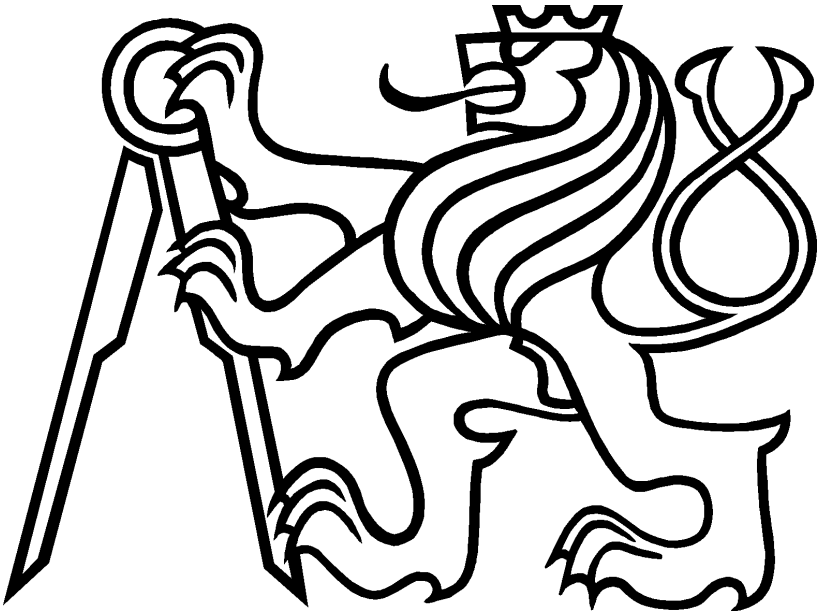
CZECH TECHNICAL UNIVERSITY IN PRAGUE

Faculty of Electrical Engineering

DIPLOMA THESIS



Tadeáš Lejsek

Dual-arm robot perceiving and manipulating soft objects – use cases

Department of Cybernetics

Thesis supervisor: **Václav Hlaváč**

Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

DIPLOMA THESIS ASSIGNMENT

Student: Bc. Tadeáš L e j s e k

Study programme: Cybernetics and Robotics

Specialisation: Robotics

Title of Diploma Thesis: Dual-Arm Robot Perceiving and Manipulating Soft Objects – Use Cases

Guidelines:

1. Familiarize yourself with the state-of-the-art in robotic manipulation with soft objects and the environment the CloPeMa testbed provides at FEL. Add on your own work in the compliant motion control.
2. Design and implement three scenarios (use cases), in which the dual-arm robot will perceive and manipulate soft objects, two of them be (1) tying/untying knots in the free space; (2) shooting from a slingshot. Find/design the third scenario. You can put more emphasis to one of the three selected scenarios.
3. Implement selected three scenarios on the CloPeMa testbed, evaluate it experimentally and document it.
4. Prepare demonstrations showing your work.

Bibliography/Sources:

[1] Spong, Mark W.; Hutchinson, Seth; Vidyasagar, M.: Robot modeling and control. Wiley, 2006, ISBN 978-0-471-64990-8.
[2] Stria J. et al.: Garment Perception and its Folding using a Dual-arm Robot, accepted to IROS 2014, Chicago, USA.

Diploma Thesis Supervisor: prof. Ing. Václav Hlaváč, CSc.

Valid until: the end of the winter semester of academic year 2015/2016

L.S.

doc. Dr. Ing. Jan Kybic
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, July 8, 2014

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Bc. Tadeáš L e j s e k

Studijní program: Kybernetika a robotika (magisterský)

Obor: Robotika

Název tématu: Dvojruký robot vnímající a manipulující s měkkými předměty – případové studie

Pokyny pro vypracování:

1. Seznamte se se stavem vědění v robotické manipulaci s měkkými objekty a experimentálním robotem projektu CloPeMa na FEL. Navažte na Vaši vlastní práci v oblasti řízení poddajného pohybu.
2. Navrhněte a implementujte tři scénáře (případové studie), v nichž dvojruký robot bude vnímat a manipulovat s měkkými objekty. Dva scénáře budou: (1) vázání/rozvazování uzlů ve volném prostoru; (2) střílení z praku. Sám navrhněte třetí scénář. Můžete jednomu ze scénářů věnovat více úsilí než zbylým dvěma.
3. Implementujte tři vybrané scénáře na experimentálním robotu CloPeMa testbed, vyzkoušejte je experimentálně a práci dokumentujte.
4. Připravte tři ukázky, které budou demonstrovat Vaši práci.

Seznam odborné literatury:

[1] Spong, Mark W.; Hutchinson, Seth; Vidyasagar, M.: Robot modeling and control. Wiley, 2006, ISBN 978-0-471-64990-8.
[2] Stria J. et al.: Garment Perception and its Folding using a Dual-arm Robot, accepted to IROS 2014, Chicago, USA.

Vedoucí diplomové práce: prof. Ing. Václav Hlaváč, CSc.

Platnost zadání: do konce zimního semestru 2015/2016

L.S.

doc. Dr. Ing. Jan Kybic
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 8. 7. 2014

Declaration

I hereby declare that I have completed this thesis independently and that I have listed all used information sources in accordance with Methodical instruction about ethical principles in the preparation of university theses.

In Prague on..... .....

Acknowledgements

I would like to thank my supervisor Václav Hlaváč. His advice given at our regular weekly meetings were highly valuable. His "top view" perspective shed new light on many aspects of writing the thesis. My deepest thanks belongs to my direct co-workers at the *CloPeMa* project Vladimír Petrík and Libor Wagner. They were always ready to thoroughly answer all of my questions. I have learned a lot from them. I would also like to thank Vladimír Smutný and Pavel Krsek. Vladimír Smutný has been my supervisor even during my high school years when I did several projects for the university. He also guided my pre-diploma work at the *CloPeMa* project.

I would like to thank Marcel who helped me to construct a better slingshot and my friends who participated in making the knot-tying videos. I would also like to thank Tereza Köppelová, a rhythmic gymnast, who showed me how to manipulate the ribbon. I am very thankful to Jéňa who can answer every imaginable question about image processing and who always had time for me and Ondra with whom I could always talk about programming in Python. I would also like to express my gratitude towards my fiancée Bára for her patience. I am very thankful to my parents who always encouraged me to study and even created the conditions that allowed me to study abroad. Last but not least, I would like to thank all of my friends and family members who were always interested in what I am doing and were always eager to see the new videos that I made.

Abstrakt

Diplomová práce zkoumá možnosti užití zpětné vazby, která napodobuje lidský zrak a hmat, a to v komplexních robotických manipulačních úlohách. Vypracoval jsem čtyři případové studie, které se zabývají manipulací s měkkými objekty pomocí dvourukého robotu *CloPeMa* jako je střelba z praku, vázání uzlu, chytání kývající se gymnastické tyče zavěšené na gymnastické stuze a přechytávání lana z jedné robotické ruky do druhé. Tyto čtyři případové studie obohatí souhrn manipulačních a percepčních schopností robotu *CloPeMa* v rámci projektu *CloPeMa* a budou užitečné i po jeho skončení.

Abstract

The thesis explores the possibilities of a various sensoric feedback loops that mimic the human visual and tactile sensing in complex robotic manipulation tasks. I developed four use cases that involve various soft object manipulation tasks with the two-arm *CloPeMa* robot such as shooting projectiles from a slingshot, tying an overhand knot, catching a swinging gymnastic pole and regrasping a piece of a rope from one robot arm to the another one. These four use cases will enrich the repository of sensing/manipulation skills in *CloPeMa* project and beyond it.

Contents

List of Figures

List of Tables

List of Appendices

1 Introduction

1.1 Motivation

Various robots and robotic production lines are used in industry and in other fields on a daily basis. However, in most cases, they are either teleoperated or programmed to repeat a previously learned motion. These robots have mostly a single arm. In case the robots and robotic applications are autonomous, the most widely used sense is the vision.

However, things are beginning to change. In the article titled "My boss the Robot" [1], David Bourne writes about the future of robots in industry. He points out that the time when robots operated in isolated cells repeating a certain motion over and over might soon change. Since humans and robots are best at different tasks (e.g. complex manipulation and precise welding), it would be very smart to efficiently combine their skills. For that reason, robots and humans would have to work side by side collaborating on a given task. In these cases, the robot has to use a variety of sensors to sense its human colleague. And this is just one of the points where different kinds of feedback other than visual come into play.

I personally find it very exciting to develop a new sense for the robots – the tactile sensing that mimics the human sense of touch. Since the time the visual feedback has been introduced, the robots have been able to accomplish many more tasks. The same promises the introduction of the tactile feedback. It involves tasks ranging from robotic surgeons up to automated manipulation with clothes.

The *CloPeMa* project aims at advancing the state-of-the-art in such areas. My work is focused at using the tactile, kinesthetic and visual feedback in different tasks such as automated knot-tying and demonstrating their functionality.

1.2 *CloPeMa* project context

The project context description is taken from a report that I created in the previous semester [2].

CloPeMa is a three year European Commission funded research project in the Framework Program 7, which advances the state of the art in the autonomous perception and manipulation of soft materials such as fabrics, textiles and garments. *CloPeMa* runs between February 2012 and January 2015. There are five partners cooperating in the project (from Greece, Italy, Scotland and two partners from the Czech Republic). *CloPeMa* testbed utilizes a dual-arm robot based on the industrial welding arm Motoman 1400. *CloPeMa* modules are integrated on top of Robotic Operating System (ROS).

In the run of *CloPeMa* project, a force compliant 13-DOF dual-arm robot formed by two industrial arms Motoman 1400 has been implemented and equipped with two off-the-shelf force/torque sensors (ATI-Mini45) in between the last joint and the gripper,

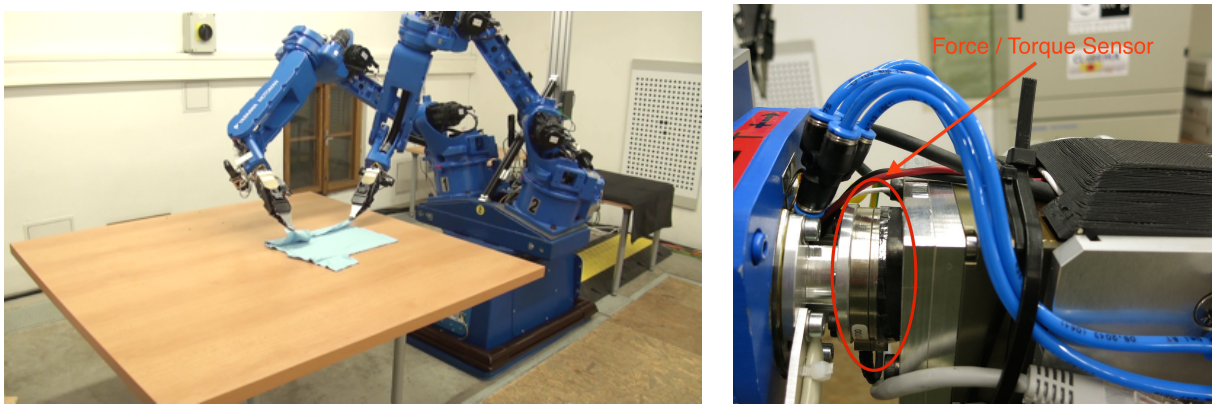


Figure 1: Left: *CloPeMa* project dual-arm robot manipulating a T-shirt. Right: The force / torque sensor Mini45 ATI mounted on *CloPeMa* robot.

see Figure 1, right side. Each arm hosts a custom build sensorized gripper with actively controlled compliance capable of rubbing motions of the fingers.

CloPeMa test-bed is rich in sensors. The sensor setup is formed by: the Kinect-like sensor ASUS Xtion on both forearms, the precise optical stereo on the central pole (formed by two SLR Nikon cameras); the gripper hosts a photometric stereo camera in the palm of the gripper, and is equipped with the 16-elements capacitive tactile sensor in the gripper finger. The view of the *CloPeMa* system folding a T-shirt is in Figure 1, left side.

Figure 2: Left: The *CloPeMa* hand with variable stiffness and tactile sensor. Right: Detail of the hand tip with the tactile sensor.

The gripper has a controllable impedance by involving hydraulics and air bubbles injected into the hydraulic oil. The hydraulic actuator is visible in Figure 2 left side. The gripper has a capacitive 16 texels tactile sensor in the upper finger in Figure 2, right side. The source of hydraulic energy is placed on the arm of the robot farther from the gripper, see Figure 3.

Figure 3: Gripper hydraulics controller box.

CloPeMa project inheritance is also in established integration procedures on top of Robotic Operating System (ROS), software design practices in C++ and Python, established project management support using Redmine tool, etc.

1.3 Goals of the thesis

Based on the given thesis assignment, I specified my own assignment for each of the given tasks more in detail.

1.3.1 Slingshot

The aim is to explore the possibilities of the compliant motion and the force feedback. The force sensor will be used mainly. The goal is to design an automated slingshot and to find out whether an automated procedure for hitting a standing target can be developed. I came up with the following scenario:

1. Develop an automated slingshot.

   (a) Initial position: one robot arm holds the slingshot, the other arm holds the projectile.
   (b) Load the projectile.
   (c) Stretch the elastic string of the slingshot so that it exerts the given force on the projectile.
   (d) Shoot the projectile at a given shooting angle.

2. Compare the theoretical analysis of the shooting (i.e. which point should be hit by the projectile based on its initial speed, shooting angle etc.) with the actual experiments.

1.3.2 Knot-tying

The goal is to tie an overhand knot in the air using the two-arm robot. Various sensors might be used, for instance Xtion kinect and the force sensor.

I came up with the following solution to the knot tying problem. The robot is already holding the rope in both grippers at the beginning of the proposed procedure.

1. Wrap the rope around the left arm in a way that a loop is created.
2. Release the rope end with the right arm and move the right arm away.
3. Turn the left arm so that the left camera sees the loop and the rope end.
4. Detect the rope end and catch it with the right arm.
5. Tighten the knot using both arms.

1.3.3 Ribbon manipulation

I chose a manipulation with the rhythmic gymnastic ribbon and pole to be the third task. The aim of it is to mimic a few capabilities of a human rhythmic gymnast. I am especially interested in tasks that involve dynamics and in investigating the capabilities of the robot grippers.

I proposed a following scenario. At the beginning, the robot is holding the pole hanging on a ribbon in its left gripper.

1. Swing the left arm in a way that the pole and ribbon start to swing as well.
2. Catch the pole with the right gripper using one of the many sensors that are present in the gripper.
3. Make sure the gripper holds the pole well (and that it was caught) and release the ribbon with the left arm.

4. Perform a few fast moves with the right arm so that the ribbon forms a certain shape, which is nice to watch.

1.3.4 Regrasping a rope

The last use case deals with the regrasping of a piece of a rope from one gripper to the other one. The light or proximity sensor should be used to detect the presence of the rope in the gripper. This section should demonstrate that it is indeed possible to swap the functions of both robot arms.

The proposed scenario is the following. The left gripper holds one end of the rope. The right gripper is open and is positioned towards the left gripper.

1. The left arm starts to move towards the right gripper.
2. When the presence of the rope is detected inside the right gripper, the motion of the left arm is stopped.
3. The right gripper closes, thus holding the other end of the rope.
4. The left gripper opens and releases the rope.
5. The left arm moves back and the functions of both arms are swapped.
6. The whole procedure is repeated again. The right arm now moves towards the left one that catches the rope.

1.4 Thesis organization and naming robot arms

The thesis is organized into seven bigger sections – one section describing the state-of-the-art, four sections for each of the given tasks, one section that describes the implementation and the last section provides conclusions. Each task description contains all the related information – assignment, analysis, experiments and discussion.

