KNOTTING ROPE BY LEARNING FROM VISUAL EXAMPLES

視覚的な例からの学習による紐結び作業の実現

by

Maiko Miyazaki
宮崎　麻衣子

A Senior Thesis
卒業論文

Submitted to
the Department of Information Science
the Faculty of Science, the University of Tokyo
on February 8, 2005
in Partial Fulfillment of the Requirements
for the Degree of Bachelor of Science

Thesis Supervisor: Katsushi Ikeuchi　池内　克史
Professor of Information Science

ABSTRACT

In the field of flexible rope manipulation, planning and manipulation methods have been studied seperately over the past years. Yet, putting together these approaches to implement one whole manipulation system gives rise to a variety of implementational problems.

This work presents a new approach to implement a knot tying system. We chose knot tying as a case study of flexible object manipulation because the knot theory was available and because knot tying operations could be clearly classified into few categories. For the planning phase, we used a method based on the Knot Planning from Observation (KPO) paradigm, which effectively generates knot tying plans based on visual examples demonstrated by human beings. For the manipulation phase, we developed a method to determine robot operations by extracting visual information from the examples used in the planning phase. This yields a simple solution to determining manipulation parameters.

The results of experimental tests are also reported, indicating that our proposed system solves some of the implementational problems in flexible object manipulation. We also show new problems that have come to light through the experiments.

論文要旨

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が浮かび上がってくる。

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。我々が紐結び作業に注目したのは、紐結びに関して結び目理論という数学的体系が存在すること、また、紐結びの動作が明確に分類可能だということに因る。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。視覚情報を用いることで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できるようになった。

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。

Acknowledgements

First, I would like to thank Professor Katsushi Ikeuchi for giving me the oportunity to work on our knot tying system and for being my advisor. I would also like to thank Dr. Jun Takamatsu for giving me support in my work and for proofreading this thesis. I appreciate Dr. Koichi Ogawara's help in using the robot. I am grateful to all other members of the laboratory for their encouragement. Last of all, I would like to show my thanks to my family and friends who have supported me in making it through.

Contents

List of Figures

Chapter 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 is an important activity the robot must learn to do and has been studied for over a decade.

Till the 1990s much of the work in robotics was dedicated to study on rigid object manipulation. On the other hand, deformable objects were left alone because the precise form and position of deformable objects are difficult to attain; they may change greatly by just a simple operation, and the changes are difficult to predict. However, deformable object manipulation has become inevitable in accomplishing complex tasks, because there are many situations in real life where we have to handle deformable objects. Everyday tasks such as connecting cables between electronic equipment and tying shoelaces all involve handling of such objects.

Roughly two types of methods were introduced for deformable object manipulation. One is a model-based method, in which the objective is to predict deformation of the object. The most common model-based method is the Finite Element Method (FEM), which models the deformation of objects caused by move of the object, external force or contact with other objects. FEM is widely used for it is applicable to a wide variety of objects. For example, Wada modeled rope [11] and cloth [10] based on this method. Because FEM requires much computation, others modelled objects in a different manner using heuristic knowledge of objects. For manipulating linear deformable objects such as rope there are methods using differential geometry [9], and for cloth methods using static 2D models [6] and 3D models [12].

Another method for deformable object manipulation is to use a vision-based method, which checks object state from visual feedback. Inaba et al. built a system which

checks and corrects manipulation operations []. Hopcroft et al. introduced a vision-based high-level language for describing knot tying and implemented a manipulation system using it [1]. Matsuno et al. used dual manipulation systems together with a vision system [4].

By using the above methods, manipulation of deformable objects became possible to a larger extent, but each robot operation to be performed was manually programmed by using robot commands. This was inefficient because the programmes 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 next challenge: how to automatically determine what operations should be done on the object to accomplish a task, in other words task planning.

One solution to this problem is the Learning from Observation (LFO) paradigm. The approach of this paradigm is to use observed data in acquiring task plans. First, the robot observes a human perform the task, and collects data from the observation. The observed data is then parsed into task primitives. Task primitives are fundamental units of movements required to accomplish the tasks. By parsing the observed data into these primitives, a sequence of task primitives is obtained. This sequence is the plan of the task. Task planning by LFO decreases programming time and eliminates the need for expertise in robot programming to teach robots new tasks. This is the reason LPO has been applied to various manipulation systems that handle rigid objects [2, 8, 3].

In this thesis we prospose a deformable object manipulation system that implement together planning and manipulation steps for tying knots in rope on the basis of the LFO paradigm. We chose the knot tying task for two reasons. First, because rope (and similar objects, e.g. wires and cords) is one 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 [7], which clearly classifies knot tying operations into few categories [5]. Planning is conducted using elements of knot tying plans (knot states, task primitives, task primitive identification method) which are already designed by our laboratory [5]. Manipulation is achieved by extending the vision-based method so that we could reuse the observed data in the planning phase to simply determine parameters necessary for translating the plans into executable robot operations.

The outline of this thesis is as follows: In chapter 2 we give a broad view of our knot tying system. Next in chapter 3 and 4 we describe the planning and manipulation methods in detail. Then in chapter 5 we show the results of experiments knot tying

tasks and discuss the evaluations of the system. Finally in chapter 6 we offer some conclusions.

Chapter 2

Outline of the Knot Tying System

The goal of our system is to tie knots in ropes that are placed on a flat surface such as a table. We decided to lay the rope on a surface to ease image recoginition and manipulation. The system works in two phases: the planning and the manipulation phase. Figure 2.1 illustrates the system architecture.

The planning phase is built on the basis of the KPO paradigm [5]. This paradigm is an application of the LFO paradigm to knot tying based on the knot theory. It provides us with methodology of describing plans and generating plans described in such a way.

In the planning phase, knot tying plans are generated from data acquired from examples shown by humans. The plans are generated in the following steps:

1. Observation
   Obtain sequential 2D images of a knot-tying performance

2. K-data Conversion
   From each 2D image, obtain a geometric representation of knot state (K-data)

3. P-data Conversion
   From each K-data, convert it to a topological representation of knot state (P-data)

4. Plan Generation
   From the sequence of P-data, identify movement primitives between each state

In the observation step, sequential snapshots of knots are taken as a human performs a knot tying task. The snapshots are in 3D, and are captured by using stereo vision are projected onto a 2D plane to make 2D images. These images are converted

