



tecnun Universidad de Navarra

MASTER THESIS

INDUSTRIAL ENGINEERING

ROBOTIC STRING-ENVELOPE OPENING

Irene Goizueta Zubimendi
San Sebastián, April 2018

Contents

ABSTRACT	ix
1 INTRODUCTION	1
1.1 Motivation	1
1.2 Project Goal	2
2 ROS AND MOVEIT!	7
2.1 Introduction to ROS	7
2.2 MoveIt!	9
2.3 How to control to UR10, Robotiq 3 finger gripper and Robotiq FT300 force/torque sensor through ROS	10
3 MOTION PLANNING	11
3.1 Involute of a circle	11
3.2 Archimedean spiral	12
3.3 Comparison between involute of a circle and Archimedean spiral	13
4 THE ALGORITHM	17
4.1 Initial configuration	17
4.2 Algorithm	18
4.2.1 Flowchart	18
4.2.2 Pseudocode	20
4.2.3 Examples	23

5	EXPERIMENTAL RESULTS	27
5.1	Experiment 1: Why use torque values instead of force	27
5.1.1	Comparison between force values and torque values	28
5.1.2	Threshold value for torque	30
5.2	Experiment 2: Minimum radius of the pivots	33
5.2.1	Radius = 4 mm	33
5.2.2	Radius = 3 mm	34
5.2.3	Radius = 2 mm	34
5.3	Experiment 3: Minimum distance between pivots	35
5.3.1	Distance = 4.25 cm	35
5.3.2	Distance = 3.5 cm	36
5.3.3	Distance = 3 cm	37
5.4	Experiment 4: Comparison of robot fail ratio with human fail ratio	37
6	CONCLUSION AND FUTURE LINES	41
6.1	Conclusion	41
6.2	Future Lines	41
7	BUDGET	43

List of Figures

1.1	Universal Robots UR10 Robot Arm.	2
1.2	Robotiq Force Torque Sensor FT 300.	3
1.3	Robotiq 3-Finger Adaptative Robot Gripper.	3
1.4	String-Envelope.	4
1.5	Assembly of the elements to compose the initial configuration of the robot.	4
2.1	ROS Summary.	8
2.2	Robots using MoveIt!	9
3.1	Involute of a circle.	12
3.2	Archimedean spiral.	13
3.3	Comparison between involute of a circle and Archimedean spiral.	14
4.1	Initial configuration of the robot and all the elments.	17
4.2	Flowchart of the algorithm to open a string-envelope.	19
4.3	Example of how the algorithm figures out the first turn.	24
4.4	Example of the case where the string is tied about the same pivot after one turn.	24
4.5	Example of the cases where the string is tied about the other pivot after one turn.	25
4.6	Example of the end of the movement.	25
5.1	Tension and No Tension cases.	28
5.2	CW case on the left and CCW case on the right for $r = 4$ mm.	33

5.3	CW case on the left and CCW case on the right for $r = 3$ mm.	34
5.4	CW case on the left and CCW case on the right case for $r = 2$ mm.	34
5.5	First experiment with a distance of 4.25 cm between pivots.	35
5.6	Experiment with a distance of 3.5 cm between pivots.	36
5.7	Experiment with a distance of 3 cm between pivots.	37
5.8	Experiment performed by 5 people with closed eyes trying to open the envelope.	38
6.1	Repository "envelope" on GitHub.	42

List of Tables

3.1	Comparison between involute of a circle and Archimedean spiral.	14
5.1	Results of force and torque readings for the case of no tension and the case of tension of the string.	29
5.2	Maximum and minimum values of force and torque.	29
5.3	Standard deviation and mean values for force and torque readings in tension and no tension case.	30
5.4	Ranges of force and torque variations in tension and no tension cases. . .	30
5.5	Results for 10 experiments of standard deviation and mean values for torque in tension and no tension case.	31
5.6	Range of variation of torques in 10 experiments.	32
5.7	Difference between means in tension and no tension case for 10 experiments.	32
5.8	Results for torque values in CW and CCW case for $r = 4\text{mm}$	33
5.9	Results for torque values in CW and CCW case for $r = 3\text{mm}$	34
5.10	Results for torque values in CW and CCW case for $r = 2\text{mm}$	35
5.11	Results for torque values when the string is tied about P1 and about P2 for a distance of 4.25 cm between pivots.	36
5.12	Results for torque values when the string is tied about P1 and about P2 for a distance of 3.5 cm between pivots.	36
5.13	Results for torque values when the string is tied about P1 and about P2 for a distance of 3 cm between pivots.	37
5.14	Number of success and fails made by the robot in the 30 experiments. . .	38
5.15	Number of success and fails made by the 5 people in the 30 experiments.	39

5.16	Comparison between robot percentage of success and human percentage of success.	39
7.1	Budget for consumables.	43
7.2	Budget for equipment.	43
7.3	Budget for the software.	44
7.4	Budget for the labour.	44
7.5	Summary of the global budget for this project.	44

ABSTRACT

Robotic manipulation of deformable objects is a challenging subject and can often be bothersome. The behavior of this kind of objects is difficult to predict because they are constantly changing their shape when they are exposed to external forces. However, deformable objects such as ropes, wires, strings, cloths, are present in industry and in our daily lifes. For that reason, we decided to focus our research on robotic manipulation of deformable objects. This project has been developed at the Robotics Institute of the Hong Kong University of Science and Technology during a period of six months. The goal of this project is to open an arbitrarily tied string-envelope with a UR10 robot, using a Robotiq FT300 wrist force/torque sensor.

INTRODUCTION

In this chapter we introduce the work that has been done at the Robotics Institute of the Hong Kong University of Science and Technology (HKUST) during a period of six months. We focus on robotic manipulation of deformable objects. We explain here the motivation to do this work and its goal.

1.1 Motivation

Let's start by talking about robotic manipulation. The research about this topic has become increasingly popular over the last 50 years [8], especially for industrial applications that focus on the execution of repetitive tasks, tasks that require big efforts, high precision or that are dangerous. This period has been accompanied by a technological maturation of robots as well, from the simple pick and place and painting and welding robots, to more sophisticated assembly robots for inserting integrated circuit chips onto printed circuit boards, to mobile carts for parts handling and delivery [12]. Several areas of robotic automation have now become "standard" on the factory floor, but until now, most of the research on robotic manipulation assumed the handled objects to be rigid.

However, many important application domains require manipulating deformable objects, especially deformable linear objects (DLOs), such as ropes, cables, wires, strings and sutures. Such objects can be found almost everywhere in the real world of industry and in human daily life and they are far more challenging to handle, as they can exhibit a much greater diversity of behaviors [17]. They add a number of difficulties to the manipulation task by continuously changing shape when exposed to external forces.

There have already been studies about robotic manipulation of deformable linear objects. For example, [17] describes a motion planner for manipulating DLOs and tying knots (self-knots and knots around simple static objects) using two cooperating robot arms. Then, [20] and [21] analyze how to adjust the motion during the manipulation of a DLO in order to eliminate vibration.

For this project, we want to continue in the same line as the works mentioned but focusing on a new problem that has not been studied before.

1.2 Project Goal

To determine the goal of this project we started with some limitations: the material available. The material provided to work with is mentioned below:

- Universal Robots UR10 robot arm.
- Robotiq Force Torque Sensor FT 300.
- Robotiq 3-Finger Adaptive Robot Gripper.

The UR10 robot arm used in this project is shown in Figure 1.1 below:

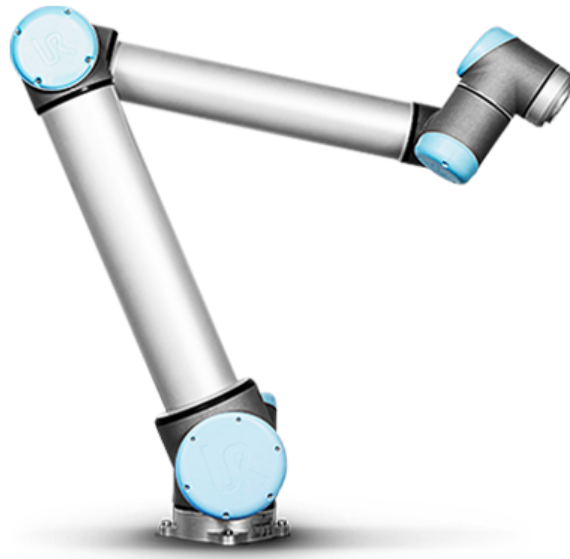


Figure 1.1: Universal Robots UR10 Robot Arm.

This is the larger industrial robot arm of the Universal Robots and it is designed for bigger tasks where precision and reliability are still of paramount importance. The UR10 robot arm allows to automate processes and tasks with payloads that weigh up to 10 kg. It has 6 degree-of-freedom and it can follow position commands like a traditional industrial robot, as well as take force commands to apply a given force or torque in/around a specified axis. In addition to standard programming, this robot has a freedrive mode for tactile programming [16].

Attached to the wrist of the UR10 robot arm, we find the Robotiq Force Torque Sensor FT 300, shown in Figure 1.2:

The FT sensor is meant to have an end-of-arm tool mounted on it, so that it can sense force and torque applied on the tool. The sensor is compatible with various tools and provides feedback that can be used for: hand guiding a robot, force control processes, assembly tasks, product testing, etc [15].



Figure 1.2: Robotiq Force Torque Sensor FT 300.

The tool mounted on the FT 300 sensor is the Robotiq 3-Finger Adaptive Robot Gripper, shown in Figure 1.3:



Figure 1.3: Robotiq 3-Finger Adaptive Robot Gripper.

The Robotiq 3-Finger Adaptive Robot Gripper is a robotic peripheral that is designed for industrial applications. Its design makes it a unique robotic end-of-arm tool to pick, place and handle a large range and volume of parts of varying sizes and shapes. It has three articulated fingers, that each have three joints. The gripper can engage up to ten points of contact with an object (three on each of the phalanges plus the palm). The fingers are under-actuated, meaning they have fewer motors than the total number of joints. This configuration allows the fingers to automatically adapt to the shape of the object they grip and it also simplifies the control of the gripper [14].

With these components, we have to determine the goal of the project being conscious of the limitations.

In HKUST, string-envelopes are very commonly used to send the correspondence because they can be re-used several times. We thought that a good challenge could be to try opening a tied envelope using the robot arm. For this, we have to figure out how to untie the string that has previously been tied in an arbitrarily manner. The envelope used for this project is shown in Figure 1.4:



Figure 1.4: String-Envelope.

The initial configuration for all the components is shown in Figure 1.5:

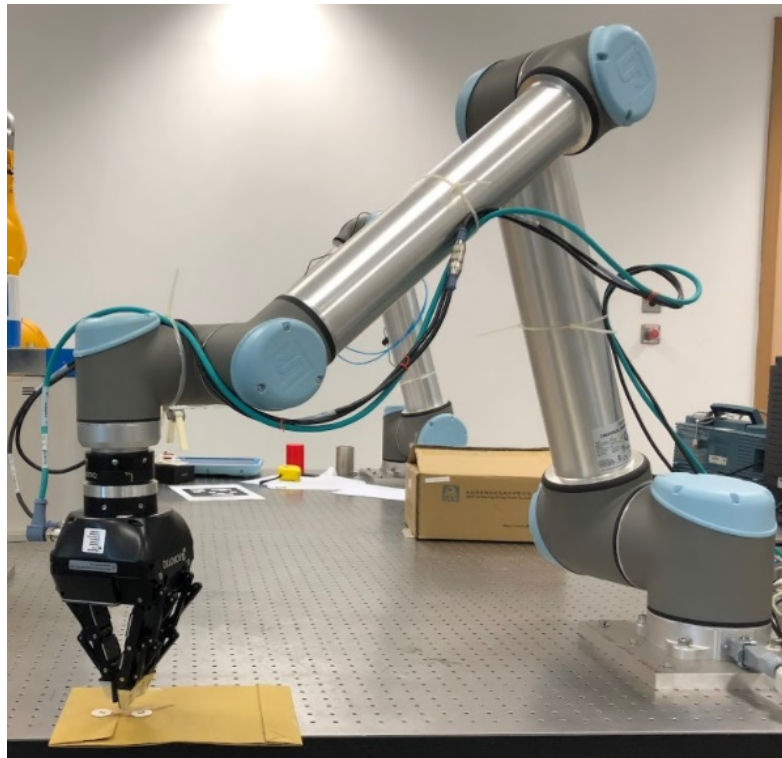


Figure 1.5: Assembly of the elements to compose the initial configuration of the robot.

