addition to displaying topological information, the tool can, through the application of topological invariants, identify if the input knot is equivalent to knots found in a lockup table (based on [1]). It permits computation of properties such as the Reduced crossing number, Determinant $|\Delta(-1)|$, $|\Delta(\exp(2\pi i/3)|$, $|\Delta(i)|$, and Vassilev invariants. Compared to KnotSketch, the software offers a non-sketching oriented drawing interface and no features such as delete/smoothing/load/save or the ability to display the crossing numbers. Moreover it does not offer functions to generate a knot from a gauss code or to regenerate a knot by reprojection onto a different plane.

Libbiarc [10] can be used to manipulate and analyze properties of knotted curves, compiled into a $C++$ library; the most frequently used functions are available as an API. The core library provides functions to interpolate point-tangent data with bi-arcs, access information such as curvature and torsion, compute the length and thickness. The library includes a viewer application, called curview, for visualisation, in which a knot can be loaded, manipulated and saved.

KnotApp [31] is a thesis that describes a program that displays a knot and other objects, such as the knot's crossing map and its trisecant curve, using the jReality JOGL Viewer. The knot may be chosen from a provided list or loaded from the file system. Then the chosen knot can be edited by dragging its vertices using the provided knot editor. The crossing map implemented displays the set of irregular projections of the underlying knot as curves on the unit sphere. The set of trisecants of the underlying knot is visualised as a curve in the 3-dimensional torus. However, the KnotApp application itself does not appear to be currently available to be tested.

Other notable knot tools, with varying functionalities include Linknot (see [19]), Knotscape [32] (uses Dowker-Thistlethwaite codes), and MING [33] (which minimizes MD-energy of polygonal knots, reduces the numbers of edges, and draws/visualizes their 3-dimensional pictures).

7. Conclusions

We have developed KnotSketch, a tool to facilitate knot exploration, with particularly interesting functionalities of permitting knot sketching followed by classical knot diagram regeneration via re-projection onto different planes via stereographic projection from the diagram drawn on the sphere. This could be particularly useful in University level educational settings, and we examine the potential capabilities via a case study. In addition, we adopt an enhanced version of Gauss code that enables the future handling of virtual knots. This ability to deal with virtual knots would likely be useful in the context of expert users, such as professional mathematicians making use of the tool in exploratory mathematics or simply to produce figures for use in publications (the use of classical knots is of course pertinent to this user class as well). One can also envisage different semantical interpretations of these types of diagrams with mixed crossing types, and this would widen the applicability even further beyond the consideration of knots.

We performed a preliminary user-based empirical study to gain insight into the usability of the toolkit, with an aim for future improvements. We plan to perform more extensive user testing, and adopt an iterative developmental methodology taking into account user-insights after testing periods. We wish to explore the utility for both Mathematics students and professional Mathematicians. Using the interface at KnotInfo [3], a user can select sets of knots and request information about them including their diagrams or compute many invariants. This interface and database (and others) could be made to be usable in conjunction with KnotSketch, thereby greatly enhancing its functionalities.

References

[1] Knot atlas take home database. At `http://katlas.org/wiki/The_Take_Home_Database`.

[2] Knot ID website. At `http://inclem.net/knotidentifier/`.

[3] Knot info website. At `http://www.indiana.edu/~knotinfo/`.

[4] A. Bangor, P. T. Kortum, and J. T. Miller. An Empirical Evaluation of the System Usability Scale. *International Journal of Human-Computer Interaction*, 24(6):574–594, 2008.

[5] F. Bertault. A force-directed algorithm that preserves edge-crossing properties. In *Graph Drawing*, volume 1731 of *Lecture Notes in Computer Science*, pages 351–358, 1999.

[6] P. Bottoni, G. Costagliola, M. De Rosa, A. Fish, and V. Fuccella. Euler diagram codes: Interpretation and generation. In *Proceedings of the 6th International Symposium on Visual Information Communication and Interaction*, VINCI '13, pages 105–106, New York, NY, USA, 2013. ACM.

