視覚的な例からの学習による紐結び作業の実現

宮崎　麻衣子 *　　池内 克史 *

あらまし　これまで柔軟物操作の分野において、プランニングと操作の手法は別々に研究されてきた。しかし、両者を統合して一つの柔軟物操作システムを構築しようとすると実装上の様々な問題を解決する必要が出てくる。
本研究では紐結び作業を行うシステムを実装する新たな手法を提案する。紐結び作業のプランニングには観察による紐結び動作学習 (KPO) パラダイムを利用し、人間の示す手本から効率的に紐結びのプランを生成した。紐の操作にはプランニング時に与えられた手本から視覚情報を抽出することでロボットの動作を生成する手法を開発した。これによりロボットを動作させるのに必要なパラメータが簡単に獲得が可能となった。
また本論文では実験結果を示し、本手法に基づくシステムでは実装上のいくつかの問題に対する解決を与えることができたことを述べる。さらに、実験を通して新たに明らかとなった実装上の問題点を報告する。

Knotting Rope by Learning from Visual Examples

Maiko MIYAZAKI**　　Katsushi IKEUCHI**

Abstract　In the field of deformable object manipulation, planning and manipulation methods have long been studied separately. Yet, to put together these methods to implement one whole manipulation system, a variety of implementational problems need to be solved.
This work presents a new approach to implement a knot tying system. For the planning phase, we used the *Knot Planning from Observation* (KPO) paradigm, which generates plans based on visual examples demonstrated by humans. For manipulation, we developed a method to determine robot commands by extracting visual information from the examples obtained in the planning phase. This yields a simple solution to determine manipulation parameters.
Experimental results are also reported, indicating that our proposed system solves some of the implementational problems in deformable object manipulation.

1 Introduction

Not only are robots presently playing an important role in manufacture, but they are also expected to accomplish a much wider variety of tasks in the future. To be able to work on more intelligent activities, manipulating objects, especially deformable objects such as a rope, is an important activity the robot must learn to do. This activity has been studied for over a decade.

Almost all research methods on rope manipulation concentrated on how to apply visual feedback. Inaba et al. built a hand-eye manipulation system that checks and corrects manipulation operations [1]. Hopcroft et al. introduced a vision-based high-level language for describing knot tying and implemented a manipulation system using it [2]. Matsuno et al. used dual manipulation systems together with a vision system [3].

These methods forced a user to manually write programs to execute rope manipulation tasks. This was inefficient because the programs took much time to write, needed to be newly written for each task, and had to be rewritten every time the task was altered. This gave rise to the demand to automatically generate the programs through determining what operations should be done on the object to accomplish a task. In short, what should be considered is so-called *task planning*.

In the first step of task planning, modeling of a deformable object was activated. The ability to predict deformation of the object when applying external force enables one to plan tasks through computer simulation. The most common modeling tool is the Finite Element Method (FEM), which is commonly used for a wide variety of objects. For example, rope [4], cloth [5], etc. was successfully modeled based on this method. But it was quite difficult to plan the tasks using the model from only the goal of the manipulation, because a deformable object allows quite various types of operations. Furthermore the FEM requires much computation.

As one solution to realize both task planning and programming for manipulation simultaneously, we focused on the Learning from Observation (LFO) paradigm. One characteristic of this paradigm is to use observed data in acquiring task plans, which dramatically decreases the difficulty of programming. As a result, automatic programming becomes possible, and the need for expertise in robot programming is completely eliminated. This is the reason the LFO paradigm has been applied to various manipulation systems that handle rigid objects [6, 7, 8].

In this paper we propose a system that implements both planning and manipulation steps for tying knots in a rope on the basis of the LFO paradigm. We chose the knot tying task for two reasons. First, because a rope (and similar objects, e.g., wire and cord) is one

* 東京大学生産技術研究所

** Institute of Industrial Science, The University of Tokyo

of the most simple but often-used deformable objects in our daily lives. Second, because knot tying has a good mathematical background called the knot theory [9], which clearly classifies knot tying operations into few categories [10]. Planning is conducted using a KPO paradigm which we have already designed [10]. Manipulation is achieved by reusing the observed data in the planning phase to simply determine parameters necessary for translating the plans into executable robot commands. We also introduce our first attempt to implement the Cross move, which is defined later.