The two arms of the *CloPeMa* robot are called *R1* and *R2*. *R1* is the right arm and *R2* is the left arm from the view of the robot. This naming convention is then used further on in the text.

2 State-of-the-art

2.1 General

A tactile feedback is useful for perceiving and manipulating soft objects. Its usage dates back to at least 1987 when Stephen Buckley, USA published his PhD on planning and teaching compliant motion strategies [3]. He was for instance interested in scenarios, in which a robot holds a T-shaped piece that has to be inserted into a corresponding hole. He worked with a robotic arm capable of Cartesian motions, but not rotations. The robot arm was modeled as a damped spring. A method how to teach the robot the wanted motions was developed.

Another example of the usage of the tactile feedback is a task where a human guides a robotic arm by hand. It was implemented at Weiss Robotics, Germany [4] (see Figure 4, left side). Such scenario might be very useful in industry and other areas. A human can teach the robot a certain motion quickly without the need to specifically program it.

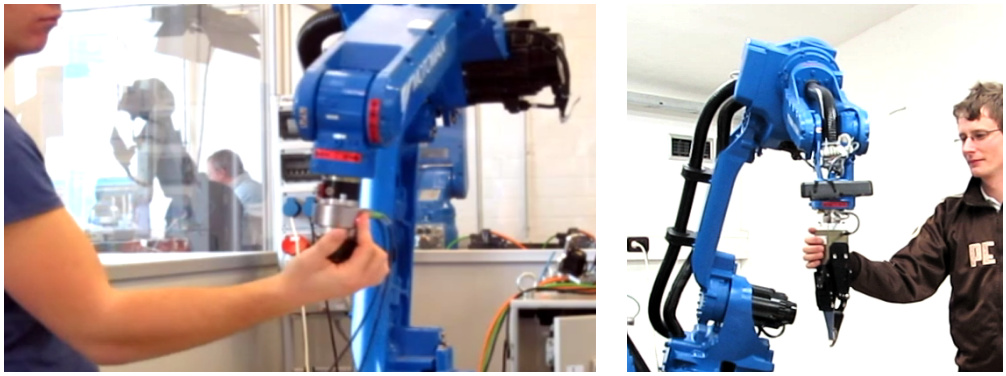


Figure 4: Left: Manually guiding the robot arm by hand at Weiss Robotics. Right: Manually guiding the robot arm by hand at *CloPeMa*.

In case the force sensor is not placed at the tip of the robot arm, the weight of everything that is placed below the force sensor has to be subtracted in order to get a correct force/torque measurement. This procedure of separating the weight caused by inertia from the influences of the environment (e.g. a human hand) is described in [5]. The same procedure is used in case of the *CloPeMa* robot, where force/torque sensors are placed in both wrists [6]. Together with my college Jan Kubeš, we implemented the manual guidance of the robot arm at the *CloPeMa* robot [2]. The experiment was recorded on video [7] and is shown in Figure 4, right side.

Another field where a tactile/haptic feedback is used is surgery. A robotic simulator using virtual force feedback was designed at Intelligent Robotics Institute, China [8]. A human surgeon should already be well trained when performing a complicated maxillofacial surgery. For this reason, a platform featuring a 6-DOF robotic arm, haptic and visual feedback was developed to help to train young surgeons.

A field in which the tactile feedback would be beneficial is the autonomous garment folding (see Figure 5). The present solution developed in the course of the *CloPeMa* project predominantly utilizes the visual feedback. The usage of the tactile feedback in *CloPeMa* testbed is planned for the future [9].

Figure 5: The *CloPeMa* robot folding a T-shirt.

A compliant robot arm can also be made cheaply. The reduction of price was achieved through several choices such as not using an expensive robot head and using stepper motors [10]. The arm is capable of playing chess (via teleoperation) and making pancakes (see Figure 6).

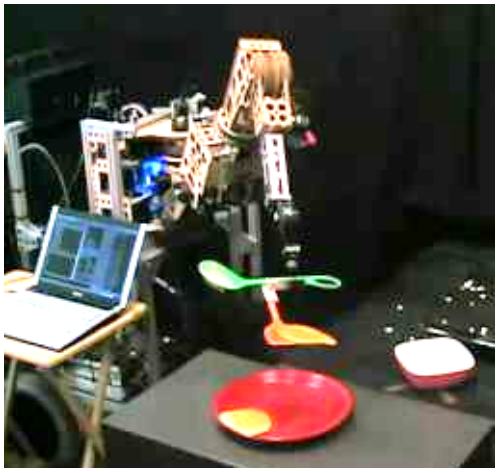


Figure 6: The 7-DOF compliant robot arm making pancakes.

2.2 Slingshot

As far as I know, nobody else has used a two-arm robot for experimenting with a slingshot. However, other scenarios have been explored.

One example is the "David and Goliath" slingshot (see Figure 7). The projectile (a stone) is thrown using a rotating slingshot. The "robotic" realization was implemented at University of Pisa, Italy, 2011. It can be seen in the video [11].

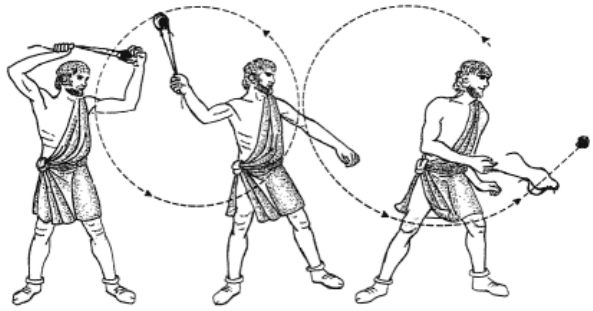


Figure 7: The principle of the rotating slingshot.

Another example is the "Trebuchet" scenario. This principle was used in siege machines in antiquity. Its realization in robotics is shown in [12] made by Lake Area Technical Institute (USA) students.

Finally, another scenario uses the principle of two fast rotating disk. When a projectile (in this case a ball) gets in between them, it is shot forwards. The realization was done by Team 5353, Fremont, USA [13].

2.3 Knot-tying

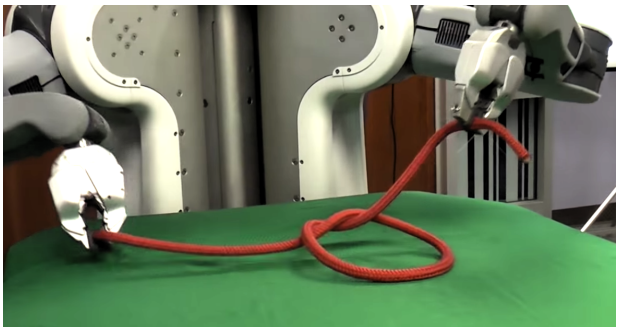


Figure 8: The PR2 robot tightening a knot at UC Berkeley.

Knot tying was done for instance in UC Berkeley, USA. They used the Willow Garage PR2 robot and their working scenario was the following. A rope was lying on the table, the robot sensed it, planed how to tie the knot and finally made a simple overhand knot while the rope was still lying on the table (see Figure 8). The whole process involves only one arm of the two-arm robot. Both arms were used only at the end of the whole procedure to

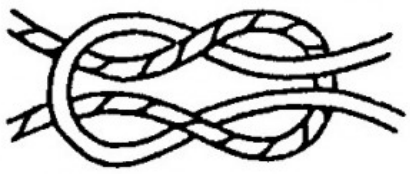


Figure 9: The suture knot.

tighten the knot. The rope was segmented based on its color. The robot was taught by a human guidance beforehand [14]. The whole procedure is shown in the video [15].

Another working scenario is to entirely omit the need to regrasp the string by using so called fixtures (knot boxes). Such work was done at Darthmouth College, Hannover, Germany [16].

A great benefit from autonomous knot tying would be in Minimally Invasive Surgery (MIS). Long Short-Term Memory neural networks were trained using supervised learning to autonomously tie suture knots on a surgical robot (see Figure 9) [17]. This work was done in cooperation of Technical University Munich, Germany and Istituto Dalle Molle di Studi sull'Intelligenza Artificiale (IDSIA), Switzerland.

2.4 Ribbon manipulation

To the best of my knowledge, nobody has used a two-arm robot for a manipulation with a modern gymnastic ribbon.

Figure 10: The Rollin' Justin Robot catching a ball.

However, scenarios where a robotic arm catches a flying object, e.g. a ball are being researched as this setup is used as a benchmark for key robotic technologies. An example

is the work done at DLR Institute for Planetary Research, Germany. A high speed 7-DOF robotic arm is used for ball catching. The whole system has to meet hard real-time operation deadlines and thus a lot of computational power is needed (a cluster with 32 CPU cores). The catch rate is reported to be $> 80\%$, the visual sensing being the weakest point of the whole system [18]. The arm is a part of Rollin' Justin Robot [19] (see Figure 10).

Fast and repeatable motions are needed in the industry. The example of that is a demo made by ABB robotics [20].

2.5 Grasping objects

Robotic object grasping is a complex topic. It was studied for instance at Carnegie Mellon University, Pennsylvania, USA [21]. The authors showed in 2011 that in order to develop a robust grasping of relatively small objects such as pens, screwdriver, cellphones and hammers, the robotic fingers should come in touch and make use of the supporting surface.

A Mobile Manipulator that is able to autonomously fetch an object lying on a flat surface was developed in 2010 at Georgia Institute of Technology [22]. The aim is to improve the everyday life of elderly, injured or disabled people (see Figure 11). A tilting laser range finder is used to acquire 3D point cloud data around the surroundings of the object to be fetched.

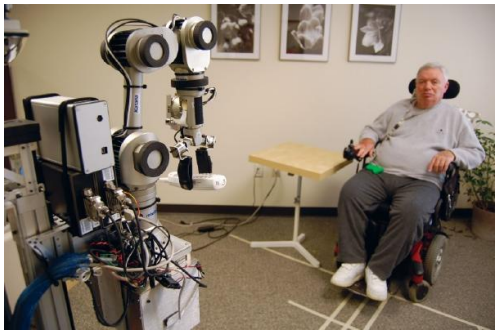


Figure 11: The mobile manipulator, EL-E, delivering an object to a patient.

3 Slingshot

3.1 Assignment

The work performed in this thesis follows closely the work, which I did in the subject called "Individual project" [2], where I focused on exploring the possibilities of a robotic slingshot. Here I continue with developing a fully automated slingshot.

The goal is to implement the following shooting procedure. The right arm holds the projectile and the slingshot is attached to the left one.

1. Load the projectile. Use force feedback to determine that the elastic string has been touched.
2. Stretch the elastic string of the slingshot by moving the attached projectile using the right arm. Stop when the exerted force exceeds a predefined threshold.
3. Adjust the shooting angle by the left arm. Shoot the projectile.

3.2 Analysis

3.2.1 Attaching the slingshot to the robot arm

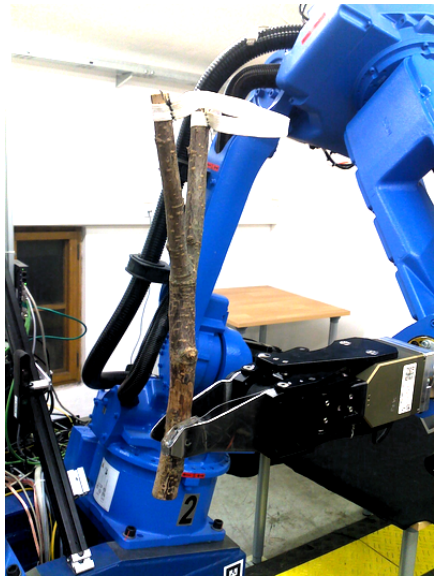


Figure 12: The traditional slingshot.

There is one major difference to my previous work [2], though. The gripper of the *CloPeMa* robot has been changed in the meantime. It is not possible to attach the elastic string directly to the gripper, because the projectile could damage the sensors in the new

gripper seriously. Therefore, I had go back to the original idea of the "traditional slingshot" (see Figure 12).

The slingshot cannot be held in the gripper. If just a small force is exerted on the elastic string, the gripper is not able to hold the slingshot anymore. Thus, the slingshot has to be attached directly to the robotic arm (see Figure 13). I did it using velcro. It is then easy to attach and remove the slingshot within a few moments. The only disadvantage is that the slingshot might move slightly, if a high force is exerted on it. However, to completely avoid this problem, the slingshot would have to be designed differently.

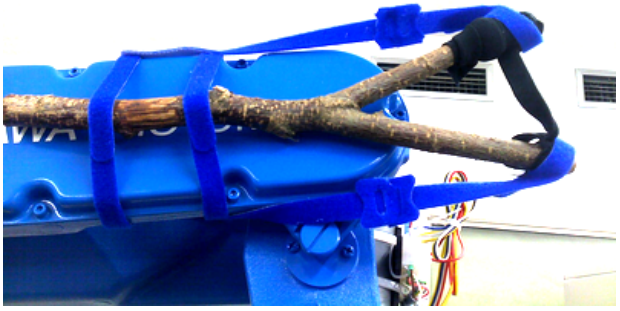


Figure 13: The attached slingshot.

3.2.2 Workflow of the shooting procedure

The procedure for shooting projectiles using the slingshot is described by the finite state machine that is shown in Figure 14.

The flow chart connected with the shooting process is shown in Figure 15.

Insert the projectile
The projectile is inserted into the gripper manually.

A little trick is used to simplify this procedure for the operator. Normally, the operator presses a button to close the gripper. But when the operator enters the safety cage with the robot to insert the projectile, it is hard to reach the keyboard. Therefore, the gripper closes, when the force exerted on it exceeds a certain threshold. Thus, no keyboard is needed to close the gripper after the projectile has been inserted (see Figure 16).

Shooting position (A)
The right arm of the robot moves next to the slingshot.

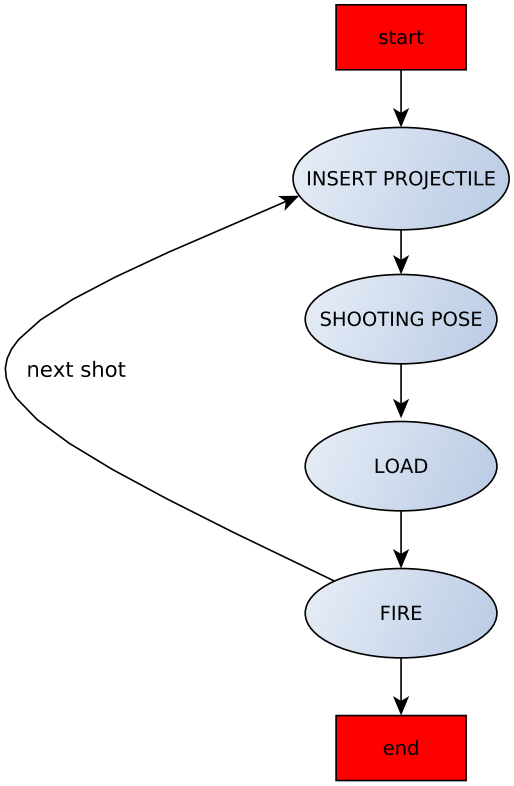


Figure 14: Slingshot workflow.

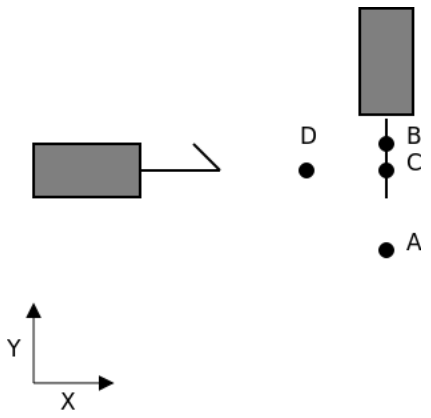


Figure 15: Slingshot process diagram – the top view.