To accomplish the main goal, it is important to first achieve the following sub-goals:

- Learn about ROS.
- Control UR10, Robotiq 3 finger gripper and get data from Robotiq FT300 through ROS.
- Plan the motion for the gripper to untie the string.

- Establish the algorithm to generalize the problem so the robot can open the envelope being tied in an arbitrarily manner.

In this document, we explain how these sub-goals were accomplished in order to achieve the main goal.

Chapter 2 talks about ROS (Robot Operating System), system used to control the robot. It also introduces MoveIt!, which provides the core functionality for manipulation in ROS.

Chapter 3 explains how to solve the problem for untying the string from the pivots, which path the end-effector must follow.

Chapter 4 shows and explains the algorithm we came up with to solve the general problem of untying arbitrarily tied string-envelopes.

Chapter 5 shows some experiments that have been done and their results.

Chapter 6 concludes this work and talks about future lines to continue developing this project.

We will start now to talk about ROS.

ROS AND MOVEIT!

In this chapter, we make a brief introduction to ROS [13], system we used to program the robot arm, the force/torque sensor and the gripper. We also talk about MoveIt!, which provides the functionality for manipulation in ROS.

2.1 Introduction to ROS

The rapid progress on the development of robotic systems has caused that robots still present some significant challenges for software developers. ROS, Robot Operating System, is a platform that is intended to ease some of these difficulties. The official description of ROS is:

ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers. [13]

Creating truly robust, general-purpose robot software is hard. From the robot's perspective, problems that seem trivial to humans often vary wildly between instances of tasks and environments. Dealing with these variations is so hard that no single individual, laboratory, or institution can hope to do it on their own [3].

The Robot Operating System (ROS) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms [6].

The goal of ROS is not to be a framework with the most features. Instead, the primary goal of ROS is to support code reuse in robotics research and development. ROS is a distributed framework of processes (aka Nodes) that enables executables to be individually designed and loosely coupled at runtime. These processes can be grouped into Packages and Stacks, which can be easily shared and distributed. ROS

also supports a federated system of code Repositories that enable collaboration to be distributed as well. This design, from the filesystem level to the community level, enables independent decisions about development and implementation, but all can be brought together with ROS infrastructure tools [4].

As a result, ROS was built from the ground up to encourage collaborative robotics software development. For example, one laboratory might have experts in mapping indoor environments, and could contribute a world-class system for producing maps. Another group might have experts at using maps to navigate, and yet another group might have discovered a computer vision approach that works well for recognizing small objects in clutter. ROS was designed specifically for groups like these to collaborate and build upon each other's work [3].

We can thus say that the advent of new open-source frameworks like ROS has made robotics more accessible to new users, both in research and consumer applications. ROS has revolutionized the developers community, providing it with a set of tools, infrastructure and best practices to build new applications and robots. A key pillar of the ROS effort is the notion of *not re-inventing the wheel* by providing easy to use libraries for different capabilities like navigation, manipulation, control (and more) [10].

Figure 2.1 below summarizes ROS:



Figure 2.1: ROS Summary.

Plumbing: ROS provides publish-subscribe messaging infrastructure designed to support the quick and easy construction of distributed computing systems.

Tools: ROS provides an extensive set of tools for configuring, starting, introspecting, debugging, visualizing, logging, testing, and stopping distributed computing systems.

Capabilities: ROS provides a broad collection of libraries that implement useful robot functionality, with a focus on mobility, manipulation, and perception.

Ecosystem: ROS is supported and improved by a large community, with a strong focus on integration and documentation. ros.org is a one-stop-shop for finding and learning about the thousands of ROS packages that are available from developers around the world.

ROS has already been implemented in Python, C++, and Lisp, and there are experimental libraries in Java and Lua [4].

ROS currently only runs on Unix-based platforms. Software for ROS is primarily tested on Ubuntu and Mac OS X systems, though the ROS community has been contributing support for Fedora, Gentoo, Arch Linux and other Linux platforms. While a

port to Microsoft Windows for ROS is possible, it has not yet been fully explored [4].

As this project focuses on robotic manipulation, let's now talk about MoveIt! which provides the core functionality for manipulation in ROS.

2.2 MoveIt!

MoveIt! is a software for mobile manipulation, incorporating the latest advances in motion planning, manipulation, 3D perception, kinematics, control and navigation. It provides an easy-to-use platform for developing advanced robotic applications, evaluating new robot designs and building integrated robotics products for industrial, commercial, R&D and other domains [1].

This software builds on multiple pillars:

- A library of capabilities: MoveIt! provides a library of robotic capabilities for manipulation, motion planning, control and mobile manipulation.
- A strong community: A strong community of users and developers that help in maintaining and extending MoveIt! to new applications.
- Tools: A set of tools that allow new users to integrate MoveIt! with their robots and advanced users to deploy new applications.

MoveIt! is the most widely used open-source software for manipulation and has been used on over 65 different robots [10].

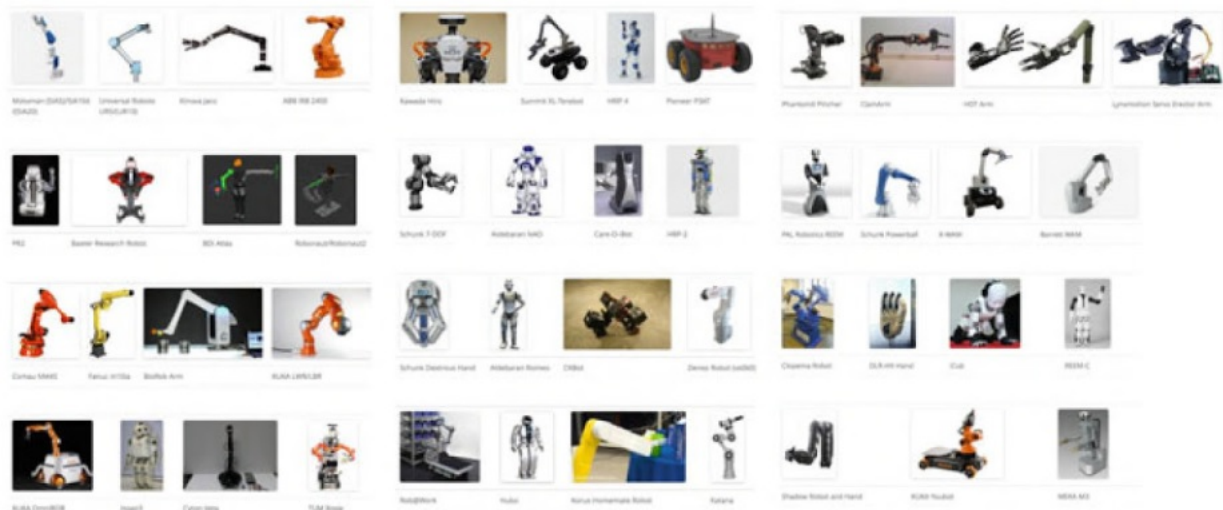


Figure 2.2: Robots using MoveIt!

Figure 2.2 shows a list of robots that MoveIt! has been used with. The robots range from industrial robots from all the leading vendors to research robots from all over the world. The robots include single arm, dual-armed robots, mobile manipulation systems,

and humanoide robots. Among these robots, we find the UR10 robot arm which we will be working with. MoveIt! has been used in applications ranging from unstructured autonomous pick and place (with industrial robots like the UR5), mobile manipulation (with the PR2 and other robots), process tasks like painting and welding, with the (simulated) Robonaut robot for target applications in the space station. MoveIt! has been used by teams in the DARPA Robotics Challenge, the ROS-Industrial Consortium, the Amazon Picking Challenge, the NASA sample retrieval challenge [10].

2.3 How to control to UR10, Robotiq 3 finger gripper and Robotiq FT300 force/torque sensor through ROS

In this section, we explain how to control UR10, Robotiq 3 finger gripper and FT300 force/torque sensor through ROS.

To control UR10 through ROS, we must install the Universal Robots package [5]. Through this package we can establish communication with UR10 controllers. Then, for the communication with Robotiq FT300 force/torque sensor and the 3 finger gripper, we must install Robotiq package [2]. Once we have installed those packages, we can run the executables to establish the connection with the three elements.

To make the connection with UR10 robot, the files shown below must be launched in three different terminals:

```
roslaunch      ur_modern_driver      ur10_bringup.launch      limited:=true
robot_ip:=192.168.1.102 [reverse_port:=REVERSE_PORT]
roslaunch      ur10_moveit_config      ur10_moveit_planning_execution.launch
limited:=true
roslaunch ur10_moveit_config moveit_rviz.launch config:=true
```

Now we can create an executable to move the robot arm through ROS.

The communication with the FT300 is made by typing in two different terminals:

```
sudo chmod 777 /dev/ttyUSB1
roslaunch robotiq_force_torque_sensor rq_sensor
```

And the connection with the gripper is made by typing in other two different terminals:

```
roslaunch robotiq_s_model_control SModelTcpNode.py 192.168.1.11
roslaunch robotiq_s_model_control SModelSimpleController.py
```

Now, we know how to control the three elements through ROS, so let's figure out which path the end-effector must follow to untie the string.

MOTION PLANNING

In this chapter we explain the solution to plan the trajectory of the end effector in order to unwind the string from the pivots.

3.1 Involute of a circle

Let's start talking about the involute of a circle. This one was discovered in 1673 by Christiaan Huygens, a Dutch mathematician and physicist. In his work titled "Horologium oscillatorium sive de motu pendulorum ad horologia aptato demonstrationes geometricae", he focused upon the theories about the motion of pendulum. He was interested in making clocks for ships at sea. Finding a clock which would keep accurate time at sea was a major problem and many years were spent looking for a solution. This issue was of vital importance since if GMT was known from a clock then, since local time could be easily computed from the sun, longitude could be easily computed. He introduced there the concept of involute or evolute of the curve and he used the circle involute in his first pendulum clock in an attempt to force the pendulum to swing in the path of a cycloid [11].

In differential geometry, the involute is defined as a curve that is obtained by attaching an imaginary string and winding or unwinding it tautly on the given curve. The locus of the free end of this attached taut string is known as an involute or an evolute [11].

The involute of a given curve can also be defined as a line segment that is tangent to the curve on one end, while the other end traces out the involute. This line segment is wound or reversely unwound along with the continuous variation in the length of line segment by a certain amount. The curve so traced by free end is called an involute [11].

In the case of this project, we have to unwind the string from circular pivots, the solution is thus the involute of a circle. This is the curve traced by the free end of a thread unwound from a circle in such a way that the thread is always tight and tangential to the circle. More practically, it is the curve traced by a hand unwinding a wire reel held in the other hand.

In this curve, all the normals are tangent to a fixed circle. Conversely, if an involute

of a circle turns around the centre of the generating circle, every tangent to the circle remains always normal to the involute.

Figure 3.1 below shows how the involute of a circle looks like:

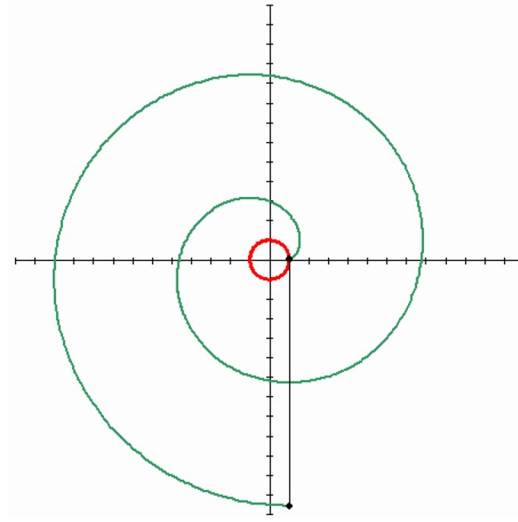


Figure 3.1: Involute of a circle.