2 Outline of the Knot Tying System

The present goal of our system is to tie a knot in a rope that is placed on a flat surface such as a table. We decided to lay the rope on a surface in order to ease image recognition. The system works in two phases: the planning and the manipulation phase.

The planning phase is built on the basis of the KPO paradigm [10]. This paradigm is an application of the LFO paradigm for knot tying. Based on the knot theory, it provides us with a methodology for describing and generating the plans. Concretely, the plans are generated from data acquired from examples shown by humans through performing the following steps:

1. Observation

   Obtain sequential images of a knot-tying performance

2. Conversion to Knot Geometry

   From each image, obtain a geometric representation of a knot state (K-data)

3. Conversion to Knot Topology

   From each geometric representation, convert it to a topological representation of a knot state (P-data)

4. Plan Generation

   From the sequence of topological representations, identify movement primitives

The generated plan is then passed onto the manipulation phase. Here, a knot tying task is executed according to the given plan by using visual information to extract robot command parameters. The manipulation phase works for each movement primitive in the following steps:

1. Object-level Parameter Acquisition

   Determine object-level parameters by using the movement primitive and the geometric representation

2. Present Knot State Acquisition

   Obtain the present state of the rope that the robot will manipulate

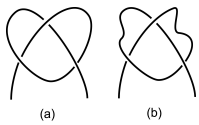

Fig. 1: Similar knots in terms of knot tying

3. Robot-level Parameter Acquisition

   Acquire robot-level parameters by using object-level parameters and the present knot state

4. Robot Command Conversion

   Generate and execute a sequence of robot commands using robot-level parameters

3 Theory for the KPO Paradigm

In this section, we roughly introduce the KPO paradigm [10], which provides us three important elements required for the planning phase, which are:

- How to represent knot states
- What to define as movement primitives
- How to identify the movement primitive that caused the knot state transition

3.1 Representation of Knot States

Considering a knot state representation appropriate for generating task plans, Morita et al. concluded that an abstract representation that does not depend on parametric information is preferable[10]. The reason can be shown through a simple example. The two knots illustrated in Fig. 1 are essentially the same in terms of how crossing occurs in the rope; that is, the knots are different in shape, but the same in the way they are knotted. Geometrically, however, they are judged as different knots. This is inconvenient because it would mislead the system to think a state transition should have occurred between the two states.

As such a representation, we use P-data representation. P-data is an array of numbers, and each column expresses the ID of crossed intersection and the attribute (vertical position and crossing direction relevant to the crossed strand) of the corresponding intersection. The topological information can be reconstructed from only P-data. An example of a 2D knot projection and its corresponding P-data is given in Fig. 2.

3.2 Definition of Movement Primitives

Morita et al. proposed the following four types of movement as movement primitives: three types of Reidemeister moves and the Cross move[10].

Reidemeister moves are introduced from the knot theory [9]. Figure 3 illustrates the three moves. The

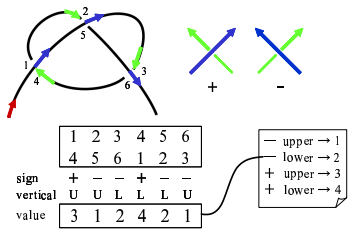

Fig. 2: 2D knot projection to P-data representation

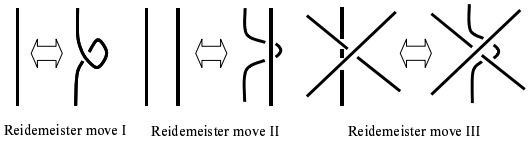

Fig. 3: The three Reidemeister moves

Cross move is introduced from the difference of target knots between in the knot theory (simple closed curve) and in knot tying (simple open curve). The illustration of a Cross move is given in Fig. 4.

The sufficiency of the movement primitives for realizing any knot tying can be easily shown by extending the Reidemeister's proof on knot-equivalence.

3.3 Movement Primitive Identification

The basic idea of identifying the primitives is by comparing the t-th P-data P_t and $(t+1)$-th P-data P_{t+1}. The P-data after the transition P_{t+1} can be estimated from the P-data before the transition P_t by using the type and parameters of the movement primitive that achieves the transition. Conversely, if P_{t+1} is equal to the P-data which is estimated from P_t under some type and parameters of a movement primitive, one can conclude the movement primitive was performed at time t.