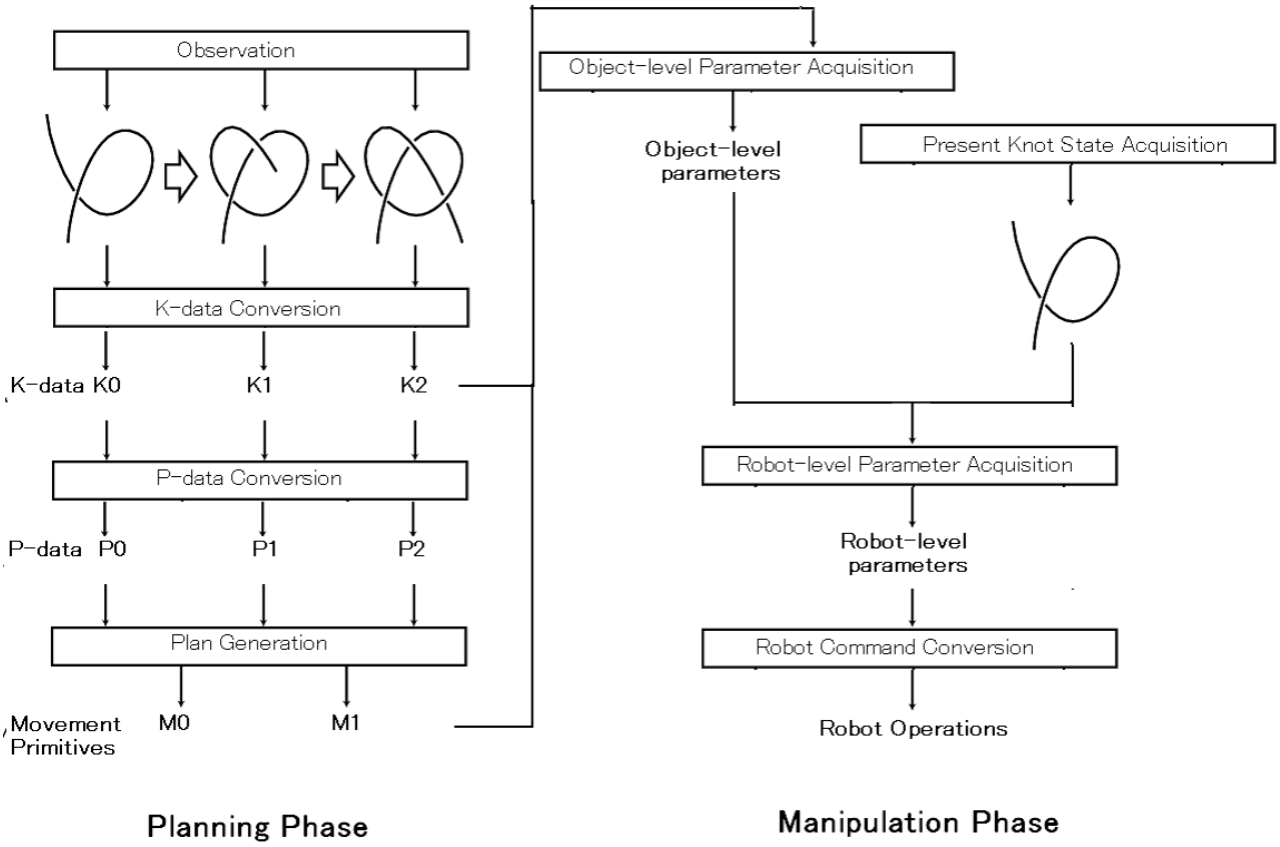


Figure 2.1: Architecture of knot tying system

to K-data, and then to P-data in the following steps. The K-data is a geometric representation of a knot, and the P-data a topological one, which we will show later. Last of all, in the plan generation step, we compare each pair of successive P-data and identify the movement primitives that make transition occur between them. As a result, we obtain a knot tying plan, which is a sequence of 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 operation 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 K-data

2. Present Knot State Acquisition
   Obtain the present state of the rope which the robot will manipulate

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 operations using robot-level parameters

In the first step, we make use of observed data in the planning phase to obtain object-level parameters. These parameters give reference to knot positions and directions without using robot coordinates. In the second step we use the vision system to capture an image of the rope the robot will work on and acquire its K-data and P-data as we do in the planning phase. Then in the third, the object-level parameters are mapped onto the present knot to acquire robot-level parameters. These are parameters that are given in robot coordinates. Finally, in the robot command conversion step, the robot operations are carried out to tie a knot in the rope.

In the next chapter we will cover the KPO paradigm which was previously introduced by Morita [5]. Then in the following chapter we will present a detailed description of our manipulation methods for executing plans.

Chapter 3

Knot Planning from Observation Paradigm

As we have seen in the previous chapter, the KPO paradigm is an application of the LFO paradigm to knot tying based on the knot theory. In a KPO system, knot tying plans can be generated from observed data of a knot tying performance.

The paradigm determines three important elements required for the planning process, which are:

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

These will be explained in the following sections.

3.1 Representation of Knot States

First we will show how knot states are represented.

In the knot theory, a knot is defined as a simple closed curve without width in a 3D space. In this paradigm, a knot is redefined as a simple open curve with two open ends.

To obtain a knot state 3D knots need to be projected onto 2D planes. For a 2D projection to be converted back to a 3D knot, it is required that the projections have the following properties:

- All intersections on a 2D knot projection are point-intersections, i.e. the crossing of part of the rope over/under another only occurs at a cross point on the rope, not over a continuous section of the rope

- The rope always crosses itself at an intersection
- There are no triple intersections in the rope; there are only simple intersections of one part of the rope over another
- The open ends on the rope do not become cross points of an intersection

Knot states are given a topological representation in the KPO paradigm. Although geometric representations give us an accurate picture of a knot, it is necessary to have an abstract representation that does not depend on parametric information. The reason can be shown through a simple example. The two knots illustrated in Figure 3.1 are essentially the same in terms how crossing occurs in the rope. That is, the knots are different in shape, but the same in the way they 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.

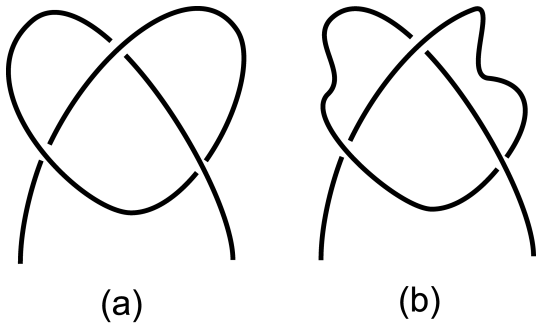


Figure 3.1: Equivalent knots

To solve this problem, we use P-data, a topological representation of knot states. P-data is generated from 2D knot projection in the following process:

1. Choose an open end as the starting point. We call this end the start terminal and the other end the end terminal.
2. From the start terminal, trace the rope to the end terminal. Give an ID number to each intersection you meet on the way. The first intersection is numbered 1. (Each intersection will be numbered twice.)
3. Again, trace the rope from the start terminal to the end terminal. At each intersection, determine the intersection's vertical position and sign.

4. For each intersection, calculate its cross attribute.

The vertical position is whether the intersection is an upper intersection (where the rope crosses over itself) or a lower one (where the rope crosses under itself). The sign of an intersection is determined by the sign of the following formula.

$$(\overrightarrow{1}_{upper} \times \overrightarrow{1}_{lower}) \cdot \overrightarrow{e_z}$$

where $\overrightarrow{1}_{upper}$ is the direction vector of the rope of the upper intersection at that intersection, and $\overrightarrow{1}_{lower}$ is that of the lower intersection, and $\overrightarrow{e_z}$ is a unit vector parallel to the normal line of the projection plane. From the vertical position and sign, the cross attribute of an intersection is determined in the following way: 1:upper/−, 2:lower/−, 3:upper/+, 4:lower/+.

An example of a 2D knot projection and its corresponding P-data is given in Figure 3.2 and 3.3. The normal line of the projection plane is upward from this paper. The i-th column holds the data for the i-th intersection on the projected knot. The first row shows the ID number of the intersection, in other words, i. The second row represents the other ID number assigned to the i-th intersection, and the third row the cross attribute of the intersection.

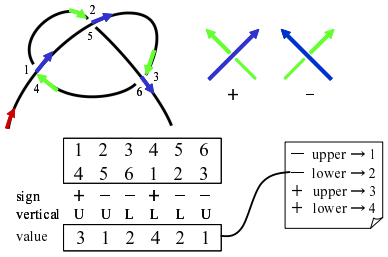


Figure 3.2: 2D knot projection

1	2	3	4	5	6
4	5	6	1	2	3
3	1	2	4	2	1

Figure 3.3: P-data of Projection

3.2 Definition of Movement Primitives

Next we will give definitions of movement primitives, which are basic movements to make a transition from a knot state to another.

3.2.1 The Four Movement Primitives

In determining movement primitives, we must first define properties movement primitives should fulfill. To do this, Reidemeister first defined equivalent knots as two knots that can be deformed to the other without cutting it. Note that stretching and

shortening of the knot are allowed. Then he proved that an equivalent knot can be obtained by either stretching, shortening the knot or finitely repeating three types of moves on the knot. Note that Reidemeister's proof was based on the definition that a knot is a simple closed curve. Whereas stretching and shortening do not change the knot state, the three moves, referred to as Reidemeister moves, result in a transition of knot state.