Its parametric equations in the Cartesian coordinates are:

$$\begin{aligned}x &= a \cdot (\cos(t) + t \cdot \sin(t)) \\y &= a \cdot (\sin(t) - t \cdot \cos(t))\end{aligned}$$

Where a is the radius of the circle and t is the angle in radians.

We noticed that the involute of a circle has a very similar shape to the Archimedean spiral.

3.2 Archimedean spiral

The Archimedean spiral is the asymptotic curve to the involute of a circle for large values of the angle t . It also is its pedal with respect to the center [19].

The pedal curve results from the orthogonal projection of a fixed point on the tangent lines of a given curve. The pedal of a curve with respect to a point O (or with pole O) is the locus of the feet of the lines passing by O perpendicular to the tangents to the curve [19].

The Archimedean spiral, also known as arithmetic spiral, is a spiral named after the 3rd century BC Greek mathematician Archimedes. It is the trajectory of a point moving uniformly (with a constant speed) on a straight line of a plane, this line turning itself uniformly (rotating with constant angular velocity) around one of its points [18]. It has

the property that any ray from the origin intersects successive turnings of the spiral in points with a constant separation distance, hence the name "arithmetic spiral" [9].

The Archimedean spiral has two arms, one for angle $t > 0^\circ$ and one for $t < 0^\circ$. The two arms are smoothly connected at the origin. Only one arm is shown in Figure 3.2. Taking the mirror image of this arm across the y-axis will yield the other arm.

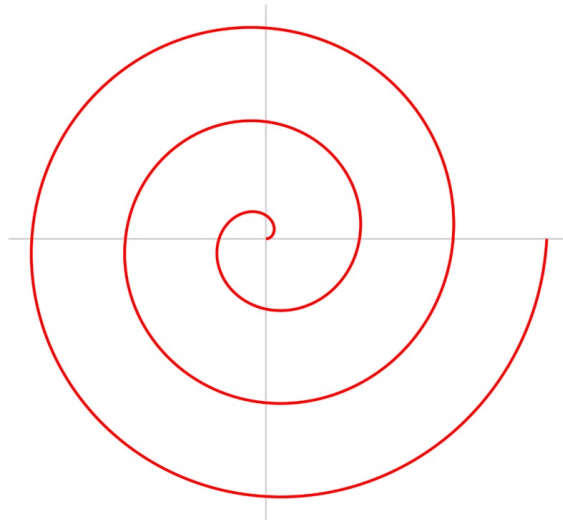


Figure 3.2: Archimedean spiral.

Its parametric equation is the one shown below:

$$\begin{aligned}x &= r \cdot t \cdot \cos(t) \\ y &= r \cdot t \cdot \sin(t)\end{aligned}$$

Where r is the radius of the spiral and t is the angle in radians.

Let's now compare the differences between the involute of a circle and the Archimedean spiral.

3.3 Comparison between involute of a circle and Archimedean spiral

As told before, involute of a circle and Archimedean spiral have very similar shapes. In this section we compare them and talk about the differences. Figure 3.3 below shows the difference in both shapes.

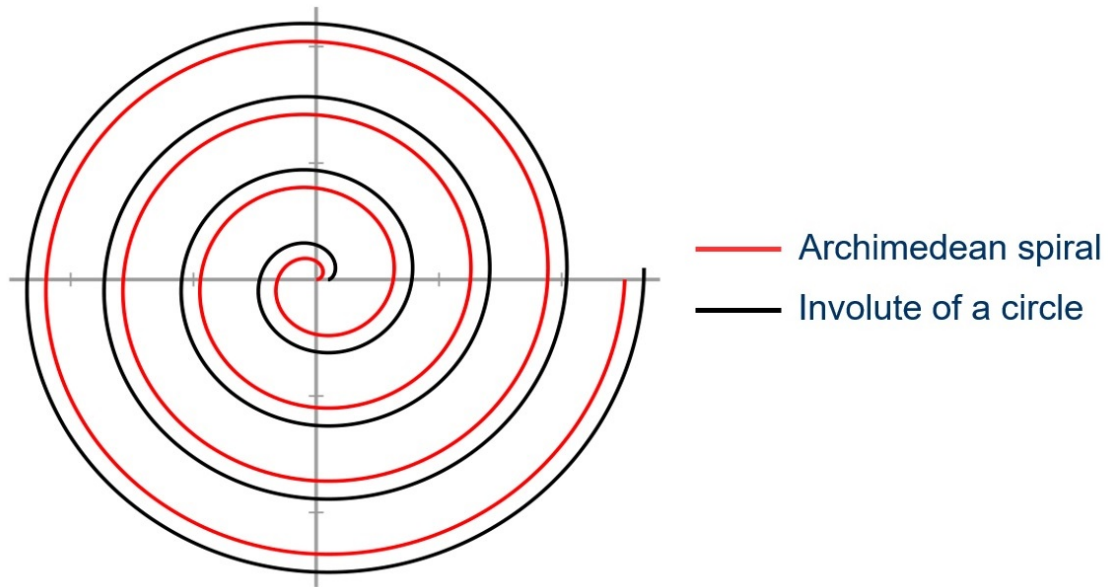


Figure 3.3: Comparison between involute of a circle and Archimedean spiral.

Archimedean spiral is represented in red and involute of a circle in black. Both radius are the same, 1 unit. They seem to have the same progress as the angle increases but let's see the real differences in Table 3.1 below.

	Archimedean spiral	Involute of a circle
Angles	$0 \leq t \leq 8\pi$	$0 \leq t \leq \frac{17\pi}{2}$
Initial position	$x = 0$ $y = 0$	$x = 1$ $y = 0$
Final position	$x = 25.12$ $y = 0$	$x = 26.7$ $y = 1$
Equation	$x = r \cdot t \cdot \cos(t)$ $y = r \cdot t \cdot \sin(t)$	$x = a \cdot (\cos(t) + t \cdot \sin(t))$ $y = a \cdot (\sin(t) - t \cdot \cos(t))$

Table 3.1: Comparison between involute of a circle and Archimedean spiral.

Let's comment the values on the table.

If we look at Figure 3.3, both curves seem to finish almost at the same angle. However, if we look to the values on the table, we see that involute of a circle finishes $\frac{\pi}{2}$ later than the Archimidean spiral.

Concerning initial and final positions, we appreciate that the difference at the beginning is 1 unit, but at the end the difference is almost 1.6 units. We can say thus that in 4 turns, the difference between involute of a circle and Archimedean spiral increases around 0.6 units, this is $0.6/4 = 0.15$ units in each turn.

Finally, if we compare both equations, we see that the equation for Archimidean spiral is much more easy to understand and to apply than the one for involute of a circle.

As the radius of our pivots is small (0.4 mm) and the length of the string (34 cm) and distance between pivots (4.25 cm) limit the number of turns (to 5 or 6 maximum), Archimidean spiral and involute will have very similar results. So, as the equation for Archimidean spiral is easier to apply, we decide to use this one in this project in order to untie the string from the circular pivots.

Let's now show and explain the algorithm we came up with to open the string-envelope.

THE ALGORITHM

In this chapter we explain the algorithm we came up with to open a string-envelope. The analysis of this algorithm is based on [7]. We start by showing the initial configuration of the robot, then we explain the algorithm by showing the flowchart followed by the pseudocode and we finish by showing some examples. The output is that the robot opens the envelope that has been previously tied in an arbitrary manner.

4.1 Initial configuration

In this section, we show the initial configuration of all the elements and the preconditions we need for the algorithm to work correctly.

The initial configuration is shown in Figure 4.1 below.

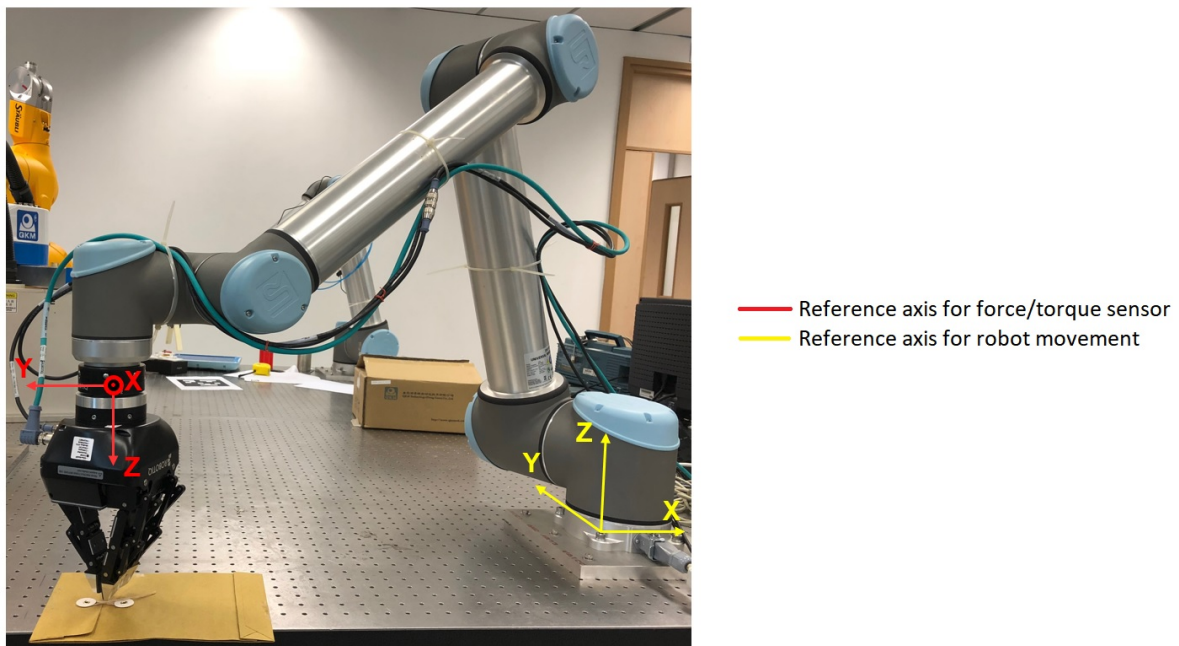


Figure 4.1: Initial configuration of the robot and all the elements.

The red axis represents the reference for the wrist force-torque sensor and the yellow axis represents the reference for the robot movement.

The preconditions for the algorithm to work are:

- The robot is already grasping the string in the middle between the two pivots.
- r_p : Pivots' radius.
- d : Distance between pivots.
- k : Threshold value from which the torque sensor reading is considered relevant.

We call Pivot 1 to the pivot that is attached to the flap of the envelope and Pivot 2 to the other one. We will use the abbreviation P1 for Pivot 1 and P2 for Pivot 2.

4.2 Algorithm

In this section, we show the flowchart, the pseudocode of the algorithm and some examples to explain how this one works.

4.2.1 Flowchart

The flowchart is shown in Figure 4.2.