4 Manipulation Phase

As the first attempt of the implementation, we chose the Cross move. Our choice of the Cross move was influenced by the fact that it is the most essential move in knot tying. That is, it is fundamentally possible to tie all types of knots by only this move. Furthermore, we concentrate on the Cross move to cross the terminal segment over the other segment. We took such a stance because the move can be realized by manipulation using only one hand. This is much easier than manipulation using both hands, which would introduce the problem of avoiding collisions; it is natural to use both hands to achieve a Cross move to cross the terminal segment under the other segment. However, once the collision can be avoided, the under-Cross move can be realized by only adding a few simple operations to the over-Cross move, and therefore it is considered appropriate to tackle the over-Cross move first.

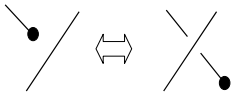

Fig. 4: The Cross move

The manipulation phase is passed on a knot tying plan from the planning phase, which is in this case a sequence of Cross moves. Each Cross move holds two pieces of information: which terminal to move and which segment to cross over/under. While this serves as adequate information for humans to conduct a Cross move, it is too abstract for robots. Humans can complement the abstract information to reproduce the move. On the other hand, robots need to be given more concrete information: the exact position and direction in robot coordinates, in other words, metric information. Thus, in this phase it is necessary to complement the abstract information so that it can be understood to robots. Only then can they be executed. In doing so, there are three things to be determined:

- what metric information we need
- how to get the information
- where to get the information from

Out of the three issues, we considered what metric information we need. As a result, we decided that three pieces of metric information would be sufficient to execute a Cross move. They are the grasp point, destination point, and the path from the grasp point to the destination point. The grasp point is the point where the robot hand grasps the rope and starts the Cross move. The destination point is where the robot hand releases the rope and ends the move. Given the position and direction of the robot hand at these two points in robot coordinates, the robot understands the start and goal of its move. In addition to these points, the robot need to be given the path from the grasp point to the destination point. The need of such path is characteristic to flexible objects. For flexible objects, the path which it moves along make great difference to the resulting state of the object. 5 shows an example of how the knot state could differ by taking two different paths with the same start and goal. The rope is held at the terminal on the left, and is moved clockwise in the upper picture and anti-clockwise in the lower one.

Then, in order to retrieve metric information, we divided the information obtaining process into two steps: first acquiring them in geometric form and then in metric. For this purpose, we defined two levels of parameters for expressing robot operation information: object-level parameters and robot-level parameters. Object-level parameters corresponds to geometric information. They give reference to a knot-relative, coordinate-independent positions and directions according to the rope's geometric structure. On

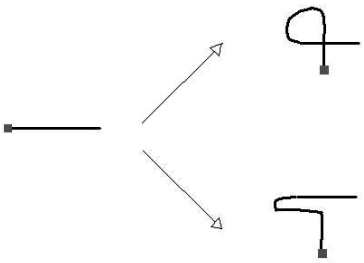

Fig. 5: Move of a rope through two different paths

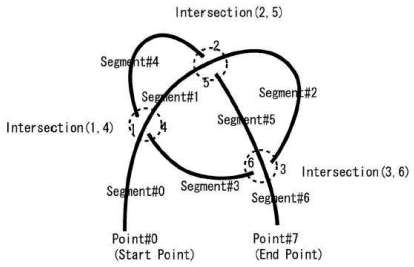

Fig. 6: Numbering of segments and points in a knot

the other hand, robot-level parameters corresponds to metric information. They give reference to the metric properties of a knot. We chose to use these two levels of representation because directly obtaining metric information would be problematic. Parameters depending on metric properties of a knot are apt to change and cannot be determined before manipulation. Dependence on metric properties would force a knot to conform to the shape. This is too restrictive. However, for the actual robot commands, metric information is essential.

Last of all, we decided to retrieve the geometric information from the knot images obtained in the planning phase. There are various methods to retrieve this information, so we should consider which method is most appropriate. There is a method to obtain necessary information through reconstruction of a knot from P-data [10]. While this gives a more flexible choice of the geometric representation to be made, it causes a new problem of choosing the most suitable representation for knot tying. Therefore, we obtained the knot's geometric data from the images acquired in the planning phase, regarding them as examples demonstrating the most suitable shapes for knot tying.