Due to this proof, we define movement primitives as moves that change the knot state. Movement primitives fulfill the following properties:

- Only one segment of the rope is moved in each primitive
- The transition by a movement primitive is direct; they move from one knot state to another without moving to an intermediate knot state in between

Reidemeister Moves

Naturally, from the above definition Reidemeister moves are movement primitives of knot tying tasks. Below, we explain each type of move.

Reidemeister move I adds or removes an intersection to the knot by creating or destroying a simple loop. Reidemeister move II adds or removes two intersections by crossing a segment of rope over another segment. Reidemeister move III moves a segment of rope across an intersection. The number of intersections does not change in this move. Figure 3.4 illustrates the three moves.

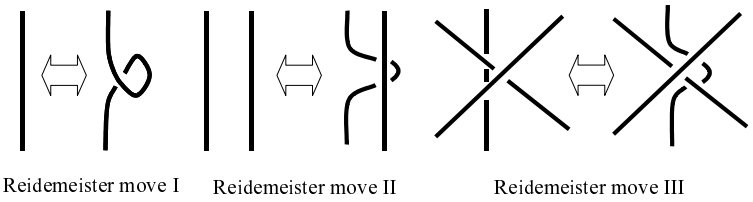


Figure 3.4: The three Reidemeister moves

Cross Move

Due to the fact that the three Reidemeister moves were defined as a knot being a simple closed curve, we add to the primitives a Cross move for dealing with simple open curves. The Cross move is adds or removes an intersection on a rope by crossing an open end over/under a segment of rope. The illustration of a Cross move is given in Figure 3.5.

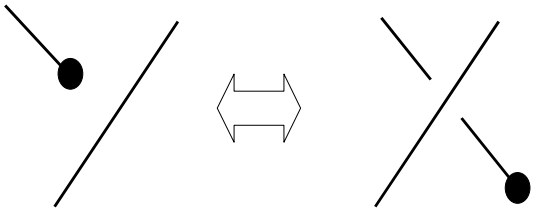


Figure 3.5: The Cross move

3.2.2 Sufficiency of the Primitives

Transition of knot states can be classified into two categories:

1. A crossing of an open end over/under a segment of rope
2. A crossing of a segment of rope other segments

The first category of transition can be realized by the Cross move. The second category can be achieved by repetition of Reidemeister moves. This can be easily shown by extending the Reidemeister's proof on knot-equivalence to simple open curves.

It can actually shown that the four primitives defined in this paper are sufficient in tying typical knots such as overhand knot, bowline knot, harness hitch, bow tie, single loop bow, two-half knot, and taut-line hitch.

3.3 Movement Primitive Identification

Lastly, we show how to identify the movement primitives that caused the transition from a P-data to the next, in other words, how to generate a knot tying plan.

3.3.1 Preliminaries

First, we define some functions used in this step.

Definition 1: Given a P-data P, *n(P)* returns the number of columns in P, which is twice the number of intersections.

Definition 2: Given a P-data P, $\sigma(i \mid P)$ returns the other ID number assigned to the P-th intersection. When it is clear what P is, we simply write $\sigma(i)$. Note that if $\sigma(i \mid P)=j$, $\sigma(j \mid P)=i$.

Definition 3: Given a P-data P, *attr(i | P)* returns the cross attribute of the i-th intersection. When it is clear what P is, we simply write *attr(i)*.

For example, by applying these function to the P-data in Figure 3.3 we obtain *n(P)*=6, σ*(2 | P)*=5, *attr(3 | P)*=2.

Now we go on to specifying the primitives. In the following explanation, P_t denotes the P-data obtained from a knot projection at time t. Primitives are specified on the following assumptions:

- Intersections of a knot increases by a transition, i.e. $n(P_{t-1}) \leq n(P_t)$
- The start point of the knot before and after the transition stays the same

The basic idea of specifying the primitives is by comparing the *t-1*-th P-data P_{t-1} and t-th P-data P_t. We prepare a function for each primitive type. Assuming that the transition to P_t is achieved by movement primitive *mp*, the function for *mp* transforms P_t to generate the P-data before the transition. If $mp(P_t) = P_{t-1}$, we can say the transition which occurred between P_{t-1} and P_t was *mp*. The transformation functions are described below.

3.3.2 Reidermeister Move I

The transition function for Reidemeister move I, $R_I(P_t|i)$, is based on two changes caused by this move. Assuming that Reidemeister move I is applied to the i-th segment($1 \leq i \leq n(P_{t-1}) + 1$) of P_{t-1} and results in P_t, the changes are:

- $n(P_t)$=$n(P_{t-1})$+2
- $\sigma(i|P_t)$=*i+1*

The first change means that one intersection is added to the knot by Reidemeister move I. The second one shows that, the new intersection will be given two continuous numbers. Figure 3.6 gives an example of this change of P-data.

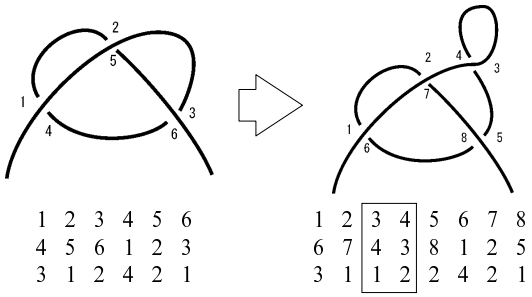


Figure 3.6: Transition caused by Reidemeister move I

From these changes, $R_I(P_t|i)$ can be defined as a function that transforms P_t by removing the i-th and $(i+1)$-th columns, and subtracting 2 from the ID numbersw of $(i+2)$-th to $n(P_t)$-th intersections.

3.3.3 Reidermeister Move II

The transition function for Reidemeister move II, $R_{II}(P_t|i)$ is defined similarly. Assume that Reidemeister move II is applied to the i-th and j-th segments($1 \leq i < j \leq n(P_{t-1}) + 1$) of P_{t-1} and results in P_t, then the changes caused by the move are:

- $n(P_t) = n(P_{t-1})+4$
- $(\sigma(i|P_t)=j+2 \quad \cap \quad \sigma(i+1 \mid P_t)=j+3) \quad \cup$
  $(\sigma(i|P_t)=j+3 \quad \cap \quad \sigma(i+1 \mid P_t)=j+2)$
- $|attr(i—P_t)\text{-}attr(i+1—P_t)|=2$

The first change means that two intersections are added, the second that the added intersections are adjacent to one another, and the third that the vertical positions (upper/lower) of the new intersections should be the same although the signs should be different. An example is shown in Figure 3.7.

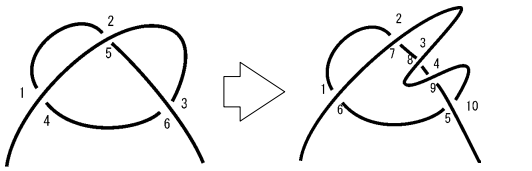


Figure 3.7: Transition caused by Reidemeister move II

From these changes, $R_{II}(P_t|i)$ is defined as a function that removes from P_t the i-th, $(i+1)$-th, $(j+2)$-th, and $(j+3)$-th columns, and subtracts 2 from the ID numbers of $(i+2)$-th to $(j+1)$-th intersections and 4 from those of the $(j+4)$-th to P_t-th intersections.