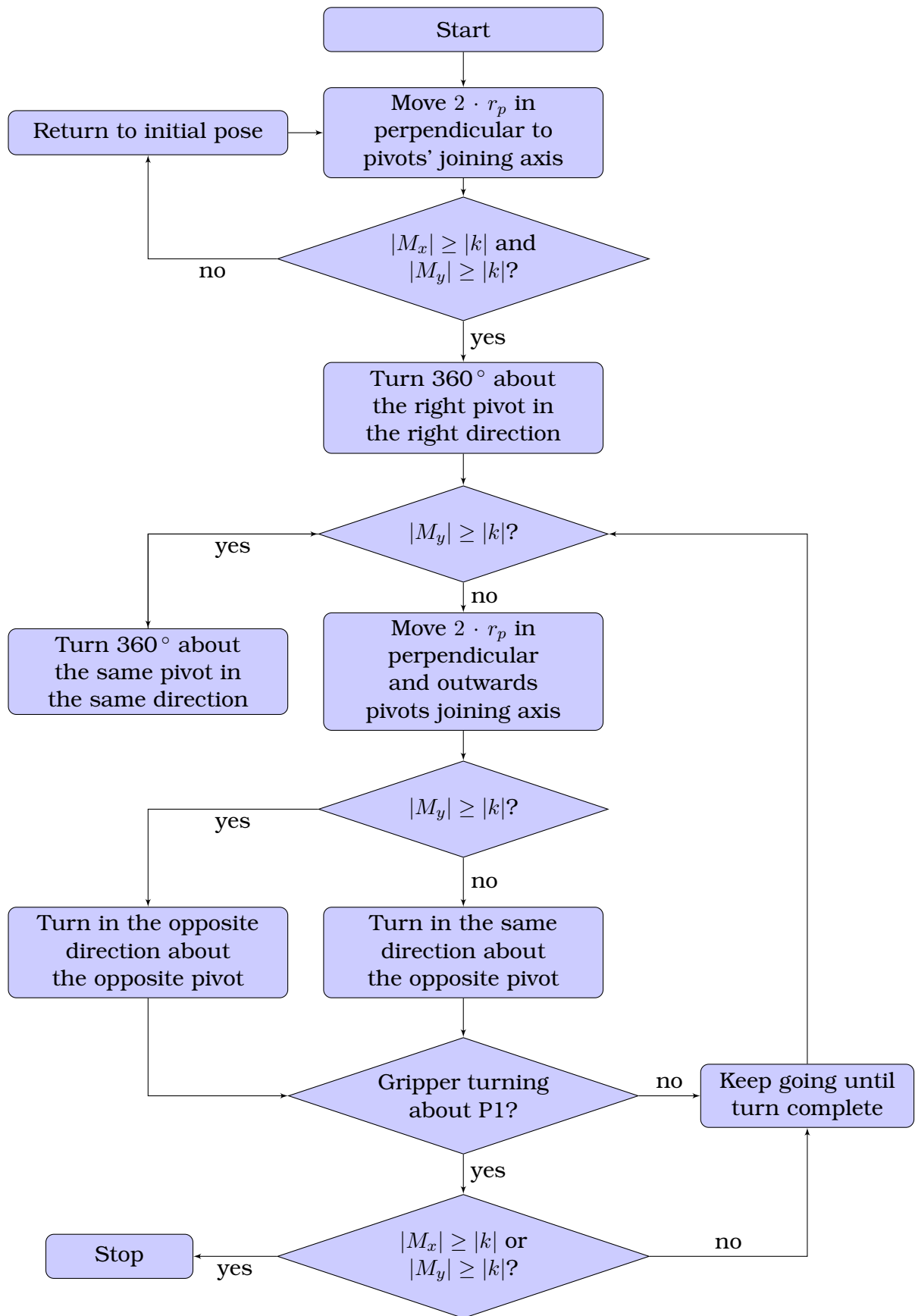


Figure 4.2: Flowchart of the algorithm to open a string-envelope.

The algorithm starts and the gripper moves $2 \cdot r_p$ in perpendicular to pivots' joining axis. If absolute values of torques around x-axis and y-axis are less than absolute value of threshold, the gripper returns to the initial pose and it moves $2 \cdot r_p$ in perpendicular to pivots' joining axis, but in the opposite direction. Otherwise, if absolute values of torques around x-axis and y-axis are greater than absolute value of threshold, then the gripper turns 360° about the right pivot in the right direction to untie the string, depending on the sign of torques.

After one turn, if absolute value of torque around y-axis is greater than absolute value of threshold, then the string is tied about the same pivot and the gripper will turn 360° about the same pivot in the same direction. Otherwise, the gripper will move $2 \cdot r_p$ in perpendicular and outwards pivots' joining axis. There, if absolute value of torque around y-axis is greater than absolute value of threshold, the gripper will turn in the opposite direction about the opposite pivot to untie the string. Otherwise, the gripper will turn in the same direction about the opposite pivot to untie the string.

To know whether the string has completely been untied, it is important to know that the beginning of the string is always attached to P1. So, if the gripper is turning about P1 to try to untie the string and absolute values of torques around x-axis and y-axis increase over absolute value of threshold, then the string is completely untied and the movement stops and the algorithm ends. The fact that torque values increase during the movement about P1 means that the string has completely been untied because its length won't increase anymore while the gripper is "opening" the movement to try to untie the string.

4.2.2 Pseudocode

The pseudocode is presented as a procedure called **UNTIE-STRING** and it is shown below.

Algorithm 1 UNTIE-STRING

```

1:  $turn \leftarrow 0$ 
2:  $[p, c] \leftarrow \text{evaluateTorques}$ 
3: Turn about  $p$  in direction  $c$ 
4: while True do
5:    $[p, c] \leftarrow \text{evaluateTorques}$ 
6:   Turn about  $p$  in direction  $c$ 
7:   if  $p = P1$  then
8:     Store  $M_x, M_y$  during the movement
9:     if  $|M_x| > k$  or  $|M_y| > k$  then
10:      break
11:   else
12:     keep going

```

The algorithm can be divided in two parts. First part goes from line 1 to 3 and second part goes from line 4 to 12.

The first part of the algorithm makes the first turn to start untying the string. There are four possible movements:

- Rotate clockwise (CW) about P1.
- Rotate counterclockwise (CCW) about P1.
- Rotate clockwise about P2.
- Rotate counterclockwise about P2.

Then, for the second part of the algorithm (**while** loop), after the first turn, there are only three possible movements:

- Rotate about the same pivot in the same direction.
- Rotate about the other pivot in the other direction.
- Rotate about the other pivot in the same direction.

Let's start by explaining how the algorithm makes the first movement (lines 1-3). In line 1, the variable *turn* is initialized to 0. This variable lets know whether we are in the first turn or after. In line 2, the function **evaluateTorques** determines which pivot, *p*, and direction, *c*, to turn in order to untie the string, depending on torque values M_x and M_y . This function is shown and explained later. In line 3, the first turn is executed.

After the first turn, we enter the **while** loop (lines 4-12). Here, the algorithm makes the rest of the turns and stops when the string has completely been untied. In line 5, again the **evaluateTorques** function determines the pivot *p* and the direction *c* to untie the string and the movement is executed in line 6.

For the algorithm to be correct and complete, it has to determine whether the string has completely been untied or not. Once the envelope is opened, the algorithm must finish.

As explained before, if the gripper is turning about P1 (line 7), then torques M_x and M_y will be stored during the movement (line 8). If torque values increase, then the movement will stop and the algorithm will finish (lines, 9, 10).

The pseudocode for the **evaluateTorques** function is shown below:

Algorithm 2 evaluateTorques

```

1: if turn = 0 then
2:    $[M_x, M_y] = \text{pullString}(0, \pm 2 \cdot r_p)$ 
3:   if  $M_y > k$  and  $M_x < -k$  then
4:      $p = P1, c = CW$ 
5:   else if  $M_y > k$  and  $M_x > k$  then
6:      $p = P2, c = CCW$ 
7:   else if  $M_y < -k$  and  $M_x < -k$  then

```

Algorithm 3 evaluateTorques (continued)

```

8:    $p = P1, c = CCW$ 
9:   else if  $M_y < -k$  and  $M_x > k$  then
10:     $p = P2, c = CW$ 
11:     $turn + = 1$ 
12:  if  $turn > 0$  then
13:    Store  $M_x, M_y$ 
14:    if  $|M_y| > k$  then
15:       $p$  and  $c$  remain the same as in previous turn
16:    else
17:       $[M_x, M_y] = \text{pullString}(0, \pm 2 \cdot r_p)$ 
18:      if  $|M_y| > k$  then
19:         $p$  and  $c$  are the opposite as in previous turn
20:      else
21:         $p$  is the opposite as in previous turn,  $c$  remains the same
return  $p, c$ 

```

The decision of which pivot and which direction to turn is made inside the **evaluateTorques** function. The decision for the first turn is made from line 1 to line 11 and the decision for the rest of the turns is made from line 12 to line 21.

For the first turn (lines 1-11), the gripper starts moving $2 \cdot r_p$ in perpendicular to pivots' joining axis (line 2).

Being on the right side of the pivots:

- If M_y is greater than k and M_x is less than $-k$, then the right movement to untie the string is to turn CW about P1 (lines 3, 4).
- If M_y and M_x are greater than k , then the right movement to untie the string is to turn CCW about P2 (lines 5, 6).

Being on the left side of the pivots:

- If M_y and M_x are less than $-k$, then the right movement to untie the string is to turn CCW about P1 (lines 7, 8).
- If M_y is less than $-k$ and M_x is greater than k , then the right movement to untie the string is to turn CW about P2 (lines 9, 10).

Then, the variable $turn$ is updated (line 11), so we know the first turn has already been made.

To figure out the rest of the turns (lines 12-21), we start by storing M_x and M_y values at the end of one turn (line 13). Then:

- If $|M_y|$ is greater than $|k|$, then the gripper has to continue turning in the same direction about the same pivot to untie the string (lines 14, 15).

Otherwise, if at the end of one turn, absolute value of torque around y-axis is less than absolute value of threshold, then the string is tied about the other pivot. To know which direction to turn, the gripper will move $2 \cdot r_p$ in perpendicular and outwards pivots' joining axis (line 17). Then:

- If $|M_y|$ is greater than $|k|$, then the right movement to untie the string is the opposite pivot and the opposite direction than in the previous turn (lines 18, 19).
- If $|M_y|$ is less than $|k|$, then the right movement to untie the string is the opposite pivot and the same direction than in the previous turn (lines 20, 21).

This function returns the pivot, p , and the direction, c , to the main function.

After the first turn, we ignore torque around x-axis value because as the gripper unties the string and goes farther from pivots, the distance between the gripper and the pivots is much greater than the distance between pivots. As at the end of one turn the string is almost perpendicular to pivots' joining axis, torque around x-axis is almost 0 and does not allow to distinguish which pivot to turn.

The pseudocode for the **pullString(0, $2 \cdot r_p$)** function (lines 2, 17), that consists of moving the gripper $2 \cdot r_p$ and storing M_x and M_y , is shown below:

Algorithm 4 pullString(0, $\pm 2 \cdot r_p$)

```

1: if  $turn = 0$  then
2:   Move 0 in x,  $+2 \cdot r_p$  in y
3:   if  $|M_y| < |k|$  then
4:     Return to initial pose
5:     Move 0 in x,  $-2 \cdot r_p$  in y
6: else
7:   Move 0 in x,  $2 \cdot r_p$  in y outwards pivots' joining axis
   return  $M_x, M_y$ 

```

For the first turn (line 1), the gripper starts moving $+2 \cdot r_p$ in y-axis (line 2). If absolute value of torque around y-axis is less than absolute value of threshold, then the gripper returns to the initial pose and moves $-2 \cdot r_p$ in y-axis (lines 3-5).

After the first turn (line 6), the gripper moves $2 \cdot r_p$ in y-axis outwards pivots' joining axis (line 7).

This function returns M_x and M_y values to the **evaluateTorques** function.

Now, we will show some examples of how the algorithm works.

4.2.3 Examples

For a better understanding of what we explained before, we show here some examples of how the algorithm works to figure out the first turn, the rest of the turns and the end of the movement.

We start by showing an example of how the algorithm figures out the first turn. Figure 4.3 shows the four possible cases for the first turn.