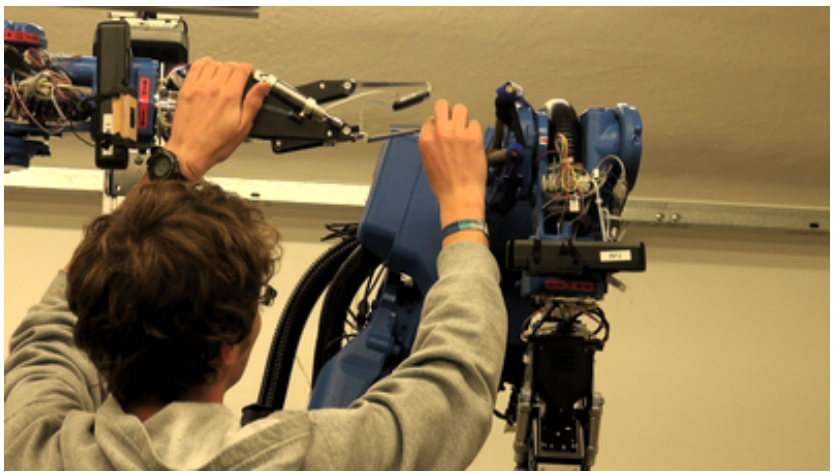


Figure 16: Inserting the projectile.

Load (B, C)

The right arm holding the projectile moves towards the elastic string. The movement stops, when the projectile touches the string (B). It is detected by the force sensor in the right arm. When the force exceeds a certain threshold force F_B, the movement is stopped. The arm then moves a bit back (C).

Fire (D)

The right arm moves backwards (D) until the desired force F_D is reached. Then the gripper opens and the projectile is released.

3.2.3 Force/torque sensor

In order to implement projectile insertion, loading and firing, a force feedback is needed. There are two ATI Mini 45 force sensors placed in the wrist of each arm. The force sensor measures the force and torque simultaneously in three axes. Thus, it provides $F_{x,y,z}$ and $T_{x,y,z}$ at the maximum frequency of 7.000 Hz. After the filtration of the outside disturbances (such as trams and light disturbances) and subtracting the weight of the gripper, the sensor provides the meaningful data at the frequency of 12.5 Hz [6]. I use the force measurements only.

3.3 Experimental setting

3.3.1 Shooting scene

Figure 17 shows two settings of the shooting with the slingshot. The left diagram shows the horizontal shooting, the right one the shooting upwards. The shooting parameters were $z_0 = 1.9\,\mathrm{m}$, $d = 1.85\,\mathrm{m}$.

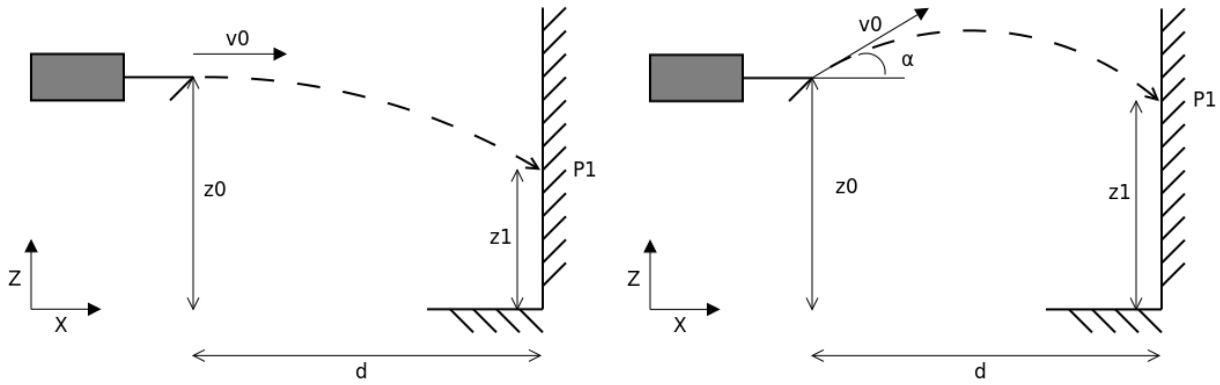


Figure 17: Left: Shooting horizontally. Right: Shooting upwards.

3.3.2 Projectile

A bent piece of wire was chosen to be the projectile (see Figure 18) for the following reasons. Firstly, the gripper can hold it, even if a high force is exerted on it. Secondly, it is light and finally, because it has a low air resistance.

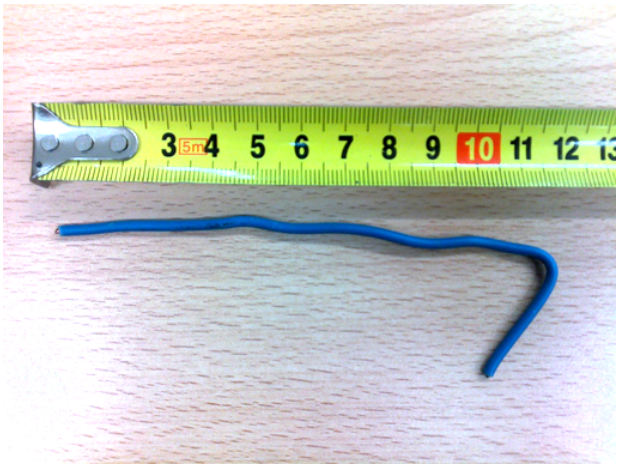


Figure 18: The projectile.

The weight of the projectile m was measured to be in the range 1 to 2 g.

When the projectile is shot, it leaves the slingshot with the initial velocity v_0. Its magnitude is given by Equation (2) and is derived assuming that the potential energy of the elastic string E_p is fully converted into the kinetic energy of the projectile E_k. The diagram connected to shooting the projectile is shown in Figure 19.

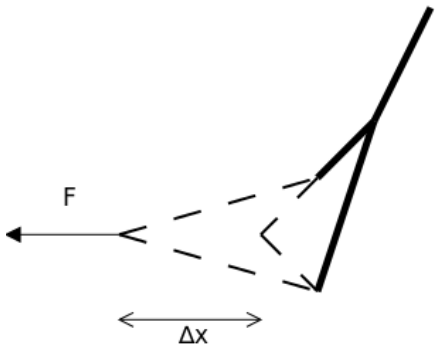


Figure 19: Shooting the projectile.

$$E_p = E_k \tag{1}$$

$$\frac{1}{2}F\Delta x = \frac{1}{2}mv_0^2$$

$$v_0 = \sqrt{\frac{F\Delta x}{m}} \tag{2}$$

The distance Δx is measured from the moment the force sensor detects a non-zero force until the moment when the desired force is reached.

3.4 Implementation

3.4.1 Force feedback

A key task is to use the force feedback. Force thresholds F_B and F_D are used to detect hitting the elastic string (B) and to detect reaching the firing position, respectively. In addition, the force threshold F_{D0} is used to detect that the elastic string starts to be stretched. This information is used to compute the distance Δx.

3.4.2 Traditional slingshot

The slingshot script version 1.0 is called the shooter. The usage of the force feedback is implemented incrementally. It means that the arm moves a certain distance (e.g. 1 cm) and then the force data is retrieved and checked whether it exceeds a certain force threshold. If not, another step is performed and the whole procedure is repeated. It results in a bit jerky movement of the robotic arm.

3.5 Mathematical description

The assumptions for the following mathematical description are the following.

1. The projectile is considered to be a point mass.
2. The projectile has no air resistance.
3. It is assumed that the motion takes place in a homogeneous gravitational field.

3.5.1 Shooting horizontally

Using the assumptions 1 and 2, the kinematics of the horizontal shooting is given by the two following Equations (3) and (4). It is connected with the left side of Figure 17.

$$x = v_0 t \tag{3}$$

$$z = z_0 - \frac{1}{2} g t^2 \tag{4}$$

The time t_1, in which the projectile hits the point $P_1 = [x_1, z_1] = [d, z_1]$ on the opposite wall, is given by:

$$t_1 = \frac{d}{v_0} \tag{5}$$

The z-coordinate z_1 is then:

$$z_1 = z_0 - \frac{1}{2} g t_1^2 = z_0 - \frac{g d^2}{2 v_0^2} \tag{6}$$

3.5.2 Shooting upwards

The setting of the shooting upwards corresponds to the right side of the Figure 17. Its kinematics is given by Equations (7) and (8).

$$x = v_0 \, t \, \cos\alpha \tag{7}$$

$$z = z_0 + v_0 \, t \, \sin\alpha - \frac{1}{2} g t^2 \tag{8}$$

The time t_1, in which the projectile hits the point $P_1 = [d, z_1]$ on the opposite wall, is given by:

$$t_1 = \frac{d}{v_0 \cos\alpha} \tag{9}$$

The z-coordinate z_1 is then:

$$z_1 = z_0 + v_0\, t_1 \sin\alpha - \frac{1}{2} g t_1^2 = z_0 + d \tan\alpha - \frac{1}{2} g t_1^2 \tag{10}$$

3.6 Experiments with the traditional slingshot

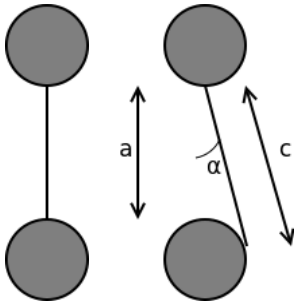


Figure 20: Left: Straight elastic string ($\alpha = 0$). Right: Bent elastic string ($\alpha > 0$).

3.6.1 Shooting horizontally

The first shooting experiment was conducted with a completely straight tightened elastic string. Thus, $\alpha = 0$ and the projectile was shot horizontally. Figure 20 left shows the side view of the slingshot.

The projectile was shot in the following setting: $F_1 = 4.4\,\mathrm{N}$, $\Delta x = 0.05\,\mathrm{m}$. The shot hit the wall at the height $z_1 = 0.5\,\mathrm{m}$. Thus, the initial velocity v_{01d} was $10.49\,\mathrm{ms}^{-1}$ according to Equation (2).

It is possible to estimate the initial velocity from the point $P_1 = [d, z_1]$. Using Equations (3) and (4), we obtain:

$$v_0 = d \sqrt{\frac{g}{2(z_0 - z_1)}} \tag{11}$$

In our case, $v_{01k} = 3.46\,\mathrm{ms}^{-1}$.

Thus, there is a substantial difference (roughly three times) between the initial velocity v_{01d} (calculation based on the parameters of the elastic string and its dynamics) and the initial velocity v_{01k} (calculation based on the kinematics of the horizontal shooting). This suggests that the theoretical model does not fit with the reality.

α [^{o}]	z_1 [cm]
5	105
10	120
15	132
20	144

Table 1: Change in α.

Δx [cm]	z_1 [cm]
5	78
6	102
7	120
8	132
9	142

Table 2: Change in Δx.

3.6.2 Shooting upwards

To make the shooting look as shooting and not as a "piece of wire falling down", α was set to be higher than zero. I accomplished this by moving the elastic string a bit to the side (see Figure 17 right). Parameters $a = 6\,\text{cm}$ and $c = 6.1\,\text{cm}$ yield:

$$\alpha = \arccos\left(\frac{a}{c}\right) \approx 10^o \tag{12}$$

Given F_1 and Δx, the z-coordinate z_1 of the point P_1 can be computed using Equation (10)). However, the theoretical calculation is very sensitive to the change in α and Δx. Table 1 shows how the change in α (given $\Delta x = 7\,\text{cm}$ and $F_1 = 4.8\,\text{N}$) influences z_1. Table 2 shows how the change in Δx (given $\alpha = 10^o$ and $F_1 = 4.8\,\text{N}$) influences z_1.

The actual Δx_1 and F_1 were measured to be $\Delta x_{1m} = 5\,\text{cm}$ and $F_{1m} = 4.8\,\text{N}$. The shots were oscillating around the height $z_{1m} = 150\,\text{cm}$.

The theoretical z_{1t} corresponding to the measured Δx_{1m} and F_{1m} is $z_{1t} = 78\,\text{cm}$.

To achieve the measured z_{1m} given F_{1m}, α and Δx would have to be e.g. $\alpha_t = 25°$ and $\Delta x_{1t} = 7\,\text{cm}$.

A video called *TraditionalSlingshot.mpg* was taken that shows the whole shooting procedure. It can be found on the attached CD.

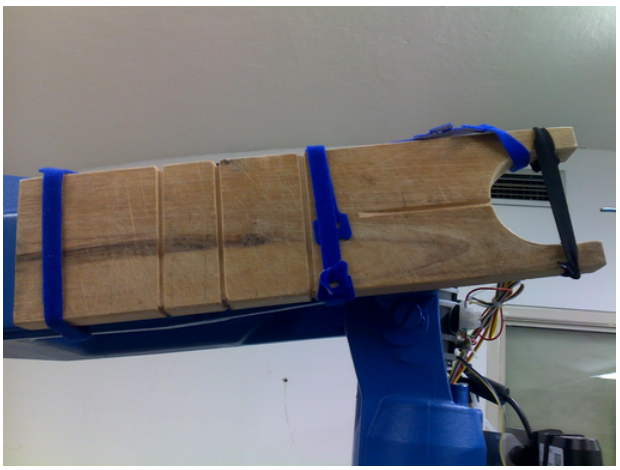


Figure 21: The flat slingshot.

3.7 Experiments with the flat slingshot

The previous experiments with the traditional slingshot had one problem. It was not possible to attach the traditional slingshot to the robot, so that it does not move while shooting. This behavior means that it is not possible to measure the shooting angle α properly and thus it is not possible to compare the measured results with the theoretical analysis.

Therefore, a new slingshot model was introduced – "flat slingshot" (see Figure 21). It is made from plank wood. Its sides are flat and thus it is possible to attach it tightly to the arm of the robot.

3.7.1 Changes to the traditional slingshot

The shooting procedure was changed a bit as well compared to the original one, see Section 3.2.2. The insertion of the projectile and moving to the shooting position was flipped.

Secondly, the implementation of loading and firing was changed, see Section 3.4.2, as well. The motion of the robotic arm is no longer jerky as it was before. Now the arm does not move incrementally in 1 cm steps and does not check the measured force value only after one step was performed. The smooth movement was achieved. However, the speed of the robot is slowed down ten times and the movement and force checking for threshold values are performed simultaneously. When the desired stopping threshold is reached, i.e. the projectile touches the elastic string, the movement is stopped immediately.

Figure 22 shows the different states of the shooting procedure.

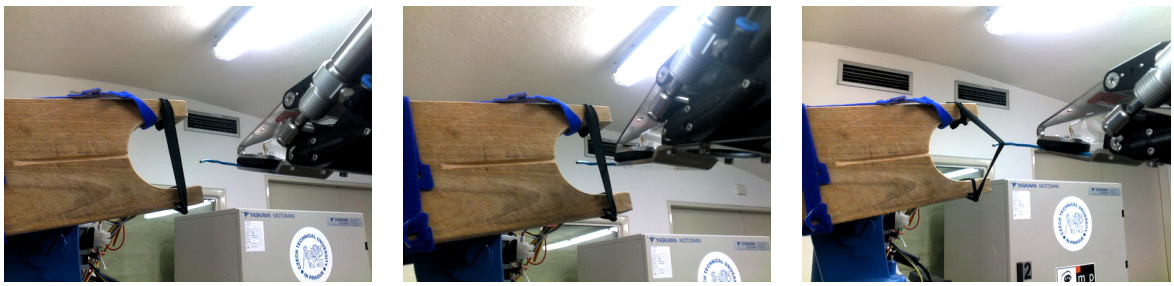


Figure 22: Left: Shooting position. Middle: Loaded. Right: Ready to fire.

Trial	z [m]	F [N]	Δx [m]
1	1.45	4.30	0.048
2	1.35	3.95	0.070
3	1.40	3.91	0.046
4	1.50	4.08	0.046
5	1.55	4.26	0.048

Table 3: Shooting 1: $\alpha = 0°$.