3.3.4 Reidermeister Move III

The transition function for Reidemeister move III, $R_{III}(P|i)$ is defined as follows. Assume that Reidemeister move III is applied to the i-th, j-th and k-th segments($2 \leq$

$i, j, k \leq n(P_{t-1}) - 1$) of P_{t-1} and results in P_t. The i-th segment is surrounded by two upper intersections, the j-th by one upper and one lower intersection, and k-th by two lower intersections. The three segments can be in any order, and compose three intersections. Each intersectios has two numbers of the set { i-1, i, j-1, j, k-1, k }. The assignment of the numbers has 8 variations, which are (See Figure 3.8):

- Type A. $\sigma(i-1) = j-1, \quad \sigma(i) = k-1, \quad \sigma(j) = k$
- Type B. $\sigma(i-1) = j-1, \quad \sigma(i) = k, \quad \sigma(j) = k-1$
- Type C. $\sigma(i-1) = j, \quad \sigma(i) = k-1, \quad \sigma(j-1) = k$
- Type D. $\sigma(i-1) = j, \quad \sigma(i) = k, \quad \sigma(j-1) = k-1$
- Type E. $\sigma(i-1) = k-1, \quad \sigma(i) = j-1, \quad \sigma(j) = k$
- Type F. $\sigma(i-1) = k, \quad \sigma(i) = j-1, \quad \sigma(j) = k-1$
- Type G. $\sigma(i-1) = k-1, \quad \sigma(i) = j, \quad \sigma(j-1) = k$
- Type H. $\sigma(i-1) = k, \quad \sigma(i) = j, \quad \sigma(j-1) = k-1$

Considering geometric restrictions, the change caused by Reidemeister move III is one of the following:

- Type A $\leftrightarrow$ Type H
- Type B $\leftrightarrow$ Type G
- Type C $\leftrightarrow$ Type F
- Type D $\leftrightarrow$ Type E

Note that unlike the other moves, the number of intersections do not change, i.e. $n(P_t) = n(P_{t-1})$. An example is shown in Figure 3.9.

Based on this change, $R_{III}(P|i)$ is defined as a function that removes from P the (i-1)-th, i-th, (j-1)-th, j-th, (k-1)-th, k-th intersections. Unlike the other moves, comparison is made between $R_{III}(P_t|i)$ and $R_{III}(P_{t-1}|i)$.

3.3.5 Cross Move

The transition functions for Cross move, $C_s(P_t|i)$ and $C_s(P_t|i)$ are defined as follows. The former is a function for Cross move applied to the start terminal, the latter

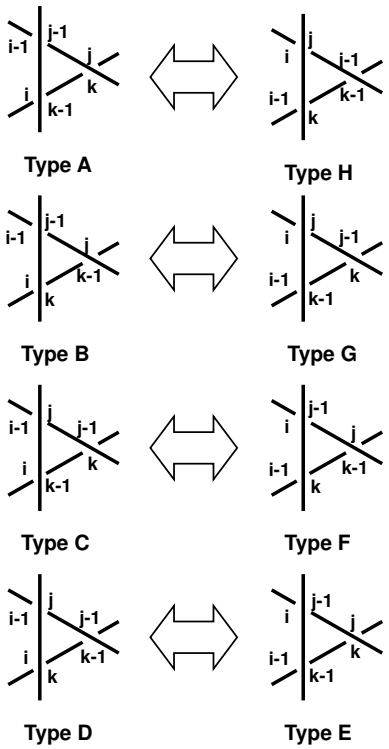


Figure 3.8: Types of Reidemeister move III

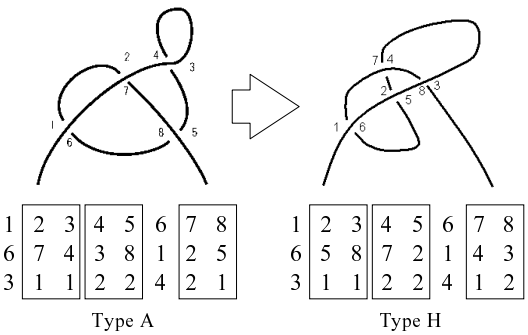


Figure 3.9: Transition caused by Reidemeister move III

the end terminal. Assume that the terminal is crossed over the j-th segment of P_{t-1} and results in P_t.

If the start terminal is crossed over, the changes caused by the move are:

- $n(P_t) = n(P_{t-1})+2$
- $i + 1 = \sigma(1 \mid P_t)$

The first change means that one intersection is added, and the second that the anew intersection is i+1. An example is shown in Figure 3.10.

From these changes, $C_s(P_t|i)$ is defined as a function that removes the 1st and (i+1)-th column and subtracts 1 from the ID numbers of the i-th to $n(P_t)$ intersections.

If the end terminal is crossed over, the changes caused are:

- $n(P_t) = n(P_{t-1})+2$
- $i = \sigma(n(P_t) \mid P_t)$

The first change means that one intersection is added, and the second that the anew intersection is i. An example is shown in Figure 3.11.

From these changes, $C_e(P_t|i)$ is defined as a function that removes the last and i-th column and subtracts 1 from the ID numbers of the (i+1)-th to ($n(P_t)$-1)-th intersections.

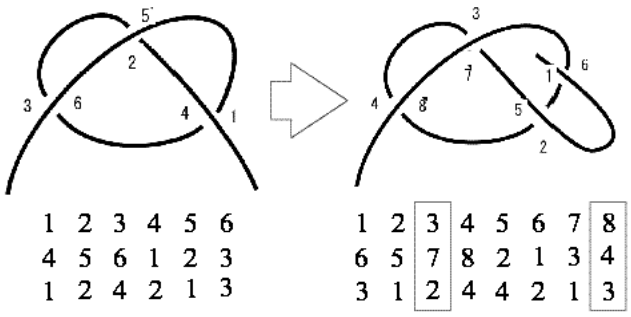


Figure 3.10: Transition caused by Cross move applied to start terminal

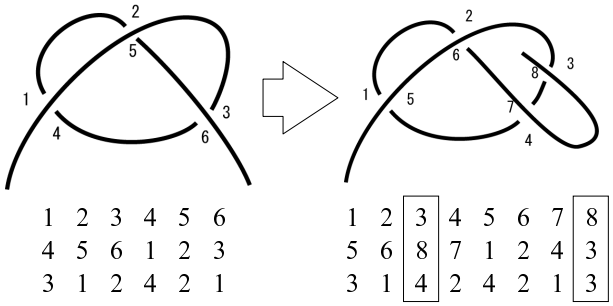


Figure 3.11: Transition caused by Cross move applied to end terminal

Chapter 4

Manipulation Methods

Our main focus was to devise a method to execute knot tying plans generated in the planning phase. Before explaining each step in detail, we discuss our basic strategies.

Firstly, we chose only to manipulate the Cross move, and following this rule we allowed in the planning phase only knot tying performances that could be parsed into a sequence of Cross moves. The main reason for this is to simplify manipulation. Our choice of the Cross move is due to 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. Overhand knots, bow ties, bowline knots and figure eight knots are examples of knots actually tied by a sequence of Cross move operations. To achieve a Cross move we always moved the terminal segment and crossed it over the other segment, regardless of requirement to cross the rope under itself. We took such a stance because it would simplify manipulation to a one hand movement, which is much easier compared to use of both hands which would arise the problem of collision avoidance; another topic to be handled later on.

Secondly, we decided to use geometric information of knots in order to determinine robot operations. Given a plan, in this case a sequence of Cross moves, we gain two pieces of information:

- which terminal to move
- which segment to cross

These are insufficient in terms of executing them. That is, they only determine abstract operations but do not give precise information needed for robot operations such as position and direction. So we decided to obtain geometric data of knots, and therefore introduced a geometric knot representation called K-data.

Thirdly, we defined two levels of parameters for expressing robot operation information: object-level parameters and robot-level parameters. Object-level parameters give reference to knot positions and directions according to the ropes's geometric structure. We defined such a parameter because parameters depending on metric properties of a knot are apt to change and cannot be determined at programming time and dependence on metric properties would force a fixtures on knots, which would be too restrictive. On the other hand, robot-level parameters refer to knot positions and directions using robot coordinates, in other words metric properties of a knot. This is because for the actual robot operations, metric information is essential.