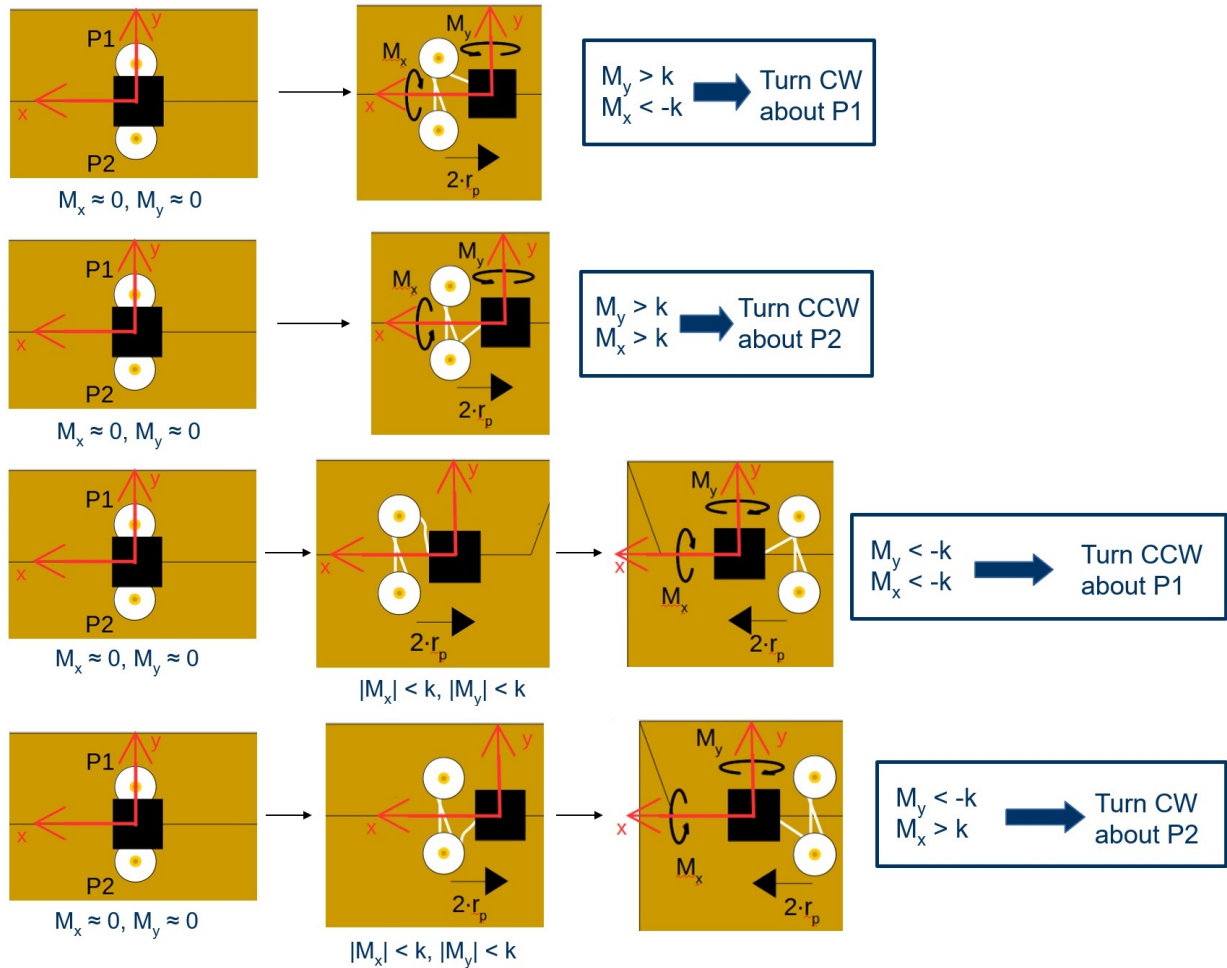


Figure 4.3: Example of how the algorithm figures out the first turn.

Let's tell the first turn was CW about P1, then Figure 4.4 shows what happens after this turn if the string is again tied about the same pivot.

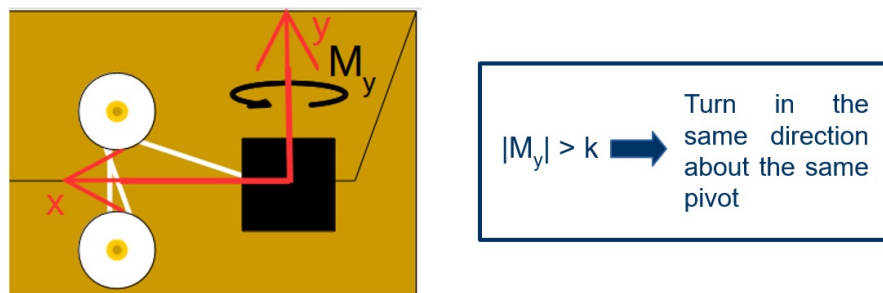


Figure 4.4: Example of the case where the string is tied about the same pivot after one turn.

Otherwise, if the string is tied about the other pivot after the first turn, Figure 4.5 shows what happens:

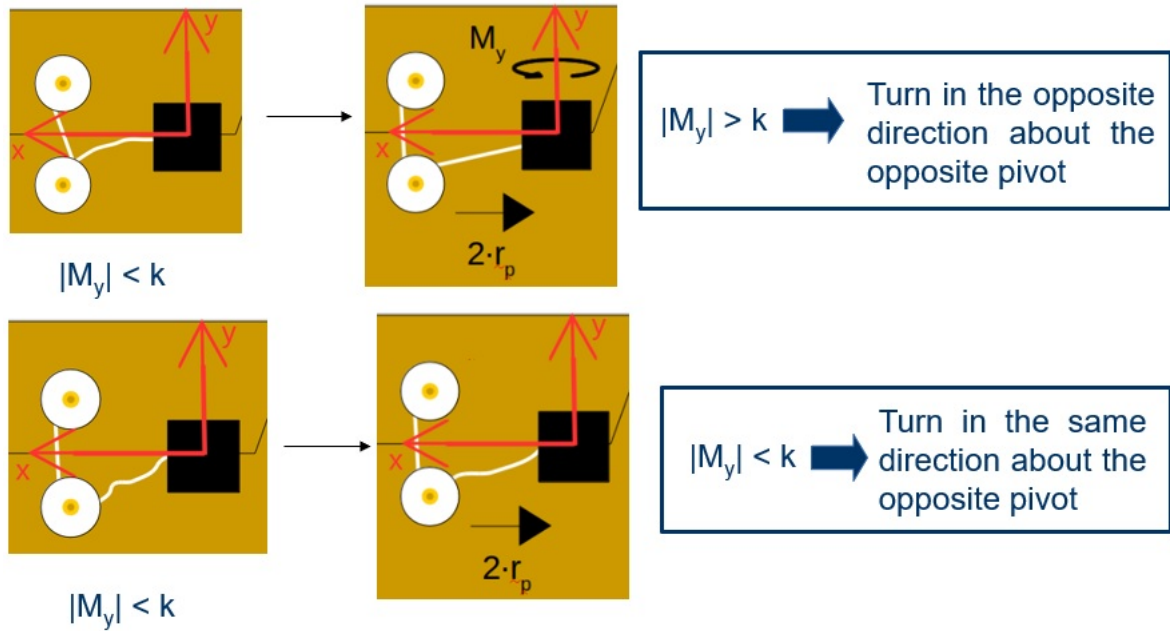


Figure 4.5: Example of the cases where the string is tied about the other pivot after one turn.

Finally, Figure 4.6 shows how the algorithm figures out the end of the movement.

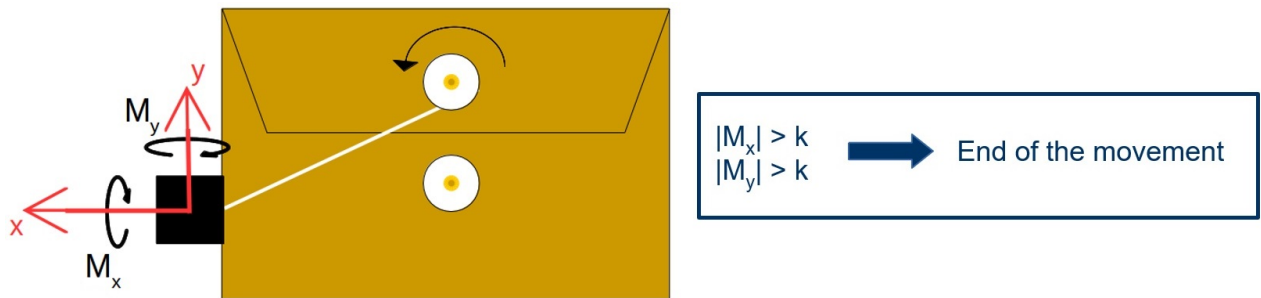


Figure 4.6: Example of the end of the movement.

EXPERIMENTAL RESULTS

In this chapter we show the experiments we have made and their results.

The four experiments we talk about here are the following:

- Experiment 1: Why use torque values instead of force.
- Experiment 2: Minimum radius of the pivot.
- Experiment 3: Minimum distance between pivots.
- Experiment 4: Comparison of robot fail ratio with human fail ratio.

The first experiment explains why we used torque values in this work instead of force values to determine the direction of string tension. We also explain how the threshold value for torque has been established.

The second experiment shows which is the minimum radius for the pivot to distinguish between CW and CCW case. We establish the minimum radius from which the pivot is considered negligible.

The third experiment consists on establishing the minimum distance between pivots to distinguish them through torque values.

Finally, the fourth experiment consists on comparing robot fail ratio with human fail ratio. There we do several experiments with the robot, and then we repeat the same experiments with different people in the same conditions than the robot (with no vision).

Let's start explaining the first experiment.

5.1 Experiment 1: Why use torque values instead of force

As told before, here we explain why we use torques instead of forces and then we establish the threshold from which torque values are considered relevant.

5.1.1 Comparison between force values and torque values

It is true that the most intuitive way to solve the problem of untying the string from the pivot would be to look at forces because these would directly tell the direction of string tension. But, let's see what happens if we look to force and torque readings of the Robotiq FT300 sensor in the case where there is no tension of the string and in the case where there is.

For this experiment, the case of no string tension is the initial pose, when the gripper is just grasping the end of the string between the two pivots, before starting the movement. The case of string tension is the case when the gripper moves $2 \cdot r_p$ in perpendicular to pivots joining axis. Figure 5.1 below shows both cases.

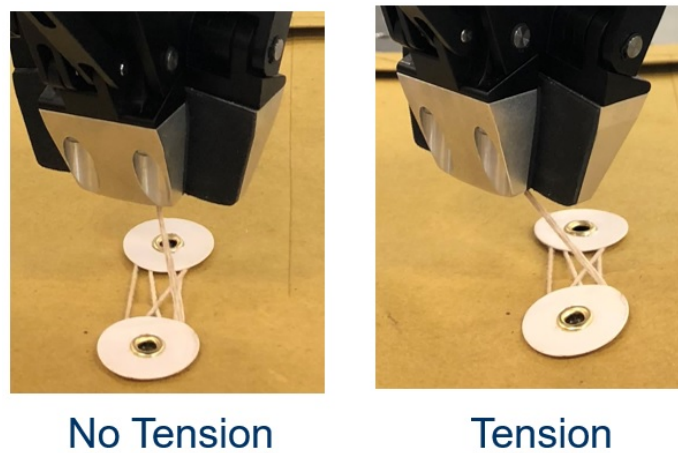


Figure 5.1: Tension and No Tension cases.

Let's take 30 values of force and torque from the force/torque sensor in both cases. Results are presented in Table 5.1.