3.7.2 Shooting at different shooting angles

The three following experiment were performed with the flat slingshot. Five shots were fired for different shooting angles $\alpha_1 = 0°$, $\alpha_2 = 10°$ and $\alpha_2 = 20°$. The height z, shooting force F and Δx were measured. Table 3, 4 and 5 show the measured results. The shooting force F_0 was set to 3.7 N.

Trial	z [m]	F [N]	Δx [m]
1	1.70	4.98	0.049
2	1.65	4.40	0.049
3	1.70	4.12	0.052
4	1.75	4.45	0.047
5	1.65	4.26	0.050

Table 4: Shooting 2: $\alpha = 10°$.

Trial	z [m]	F [N]	Δx [m]
1	1.65	4.37	0.050
2	1.65	4.43	0.047
3	1.70	4.37	0.049
4	1.65	4.47	0.073
5	1.70	3.94	0.049

Table 5: Shooting 3: $\alpha = 20°$.

Trial	z_m [m]	z_t [m]	Δz [m]
1	1.45	1.09	0.36
2	1.45	1.29	0.06
3	1.40	0.97	0.43
4	1.50	1.00	0.49
5	1.55	1.08	0.47

Table 6: Shooting 1 comparison ($\alpha = 0°$).

3.7.3 Comparison with the theoretically calculated values

For each trial, a theoretical height z_t was computed according to Equations (6) and (10). Then a difference $\Delta z = z_m - zt$ was computed for each trial. Tables 6, 7, 8 show the obtained results.

The results suggest that Trial 2 and Trial 4 in Shooting 1 and Shooting 3 are badly measured since Δx is far off the other Δx measured. These values were omitted in the following analysis.

Finally, the means μ and standard deviations σ were computed for the differences Δz. Table 9 shows the results.

3.8 Discussion

3.8.1 Traditional slingshot

The obtained results suggest that the theoretical model does not fit with the actual experiments. However, since it is not possible to attach the traditional slingshot tightly to the robotic arm, there is not much that can be done about it.

Trial	z_m [m]	z_t [m]	Δz [m]
1	1.70	1.51	0.18
2	1.65	1.42	0.23
3	1.70	1.42	0.28
4	1.75	1.40	0.35
5	1.65	1.48	0.17

Table 7: Shooting 2 comparison ($\alpha = 10°$).

Trial	z_m [m]	z_t [m]	Δz [m]
1	1.65	1.70	0.05
2	1.65	1.66	-0.01
3	1.70	1.69	0.01
4	1.65	1.99	-0.34
5	1.70	1.59	0.11

Table 8: Shooting 3 comparison ($\alpha = 20°$).

α^o	μ [m]	σ [m]
0	0.44	0.06
10	0.24	0.07
20	0.02	0.07

Table 9: Statistical analysis of Δz.

However, the experiments show that a certain repeatability is indeed reachable. Out of six shots fired with a given force and shooting angle, three hit the target hanging on the wall. I recorded the whole experiment on video called *TraditionalSlingshot.mpg* that can be found on the attached CD.

3.8.2 Flat slingshot

The results suggest that there is a systematic error difference between the theoretical computed heights z_t and actually measured values z_m. As the shooting angle α was gradually increased, the difference between the theoretical and measured values decreased to values close to zero.

The reason could be the fact that the projectile is not held "horizontally" in the left gripper, but slightly upwards. To compensate that the way the shooting angle is adjusted would have to change. It seems that it is not enough that the shooting angle is controlled only by the rotation of the left arm. The right one would have to turn accordingly as well. Another possibility would be to redesign the slingshot again. Instead of having the elastic string tightened vertically, it could be placed horizontally. The shooting angle would be then controlled solely by changing the position of the right arm.

4 Knot tying

4.1 Assignment

The work done on knot tying follows the work performed in UC Berkeley, USA. I also follow up on my own work performed in the previous semester [2].

The goal is to be able to tie a simple overhand knot (see Figure 23) in the air. The procedure should be fully automated.

Figure 23: The over hand knot.

My working scenario is the following. The knot-tying process starts from a position when the robot holds the rope in both arms in the air. The knot is then made while still holding the rope the whole time in the air, i.e. it is not allowed to place the rope on a table. The robot thus has to release one end of the rope at a certain point of the tying process and has to "recatch" it again.

To figure out the movements of the robot to tie an over hand knot, I decided to follow a human example. I simply imagined how I would tie a knot if I could only use two fingers (the gripper of the robot only has "two fingers"). I came up with the following solution that starts from a position where the robot holds the rope in both arms.

1. Wrap the rope around the left arm in a way that a loop is created.
2. With the right arm release the rope end and move the right arm away.
3. Turn the left arm so that the right camera sees the loop and the rope end.
4. Detect the rope end and catch it with the right arm.
5. Tighten the knot using both arms.

Figure 24: Available ropes.

Rope	color	type	diameter ϕ [mm]
R1	red	twisted	14
R2	white	twisted	19
R3	white	braided	19
R4	blue	kernmantled	23
R5	blue	twisted	26
R6	orange	flat	1.4

Table 10: Available ropes.

4.2 Experimental setting

4.2.1 Available Ropes

I bought six 2 m long rope segments of different colors made from different materials (see Figure 24). The ropes are numbered from left to right. Table 10 shows their properties. The *twisted* rope is made of a three strands, the *braided* rope has a braided tubular jacket over strands of fiber and the *kernmantled* rope has a kern covered with a braided tubular jacket.

4.2.2 Examining the Asus Kinect sensor

Kinect sensor provides the point cloud, i.e. the distance measurements from the observer's point of view. Kinect sends messages at an average rate of 4 Hz. The message is an array of size 640 x 480 x 6. It resembles an image that contains information about color

(RGB) and x, y and z-coordinate of the given point in every pixel. Thus a depth image is obtained from the point cloud just by taking the z-coordinate. Furthermore, if a certain point is found in the RGB image (e.g. rope end), its coordinates in the camera coordinate system are known immediately.

4.3 Theoretical analysis

4.3.1 Used sensors

To be able to follow the tying procedure (see Section 4.1), several sensors have to be used. In order to detect the rope end visually, the Kinect sensor is needed. To fasten the knot, the force feedback from the force/torque sensor is needed.

4.3.2 Rope requirements

The rope should be rather thick. Firstly, it is then easier to detect it in the depth image provided by the Kinect sensor. Secondly, it is less probable that a thicker rope gets stuck on the left arm while fastening the knot. On the other hand, the rope should not be thicker than 42 mm which is the maximum pitch of the robot gripper.

4.4 Learning from a human example

4.4.1 Task assignment

In order to test the theoretical solution proposed in Section 4.1, I asked two subjects (A – sees well, B – is blind) to tie an overhand knot on a rope. The hypothesis is that both subjects would use fingers to tie a knot at first. When only two fingers are allowed, they would come up with a similar solution as I did – wrapping the rope around the second arm. Subject B would primarily rely on the tactile feedback to recatch the rope while subject A would primarily rely on the visual feedback. I also hypothesized that subject B would spend a lot more time analyzing the properties of the rope, e.g. its stiffness. To simulate the robotic hands even better, two scenarios will be used. In the first one, the subjects could use two fingers only and in the second one they will have to use pliers to manipulate with the rope. The following list summarizes all aforementioned scenarios.

1. Tie the knot as you like, i.e. you can use fingers.
2. Tie the knot using the pointing and the middle fingers only.
3. Tie the knot using pliers.

4.4.2 Observing how a normally seeing and a blind person ties a knot

The whole experiment was recorded on a video called *SubjectAB.mpg* that can be found on the attached CD.

All fingers allowed, 1

Both subjects had no problems tying the knot if they could use all fingers normally. In fact, they mostly used fingers to tie the knot. Subject B was a bit slower than subject A. To fasten the knot at the end, subject B tied it in series of movements checking how tight the knot is in every step.

Two fingers only, 2

Both subjects had considerable problems to tie the knot. Subject B was not able to succeed at all, while subject A came up with a very complicated solution.

In the end, I had to change the original plan and show them my solution and asked them to repeat it. Subject A was able to follow it after I visually showed him how to do it. However, subject B had to be taught by a direct guidance, hand by hand.

Subject B had problems making the loop around the second arm. Sometimes he created no loop at all and thus was unable to tie the knot. Even if he succeeded in making the loop, it was uneasy for him to find the rope end using only tactile feedback (see Figure 25, the right side). He sometimes did not catch the rope end, but the loop instead. However, he was able to correctly distinguish them by trial and error.

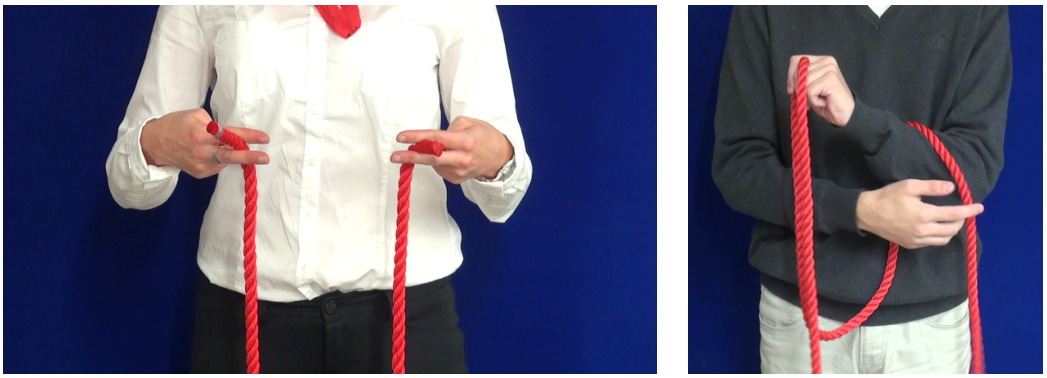


Figure 25: Left: Subject A holds the rope with two fingers. Right: Subject B caught the rope end.

When tightening the knot, the rope sometimes got stuck on the left arm of subject B. It required an extra effort to make the loop slide down his forearm.

Pliers, 3

Subject A had no problems to tie the knot using pliers. As hypothesized, it was simple to recatch the rope.

Subject B surprised me a lot. I hypothesized that since he had no tactile feedback from the pliers, he would be unable to tie the knot at all. However, he switched from recatching the rope end to recatching the pliers holding the rope end. He used his second arm to press the pliers against his body to prevent them from falling down (see Figure 26).

Figure 26: Subject B made the knot using pliers.

4.4.3 Discussing the observed behavior

As expected, Subject A was generally faster in tying the knot than Subject B. However, subject B was able to came up with solutions that I could not imagine. It proved that subject B would not be replaceable by a normally seeing person who would only close his eyes.

Both subjects had a lot more problems to tie the knot using two fingers only than using pliers. I think it is because of the fact that they both already worked with pliers before and thus were accustomed to using them. However, to work with two fingers only was something completely new to them.

Subject B refrained from long and fast movements especially when tightening the knot. I think it is because he could have hit an unseen obstacle. In this way, he protected himself.

Finally, I had to mention that the 2 m long rope segment is well suited for the *CloPeMa* robot, but it is too long to be worked with for a human.

4.4.4 Implications for the robotic knot-tying

The *CloPeMa* robot is in a way similar to Subject B. Although it can use visual feedback, it is slow, because it takes a lot of time to process it.

Based on the observation of the behavior of Subject B, I came up with the following guidelines for tying the knot programmatically.

- Make the loop around the left arm big enough so that there is enough space later to recognize and recatch the rope end correctly.
- Use visual feedback to detect the rope end as it is much faster and more reliable than the tactile feedback only.
- While tightening the knot, the left arm should move in a way to help the loop slide down and not get stuck.

4.5 Finding rope end

4.5.1 Segmentation methods

The main task of the knot-tying procedure is to find the previously released rope end again. I decided to use the visual information for this purpose.

There are basically a few basic detection approaches how to segment the rope and subsequently find its end:

1. Background subtraction;
2. RGB based;
3. Depth image based.

The Case 1 requires that the background does not change while tying the knot. First, a model of the background is made from several images of the background. Only afterwards is the rope moved against it, a new image is taken and the segmentation starts. The condition that the background does not change holds in the case if the rope is wrapped around the left arm and the right one then tries to catch the rope again. However, this condition will not hold if the arms are used vice versa. In that case, the rope would hang against the safety door of the robotic cell, so there could be a change in the background when for instance a person comes in. This method was rejected because it is not general enough.

The Case 2 uses the color of the rope to segment it. However, I have a few ropes in different colors at disposal. Thus, a human would have to specify the color of the rope before the start of the knot tying. Furthermore, some ropes are white and it is obviously hard to segment such ropes against a white wall. For these reasons, this approach was rejected as well.

Finally, the Case 3 uses the depth image to segment the rope. It is possible to get an estimate of how far the rope is from the camera from the positions of the arms of the robot. There are normally no other objects around this distance as no person can come close to the robot when it moves. Furthermore, the available ropes are thick enough to be seen in the depth image. For these reasons, I decided to implement rope segmentation based on the depth image.

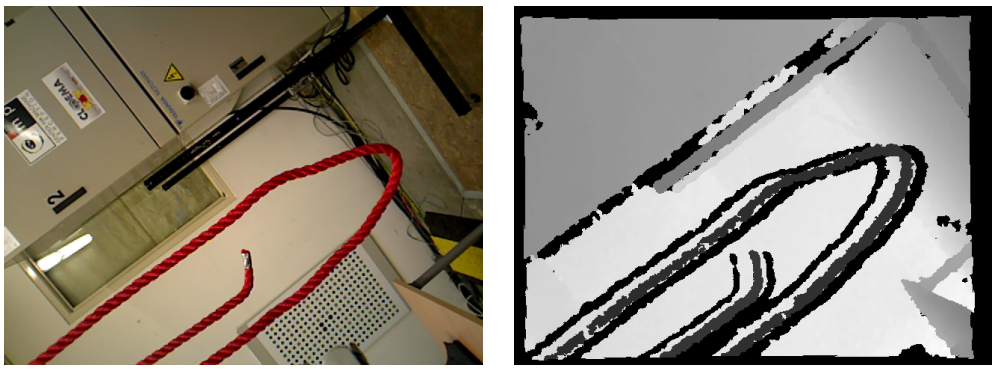


Figure 27: Rope end. Left: RGB image. Right: Depth image.

4.5.2 Implementation of the detection algorithm

The detection algorithm works with the point cloud obtained from the Kinect sensor. Figure 27, right side, shows the RGB image obtained from the point cloud and the left side shows the corresponding depth image. The images are turned almost upside down because of the position of the Kinect sensor on the right arm. I will assume the following.

1. There is no other object around the same distance as the rope in the image.
2. There are always two pieces of the rope in the image: the rope loop and the rope end.

The detection algorithm works with depth image only (so it does not matter which color the rope has). It also uses information from the geometry of the robot – the distance between the origin of the camera coordinate system and the tip of the left arm called rope distance r_d. The rope end detection works in the following way.

1. Compute the distance r_d of the rope from the camera.
2. Segment the rope: Mark everything within $[r_d - 0.1, r_d + 0.5]$ as the rope.
3. Find the first connected component C_1 (see Algorithm 1).
4. Find the second connected component C_2 (see Algorithm 1).
5. The smaller component (in number of pixels) is considered to be the rope end, the bigger one is considered to be the loop.
6. Find the tip of the rope end P_{rt} as the point that has the minimal x-coordinate index.

The algorithm for finding connected components (see Algorithm 1) has one important parameter – the width w [pixels]. This parameter is introduced so that the algorithm can connect pieces that are only separated by a short distance. The reason is that the depth image might not contain only two components (the rope loop and the rope end), but these components might be a bit "scattered". The higher the width w is, the faster the algorithm

is. However, there is an upper bound on w. The algorithm should not step over the gap between the rope loop and the rope end, i.e. it should not connect these two components into one.