[7] J. Brooke. Sus: A quick and dirty usability scale. In P. W. Jordan, B. Weerdmeester, A. Thomas, and I. L. Mclelland, editors, *Usability evaluation in industry*. Taylor and Francis, London, 1996.

[8] D. D. Buck. EP/H031367/1 SANDPIT : Knots and Evolution - Topologically Driven Integrase Mutagenesis, and EP/G039585/1 DNA Knotting and Linking: Applications of 3-Manifold Topology to DNA-Protein Interactions, 2009.

[9] G. Cairns and D. Elton. The Planarity Problem II. *Journal of Knot Theory and its Ramifications*, 5:137–144, 1996.

[10] M. Carlen. *Computation and Visualisation of Ideal Knot Shapes*. Ph.d. thesis, EPFL, Lausanne, 2010.

[11] J. Carter. Classifying immersed curves. In *Proc. Amer. Math. Soc*, volume 111, pages 281–287, 1991.

[12] J. H. Conway. An enumeration of knots and links, and some of their algebraic properties. *Computational Problems in Abstract Algebra*, pages 329–358, 1970.

[13] G. Costagliola, M. De Rosa, A. Fish, V. Fuccella, and R. Saleh. Curve-based diagram specification and construction. In *Proc. VL/HCC*, pages 39–42, 2013.

[14] G. Costagliola, V. Fuccella, and M. Di Capua. Interpretation of strokes in radial menus: The case of the keyscretch text entry method. *Journal of Visual Languages & Computing*, 24(4):234 – 247, 2013.

[15] P. Cromwell. *Knots and Links*. Cambridge University Press, 2004.

[16] M. Dennis and A. Taylor. Vortex knots in tangled quantum eigenfunctions. *Nature Communications*, (12346), 2016.

[17] C. Gauss. Werke. *Band 8. Teubner*, 1900.

[18] N. Gilbert and T. Porter. *Knots and Surfaces*. Oxford Science Publications, 1997.

[19] S. Jablan and R. Sazdanovi. *LinKnot: Knot Theory by Computer*, volume 21 of *Series on Knots and Everything*. World Scientific, 2007.

[20] L. Kauffman. Virtual knot theory. *European Journal of Combinatorics*, 20:663–691, 1999.

[21] Kurlin. Gauss paragraphs of classical links and a characterization of virtual link groups. *Math. Proc. Camb. Phil. Soc.*, 145:129–140, 2008.

[22] R. Metzler. Knots, bubbles, unwinding, and breathing: Probing the topology of DNA and other biomolecules. *Handbook of Theoretical and Computational Nanotechnology, edited by M. Rieth and W. Schommers*, 1:1–54, 2005.

[23] J. Nagy. Über ein topologisches Problem von Gauss. *Mathematische Zeitschrift*, 26(1):579–592, 1927.

[24] OGDF. Open graph drawing framework. `http://www.ogdf.net`.

[25] K. Reidemeister. Elementare begrøundung der knotentheorie. *Abh. Math. Sem. Univ. Hamburg*, 5:24–32, 1926.

[26] R.G.Scharein. *Interactive topological drawing*. Ph.d. thesis, The University of British Columbia, 1998.

[27] D. Rolfsen. *Knots and Links.* AMS Chelsea Press, 2003.

[28] P. Rosenstiehl and R. Tarjan. Gauss codes, planar Hamiltonian graphs, and stack-sortable permutations. *Journal of algorithms*, 5(3):375–390, 1984.

[29] R. Scharein. *Interactive Topological Drawing*. PhD thesis, University of British Columbia, 1998.

[30] P. Simonetto, D. Archambault, and C. Scheidegger. A simple approach for boundary improvement of Euler diagrams. *IEEE Transactions on Visualization and Computer Graphics*, 22(1):678–687, 2016.

[31] M. Sommer. *Visualization in Geometric Knot Theory - Understanding the mathematical structure of trisecants*. Ph.d. thesis, Technische Universität Berlin, 2007.

[32] M. Thistlethwaite. Knotscape. At `http://www.math.utk.edu/~morwen/`.

[33] Y.-Q. Wu. Ming. At `http://www.math.uiowa.edu/~wu/`.