NO TENSION (INITIAL POSE)				TENSION (CW P1)			
$F_x(N)$	$F_y(N)$	$M_x(N \cdot m)$	$M_y(N \cdot m)$	$F_x(N)$	$F_y(N)$	$M_x(N \cdot m)$	$M_y(N \cdot m)$
1.19	1.41	0.072	0.032	2.55	-0.04	-0.177	0.422
0.76	0.27	0.023	0.035	2.81	1.56	-0.199	0.431
-0.04	-0.19	0.060	0.007	2.25	-0.40	-0.194	0.432
0.13	-1.28	0.057	0.006	2.34	0.06	-0.170	0.412
1.96	0.13	0.056	0.030	3.31	-1.18	-0.184	0.422
0.71	-0.63	0.033	0.037	3.75	1.00	-0.215	0.440
1.61	-0.26	0.043	0.045	2.71	1.00	-0.214	0.446
1.79	-0.90	0.056	0.010	3.96	1.24	-0.214	0.434
0.13	0.28	0.049	0.039	3.28	2.52	-0.190	0.411
-0.65	0.31	0.009	0.044	5.23	-0.31	-0.196	0.448
-0.88	-0.81	0.063	0.045	2.18	-0.44	-0.212	0.425
-0.36	1.10	0.069	0.014	3.60	1.04	-0.198	0.411
0.43	-0.49	0.056	0.049	3.47	2.06	-0.191	0.406

-0.64	-0.10	0.065	0.029	2.41	0.46	-0.157	0.430
0.50	-0.22	0.058	0.032	3.65	1.72	-0.169	0.416
-0.42	-0.89	0.067	0.031	3.04	-0.91	-0.191	0.409
0.15	-1.66	0.074	0.020	3.23	0.08	-0.177	0.403
-0.74	0.89	0.074	0.028	3.18	1.10	-0.190	0.415
1.38	-1.86	0.071	0.037	4.07	0.63	-0.192	0.407
0.21	-1.05	0.073	0.048	4.28	-0.57	-0.197	0.428
0.26	-0.95	0.072	0.022	3.86	0.14	-0.207	0.390
0.38	-1.54	0.061	0.035	3.26	-1.17	-0.188	0.442
1.43	-2.17	0.058	0.057	3.17	0.10	-0.199	0.368
0.16	-0.71	0.013	0.058	3.05	0.06	-0.173	0.438
-0.67	-0.21	0.078	0.045	4.87	1.35	-0.182	0.444
0.21	-1.97	0.065	0.021	2.30	0.67	-0.155	0.403
0.06	-0.46	0.063	0.001	3.23	3.23	-0.200	0.409
1.43	-1.37	0.044	0.026	4.64	-0.95	-0.180	0.417
-1.43	-0.77	0.042	0.049	3.67	2.00	-0.171	0.407
1.08	-0.83	0.073	0.015	1.12	-0.19	-0.199	0.430

Table 5.1: Results of force and torque readings for the case of no tension and the case of tension of the string.

Now let's look to the maximum and minimum values in each case (Table 5.2).

	NO TENSION (INITIAL POSE)				TENSION (CW P1)			
	$F_x(N)$	$F_y(N)$	$M_x(N \cdot m)$	$M_y(N \cdot m)$	$F_x(N)$	$F_y(N)$	$M_x(N \cdot m)$	$M_y(N \cdot m)$
Min	-1.43	-2.17	0.009	0.001	1.12	-1.18	-0.215	0.368
Max	1.96	1.41	0.078	0.058	5.23	3.23	-0.155	0.448
Max - Min	3.39	3.58	0.069	0.057	4.11	4.41	0.060	0.080

Table 5.2: Maximum and minimum values of force and torque.

We appreciate that the difference between the maximum and the minimum value for force readings is around $4N$, while for torques it is around $0.070N \cdot m$. The sensor is much noisier for force readings than for torques. Furthermore, if we compare maximum and minimum values in *tension* and *no tension* case, we see that force values are overlapped in both cases:

$$F_{x,max,notension} = 1.96N > 1.12N = F_{x,min,tension}$$

$$F_{y,max,notension} = 1.41N > -1.18N = F_{y,min,tension}$$

While torques are clearly distinguishable:

$$M_{x,min,notension} = 0.009N \cdot m > -0.155N \cdot m = M_{x,max,tension}$$

$$M_{y,max,notension} = 0.058N \cdot m < 0.368N \cdot m = M_{y,min,tension}$$

As there is an error in sensor readings, we decided to take the mean values as the most accurate values to compare both cases. Let's look now to mean values and standard deviation in Table 5.3.

		$F_x(N)$	$F_y(N)$	$M_x(N \cdot m)$	$M_y(N \cdot m)$
NO TENSION	St. dev.	0.86	0.89	0.018	0.015
(INITIAL POSE)	Mean	0.34	-0.56	0.057	0.032
TENSION	St. dev.	0.87	1.12	0.016	0.018
(CW P1)	Mean	0.49	0.53	-0.189	0.42

Table 5.3: Standard deviation and mean values for force and torque readings in tension and no tension case.

With this values of standard deviation and mean, we establish the ranges of force and torque variations for tension and no tension case. The results are shown in Table 5.4.

	No Tension (Initial pose)	Tension case (CW P1)
$F_x \pm st.dev.(N)$	$-0.52N < F_x < 1.2N$	$-0.38N < F_x < 1.36N$
$F_y \pm st.dev.(N)$	$-1.45N < F_y < 0.33N$	$-0.59N < F_y < 1.65N$
$M_x \pm st.dev.(N \cdot m)$	$0.039N \cdot m < M_x < 0.075N \cdot m$	$-0.205N \cdot m < M_x < -0.173N \cdot m$
$M_y \pm st.dev.(N \cdot m)$	$0.017N \cdot m < M_y < 0.047N \cdot m$	$0.402N \cdot m < M_y < 0.438N \cdot m$

Table 5.4: Ranges of force and torque variations in tension and no tension cases.

Looking to forces values, we see that both ranges for F_x and F_y are overlapped in case of tension and no tension. Variation in force readings is too big compared to the little forces (tension of the string) we are working with. We can thus not distinguish between the tension case and no tension case with force values. Force cannot be used to determine the direction of string tension in this case.

Otherwise, if we look to torque values, we see that ranges for tension case and no tension case are clearly distinguishable. Torque readings are much more precise than force readings and that is why we use torque values in this work.

5.1.2 Threshold value for torque

After explaining why torque values are more accurate for this project, let's establish now the threshold value for torque readings to be considered relevant.

To do that, we made 10 experiments where we took 30 values of torque in tension and in no tension case. Then, with the mean and the standard deviation we compare both cases of each experiment. We show directly the results for the mean and standard deviation in each experiment. These are shown in Table 5.5 below:

		NO TENSION (INITIAL POSE)		TENSION (CW P1)	
		St. dev.	Mean	St. dev.	Mean
Experiment 1	$M_x(N \cdot m)$	0.018	0.057	0.016	-0.189
	$M_y(N \cdot m)$	0.015	0.032	0.018	0.42
Experiment 2	$M_x(N \cdot m)$	0.022	0.042	0.021	-0.168
	$M_y(N \cdot m)$	0.016	0.033	0.026	0.140
Experiment 3	$M_x(N \cdot m)$	0.017	0.028	0.024	-0.072
	$M_y(N \cdot m)$	0.024	0.042	0.022	0.184
Experiment 4	$M_x(N \cdot m)$	0.019	0.052	0.024	-0.056
	$M_y(N \cdot m)$	0.014	0.047	0.015	0.143
Experiment 5	$M_x(N \cdot m)$	0.015	0.031	0.018	-0.173
	$M_y(N \cdot m)$	0.022	0.022	0.019	0.215
Experiment 6	$M_x(N \cdot m)$	0.016	0.009	0.024	-0.240
	$M_y(N \cdot m)$	0.015	0.020	0.016	0.199
Experiment 7	$M_x(N \cdot m)$	0.021	0.053	0.020	-0.291
	$M_y(N \cdot m)$	0.028	0.045	0.023	0.284
Experiment 8	$M_x(N \cdot m)$	0.028	0.055	0.016	-0.078
	$M_y(N \cdot m)$	0.021	0.031	0.011	0.152
Experiment 9	$M_x(N \cdot m)$	0.026	0.038	0.018	-0.085
	$M_y(N \cdot m)$	0.015	0.017	0.021	0.141
Experiment 10	$M_x(N \cdot m)$	0.023	0.053	0.021	-0.115
	$M_y(N \cdot m)$	0.027	0.056	0.018	0.202

Table 5.5: Results for 10 experiments of standard deviation and mean values for torque in tension and no tension case.

The maximum value of standard deviation is $0.028N \cdot m$ in experiments 7 and 8, so the threshold value must be greater than $0.028N \cdot m$.

To see if any torque value is overlapped in tension and no tension case, let's establish the ranges of variation of torque in each experiment. These ranges are shown in Table 5.6:

		NO TENSION (INITIAL POSE)	TENSION (CW P1)
Exp. 1	$M_x \pm st.dev.(N \cdot m)$	$0.039 < M_x < 0.075$	$-0.205 < M_x < -0.173$
	$M_y \pm st.dev.(N \cdot m)$	$0.017 < M_y < 0.047$	$0.402 < M_y < 0.438$
Exp. 2	$M_x \pm st.dev.(N \cdot m)$	$0.02 < M_x < 0.064$	$-0.189 < M_x < -0.147$

	$M_y \pm st.dev.(N \cdot m)$	$0.017 < M_y < 0.049$	$0.114 < M_y < 0.166$
Exp. 3	$M_x \pm st.dev.(N \cdot m)$ $M_y \pm st.dev.(N \cdot m)$	$0.011 < M_x < 0.045$ $0.018 < M_y < 0.066$	$-0.096 < M_x < -0.048$ $0.162 < M_y < 0.206$
Exp. 4	$M_x \pm st.dev.(N \cdot m)$ $M_y \pm st.dev.(N \cdot m)$	$0.033 < M_x < 0.071$ $0.033 < M_y < 0.061$	$-0.08 < M_x < -0.032$ $0.128 < M_y < 0.158$
Exp. 5	$M_x \pm st.dev.(N \cdot m)$ $M_y \pm st.dev.(N \cdot m)$	$0.016 < M_x < 0.046$ $0.000 < M_y < 0.044$	$-0.191 < M_x < -0.155$ $0.196 < M_y < 0.234$
Exp. 6	$M_x \pm st.dev.(N \cdot m)$ $M_y \pm st.dev.(N \cdot m)$	$-0.007 < M_x < 0.025$ $0.005 < M_y < 0.035$	$-0.264 < M_x < -0.216$ $0.183 < M_y < 0.215$
Exp. 7	$M_x \pm st.dev.(N \cdot m)$ $M_y \pm st.dev.(N \cdot m)$	$0.032 < M_x < 0.074$ $0.017 < M_y < 0.073$	$-0.311 < M_x < -0.271$ $0.261 < M_y < 0.307$
Exp. 8	$M_x \pm st.dev.(N \cdot m)$ $M_y \pm st.dev.(N \cdot m)$	$0.027 < M_x < 0.083$ $0.010 < M_y < 0.052$	$-0.094 < M_x < -0.062$ $0.141 < M_y < 0.163$
Exp. 9	$M_x \pm st.dev.(N \cdot m)$ $M_y \pm st.dev.(N \cdot m)$	$0.012 < M_x < 0.064$ $0.002 < M_y < 0.032$	$-0.103 < M_x < -0.067$ $0.120 < M_y < 0.162$
Exp. 10	$M_x \pm st.dev.(N \cdot m)$ $M_y \pm st.dev.(N \cdot m)$	$0.030 < M_x < 0.076$ $0.029 < M_y < 0.083$	$-0.136 < M_x < -0.094$ $0.184 < M_y < 0.220$

Table 5.6: Range of variation of torques in 10 experiments.

We appreciate that in the 10 experiments, ranges in tension case and in no tension case are never overlapped.

Finally, we look to the difference between the means of both cases in each experiment (Table 5.7).

	$M_{x,tension} - M_{x,notension}(N \cdot m)$	$M_{y,tension} - M_{y,notension}(N \cdot m)$
Experiment 1	-0.246	0.388
Experiment 2	-0.21	0.107
Experiment 3	-0.100	0.142
Experiment 4	-0.108	0.096
Experiment 5	-0.204	0.193
Experiment 6	-0.249	0.179
Experiment 7	-0.344	0.239
Experiment 8	-0.133	0.121
Experiment 9	-0.123	0.124
Experiment 10	-0.168	0.146

Table 5.7: Difference between means in tension and no tension case for 10 experiments.