Algorithm 1 Find connected components.

```
1: procedure FindComponent(maskRef, w)
2:     maskNew = 0                    ▷ Init maskNew - Array of the same size as maskRef
3:     points.add(p_i)                                   ▷ Randomly initialize a point.
4:     while points.size > 0 do
5:         p = points.pop()
6:         cut = GetArea(p, w, maskRef)        ▷ Select rectangular area around a point.
7:         maskNew[cut == 1] = 1       ▷ Mark pixels as 1 (belonging to the component)
8:         pToAdd = PointsToAdd(cut)                          ▷ Find 4 corner points.
9:         while pToAdd.size > 0 do pNew = pToAdd.pop()
10:            if pNew not in pointsVisited then      ▷ Add new point if it is not already
   visited
11:                pointsVisited.add(pNew)
12:                points.add(pNew)
13:            end if
14:        end while
15:    end while
16:    return maskNew
17: end procedure
```

4.5.3 Testing the detection algorithm

First, I analyzed the algorithm for finding connected components and experimentally estimated the parameter width w. The results below are from analyzing the recorded rosbag with data from working with the red rope.

For width $w = 20$, the algorithm worked sometimes correctly, but sometimes it connected both components C_1 and C_2 into one. As there is just a little time difference for width $w = 20$ and $w = 15$, width was chosen to be $w = 15$. Table 11 summarizes the measured times and iterations of the algorithm. The trend in the measured times is shown graphically in Figure 28.

4.5.4 Experimental results

Figure 29 shows the results of the detection algorithm. The first image shows the segmentation of the rope in the depth image based on the guessed rope distance d_r. The second image shows the first connected component C_1 and the third image shows the second connected component C_2. In our case, C_1 is bigger in size (the amount of pixels) than C_2,

Width w [pixels]	t_{C_1} [s]	t_{C_2} [s]	i_{C_1}	i_{C_1}
5	3.927	0.782	12626	2706
10	3.677	0.571	8738	1459
15	2.415	0.394	4623	820
20	2.384	0.410	3864	640

Table 11: Measuring Connected Components Algorithm.

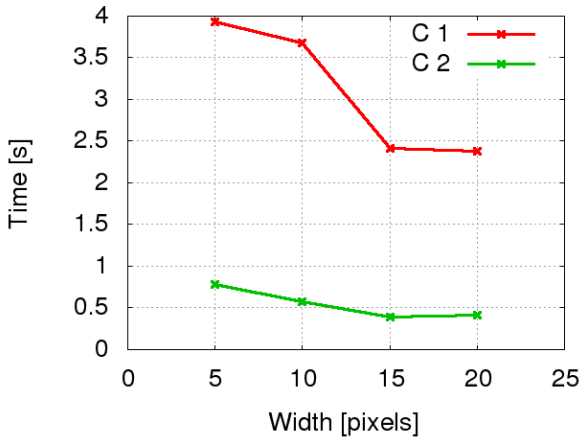


Figure 28: Measuring Connected Components Algorithm.

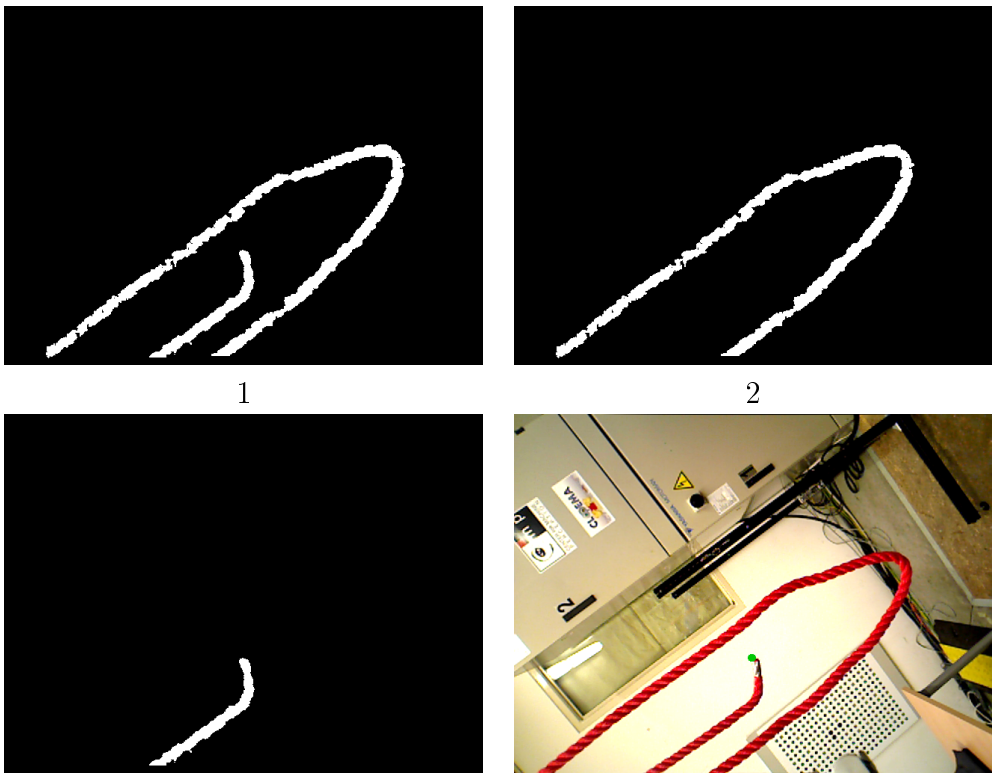


Figure 29: 1: Rope segmentation, 2: C_1 rope loop, 3: C_2 rope end, 4: Rope tip P_{rt} detected.

so C_1 is considered to be the rope loop and C_2 the rope end. The tip of the rope end P_{rt} is denoted by a green dot in the fourth image.

4.6 Knot-tying procedure

Based on the theoretical analysis, a work flow of the knot-tying procedure was designed (see Figure 30). Below is a more detailed description of the states of the knot-tying procedure. A diagram connected with each state is shown in Figure 31 and Figure 32.

INIT
The tying procedure starts with the two grippers facing each other, right in pose P_I and left in P_0 (see Figure 31 - 1). Both grippers hold the ends of the rope.

WRAP
The right arm goes through poses P_A, P_B, P_C and P_D (see Figure 31 - 2). The rope thus forms a loop around the left arm. Finally, the right gripper releases the rope end.

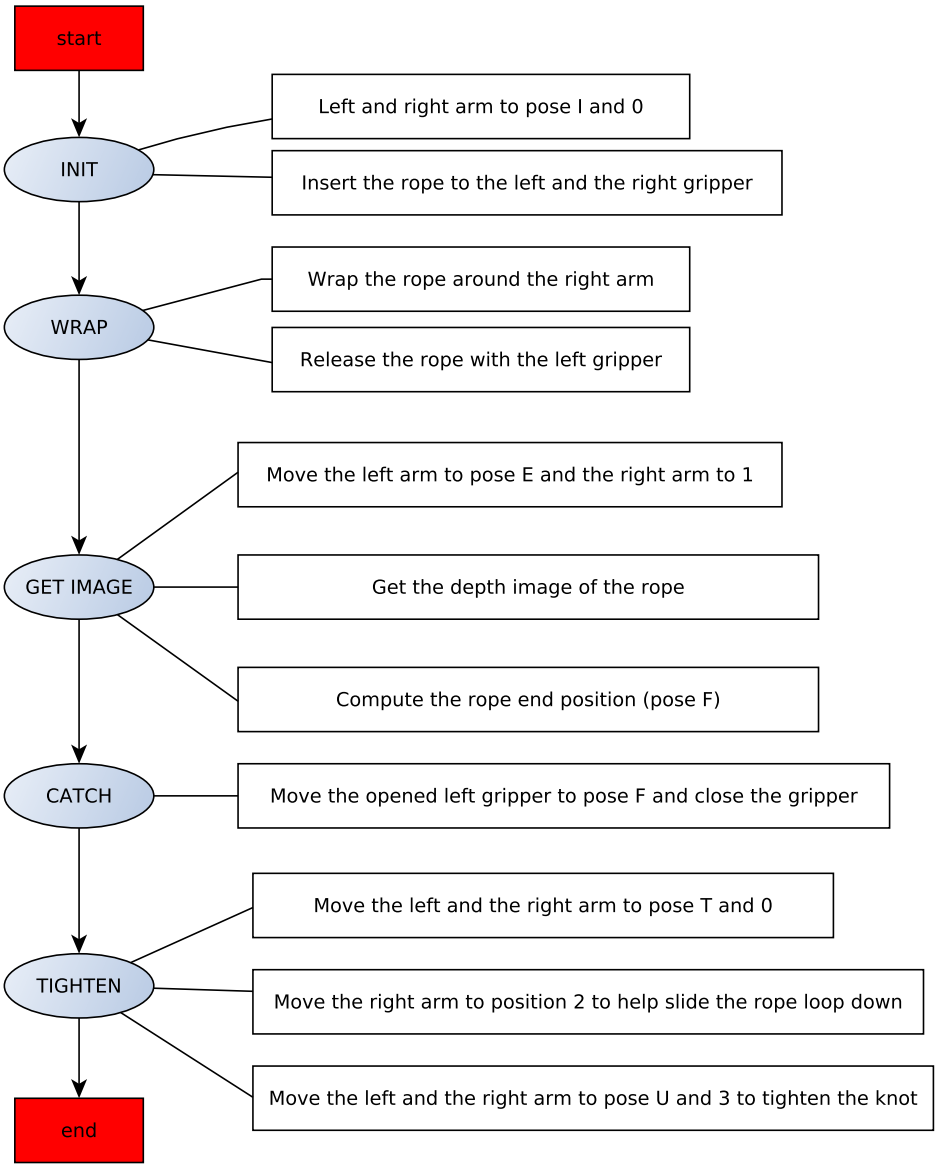


Figure 30: Knot-tying work-flow.

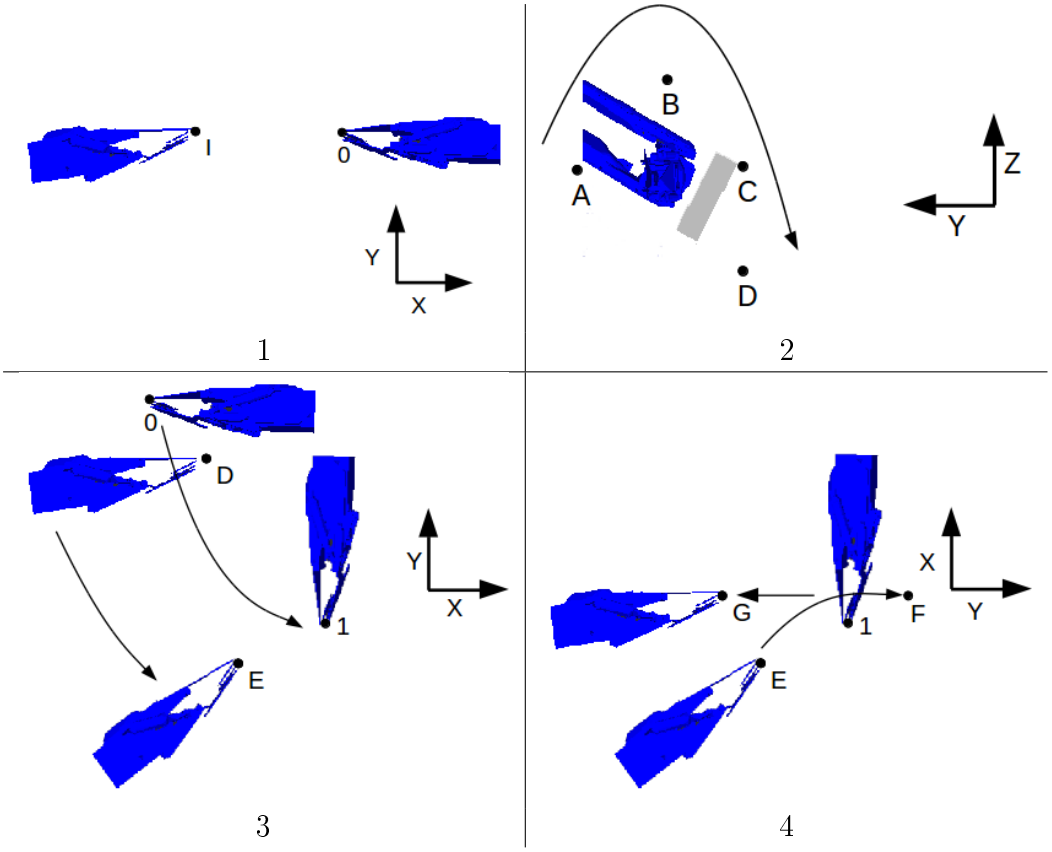


Figure 31: 1: INIT, 2: WRAP, 3: GET IMAGE, 4: CATCH.

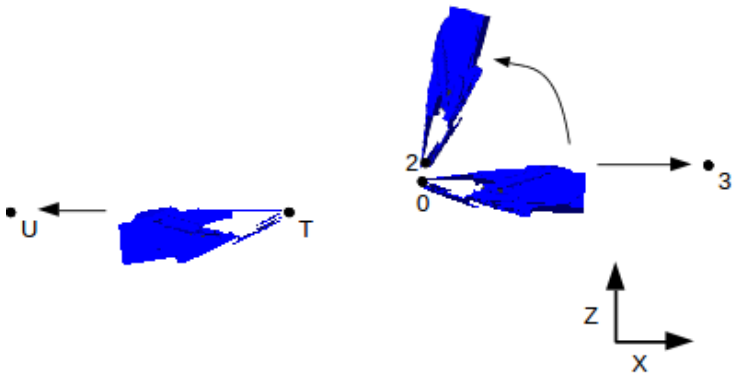


Figure 32: TIGHTEN.

CAPTURING THE IMAGE
First, the left arm turns to pose P_1 to enable catching the rope end later. Second, the right arm moves to pose P_E in order to have a good view of the rope loop and the rope end hanging on the left arm (see Figure 31 - 3). Third, the right gripper closes and moves to get out of sight. The point cloud from the Kinect sensor is captured. Finally, the rope end is detected from the depth image. Pose of the rope end P_F is computed.

CATCH
The right gripper opens, the right arm moves to pose P_F to catch the rope end. The gripper closes again. Next, the right arm moves back out of the loop to pose P_G (see Figure 31 - 4).

TIGHTEN
After the rope end is caught, both arms move to poses P_T and P_0. The left arm then moves to pose P_2 to help the rope loop slide down of the robot arm . Finally, both arms start slowly moving from each other to poses P_U and P_3 (see Figure 32). The magnitude of the z-coordinate of the force from the right force sensor F_z is measured. If the measured force exceeds the experimentally estimated threshold $F_{thresh} = 10\,\mathrm{N}$ (meaning that the knot is already tightened), the motion of both arms is stopped.

4.7 Experiments

I tested the developed knot-tying script on ropes R1-5 (see Table 10). Since R6 is flat and thus it is hard to detect it in the depth image, it was omitted from the experiments.

I also changed the width parameter w to 10. I did it due to further experiments with the red rope. Sometimes the rope end and rope loop were very close to each other. If $w = 15$, both connected components merged into one and the rope end detection was not possible.

The observations are listed below:

- The proposed solution was successful in the case of R1 and R4 (successful rope end detection in Figure 34). It took around 65 seconds to tie the knot (see video *KnotTying.mpg* that can be found on the attached CD). Figure 33, left side shows the knot tying process of R1 and Figure 33, right side shows the successfully made knot on the rope R4.
- The rope end was sometimes not found correctly in the case of R2 and R3 (see Figure 35).
- R5 got stuck on the left arm during tightening.