4.1 Preliminaries

4.1.1 K-data

K-data is a geometric representation of a knot state. It is made from 2D knot projections on the flat surface on which the rope is laid. K-data holds essentially two pieces of information, which are a vector of specific points on the rope and a vector of rope segments.

The vector of points in K-data consists of intersections and the two terminals of the rope. The points are numbered, starting from one terminal of the rope (the start terminal), then tracing along the rope, numbering each intersection according to its meeting order, and finally ending at the other terminal (the end terminal). Note that intersections will be given numbers twice. Each point holds information to tell whether it is an upper cross point (a point where the rope crosses over itself), a lower cross point (a point where the rope crosses under itself), or a terminal.

The segments are sections of a rope starting and ending at either an intersection or a terminal. The vector of segments is ordered similarly to the points; starting to number the segment with the starting terminal, then tracing along the rope, and numbering each segment according to its meeting order till reaching the segment with the end terminal. Each segment has properties of the starting and ending points and also a chain of directions. This chain describes the shape of the rope segment. We regard the segment as a chain of points for visual feedback purposes. For each point we obtain its direction, and hold this information as a direction chain.

Figure 6 is an example of a knot with its points and segments numbered as in the K-data.

4.1.2 Robot Commands

We present the robot commands we provided for knot tying. The function arguments are given in the robot coordinates. We assume that the Z-axis is in the direction upward from the flat surface which the rope is on. Robot commands are defined as follows:

- graspRope($\mathbf{p}$, θ)

  Grasp the rope at the position $\mathbf{p}$ ($\in R^3$). The robot hand will be rotated so that it will be parallel to the direction θ.

- moveRope($\mathbf{p}$, θ)

  Move the rope from the present position to the position $\mathbf{p}$. The robot hand should be in the direction θ at that position.

- releaseRope()

  Release the rope at the present position.

4.2 Object-level Parameter Acquisition

For each knot tying move, we obtained object-level parameters from the acquired task plan and the K-data of the two successive knots. In the plan we find the two segments (the terminal segment and the segment that it crosses) involved in the Cross move. Then, by comparing those segments in the pre-Cross-move K-data and pro-Cross-move K-data we completely extract object-level parameters. The Cross move parameters are defined as follows:

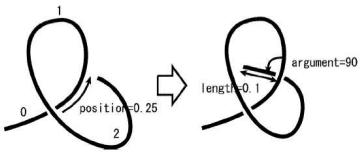

Fig. 7: Example of object-level parameters acquired from K-data

- *terminal*

  Which terminal to move (start/end_terminal)

- *segment*

  Segment number to cross

- *position*

  Relative position in the segment to cross ($0 < position < 1$, in which 0 is the starting point and 1 the ending point of the segment. The coordinates are named segment coordinates.)

- *direction*

  Terminal's vertical direction relative to the segment (over/under)

- *argument*

  Argument of the terminal's direction vector relative to the segment

- *length*

  Relative length of the new segment to the total rope length (Let the length be 1, then $0 < length < 1$. The coordinates are named rope coordinates.)

For example, parameters for the Cross move shown in Fig. 7 are determined as follows: *terminal* = end_terminal, *segment* = 1, *position* = 0.25, *direction* = under, *argument* = 90 [deg], *length* = 0.1.

4.3 Present Knot State Acquisition

In the present state acquisition step, we acquire geometric information of the knot that will be manipulated. First a 3D image of the knot is captured by stereo vision. Next the image will be projected onto the 2D plane and then converted to K-data. This will all be done in the same manner as in the observation, projection, and K-data conversion steps in the planning phase.

4.4 Robot-level Parameter Acquisition

In the robot-level parameter acquisition step, the aim is to obtain metric information of the grasp point, destination point, and a path that the rope should move along. The path is defined from positions and directions of the following three specific points on the path: the grasp point, the cross point, and the destination point. Note that we need to specify only one set of these three points per movement primitive.

The position of the grasp point G_{rope} is acquired in the rope coordinates as Equation (1).

$$G_{rope} = \begin{cases} length & (terminal = \text{start\_terminal}) \\ 1 - length & (terminal = \text{end\_terminal}) \end{cases} \quad (1)$$

In the implementation, we adjust G_{rope} so that it would not be too near or too far from the terminal of the rope in order to make a firm grip on the rope. We determine the position in the robot coordinates by finding in the 3D knot image the point that corresponds to G_{rope}. And we also determine the grasp direction using the direction chain of the segment.