We appreciate that the minimum difference between the 2 cases is for M_y in experiment 4, and the value is $0.096N \cdot m$. We decided thus, to establish a threshold value of $0.095N \cdot m$ from which we consider that the string is taut. Furthermore, this value is greater than $0.028N \cdot m$, which is the maximum standard deviation.

5.2 Experiment 2: Minimum radius of the pivots

In this section we show the experiment made to figure out which is the minimum radius of the pivots for the sensor to distinguish between CW and CCW case.

For this, we took different objects with different radius and we wound the thread around them. By moving $2 \cdot r_p$, we see when torque values do not allow to distinguish anymore between CW and CCW case.

5.2.1 Radius = 4 mm

We start with the real radius of our pivots. This is 4 mm. In Figure 5.2 below, we can see the difference between CW and CCW case:

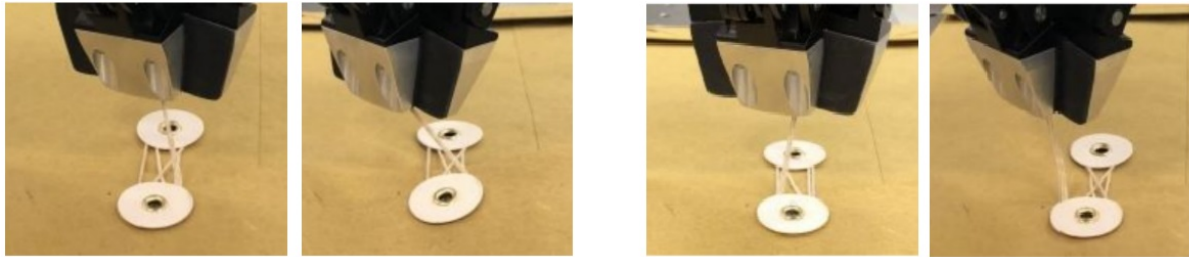


Figure 5.2: CW case on the left and CCW case on the right for $r = 4$ mm.

Let's see the results for torques in Table 5.8.

	r = 4 mm		
	$ M_{x,tension} - M_{x,notension} $	$ M_{y,tension} - M_{y,notension} $	Comparison to threshold
CW case	$0.101N \cdot m$	$0.142N \cdot m$	$> 0.095N \cdot m$
CCW case	$0.053N \cdot m$	$0.018N \cdot m$	$< 0.095N \cdot m$

Table 5.8: Results for torque values in CW and CCW case for $r = 4$ mm.

We appreciate that when the gripper moves $2 \cdot r_p$ in perpendicular to pivots' joining axis in CW case both $|\Delta M_x|$ and $|\Delta M_y|$ are greater than threshold value while in CCW case they are smaller. We can thus distinguish both cases for a pivot's radius of 4 mm.

Let's see what happens with a radius of 3 mm.

5.2.2 Radius = 3 mm

For this experiment, we took a pen with a radius of 3 mm and we winded the string of the envelope around it CW and then CCW. Figure 5.3 shows both cases:



Figure 5.3: CW case on the left and CCW case on the right for $r = 3$ mm.

Let's see the results for torques in Table 5.9.

	$r = 3$ mm		
	$ M_{x,tension} - M_{x,notension} $	$ M_{y,tension} - M_{y,notension} $	Comparison to threshold
CW case	$0.254N \cdot m$	$0.185N \cdot m$	$> 0.095N \cdot m$
CCW case	$0.020N \cdot m$	$0.056N \cdot m$	$< 0.095N \cdot m$

Table 5.9: Results for torque values in CW and CCW case for $r = 3$ mm.

We appreciate the same as in the previous case. In CW case both $|\Delta M_x|$ and $|\Delta M_y|$ are greater than threshold value while in CCW case they are smaller. We can thus distinguish both cases for a pivot's radius of 3 mm.

Let's see now what happens with a radius of 2 mm.

5.2.3 Radius = 2 mm

For this experiment, we took an Allen key with a 2 mm radius and we winded the string around it CW and then CCW. Both cases are shown in Figure 5.4:

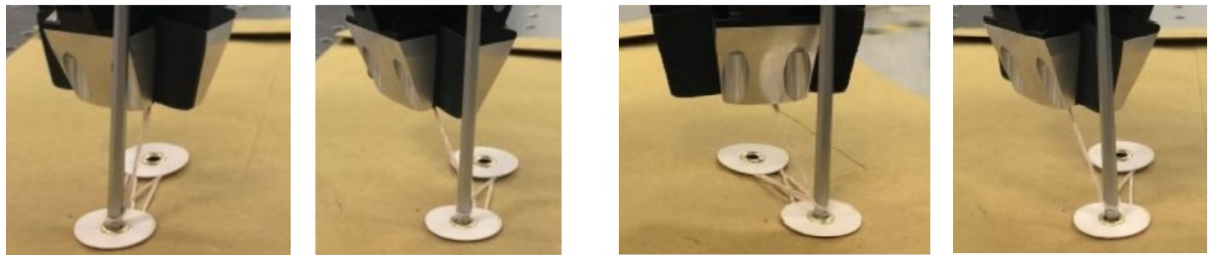


Figure 5.4: CW case on the left and CCW case on the right case for $r = 2$ mm.

The results for torques are shown in Table 5.10.

	r = 2 mm		
	$ M_{x,tension} - M_{x,notension} $	$ M_{y,tension} - M_{y,notension} $	Comparison to threshold
CW case	$0.405N \cdot m$	$0.320N \cdot m$	$> 0.095N \cdot m$
CCW case	$0.566N \cdot m$	$0.407N \cdot m$	$> 0.095N \cdot m$

Table 5.10: Results for torque values in CW and CCW case for $r = 2\text{mm}$.

In this case, we see that both in CW and CCW case, if the gripper moves $2 \cdot r_p$ in perpendicular to pivots' joining axis, $|\Delta M_x|$ and $|\Delta M_y|$ are greater than threshold value. In this case we can thus not distinguish between CW and CCW case. For this algorithm to work with this force/torque sensor, pivots' radius must be bigger than 2 mm. A radius of 2 mm or smaller could be considered as negligible.

5.3 Experiment 3: Minimum distance between pivots

In this section we show different experiments where we modify the distance between pivots to figure out which is the minimum distance we could have for the force/torque sensor to distinguish between P1 and P2.

Let's start by analyzing what happens with the real distance between pivots (4.25 cm).

5.3.1 Distance = 4.25 cm

For this experiment, we first tied the string around P1 and then about P2 and we took torque values when the gripper moved $2 \cdot r_p$ in perpendicular to pivots' joining axis.

In Figure 5.5 this first experiment is shown:

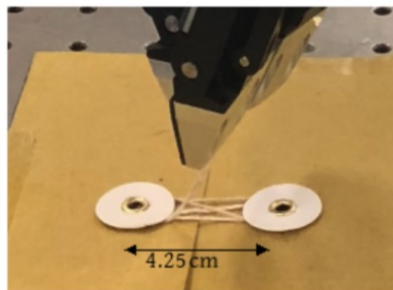


Figure 5.5: First experiment with a distance of 4.25 cm between pivots.

Values for torques when the string is tied about P1 and when the string is tied about P2 are shown in Table 5.11 below.

d = 4.25 cm				
	$M_{x,notension}$	$M_{x,tension}$	$M_{x,tension} - M_{x,notension}$	Comparison to threshold
P1	$0.026N \cdot m$	$-0.134N \cdot m$	$-0.160N \cdot m$	$< -0.095N \cdot m$
P2	$-0.035N \cdot m$	$0.067N \cdot m$	$0.102N \cdot m$	$> 0.095N \cdot m$

Table 5.11: Results for torque values when the string is tied about P1 and about P2 for a distance of 4.25 cm between pivots.

In the case where the string is tied about P1, ΔM_x is less than the negative value of the threshold when the gripper moves $2 \cdot r_p$, while when the string is tied about P2, ΔM_x is greater than the positive value of the threshold. We can thus distinguish if the string is tied about P1 or P2 for a distance of 4.25 cm between pivots (real case).

Let's see what happens for a distance of 3.5 cm between pivots.

5.3.2 Distance = 3.5 cm

For this experiment, we did the same thing as in the previous one but we changed pivots' distance to 3.5 cm. This is shown in Figure 5.6 below:

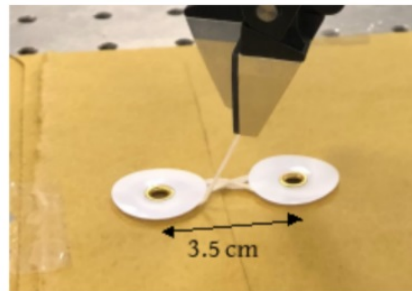


Figure 5.6: Experiment with a distance of 3.5 cm between pivots.

Values for torques when the string is tied about P1 and when the string is tied about P2 are shown in Table 5.12 below.

d = 3.5 cm				
	$M_{x,notension}$	$M_{x,tension}$	$M_{x,tension} - M_{x,notension}$	Comparison to threshold
P1	$0.004N \cdot m$	$-0.097N \cdot m$	$-0.101N \cdot m$	$< -0.095N \cdot m$
P2	$-0.032N \cdot m$	$0.073N \cdot m$	$0.105N \cdot m$	$> 0.095N \cdot m$

Table 5.12: Results for torque values when the string is tied about P1 and about P2 for a distance of 3.5 cm between pivots.

Again, in this case as in the previous one, when the string is tied about P1, ΔM_x

is less than the negative value of the threshold, while when the string is tied about P2, ΔM_x is greater than the positive value of the threshold. We can thus distinguish whether the string is tied about P1 or P2 for a distance of 3.5 cm between pivots.

Let's see now what happens for a distance of 3 cm between pivots.

5.3.3 Distance = 3 cm

For this experiment, we changed pivots' distance to 3 cm. This is shown in Figure 5.7 below:

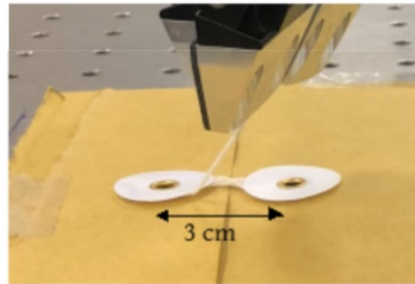


Figure 5.7: Experiment with a distance of 3 cm between pivots.

Values for torques when the string is tied about P1 and when the string is tied about P2 are shown in Table 5.13 below.

d = 3 cm				
	$M_{x,notension}$	$M_{x,tension}$	$M_{x,tension} - M_{x,notension}$	Comparison to threshold
P1	$0.033N \cdot m$	$-0.026N \cdot m$	$-0.007N \cdot m$	$> -0.095N \cdot m$
P2	$0.032N \cdot m$	$0.046N \cdot m$	$0.014N \cdot m$	$< 0.095N \cdot m$

Table 5.13: Results for torque values when the string is tied about P1 and about P2 for a distance of 3 cm between pivots.

In this case, when the string is tied about P1 as well as when it is tied about P2, $|\Delta M_x|$ is less than the threshold value, when the gripper moves $2 \cdot r_p$. We can thus not know which pivot the string is tied about for a distance of 3 cm between pivots or less.

5.4 Experiment 4: Comparison of robot fail ratio with human fail ratio

For this experiment we performed 30 tests where the robot had to open the envelope that had been previously tied arbitrarily. Then, we performed this same 30 tests with 5

different human being with closed eyes, so they were in the same conditions than the robot. Finally we compared the results to see who succeed more times.

Let's start by seeing robot results of success and fails in Table 5.14.

	Robot
Success	19
Fails	11

Table 5.14: Number of success and fails made by the robot in the 30 experiments.