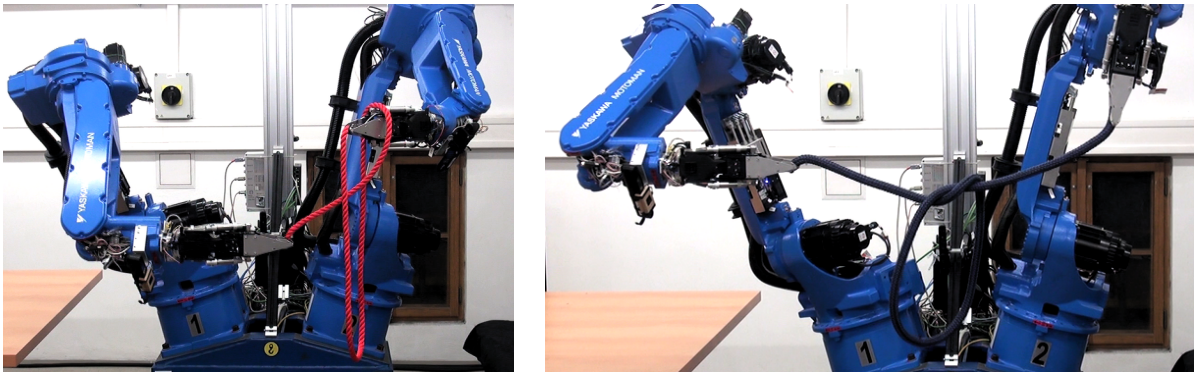


Figure 33: Left: Tightening the knot on the rope R1, Right: The knot made on the rope R4.

Figure 34: Left: R4 segmentation, Right: R4 rope end found.

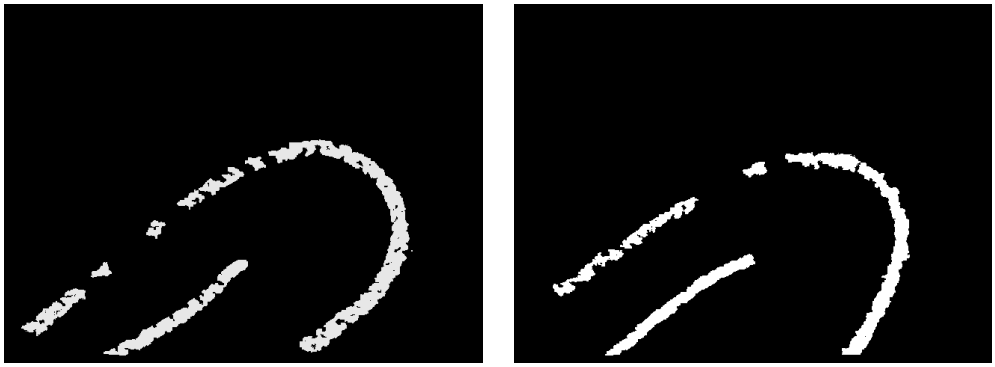


Figure 35: Left: R2 segmentation, Right: R3 segmentation.

4.8 Discussion

It turned out that the proposed solution is mainly successful for non-white and non-twisted ropes. The rope should not be flat, but should also not be thicker than the gripper pitch.

If the rope is white, the obtained segmentation in the depth image has huge gaps between the many connected components (see Figure 35). Thus, the precondition that the segmentation should contain only two connected components is violated and the rope end detection fails.

The twisted ropes not only cause the same problems during rope end detection as the white ones, but they also tend to get stuck on the robot arm.

The mentioned issues relate to image segmentation. I used a simple segmentation algorithm because the image segmentation is an unsolved problem. Image segmentation is the object of the intensive research by others.

5 Ribbon manipulation

5.1 Assignment

The manipulation with the rhythmic gymnastics ribbon was chosen to demonstrate the dynamic motion combined with the visual/tactile perception on the *CloPeMa* robotic testbed.

A common task for a human rhythmic gymnast is to swing the metallic pole hanging on a ribbon with one hand and catch it with the other hand (see Figure 36, the left side). Another rhythmic gymnast task is fast back and forth movements causing the ribbon to form spirals and other shapes that are nice to watch (see Figure 36, the right side).

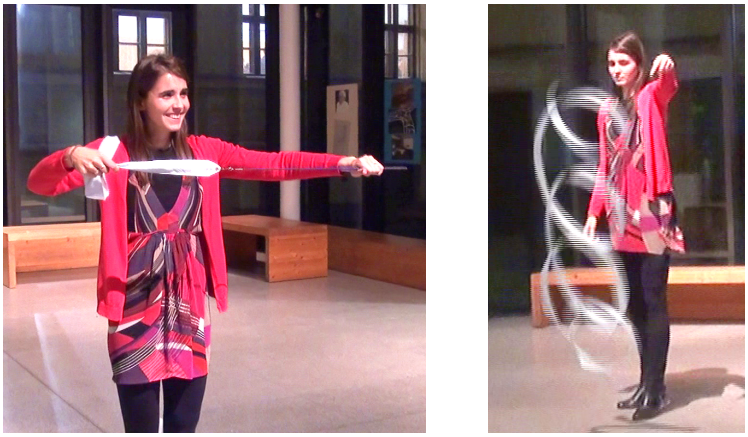


Figure 36: Left: Rhythmic gymnast catches the pole. Right: Rhythmic gymnast swings the ribbon.

CloPeMa "robot rhythmic gymnast" should perform a similar scenario that is defined to be the following. It starts from the state, in which the robot holds the pole hanging on the ribbon in its left gripper.

1. Swing the left arm in a way that the pole and the ribbon start to swing as well.
2. Grasp the pole with the right gripper using one of the sensors present in the gripper.
3. Make sure the pole was grasped, is held properly and release the ribbon from the left gripper.
4. Perform a few fast moves with the right arm so that the ribbon forms a certain shape nice to watch.

5.2 Analysis

In order to fulfill the proposed scenario given in Section 5.1, I had to check a few capabilities of the *CloPeMa* robot.

5.2.1 Available sensors in the gripper

The gripper contains tactile, light, magnetic and proximity sensor in its tip T [23].

Figure 37: Gripper description.

5.2.2 Verifying the ability to catch the swinging pole

I decided to use the light sensor to detect the swinging pole. The output of the light sensor, the relative intensity h, goes up as the pole approaches it. If the pole swings with the gripper positioned at the turning point of the swinging motion, two peaks appear at the output of the light sensor. The first one appears when the pole passes the tip T of the gripper on its way inside and the second one on its way out (see Figure 37). The threshold for detecting the pole was set to $h = 30$. The time between the two peaks was measured. Figure 38 shows the obtained results. The total time during which the pole was inside the gripper was estimated to be $t_p = 520\,\text{ms}$.

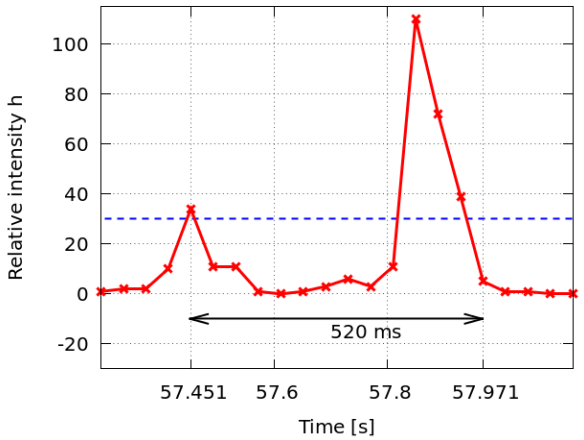


Figure 38: Swinging pole detection.

5.2.3 Gripper closing speed

In order to catch a swinging pole on a ribbon, the capabilities of the gripper have to be explored.

Gripper state	n	t_c [ms]
full open	0	600
full close	100	850
half close	50	420

Table 12: Gripper speed.

To control how much the gripper opens, parameter n is introduced. It controls the gripper on a percentage like basis: $n = 100$ for a fully closed gripper and $n = 0$ for a fully opened gripper.

The gripper was set to maximum speed (gripper frequency was set to $f_g = 25,000$) and the times t_c (worst-case) the gripper needs to open and close were measured. The results are shown in Table 12.

To be able to catch the pole, the gripper cannot be fully opened as it cannot close fast enough. The gripper has to be only half opened to be able to catch the pole.

5.2.4 Maximal speed of the *CloPeMa* robot

The experiments performed to determine the maximum speed of the robot are described in [24]. The robot can move faster if the executed trajectory has a low control point density.

The *CloPeMa* testbed robot moves the fastest when it is operated directly through its control system on a trajectories taught in by a teach pendant. If the robot is managed by a program through ROS, its speed is limited to 20% of its maximum speed at the level of the robot control system. For this reason, the following experiments were performed directly on trajectories taught in by the *CloPeMa* researcher Libor Wagner. Firstly, the robot had to perform a back and forth motion along a straight line in a general direction in 3D space (i.e. involving the motion of many robot joints simultaneously) at its maximum speed while holding the pole and the ribbon. The result was that the whole robot started shaking considerably due to the huge changes in acceleration. For this reason, for performing fast motions I decided to move the robotic arm using only one (and preferably the last) joint to reduce the motion impact on the whole construction.

The conclusion is that the needed fast motions (to swing the pole and to make fast motions with the ribbon) will be performed using one joint only. In addition, to perform back and forth motions only the start and end point of the trajectory will be used to reduce the point density. The resulting motions are thus arcs.

5.3 Robotic rhythmic gymnast

5.3.1 Equipment

I had a 6 m long gymnastic ribbon (see Figure 39) and two gymnastic poles at my disposal. The blue pole is shown in Figure 40, left side and the white pole in Figure 40, right side. The properties (length d, weight m and diameter ϕ) of the gymnastic poles are shown in Table 13. The diameter ϕ is measured at the point where the pole is grasped by the right gripper.

Figure 39: Rhythmic gymnastics ribbon.

Pole	Length d [cm]	Weight m [g]	diameter ϕ [mm]
blue	59	29	0.84
white	57	22	0.72

Table 13: Properties of the rhythmic gymnastics poles.

5.3.2 Experimental setup

The system consists of the end of the robot arm (left one), the ribbon and the pole (see Figure 44).

Based on the given assignment in Section 5.1 I designed a flow chart (see Figure 42). It is explained below. I estimated the swinging angles α and θ, catching threshold h_{catch} and recatching threshold $r_{recatch}$ experimentally.

The swinging angle α and θ are defined according to Figure 41.

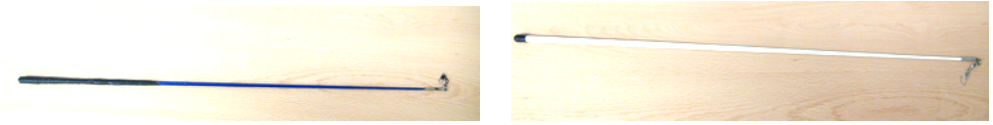


Figure 40: Left: Blue rhythmic gymnastics pole. Right: White rhythmic gymnastics pole.

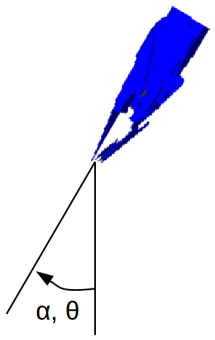


Figure 41: Definition of the swinging angles α and θ.

INITIAL POSITION
The right gripper holds the ribbon and the pole. The left arm is in the catching position. It means the right gripper should be positioned in the same plane in which the ribbon and pole swing. The turning point of the swinging motion should lie inside the gripper. It means that the pole should get inside the gripper at a certain point, but it should not hit the gripper.

SWING THE RIBBON AND POLE
The left robot arm swings back and forth between points P_0 and P_1 causing the pole and the ribbon to swing as well. The parameters $l = 1\,\mathrm{m}$ and the swinging angle $\alpha = 22.5^o$ were used in experiments.

CATCH THE POLE
The right gripper starts closing from a half opened state as soon as the output of the light sensor exceed the closing threshold $h_{catch} = 30$.

Is the pole caught well?
To determine whether the pole was caught inside the gripper (see Figure 43 Left) or caught well (see Figure 43 Right) the proximity sensor is used. If the recatching threshold exceeds $r_{recatch} = 5200$ an attempt to recatch the pole is made. It means that the gripper is opened for a short moment (to $n_{recatch}$) so that the pole moves towards its tip and closes back again ($n_{close} = 100$) so that the pole stays inside the gripper.

MOVE TO THE DANCING POSITION
The left arm, which held the ribbon, releases it and moves away. The right arm holding the pole moves up.

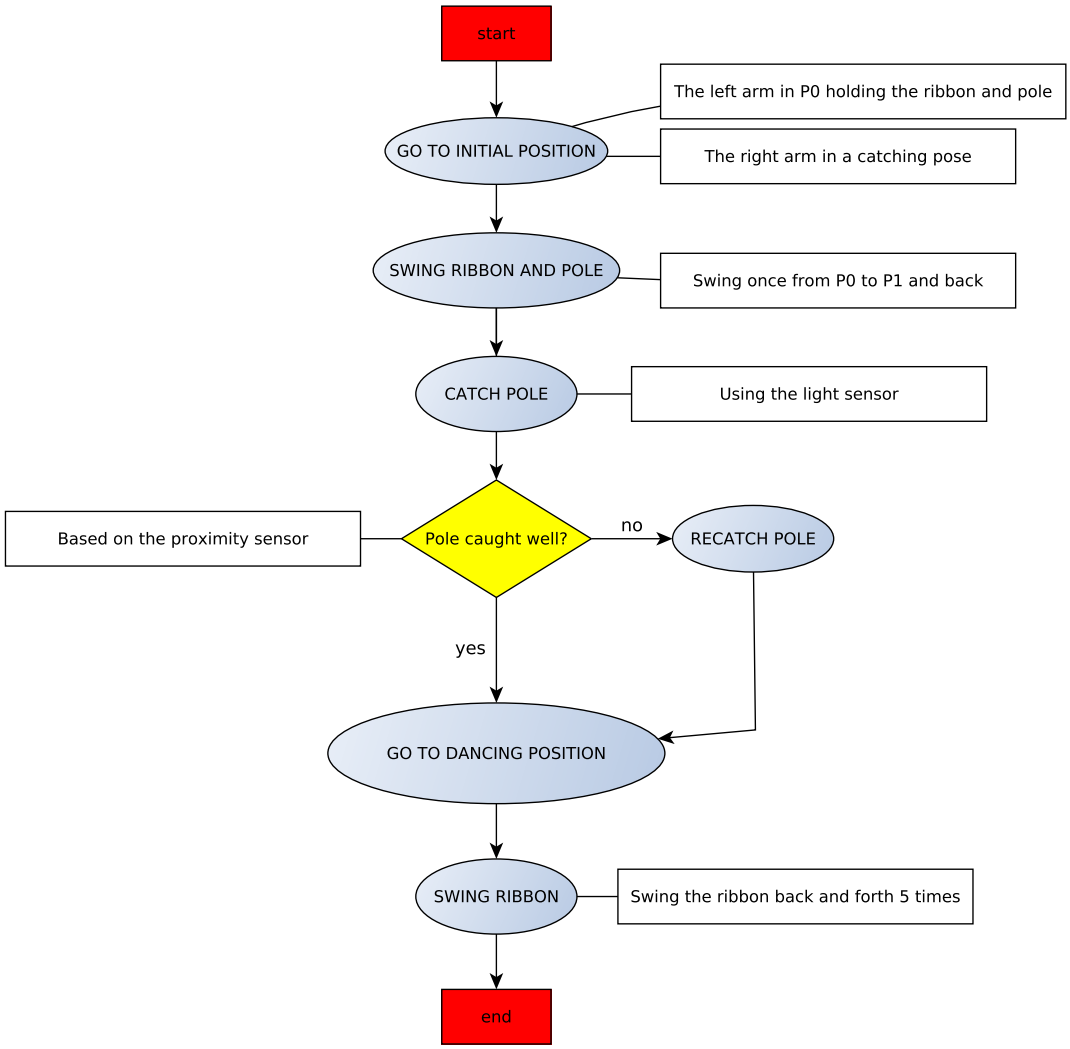


Figure 42: Robotic rhythmic gymnast workflow.