The cross point is the point where the rope should make a crossing over/under itself. The position of the cross point C_{seg} is acquired in the segment coordinates as Equation (2).

$$C_{seg} = position \times (segment)\text{-th\_segment\_length} \quad (2)$$

The position and the direction in the robot coordinates can be determined as the same manner.

The destination point is the final point that the grasp point should move to after making a crossing at the cross point. The point lies in the direction of the cross point direction. The distance d between the cross and destination point is determined by Equation (3)

$$d = \begin{cases} (length - G_{rope}) \times \text{total\_rope\_length} \\ \qquad (terminal = \text{start\_terminal}) \\ (length - (1 - G_{rope})) \times \text{total\_rope\_length} \\ \qquad (terminal = \text{end\_terminal}) \end{cases} \quad (3)$$

Having considered the grasp, cross, and destination points, we consider the path. The path from the grasp point to the cross point is defined using the cubic Bezier curve, and the path from the cross point to the destination point is defined as the straight line. Then we concretely describe the setup of the cubic Bezier curve.

The curve has four control points. That is, we only need to set up the points. The first and fourth control points are the grasp and cross points so that they will be the two ends of the Bezier curve. The second control point is located on a half line starting from the grasp point and extending in the rope direction of the grasp point. The distance between the first and the second control point is given as half the distance between the grasp and cross points. We determined this distance, regarding it reasonable for the rope not to take too long a way around to the cross point. The third control point is determined in a similar way to the second, only that we use a half line starting from the cross point and extending in the opposite direction of the rope direction of that point.

4.5 Robot Command Conversion

Finally, a Cross move is translated into several robot commands and executed in the execution step. The Cross move will be executed in the following sequence:

1. graspRope($\mathbf{p}_{grasp}$, θ_{grasp})
2. Call moveRope(...) several times for realizing the path generated by the cubic Bezier curve.
3. moveRope($\mathbf{p}_{cross}$, θ_{cross})
4. moveRope($\mathbf{p}_{dest}$, θ_{dest})
5. releaseRope()

$\mathbf{p}_{grasp}$, $\mathbf{p}_{cross}$, and $\mathbf{p}_{dest}$ stand for positions of the grasp, cross, and destination points, respectively, and θ_{grasp}, θ_{cross}, and θ_{dest} stand for directions of the grasp, cross, and destination points, respectively. Note that while moving the rope, the Z-coordinates are set up to be appropriately far away from the surface that the rope is on (in this implementation 10[cm] higher).

5 Experimental Results

In this section, we discuss the experiments conducted for this paper. First we introduce the platform on which we built our system. Second, we describe how a knot image is acquired. Last, we some results of knot tying tasks.

5.1 Platform

The robot of our system is designed to imitate the upper part of the human body, especially the vision system (i.e., eyes), arms, and hands.

The vision system consists of a 9-eye stereo vision system for 3D observation. These eyes are set onto the head of the robot. The 9-eye stereo vision system is made by Komatsu and uses the multi-baseline method to recognize its surrounding environment in real-time.

The robot also has left and right arms, each with a hand at the end. The arms are PA10 robot arms made by Mitsubishi Heavy Industries. They have 7 degrees of freedom (DOFs). This is enough for the robot hand to move around in a wide area of three-dimensional space.

The robot hands imitate those of a human. Both hands have a thumb, and fingers that face the thumb. The left and right hands are composed of three and four fingers, respectively. The fingers are fewer than the human hand, considering the size and shape of the hand. Each finger has 3 joints, thus 3 DOFs. For the movement of joints we use the finger joint actuator made by Yaskawa Electric Corporation.

5.2 K-data Extraction

Here we explain how we acquire a 2D knot projection by using the vision system and how we extract K-data from the projection.

5.2.1 2D Knot Projection Acquisition

First, 3D images are captured through a vision system, and RGB and disparity data are acquired. From the disparity data we calculate the depth at each point in the captured image. The depth is given in camera coordinates, in which the coordinate center is fixed to the center of the 9-eye vision system. We then convert depth in the camera coordinates to that in the robot coordinates. As we always place the knot on a 2D plane, a 2D knot projection is simply the same as a 3D image data without the Z-coordinates.

5.2.2 K-data Conversion

From the 2D knot projection, we extract K-data. The conversion process is as follows:

1. Background subtraction

   Subtract background data from the image to extract only the knot. The field of view is assumed to be fixed for the subtraction.