Fourthly, and most importantly, we decided to retrieve geometric data from the knot images obtained in the planning phase. As we have mentioned in our second strategy, we made use of geometric information for determining robot operations; the problem was from where to gain such data. To this problem Morita et al devised a method to reconstruct a knot from P-data and obtain necessary information from it[]. While this gives a more flexible choice of the geometric representation to be made, it arises a new problem of choosing the most suitable representation for knot tying. Therefore, we obtained knots' geometric data from the images acquired in the planning phase, regarding them as examples demonstrating the most suitable shapes the knot should form at each step.

In this chapter we will first introduce a geometric representation for knots and robot operations. Then we will explain each manipulation step in detail. The first three steps are dedicated to acquiring robot operation paramaters, and in the last step a sequence of robot operations will be executed.

4.1 Preliminaries

4.1.1 Geometric Representation of Knot States

Here we give the definition of K-data and also show how K-data corresponds with P-data.

K-data

As in the KPO paradigm, we define a knot as a simple open curve with two open ends in a 3D space and from it we may obtain a 2D projection.

K-data is a geometric representation of knot states, which is made from 2D knot projections. 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 cross points of intersections and the two open ends of the rope. The points are numbered, starting from one end of the rope (the start terminal), then tracing along the rope meeting each cross point, and finally ending at the other end (the end terminal). Note that cross points 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) or a lower cross point (a point where the rope crosses under itself) or an open end.

The segments are sections of a rope starting and ending at either a cross point or an open end. The vector of segments are ordered similarly to the points; first with the segment starting from the start terminal, followed by the segment adjacent to it, then the next segment adjacent to the previous one, till it reached the segment ending at the end terminal. Each segment has as its property the starting and ending points and also a chain of directions. This chain describes the shape of the rope segment. We discretize the segment of rope as a chain of points for visual feedback purposes. For each point we obtain its direction vector, and hold this information as a direction chain.

Figure 4.1 is an example of a knot with its points and segments numbered as in the K-data.

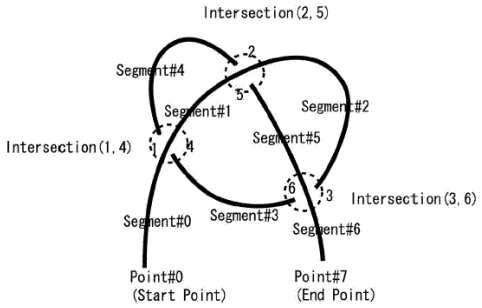


Figure 4.1: Numbering of segments and points in a knot

K-data and P-data

While K-data represents geometric information, P-data represents topologic. However, conversion from K-data to P-data is simply achieved by applying the P-data generation algorithm introduced in 3.1. By excluding the two end points in the K-data,

intersections of P-data are obtained. The two corresponding intersections and cross attributes are acquired easily in a similar way.

4.1.2 Robot Operations

In this section we present the robot operation functions we provided for knot tying. The function arguments are given in robot coordinates. We assume that rope will be placed on a 2D plane such as a table, and that the Z-axis is in the direction upward from the plane. Operations are defined as follows:

- graspRope(x, y, z, θ)
  Grasps the rope at coordinate (x, y, z). The robot hand will be rotated so that it will be parallel to the direction of the rope θ.

- liftRope(z)
  Move the rope by z in the direction of the z axis

- moveRope(x, y, z, θ)
  Move the rope from the present position to point (x, y, z). The rope should be in the direction of θ at that point.

- 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 Cross move acquired from the plan and the K-data of the two knots between which the Cross move occurred. In both K-data we find the two segments (the terminal segment and the segment which it crosses) invloved in the Cross move. By comparing those segments in the pre-Cross-move K-data and pro-Cross-move K-data we exract obhect-level parameters. The Cross move parameters are defined as follows:

- *terminal*
  Which terminal to move (start terminal / end terminal)

- *segment*
  Segment number to cross

- *position*
  Position in the segment to cross ($0 < position < 1$, in which 0 is the starting point and 1 the ending point of the segment)

- *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
  (Let the total rope length be 1, then $0 < length < 1$)

An example of parameters for the Cross move is shown in Figure 4.2 will be *terminal*=end terminal, *segment*=1, *position*=0.25, *direction*=under, *argument*=90, *length*=0.1.

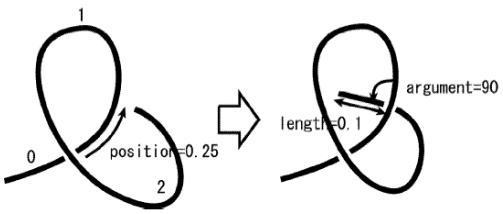


Figure 4.2: Example of object-level parameters acquired from K-data

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 a 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 a path which the rope should move along. The acquired object-level parameters are mapped onto the

present knot K-data to obtain robot-level parameters to determine the path. More specifically, the parameters are the positions and directions of the grasp, cross and destination points. Note that we need to specify only one set of these three points per movement primitive.

The position and direction of the grasp point are acquired as below.

1. Map the grasp point onto present knot K-data
   Acquire the *grasppoint*-th point (counting from the terminal) on the *terminal* segment, where

   $$grasppoint = min\{(length \times (total\ rope\ length)), (length\ of\ terminal\ segment)/2, segmentoffs$$

   Segmentoffset is the minimum distance the point should be from the terminal, if possible. We set this value to 50. The *grasppoint* is chosen from the threee candidates so that it would not be too near or too far from the end of the rope in order to make a firm grip on the rope.

2. Determine the grasp point on robot level
   Find in the 3D knot image the point that corresponds to the grasp point in K-data, and acquire its 3D robot coordinate

3. Determine the grasp direction from the K-data
   On the direction chain of the *terminal* segment, obtain the direction at the grasp point

The position and direction of the cross point are acquired as follows:

1. Map the cross point onto present knot K-data
   Acquire the (*position* $\times$ (*segment*-th segment length)) point on the *segment*-th segment

2. Determine the cross point on robot level
   Find in the 3D knot image the point that corresponds to the cross point in K-data, and acquire its 3D robot coordinate

3. Determine the rope direction at the cross point from K-data
   On the direction chain of the *segment*-th segment, obtain the direction at the cross point. Calculate the cross rope direction by adding to the direction *argument*.

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. Therefore the direction of the destination point is the same as that of the cross point. The position is determined in the following way

1. Determine the distance of the destination point from the cross point by calculating

$$distance \quad = \quad min\{(length - grasppoint), distanceoffset\}$$

2. Determine 3D robot coordinates by calculating

$$\begin{aligned} destination\_x &= cross\_x + distance \cdot \cos(destination\_dir) \\ destination\_y &= cross\_y + distance \cdot \sin(destination\_dir) \\ destination\_z &= cross\_z \end{aligned}$$

   where *destination_x* is the 3D robot X-coordinate and *destination_dir* the direction of the destination point and likewise.

Distanceoffset is the minimum distance the rope should be moved past the cross point. It is given in the number of segment points. We set this value to 30.

Determining the path from the grasp point to the cross point is also required in rope manipulation because the results may change greatly depending on the path taken. Figure 4.3 is an example of two paths both with the same grasp and cross points, but different paths.

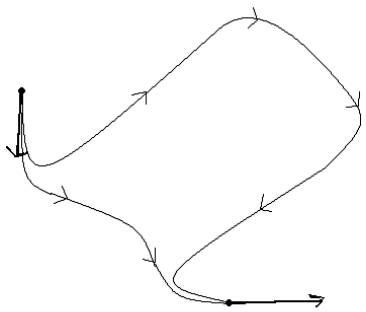


Figure 4.3: Different paths for movement between to points

We considered that the movement path would be a simple curve with the grasp and cross points as the two ends. For obtaining this path, we used cubic Bezier curve.