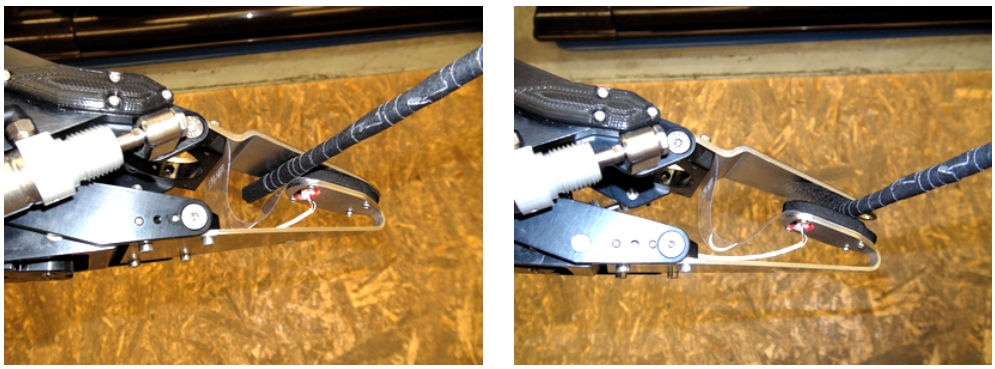


Figure 43: Left: The pole caught inside the gripper. Right: The pole caught well.

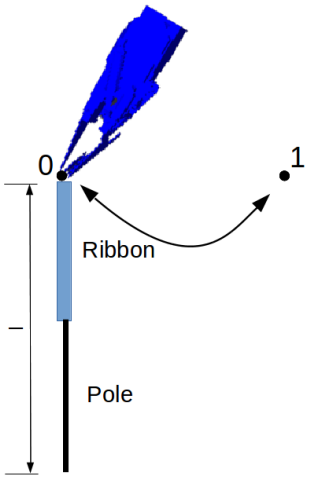


Figure 44: The hanging pole and the ribbon.

SWING THE RIBBON
The last joint of the right arm starts moving five times back and forth. The swinging angle is $\theta = 30.0^o$, so the total angle is two times bigger.

5.3.3 Mathematical description of the swinging pole

My first idea was to describe the whole system mathematically as a double pendulum and then solve the appropriate differential equations. After that, I thought I would use the control theory to compute the motion of the left robot arm that will cause the maximum swinging amplitude of the pole hanging on the ribbon. However, after I finally came up with the system description using the Lagrange formalism, I realized that this approach is too complicated.

For this reason, I tried to use the simplest description of a pendulum – the mathematical pendulum (see Equation (13)).

$$\ddot{\varphi} = -\frac{g}{l}\sin(\varphi) \tag{13}$$

The natural oscillations of a mathematical pendulum are described by Equation (14). In our case ($l = 1\,\mathrm{m}$), $T_0 = 2.01\,\mathrm{s}$.

$$T_0 = 2\pi\sqrt{\frac{l}{g}} \tag{14}$$

I assumed that the swinging pole and ribbon resemble the mathematical pendulum (no damping, no air resistance, the whole mass concentrated only at the tip of the pendulum

as a point mass, only small angle oscillations etc.). Thus, to achieve resonance and hence a maximal amplitude of the forced oscillations, the driving period of the robotic arm T_r has to equal T_0.

5.3.4 Experiments

Based on the designed workflow (see Figure 42), I made a video with two different gymnastic poles – the white one and the blue one. It is called *Gymnastics.mpg* and can be found on the attached CD. The end of the blue pole was wrapped in tape so that it is easier for a human to hold it and it was thicker than the end of the white pole.

Although the swinging ribbon and the pole break almost all the assumptions connected with the mathematical pendulum, the experiments with the real swinging pole and ribbon show that the maximal amplitude increases as the period of the driving oscillations T_r approaches T_0. Furthermore, the driving oscillations are applied in a way that the whole pendulum is swung by its end.

Figure 45 shows the successful catching of the swinging white pole.

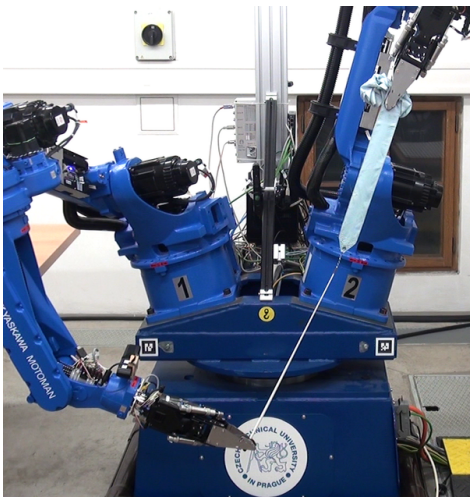


Figure 45: The white pole caught.

5.3.5 Discussion

The success rate of the pole catching depends on how the ribbon is held by the left arm. Since the gymnastic ribbon is 6 m long it has to be folded so that the left gripper can hold it. However, the position of the plane in which the ribbon and pole will swing later greatly depends on how the gripper holds it. For this reason, the correct catching position of the right gripper has to be adjusted manually every time the ribbon is placed into the left gripper. However, once the catching position is properly adjusted, the success pole catching rate is $> 90\%$.

In approximately 90% of the cases the pole is caught inside the gripper and has to be recaught. In approximately 10% of the cases the pole is caught well.

The recatching parameter was determined experimentally. $n_{recatch\_white} = 80$ for the white pole and $n_{recatch\_blue} = 69$ for the blue pole (the thicker one). This parameter has to be set in advance by a human.

The whole process runs fully automatically with the exception of setting $n_{recatch}$ and determining the correct catching position. The fact that I used two different poles of different thickness and weight shows the generality of the proposed solution.

6 Regrasping a rope

6.1 Assignment

The last use case concerns regrasping a piece of a rope from one gripper to the other.

The proposed scenario is the following. It starts with the left gripper holding one end of the rope. The right gripper is open and is positioned towards the left gripper (see Figure 46).

1. The left arm starts to move towards the right gripper.
2. When the presence of the rope is detected inside the right gripper, the motion of the left arm is stopped.
3. The right gripper closes, thus holding the other end of the rope.
4. The left gripper opens and releases the rope.
5. The left arm moves back and the functions of both arms are swapped.
6. The whole procedure is repeated again. The right arm now moves towards the left one that catches the rope.

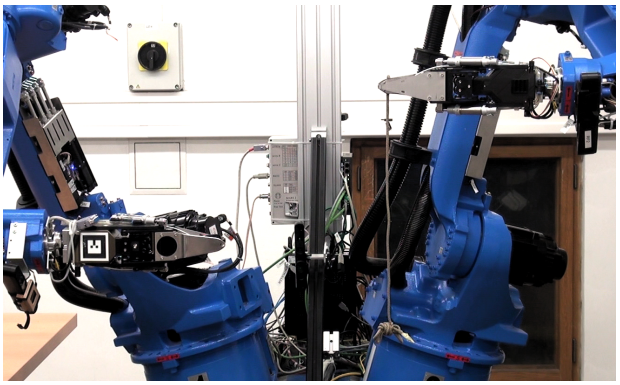


Figure 46: The initial position for regrasping the rope.

6.2 Regrasping workflow

In the following text, I use the definition of points shown in Figure 47 (front view). Points $P_{a,b,c}$ are connected with the left arm, while points $P_{1,2,3}$ are connected with the right arm. Point P_0 is used as the default point from which the rest of the points is calculated using the parameters l and h, l denotes the distance between the grippers and h the length of the rope.

Based on the assignment, I designed the rope regrasping workflow that is shown in Figure 48. The states of the procedure are shown in Figure 49 and are described below.

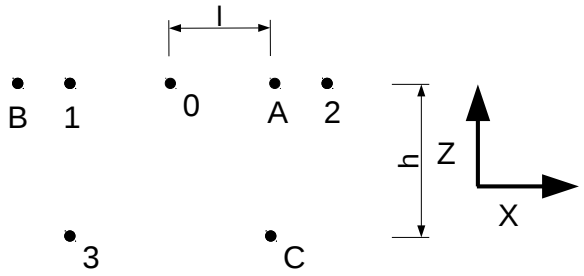


Figure 47: Defining the points.

GRASP INIT
The left arm moves to point P_a and the right one to point P_3. The right gripper holds the rope that is hanging down freely. The left gripper is fully opened and is positioned in such a way that the rope that is going to be moved towards it in the next step will enter the inside of the gripper (see Figure 49, 1).

GRASP LEFT
The left arm starts moving slowly towards point P_b. The rope thus enters the right gripper. As soon as the rope is detected (see Section 6.3), the movement of the left arm is stopped and the right gripper closes, thus catching the rope (see Figure 49, 1).

SWITCH ARMS LEFT
First, the left gripper releases the upper end of the rope. It falls down, now hanging just from the right gripper. Second, the left arm moves to point P_c thus reaching a pose in which it will later be able to catch the moving rope. Finally, the right gripper turns by 180° to unwrap the rope from the gripper (see Figure 49, 2).

GRASP RIGHT
See **GRASP LEFT**, the function of both arms is swapped (see Figure 49, 3).

SWITCH ARMS RIGHT
See **SWITCH ARMS LEFT**, the function of both arms is swapped (see Figure 49, 4).

The gripper rotates 180° around the x-axis upon transition from points P_3 to P_1 in the case of the right gripper and from P_C to P_A in the case of the left gripper. Let us know examine the case of the right gripper. The rope end that has fallen down from the left gripper might now be a bit "wrapped" around the right gripper (see Figure 50). Therefore, the right gripper has to rotate to ensure that the rope gets "unwrapped" and can hang down freely.

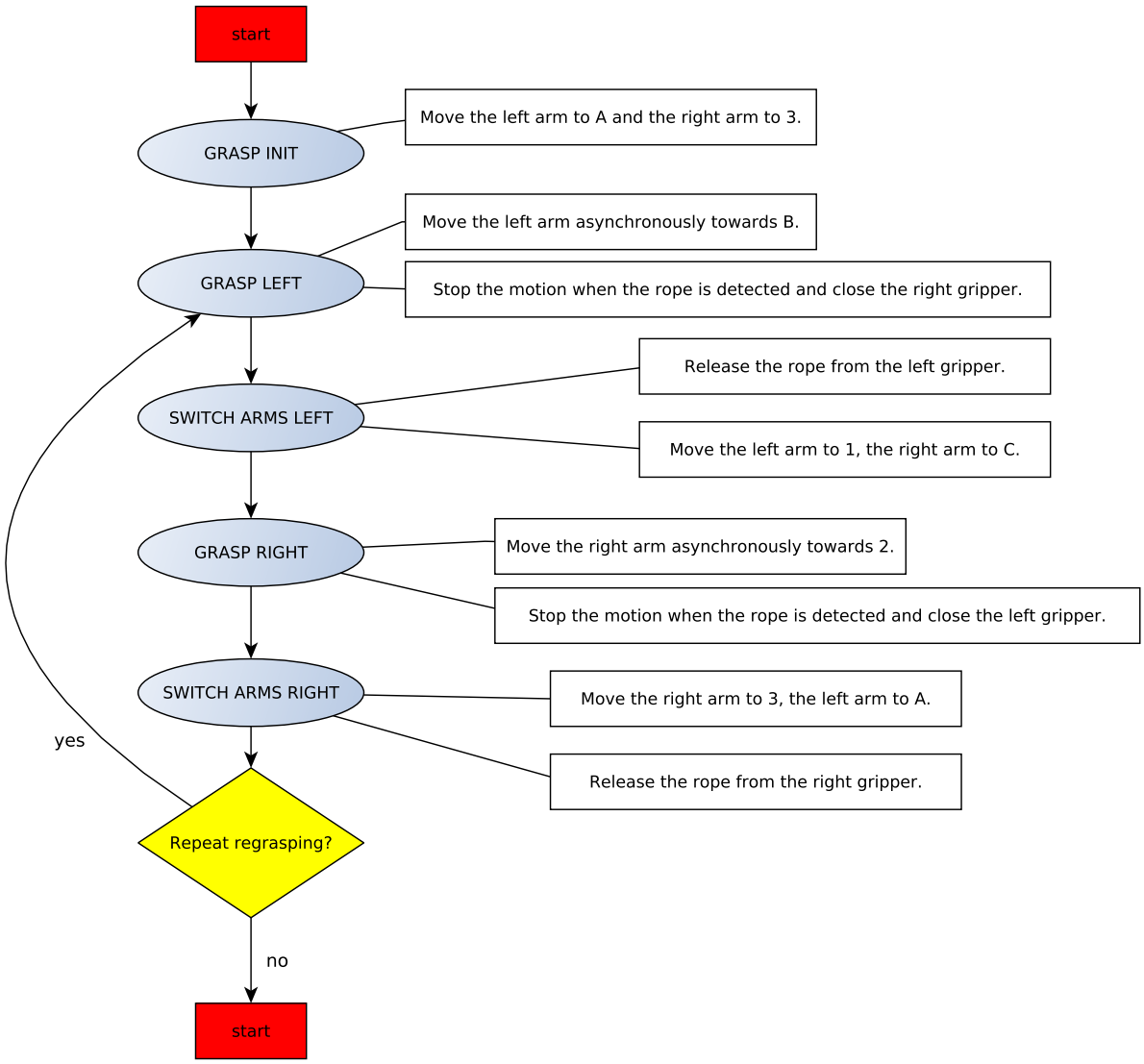


Figure 48: The regrasping workflow.

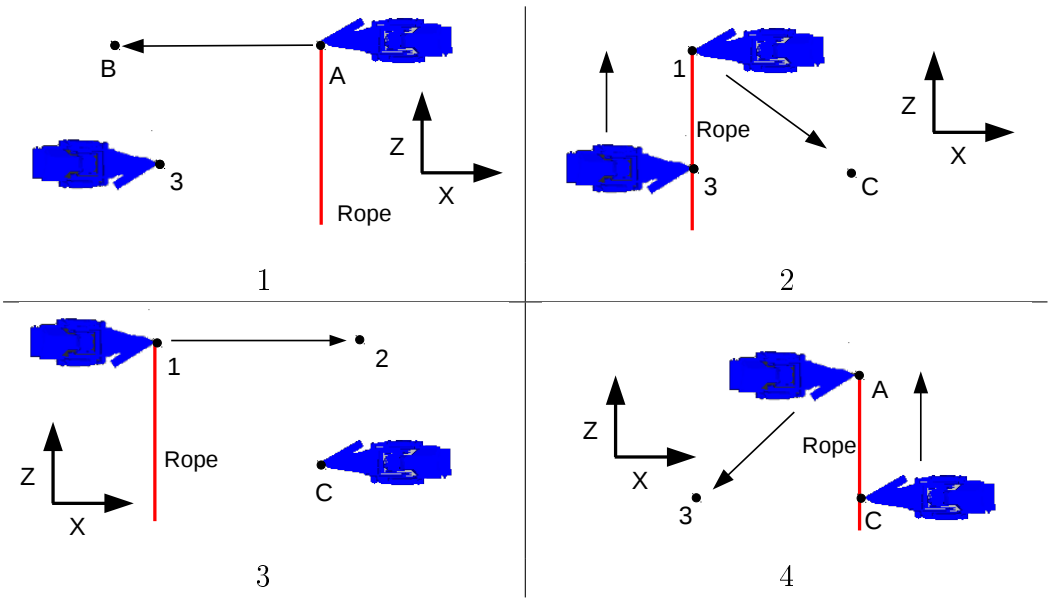


Figure 49: 1: Grasp left, 2: Switch arms left, 3: Grasp right, 4: Switch arms right.