The robot succeeded 19 times to open the envelope and failed 11 out of 30 attempts. The reasons why the robot failed are:

- Bad reading of the torque by the sensor: a bad reading of torque makes the robot to take the wrong action. For example, it turns around the wrong pivot or in the wrong direction.
- Error in the movement of the robot: if there is an error in the movement and the spiral movement is not perfect, the gripper may pull the string too much during the movement (and break it) or it may not finish the turn where it is supposed to and the envelope will not be opened.

Let's see now the results for humans. In Figure 5.8 we can see how the experiment was performed by the 5 people:



Figure 5.8: Experiment performed by 5 people with closed eyes trying to open the envelope.

In Table 5.15, the results for these 5 people are shown:

	Human 1	Human 2	Human 3	Human 4	Human 5
Success	12	21	17	7	26
Fails	18	9	13	23	4

Table 5.15: Number of success and fails made by the 5 people in the 30 experiments.

We appreciate that results for different humans are very different. That is why we took different people to perform the experiments, because every person is different while the algorithm for the robot will always work the same way. The reasons why people failed are:

- They did not know which direction to turn.
- They did not know where they were in the space and they gave up.
- They thought the envelope was already opened and it was not.

Let's compare now the percentage of success for the robot and for an average person in Table 5.16:

	Robot	Humans
Percentage of success	63.30%	55.30%

Table 5.16: Comparison between robot percentage of success and human percentage of success.

We see that the percentage of success for the robot is higher than for an average human.

The main difference between humans and the robot was that people could correct their movement if they realized they were doing it wrong, while the robot is not able to correct the movement and will not open the envelope if it doesn't figure out the good movement the first time.

We also noticed that when a person succeeded to open the envelope, he or she took much less time than the robot. The robot takes around 4 minutes to open the envelope while the human did it in about 30 seconds.

CONCLUSION AND FUTURE LINES

6.1 Conclusion

To conclude this work we can say that the algorithm to open the envelope works correctly. The spiral movement is a good approach to plan the trajectory of the end effector in order to unwind the string from the circular pivots. We have demonstrated that it is possible for a robot to open a string-envelope only with a wrist force/torque sensor. Furthermore, even if the algorithm is not perfect, it succeeds more than an average human in the same conditions.

6.2 Future Lines

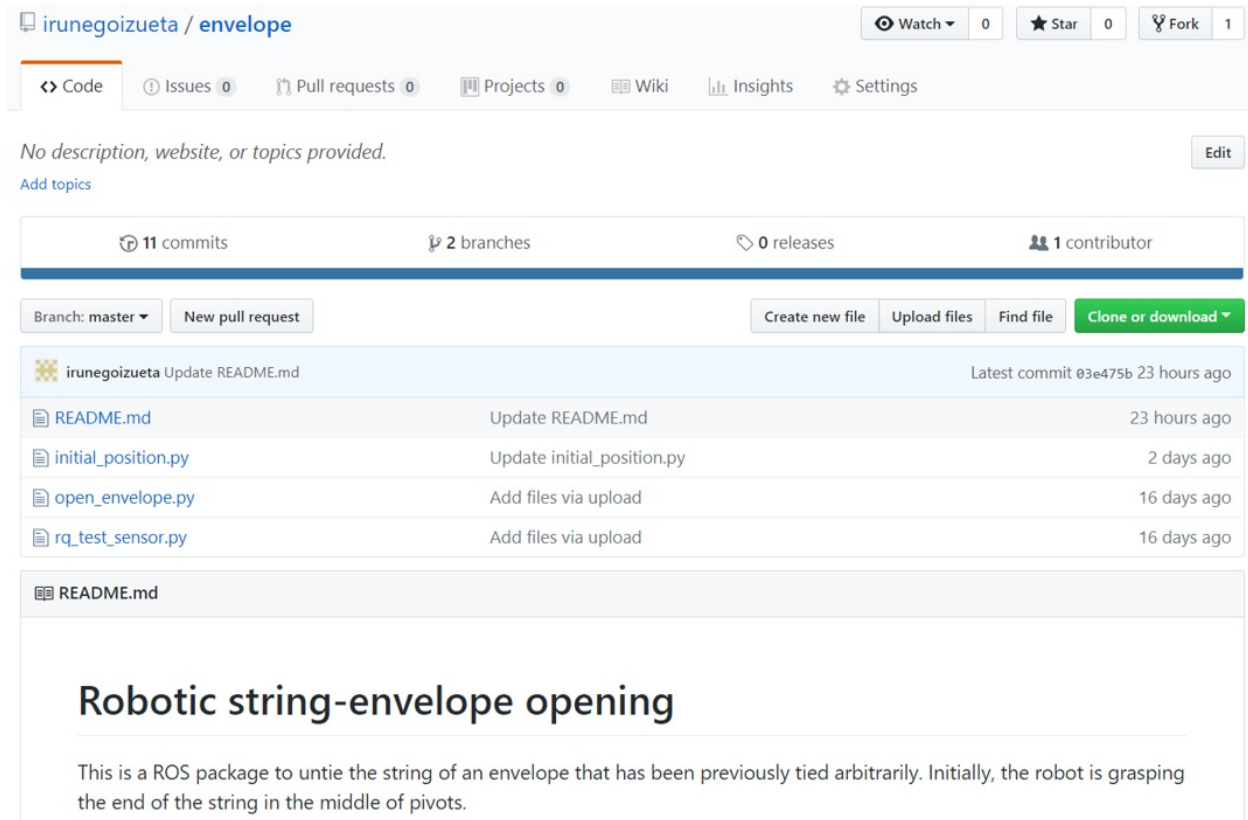
In order to continue developing this project, we could add corrective movements. We saw that when humans succeeded to open the envelope, it was because they were able to correct their movement. If we add corrective movements to the robot, its percentage of success could increase.

We also noticed that when humans opened the envelope correctly, they did it in less time than the robot. The speed of the robot's movement can be increased to such point that it could be much faster than a human. We could add superhuman capabilities to the robot by increasing its speed, so it would open the envelope in less than 5 seconds.

In order to enhance autonomy, other sensors such as visual sensing could be added to the robot. If we provide the robot with visual sensing, it could for example detect the end of the string at the beginning and grasp it, or measure the distance between pivots and pivots' radius. This way, we would get rid of the preconditions we have and the algorithm would become more general and work in many different situations and with different parameters of envelopes.

Finally, we created a project on GitHub (Figure 6.1). This is a platform for collaborative development that contains from open source projects to private team repositories. There, developers can discover, share, and build better software. The repository created for this project is called "envelope" and it can be found in the following link: <https://github.com/irunegoizueta/envelope/>. There, we can find all the

executables needed for the algorithm to work correctly and an explanation of how to install and run them.



irunegoizueta / envelope

Watch 0 Star 0 Fork 1

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

No description, website, or topics provided. [Add topics](#) [Edit](#)

11 commits 2 branches 0 releases 1 contributor

Branch: master New pull request Create new file Upload files Find file Clone or download

File	Commit Message	Time
README.md	Update README.md	23 hours ago
initial_position.py	Update initial_position.py	2 days ago
open_envelope.py	Add files via upload	16 days ago
rq_test_sensor.py	Add files via upload	16 days ago

Robotic string-envelope opening

This is a ROS package to untie the string of an envelope that has been previously tied arbitrarily. Initially, the robot is grasping the end of the string in the middle of pivots.

Figure 6.1: Repository "envelope" on GitHub.

Chapter 7

BUDGET

Here, we show the budget corresponding to the development of this project. This one is split in different lines:

- Consumables: The consumables used in this project are the string-envelopes.
- Equipment: The equipment used in this work is the UR10 robot, the Robotiq force/torque sensor, the Robotiq 3 finger gripper and the computer.
- Software: Here we find Ubuntu 16.04.3 LTS, Windows 10 and ROS.
- Labour: This line corresponds to the engineering hours spent on this work.

In Table 7.1 below, the budget for consumables is shown:

Quantity	Reference	Description	Unitary Price (\$)	Total Price (\$)
20	65035	String-Envelope	1.4	28
Total consumables				28

Table 7.1: Budget for consumables.

Table 7.2 shows the budget for the equipment used in this project:

Equipment	Cost of acquisition (\$)	Depreciation period (years)	Depreciation mensual cost (\$)	Time of use (month)	Depreciation (\$)
Robot	50000	4	1041	6	6246
Force/Torque Sensor	4000	4	83	6	498
Gripper	18000	4	375	6	2250
Computer	700	4	15	6	90
Total equipment					9084

Table 7.2: Budget for equipment.

The budget for the software is shown in Table 7.3:

Software	Cost of acquisition (\$)	Depreciation period (years)	Depreciation mensual cost (\$)	Time of use (month)	Depreciation (\$)
Ubuntu 16.04.3 LTS	0	-	-	6	-
Windows 10	12000	1	1000	1	1000
ROS	0	-	-	6	-
Total software					1000

Table 7.3: Budget for the software.

Finally, the budget for the labour is shown in Table 7.4:

Task	Duration (hours)	Unitary Price (\$)	Total Price (\$)
Engineering	960	50	48000
Total labour			48000

Table 7.4: Budget for the labour.

The summary of the global budget for this project amounts to 77347 \$ and is presented in Table 7.5:

Line	Partial Amount (\$)	Accumulated Amount (\$)
Consumables	28	28
Equipment	9084	9112
Software	1000	10112
Labour	48000	58112
Indirect costs (10%)		5811.2
Total without VAT		63923.2
Total with VAT		77347

Table 7.5: Summary of the global budget for this project.

Bibliography

- [1] Moveit! <https://moveit.ros.org/>. Accessed: 2017-11-13.
- [2] robotiq. <http://wiki.ros.org/robotiq>. Accessed: 2017-10-22.
- [3] Ros. <http://www.ros.org/>. Accessed: 2017-10-20.
- [4] Ros. <http://wiki.ros.org/ROS/Introduction>. Accessed: 2017-10-20.
- [5] universalrobot. http://wiki.ros.org/universal_robot. Accessed: 2017-10-22.
- [6] Michael Cashmore, Maria Fox, Derek Long, Daniele Magazzeni, Bram Ridder, Arnau Carrera, Narcis Palomeras, Natalia Hurtos, and Marc Carreras. Rosplan: Planning in the robot operating system. In *ICAPS*, pages 333–341, 2015.
- [7] Thomas H Cormen. *Introduction to algorithms*. MIT press, 2009.
- [8] Richard Fanson. *Robotic Manipulation of Deformable Objects Using Robust Output Regulation*. PhD thesis, 2010.
- [9] HC Holland. The archimedes spiral. *Nature*, 179(4556):432, 1957.
- [10] Anis Koubaa. *Robot operating system (ros): The complete reference*, volume 1. Springer, 2016.
- [11] John McCleary. *Geometry from a differentiable viewpoint*. Cambridge University Press, 2013.
- [12] Richard M Murray. *A mathematical introduction to robotic manipulation*. CRC press, 2017.
- [13] Jason M O’Kane. A gentle introduction to ros. 2014.
- [14] Robotiq. *Robotiq 3-Finger Adaptative Robot Gripper Instruction Manual*. Robotiq, 2014.
- [15] Robotiq. *Robotiq Force Torque Sensor FT 150/300 Instruction Manual*. Robotiq, 2016.
- [16] Universal Robots. *UR10/CB3 User Manual*. Universal Robots, 2014.
- [17] Mitul Saha and Pekka Isto. Manipulation planning for deformable linear objects. *IEEE Transactions on Robotics*, 23(6):1141–1150, 2007.

- [18] N.J.A. Sloane. "sequence a091154". In *The On-Line Encyclopedia of Integer Sequences*.
- [19] Eric W Weisstein. Pedal curve. 2003.
- [20] Shigang Yue and Dominik Henrich. Manipulating deformable linear objects: Force/torque sensor-based adjustment-motions for vibration elimination. 2001.
- [21] Shigang Yue and Dominik Henrich. Manipulating deformable linear objects: sensor-based fast manipulation during vibration. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 3, pages 2467–2472. IEEE, 2002.