2. Thinning

   Change the knot into a thin line by using Hilditch filter.

3. Graph-Representation Extraction

   Find intersections and terminals in the knot by counting the neighboring points for all pixels of the thin line image. And find all segments by following neighboring pixels from the intersection or the terminal until reaching another.

4. Reordering

   Reorder the segments and points (intersections and terminals) so that they are in an order that can be traced from one terminal of the knot to the other.

5. Vertical Position Extraction

   By using disparity data obtained from the vision system, determine vertical positions of the intersections.

5.3 Results of Tying a Simple Knot

We conducted an experiment to tie a simple knot in a rope. Due to the limitation of pages, we illustrated a part of the knot tying. The knot images captured from the human performance are shown in Fig. 8 (a). From these images, K-data is acquired. Figure 8 (b) illustrates how the knot states are recognized after K-data conversion. Their object-level parameters are given in Fig. 8 (c). Figure 9 is an illustration of the calculated path for the Cross move in Figure 10. Figure 10 shows the results of manipulation using the parameters.

Looking at the path generated for the Cross move, it appears that it was appropriate for producing the

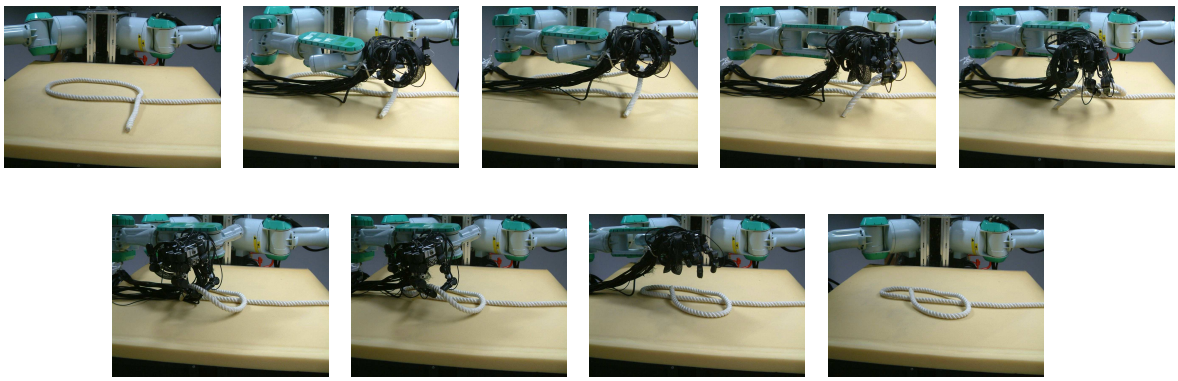

Fig. 10: Manipulation of Cross move

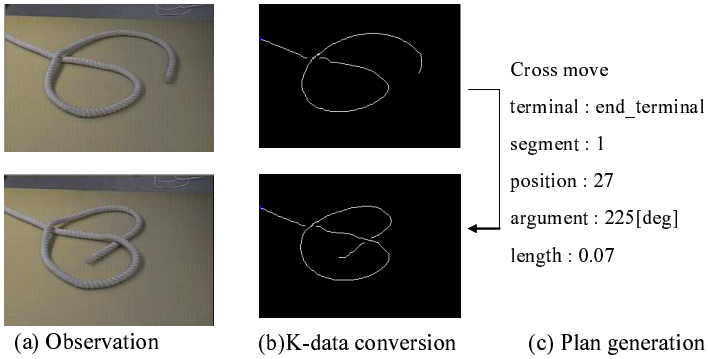

Fig. 8: Knotting of a part of a simple knot

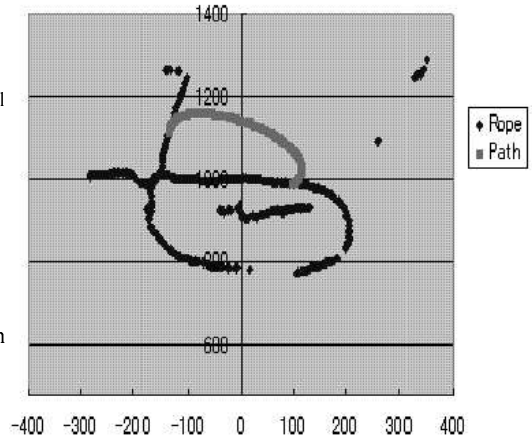

Fig. 9: Calculated path of the Cross move in Fig. 10

target knot state. We can also see that manipulation was conducted successfully following the given path.