Let the two dimensional coordinate (x and y coordinates) of the grasp point be G and its rope direction be ang_G, the two dimensional coordinate of the cross point be C and its rope direction be ang_C, and the distance between the grasp and cross point be *dis*. Then the four control points $P_i (0 \leq i \leq 3)$ are determined as follows:

$$\begin{aligned} P_0 &= G \\ P_1 &= G + 0.5 \cdot dis \cdot (\cos(ang_G), \quad \sin(ang_G)) \\ P_2 &= C - 0.5 \cdot dis \cdot (\cos(ang_C), \quad \sin(ang_C)) \\ P_3 &= C \end{aligned}$$

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 second control point is given as half the distance between the grasp and cross points. We determined this distance regarding it reasonable enough for the rope not to take a too long 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 of rope direction of that point.

A point $P(t)$ on the Bezier curve (t is a parameter between 0 to 1) can be calculated by the formula below:

$$\begin{aligned} P(t) \quad &= \quad P_0 B_0^3(t) + P_1 B_1^3(t) + P_2 B_2^3(t) + P_3 B_3^3(t) \\ where \quad & B_0^3(t) = (1-t)^3, \quad B_1^3(t) = 3t(1-t)^2, \\ & B_2^3(t) = 3t^2(1-t), \quad B_3^3(t) = t^3 \end{aligned}$$

From the Bezier curve, we choose two middle points (this time we chose t=0.3 and t=0.7) that the rope should pass through when moving from the starting to the cross point. Besides these middle points, divided the Bezier curve into many small sections, and at the point where a new section begins, we checked if the direction of the rope changes from the previous section, and if it did, we made the rope also pass through those points. The direction of the rope at those points is calculated from the gradient of the tangent line.

4.5 Robot Command Conversion

Last of all, a Cross move is translated into robot level operations and executed in the execution step.

Using the given robot operations, the Cross move will be executed in the following sequence:

1. graspRope(grasppoint_x, grasppoint_y, grasppoint_z, grasppoint_θ)
2. liftRope(lift_z)
3. For all middlepoints the rope must pass through, moveRope(middlepoint1_x, middlepoint1_y, middlepoint1_z, middlepoint1_θ)
4. moveRope(crosspoint_x, crosspoint_y, crosspoint_z, crosspoint_θ)
5. moveRope(destinationpoint_x, destinationpoint_y, destinationpoint_z, destinationpoint_θ)
6. releaseRope()

The parameters grasppoint_x, grasppoint_y, grasppoint_z, grasppoint_θ stand for the x, y, z coordinates and rope direction of the grasp point. The coordinates and rope direction for the cross point, destination point and the middle points on the path are expressed similarly. The parameter lift_z denotes how much to lift the rope. We set this value to 10cm.

Chapter 5

Experimental Results

We will show experiments conducted in this research. First we will introduce the platform on which we built our system. Second, we will describe how a knot image is acquired. Last we will give some results of knot tying tasks.

5.1 Platform

We built our system on a robot illustrated in Figure 5.1. The robot is designed to imitate the upper part of the human body, especially the eyes, arms, and hands. We chose a robot that had functions similar to human beings because for one thing it was one our interests to make the robot accomplish intelligent tasks in a human-like manner, and for another we considered such robots to have the most general-purpose design to accomplish a wide variety of tasks.

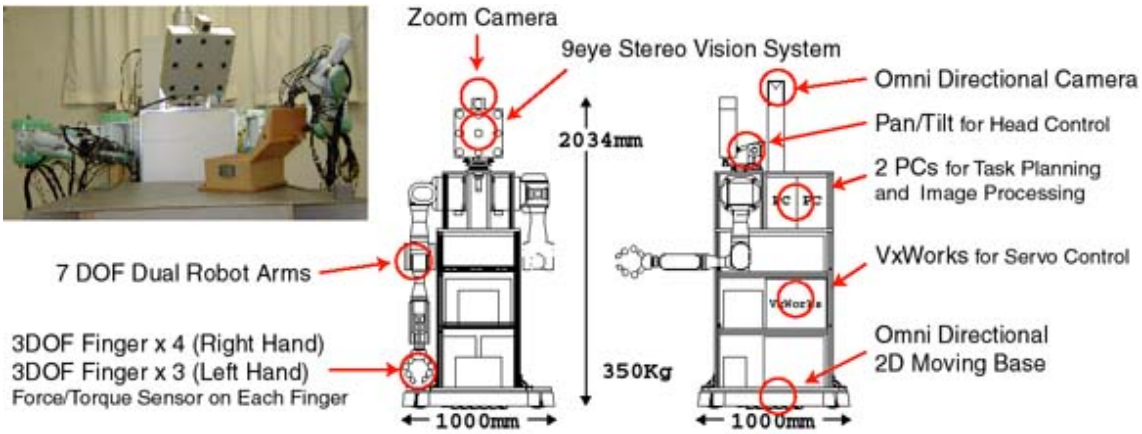


Figure 5.1: Platform of knot tying system

We will take a look at the three main parts of the robot.

5.1.1 Vision System

The vision system consists of a 9-eye stereo vision system for 3D observation. These are set onto the head of the robot. The head can be panned and tilted so that the robot may adjust its viewing direction.

The 9-eye stereo vision system is made by Komatsu and uses the multi-baseline method to recognize its surrounding environment in realtime. The system has the following features:

- Robust stereo matching
  By stereo matching, the system calculates 8 stereo pairs(pairs of the central camera and one of the 8 surrounding cameras). For each pixel, the most reliable pair is chosen as the distance data.

- Realtime processing using hardware board
  A 280 $\times$ 200 distance image is generated in realtime by doing stereo calculation on the hardware(15fps to 30fps).

- Easy camera setting
  The camera can easily be set up according to the users' demands. Here we set the focus distance longer and adjusted the 8 surrounding cameras to face inwards in order to fastly generate high resolution distance images.

5.1.2 Hardware System

The robot has a head with the vision system mentioned above, and 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 dimesional space.

The robot hands imitate those of a human. Both hands have a thumb, and fingers that face the thumb: four for the right hand and three for the left. The fingers are fewer than the human hand, considering the size and shape of the hand. The fingers have 3 joints, thus 3 DOFs. Sensors are attached to the hand: contact sensors on the palm and each finger cushion, force and torque sensors on all fingertips. For the movement of joints we use the finger joint actuator made by Yaskawa Electric Corporation, and for the fingertip sensors the BL NANO Sensor by BL AUTOTEC.

The robot body is built on wheels which can move in omnidirection. This enables the robot to dynamically change its viewing direction or working space, which results in an expanded view and operation scope.

5.1.3 Software system

Robot software management is done by the Common Object Request Broker Architecture (CORBA). CORBA is an open distributed object computing infrastructure. Merging softwares developed on different machines on a network is made easy by using CORBA as a bus. Adding external equipment, for example data gloves, also becomes simple. We use the TAO ORB implementation of CORBA.

5.2 Image Processing

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

We fixed the position of the robot body, the pan and tilt of the head, and the position od the table on which the rope was placed to fix the vision system's photographing region.

5.2.1 2D Knot Projection Acquisition

First, 3D images are captured through a vision system, and RGB and disparity data is 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 camera coordinates to that in robot coordinates. As we always place the knot on a 2D plane, a 2D knot projection is simply the same as 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 substraction
   Substract background data from the image to extract only the knot

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

3. Point Extraction

   Find cross points and open ends in the knot by counting the neighboring points for each knot point. Add them to the K-data point vector.

4. Segment Generation

   For all the ponits found in the previous step, find any segment that connects it to another point. If there is any, add them to the K-data segment vector. Note that the segments will be counted twice.

5. Redundant Segment Elimination

   Remove from the segment vector segments that are too short, regarding them as recoginition error of the vision system.

6. Double Count Elimination

   Remove one of the double-counted segments to leave only segment data for each segment.

7. Point Merging

   Merge points on the knot that are too close to one another, regarding them as recognization error of the vision system. Make changes in the segment vector accordingly.

8. Reordering

   Reorder the segments and points so that they are in an order that can be traced from one end of the knot to the other.

9. Vertical Position Extraction

   By using disparity information obtained from the vision system, determine vertical positions of the intersections at each point.

5.3 Results of Tying a Simple Knot

We conducted an experiment to tie a simple knot in a rope. A simple knot is illustrated in step4 of Figure 5.2. It is the most basic type of knot and can be made by executing only Cross moves.

The knot images of captured from the human performance are shown in Figure 5.2 (a). From these images, K-data is acquired. Figure 5.2 (b) illustrates how the knot states are recognized after K-data conversion. The states we named state1, state2, state3, and state4. The sequence of movement primitives and their object-level parameters are given in Figure 5.2 (c).

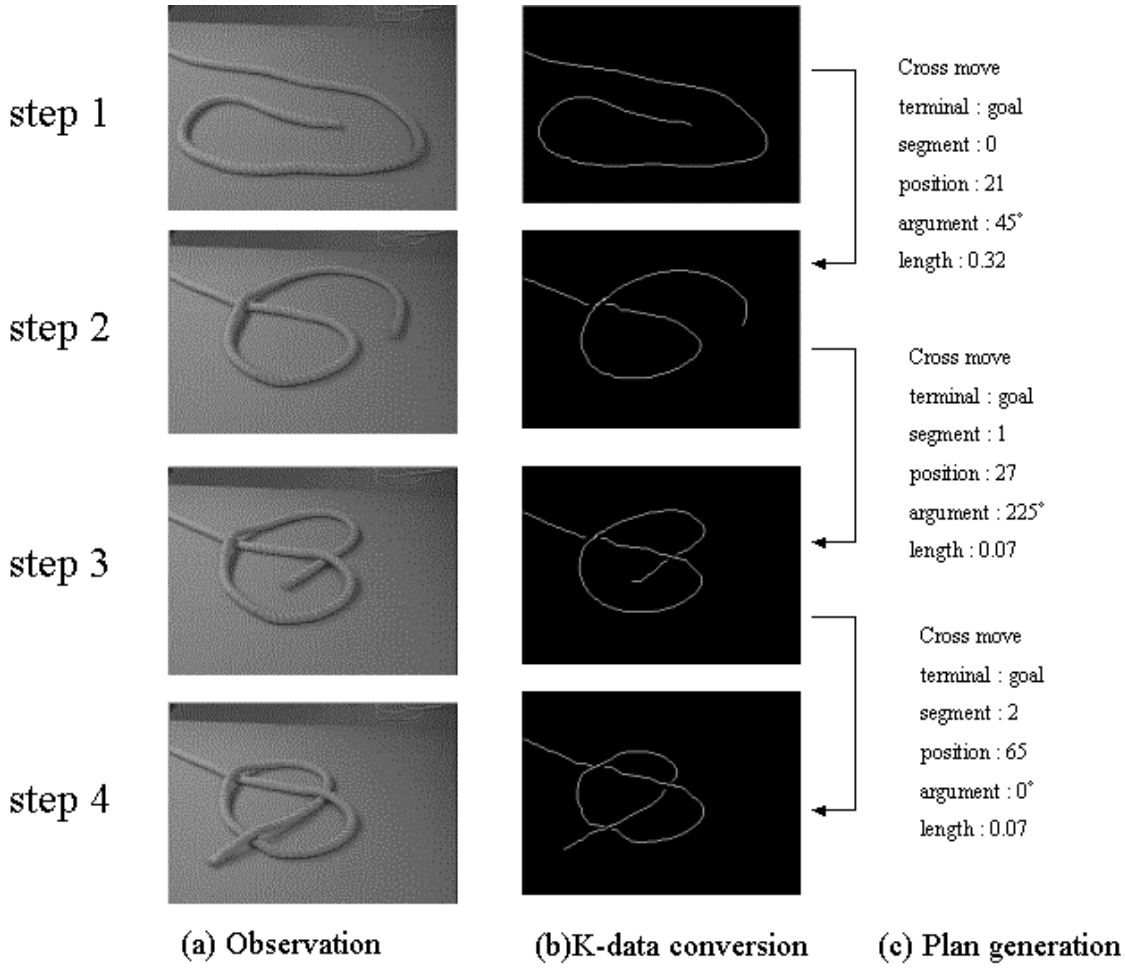


Figure 5.2: Knotting steps of a simple knot

We manipulated the Cross move from state1 to state2 and state2 to state3 seperately as the first step towards tying a simple knot. Figure 5.3 and 5.4 each show the results of manipulation together with an illustration of the calculated movement path. The coordinates in the figure of the path are X- and Y-robot coordinates. Looking at the movement paths generated for each Cross move, it appears that they were appropriate for producing the target knot state. We can also see that manipulation was conducted following the given path.

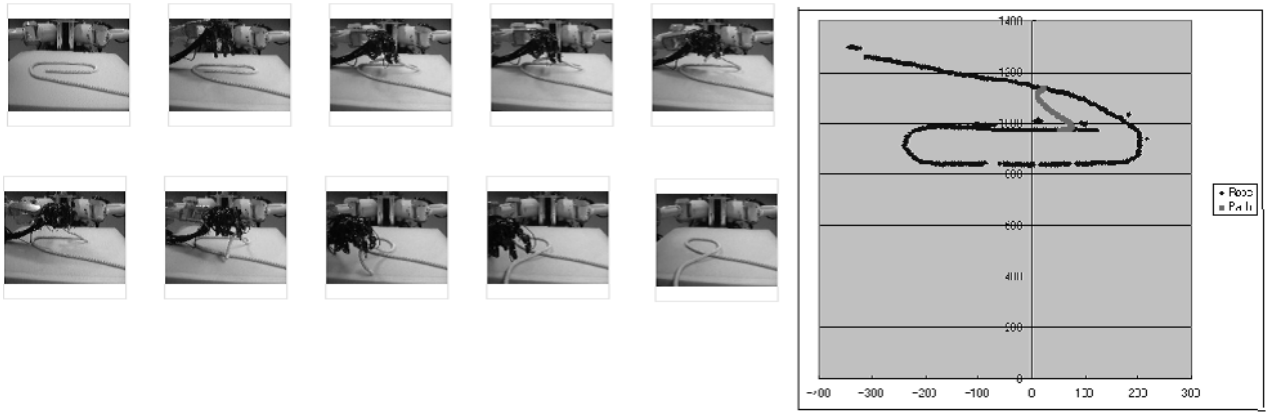


Figure 5.3: Manipulation of Cross move from state1 to state2

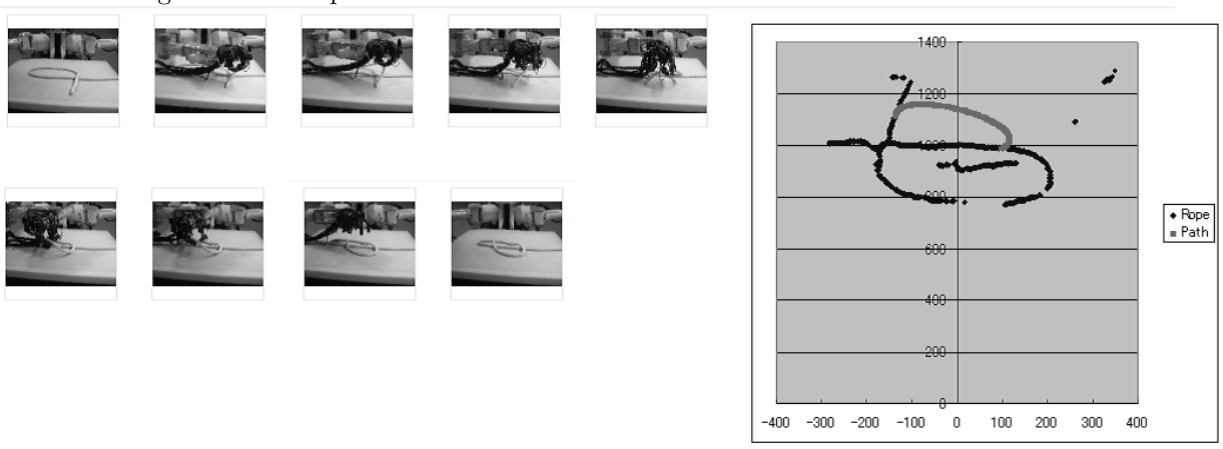
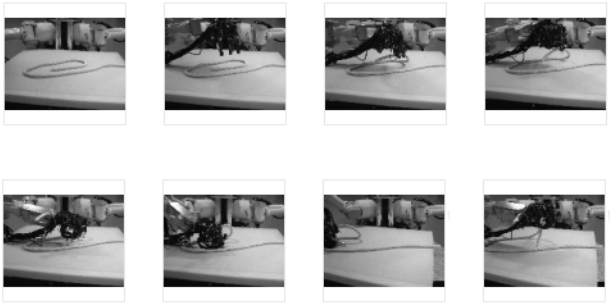