However, in some cases, there were failures in the manipulation although the calculated path seemed to be sufficient in executing the desired Cross move. The failure occurred when the hand rotated to change the rope direction. When manipulating rope, rotating the other way around is undesirable because it could change the outcome of the rope state. The robot hand needs to rotate continuously. But there is a limit to how much the robot hand can be twisted because the hand may collide with the arm that it is connected to. Therefore, the hand is rotated within the limit. However, when it becomes impossible to rotate any further, the hand switches to rotate the other way around to reach to the same direction. Thus, when there arises a situation that the robot hand has reached its rotation limit but still has to rotate, the robot comes to a halt.

To solve this hand-arm collision problem, there are serveral possible consideration to make. One is to improve the robot's solution to inverse kinematics. We could optimize the solution selection so that the solution is chosen in which collision is least probable to occur. Another is to put down the rope and regrasp it when collision may occur.

6 Conclusion

We proposed a manipulation method for a knot tying system to carry out knot tying plans generated on the basis of the KPO paradigm. We concentrated on manipulating the Cross move because it was the most simple and fundamental movement primitive.

First we presented a method to automatically determine a sequence of robot commands from task plans. We made extensive use of geometric data that can be obtained from observation in the planning phase in order to determine the details of manipulation movements. We obtained a movement path using Bezier curves, of which parameters are determined by three specific points (grasp, cross, and destination points) that were acquired from observation.

Then we conducted experiments to evaluate our method, taking simple knot tying as an example. The experimental results indicated that the movement paths generated are on the whole appropriate for making a transition to the next knot state in a knot tying task. Thus we conclude that our method effectively determines manipulation procedures.
Carrying out the procedures, however, there still re-

mains hand-arm collision problems. These we will tackle in our future works. Implementation of other movement primitives will also be considered in future works. We believe they can be implemented in the same manner as the Cross move.

References

[1] M. Inaba and H. Inoue. Hand eye coordination in rope handling. *J. of Robotics Society Japan*, Vol. 3, No. 6, pp. 32 – 41, 1985.

[2] J. E. Hopcroft, J. K. Kearne, and D. B. Krafft. A case study of flexible object manipulation. *Int. J. of Robotics Research*, Vol. 10, No. 1, pp. 41 – 50, 1991.

[3] T. Matsuno, T. Fukuda, and F. Arai. Flexible rope manipulation by dual manipulator system using vision sensor. *IEEE Int. Conf. on Advanced Intelligent Mechatronics*, pp. 677 – 682, 2001.

[4] T. Wada, B. J. McCarragher, H. Wakamatsu, and S. Hirai. Modeling of shape bifurcation phenomena in manipulations of deformable string objects. *Int. J. of Advanced Robotics*, Vol. 15, No. 8, pp. 833 – 846, 2001.

[5] T. Wada, S. Hirai, and S. Kawamura. Analysis and planning of indirect simultaneous positioning operation of deformable objects. *Journal of the Robotics Society of Japan*, Vol. 18, No. 5, pp. 675 – 682, 2000.

[6] K. Ikeuchi and T. Suehiro. Toward an assembly plan from observation part i: Task recognition with polyhedral objects. *IEEE Trans. on Robotics and Automation*, Vol. 10, No. 3, Jun. 1994.

[7] J. Takamatsu, H. Tominaga, K. Ogawara, H. Kimura, and K. Ikeuchi. Extracting manipulation skills from observation. *IEEE Int. Conf. on Intelligent Robots and Systems*, Vol. 1, pp. 584 – 589, 2000.

[8] Y. Kuniyoshi, M. Inaba, and H. Inoue. Learning by watching: Extracting reusable task knowledge from visual observation of human performance. *IEEE Trans. on Robotics and Automation*, Vol. 10, No. 6, Dec. 1994.

[9] K. Reidemeister. *KNOT THEORY*. BCS Associates, 1983.

[10] T. Morita, J. Takamatsu, K. Ogawara, H. Kimura, and K. Ikeuchi. Knot planning from observation. *IEEE Int. Conf. on Robotics and Automation*, 2003.