Figure 5.4: Manipulation of Cross move from state2 to state3

5.4 Discussion

While manipulation was successfully conducted as in the above examples, there were some cases in which manipulation failed. An example of such cases is given in Figure 5.5. This is a manipulation of a Cross moves from state1 to state2, It is given a rope state different from that of to map the object-level parameters onto.



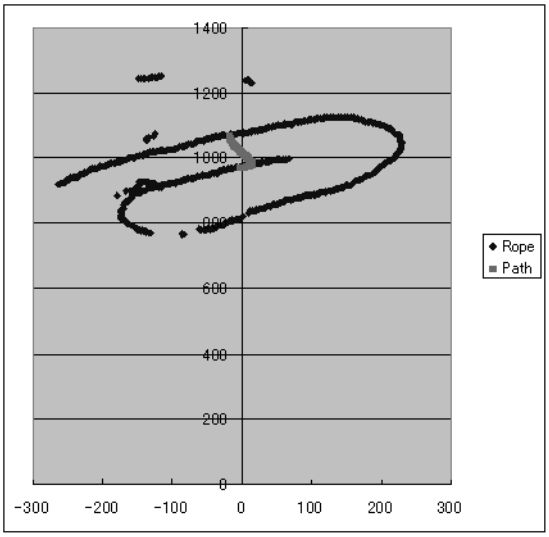


Figure 5.5: Example of manipulation failure

Looking at the paths, they seem to be as sufficient as those in the successful examples for the rope to move on. However, the actual result of knot state after the manipulation is quite different from what is expected. This we consider was due to the bodily constraints of the robot. The problem seemed to be caused especially by the rotation of the hand. In the successful examples, the right hand moved within its limits in which collision with its own arm would not occur. In contrast, the robot hand had to rotate in the direction opposite to which it was rotating in so that it could avoid hand-arm collision. This change in rotatory direction is thought to have resulted in an unexpected knot state after the manipulation.

The problem of the robot's movement limitations became even more obvious when we executed the Cross moves sequentially to perform a complete knot tying task. The robot hand and arm had to move and rotate in a wider range, increasing possibility of collisions.

To solve this problem of hand-arm collision, the following considerations would be

possible.

First, and most important, we could improve the robot's solution to inverse kinematics. Presently, the robot pose is determined by solving inverse kinematics. However, the multiple solutions may exist for a pose, and the choice of which solution to use is done in the manner of redundant manipulation. Therefore, we could optimize the solution selection so that the solution is chosen which hand-arm collision is least probable to occur.

Second, we could optimize the work space for knot tying. Though the current work space is adequate for pick and place or insertion movements, where moves are mostly linear, it is seems too small for rotation movements. Improvement can be made by reconsidering the position, in regard to the robot arm and hand, of the 2D plane on which the rope is placed, and by permitting robot body movement during the manipulation.

Third, we could design a method to make the robot regrasp the rope when collision may occur. When there is danger of a hand-arm collision, the robot may put down the rope and grasp it again in a pose which will avoid collision. This will arise a need to devise a collision recognition module which models the hand and arm shape and calculates their position relative to each other. Additional rope recognition will also need to be conducted each time the rope is released and picked up again.

Fourth, we could implement move of rope using the robot fingers. The movement of rope was realized by the arm move, and the hand was used only to hold the rope. Thus we may introduce a robot operation in which the rope pose can be change by twinsting the rope using robot fingers. Since the movement will be only within the hand of the robot, hand-arm collision will be avoided.

Chapter 6

Conclusion and Future Works

6.1 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 robot operations 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 from grasping, crossing and destination points, which were acquired from observation, a movement path using Bezier curves.

Next we implemented the observation and manipulation modules. In the observation module we successfully acquired task plans and parameters for manipulation using several image processing techniques. We achieved visual feedback by connecting the observation and manipulation modules.

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 may conclude that our method effectively determines manipulation procedures on a thoeretical level, where the robot's physical limitations can be disregarded.

However, our findings also imply that the robot's physical limitations are unignorable, in fact noticeable problems in manipulating deformable objects, where the course taken by the robot hand grasping the object brings about great change in the resulting object state. Thus we can also conclude that solving the robot's physical limitation problems is essential to deformable object manipulation.

6.2 Future Works

In the next stage of the research, we plan to work on the following tasks:

- Hand-arm collision avoidance
  We will devise a new method to overcome the physical limitations that we come face to face in knotting rope. We will work on handling hand-arm collision on the basis of considerations we made in the discussion section.

- Dual hand manipulation
  When we take into account vertical positions (whether the rope is crossing over itself or under) of the rope at an intersection, we need to make use both hands: with one hand lift the rope, and with the other pass the rope under it. In implementing dual hand manipulation, we will also have to tackle the problem of hand-hand collision.

- Implementation of the other movement primitives
  In this research we limited manipulation to the Cross move. To allow a wider variety in moves, implementation of other primitives is necessary. Implementation is thought to be possible by taking an approach to similar to that of the Cross move.

- Knotting in the air
  Our knot was placed on a 2D plane for ease of recognition. In the future, we hope to let the robot hold up the knot and manipulate it in the air. To do this expansion of knot recognition schemes would be necessary. For example, consideration of how to determine the projection plane, how to maintain the knot state while manipulation is conducted.

- Improvement of the vision system
  We plan to make an improvement in the vision system. The precision of the current system is inadequate in that it is error-prone especially in determining the knot coordinates and the vertical position of the rope at an intersection. We are considering of replacing the current system with a range sensor which has higher accuracy.

References

[1] J. E. Hopcroft, J. K. Kearne, and D. B. Krafft. A case study of flexible object manipulation. *Int. J. of Robotics Research*, 10(1):41 – 50, 1991.

[2] K. Ikeuchi and T. Suehiro. Toward an assembly plan from observation part i: Task recognition with polyhedral objects. *IEEE Trans. on Robotics and Automation*, 10(3), Jun. 1994.

[3] 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*, 10(6), Dec. 1994.

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

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

[6] D. M. Pollet P. M. Taylor and M. T. Grieber. Analysis and design of pinching grippers for the secure handling of fabric panels. *Proc. EURISCON '94*, (4):1847 – 1856, 1994.

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

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

[9] T. Wada, S. Hirai, and S. Kawamura. Modeling of plain knitted fabrics for their deformation control. *Proc. IEEE Int. Conf. Robotics and Automation*, (3):1960 – 1965, 1991.

[10] 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*, 18(5):675 – 682, 2000.

[11] 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*, 15(8):833 – 846, 2001.

[12] J. Weil. The synthesis of cloth objects. *SIGGRAPH '86*, 20(4):49 – 54, 1986.