6.3 Detecting the rope in the gripper

The rope detection algorithm was originally tested on the rope R2 (white and twisted rope, see Table 10). The left gripper holding the R2 rope was moved to point P_b and the output of the proximity and light sensor placed in the right gripper was measured.

I decided to use the proximity sensor over the light sensor for the rope detection, because the peak in the output of the proximity sensor was more distinct than the peak in the output of the light sensor (see Figure 51, left side).

The motion of the arm holding the rope is stopped, when a peak in the output of the proximity sensor passes. This approach proves more robust for ropes of different thicknesses (a thicker rope produces a higher peak) than a simple thresholding. However, a threshold is used to detect the rise of the peak. Initially, I set up the threshold to $r_0 = 100$ (see Figure 51, right side). The threshold r_0 makes sure that the rope detection process does not react on noise in the output of the proximity sensor.

6.4 Experiments

I found out that the rope R2 is not suitable for the regrasping procedure. It did not become straight after it was regrasped, it was rather bent to one side. This behavior prevents the successful consecutive regrasping. For this reason, I decided to use a different rope (see Figure 52). To ensure that it becomes straight again, it has a weight on each end.

I did not have to change the rope detection algorithm, it worked well for a thinner rope also. The only thing I had to adjust was to lower the threshold r_0 to 20, since the peak in

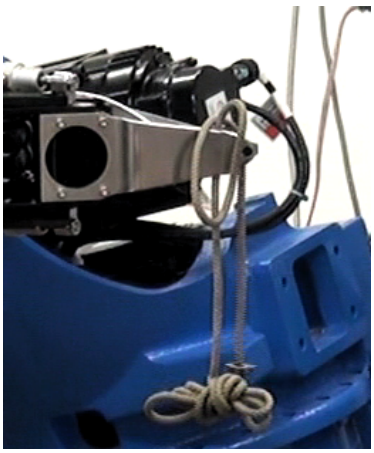


Figure 50: The rope wrapped around the gripper.

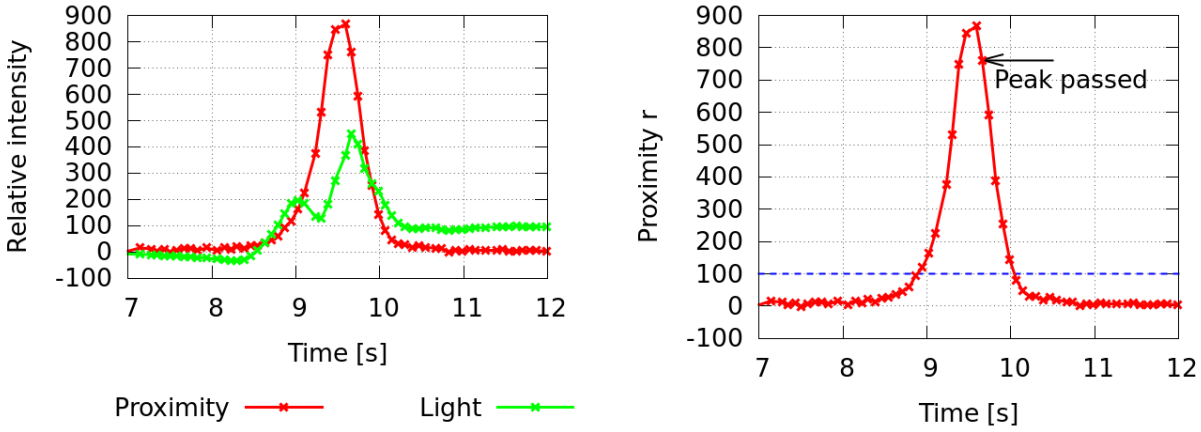


Figure 51: Left: Comparing the proximity and light sensor. Right: Detecting the peak.

the output of the proximity sensor became smaller.

The procedure ran successfully for regrasping the rope from the right gripper to the left and back. It took 34 s. The experiment was recorder on video that is called *RegraspRope.mpg* and can be found on the attached CD.

6.5 Discussion

Although the robot managed to regrasp the rope successfully, a few additional features would have to be implemented in order to make the process continuous (i.e. to be able to regrasp the rope many times in a row). The point in which the upper gripper releases the rope should be moved a bit to the side. In this way, the rope would fall to a predictable side of the lower gripper. Thus, it would be possible to always unwrap the rope from the gripper by turning it by 180°. This does not happen in the present case, since the rope

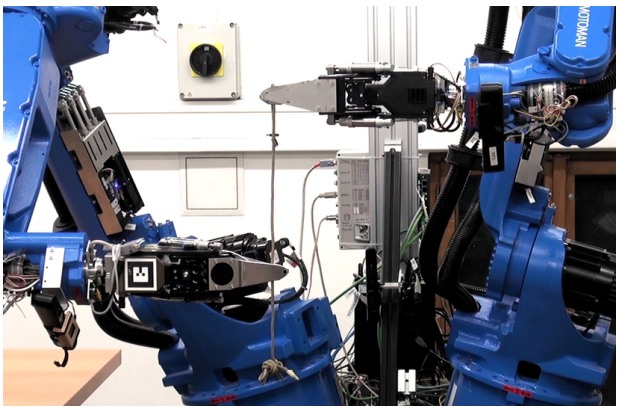


Figure 52: The left arm regrasps the rope.

is released directly above the gripper and thus it is not predictable on which side of the gripper it falls down.

7 Implementation

I programmed all the needed scripts and algorithms in Python 2.7.3 [25]. The communication with the various installed sensors was done through *Robotic Operating System* (ROS) [26]. I used the *Moveit!* package to control the movements of the robot [27].

7.1 How to run the code

The following manual is applicable for the PC's at the *CloPeMa* laboratory. The required operating system is Ubuntu 12.04 with installed ROS Hydro.

To launch the *CloPeMa* Rviz 3D Visualization Tool 53, run the following code in the console.

```
1 roscore
2 roslaunch clopema_launch start_robot.launch
```

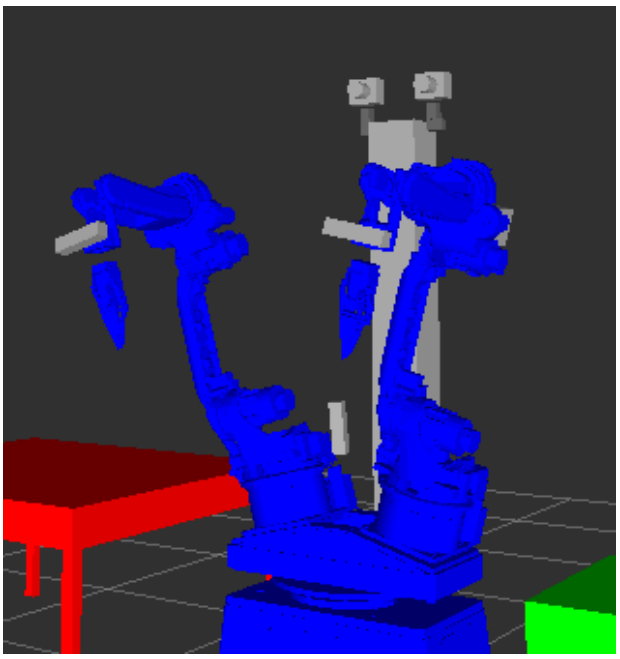


Figure 53: Rviz 3D Visualization Tool.

There is a Moveit! plugin, so one can move the robotic arms simply by dragging them in the visualization.

The code that I wrote can be found in the *clopema_compliance_2* package. Type the following to the console to navigate to the corresponding folder.

```
roscd clopema_compliance_2
```

The main code for the four tasks consists of four classes each placed in a separate module (class Shooter in shooter2.py for the slingshot, Tyer in tyer2.py for the knot-tying, Pendulum in pendulum2.py for the rhythmic gymnastics and Regrasper in regrasper.py for the regrasping). Each file contains a main() method that starts the main sequence for that particular module. I prepared the so called launch files that are used not only to start the main method, but also to launch all the required sensors. They are called *shooter.launch*, *tyer.launch*, *pendulum.launch* and *regrasper.launch*. To run the launch file, type:

```
roslaunch clopema_compliance_2 *.launch
```

If you wish to inspect and run the different methods of the classes separately, use the *IPython* console. There are four test files that only initialize an instance of that particular class. One can then run the different methods one by one. The following code is an example for the *Regrasper* class.

```
# First navigate to the corresponding folder.
ipython # Start the IPython console.

# Initialize an instance of Regrasper class.
run regrasper2_test.py

# Then run the different methods individually, for example:
r.init() # The robot moves to the initial position.
r.insert_rope_left() # Insert the rope to the left gripper.
r.grasp_left() # Regrasp the rope.
```

Note that you will have to start all the required sensors manually.

```
roslaunch clopema_launch force_sensors.launch # Start both force sensors.
roslaunch clopema_launch xtion1.laucnh # Start the xtion sensor in the
    r1 arm.
```

The documentation for all four modules can be found in the Use Cases Reference in Appendix A or on the attached CD.

7.2 Description of the developed reusable code

I successfully implemented a few classes (I call them managers) in Python that simplify the work with the *CloPeMa* robot and that are made to be used by others as well. These classes build on top of the *CloPeMa* testbed and provide more advanced functionality.

MoveManager covers many areas that are connected with controlling the movements of the robot arms. There are functions available for moving the arm along a straight line and for turning only one joint. Furthermore, there are functions that enable computing distance between the different coordinate systems of the robot and for transforming poses using the axis-angle notation.

GripperManager covers the operation of the robot gripper. One can set its closing speed. Not only can one open and close the gripper fully, but there are functions that enable partial opening as well.

ForceManager provides the functionality connected with the force sensor. One can stop the motion of the robot arm or open/close the gripper, if the measured force exceeds a certain threshold.

CameraManager provides the functionality connected with the Kinect sensor. There are function to obtain either the RGB or depth image of the scene. The algorithm for finding connected components is implemented in this module.

ProximityManager provides functions connected with the light and proximity sensors that are placed in the grippers. It can be used for closing the gripper, if the output of the light or proximity sensor exceeds a certain threshold.

I also generated a documentation that contains the most used methods as well as a short description of the developed modules. Below is an excerpt taken from the automatically generated documentation describing the managers. The detailed description for the developed methods and classes can be found in the Manager Reference in Appendix A. The full generated documentation in pdf and in HTML can be found on the attached CD.

CHAPTER

ONE

MANAGERS

1.1 Move Manager

The class `MoveManager` from the module `move_manager` provides the functionality connected with operating the *CloPeMa* robot.

1.1.1 Robot movement

There are two methods for moving the robot arms along a straight line: `Move()` for one arm only and `MoveBoth()` for both arms at the same time. Both methods can be executed asynchronously. Use `StopExecution()` to stop the execution of an asynchronous movement.

There are also two example methods for turning the arm in a specified joint: `TurnR2R()` for joint *r2_joint_r* and `TurnL2T()` for joint *r1_joint_t*. The speed of the movement can be specified directly.

There are methods for changing the speed of the whole robot: `SetRobotSpeed()` and `GetRobotSpeed()`. Note that the maximum is 0.2 (20% of the speed that can be achieved through the touch pendant).

The methods `Home()`, `RightHome()` and `LeftHome()` move both or one arm to its home position.

In case one does not need any advanced functionality concerning the grippers, one can use `OpenGripper()` and `CloseGripper()` to fully open or close the gripper.

1.1.2 Manipulation with the poses

The method `CreatePose()` is used to create a pose (*PoseStamped*). The method `RotatePose()` rotates the pose around a specified axis and angle. The method `TransformPoint()` transforms a point (*Point*) from one coordinate system to another.

The method `GetCurrentPose()` returns the pose of the specified coordinate system.

The method `GetXYDistanceBetweenFrames()` returns the distance between two coordinate systems from the top view.

1.2 Gripper Manager

The class `GripperManager` from the module `gripper_manager` provides an advanced functionality for the *CloPeMa* grippers.

The user can get/set the current opening/closing speed of the gripper through `GetGripperFrequency()` and `SetGripperFrequency()`. The maximum is 25000.

The gripper can be opened only partially using `MoveAbsolutePercentage()` on the percentage like basis. 0 is fully open and 100 fully close.

1.3 Force Manager

The class `ForceManager` from the module `force_manager` provides the functionality connected with the two force/torque sensors that are placed in each wrist of the *CloPeMa* robot.

The method `GetForce()` reads the force data from the force sensor.

The methods `CloseOnForce()` and `OpenOnForce()` are used to simplify the work of the user when inserting objects to the gripper. The gripper opens/closes when the user exerts a force on it. I use the `CloseOnForce()` every time I have to manually place something into the gripper.

The methods `WaitUntilForceX()` and `WaitUntilForceZ()` interrupt the execution of the program until the force exceeds a certain threshold. This is useful in situations when for instance the movement of the robotic arm should be stopped. These methods provide the main functionality for the **slingshot** (`shooter2.Shooter` class). They are also used during the **knot-tying** (`tyer2.Tyer` class) to stop the motion of the robotic arms when the knot is already tightened.

1.4 Camera Manager

The class `CameraManager` from the module `camera_manager` works with the *PointCloud2* message obtained from the *xtion1* or *xtion2* sensors.

One can get the raw PointCloud message using `GetPointCloud()`. The RGB image is obtained using `GetRgbImage()` and the depth image `GetDepthImage()`. If one wishes to work with both contained in one array, one can use `GetXYZ()`. This method returns a m x n x 6 numpy array. 1 to 3 are the x, y and z point coordinates and 4 to 6 the R, G and B.

The algorithm for finding connected components is implemented as the method `find_component2()`. The method `find_rope_end()` segments the rope and finds the rope end. This is crucial part in **knot-tying** (`tyer2.Tyer` class).

1.5 Proximity Manager

The class `ProximityManager` from the module `proximity_manager` provides the functionality connected with the tactile, light and proximity sensors that are placed in each gripper.

The method `GetProximity()` reads the current data from all the tactile, light and proximity sensors from both grippers. The data is returned in a big array.

The method `CloseOnProximity()` closes the gripper when the output of the light sensor in that gripper exceeds a certain threshold. This method is used when catching the swinging pole and ribbon in the robotic **gymnastics** (`pendulum2.Pendulum` class).

The method `WaitUntilProximityPeak()` interrupts the execution of the program until the maximum of a peak in the output of the proximity sensor has gone. This method is the core stone of stopping the movement of the robotic arm in the **regrasping** (`regrasper.Regrasper` class).

8 Conclusions

In the course of working on the master thesis, I implemented four use case scenarios. The feedback from the various sensors of the two arm *CloPeMa* robot is used to accomplish tasks that involve perception and manipulation with soft objects.

8.1 Slingshot

I designed two slingshots to shoot from – the so called traditional slingshot (see Figure 54, left side) and the flat slingshot (see Figure 54, right side). Both had to be attached directly to the robot arm. Since the design of the grippers have changed, it was not possible to attach the elastic string directly to it as the projectile could damage some of the sensors placed in the gripper.

I developed a fully automated shooting procedure that includes loading the projectile, stretching the elastic string to a specified force and adjusting the shooting angle. Thus, a few shots with the same parameters could be shot in a row. The force feedback was used.

I made a few experiments with both slingshots. It involved shooting on a target hanging on the wall and measuring the height, at which the projectile hit the wall. I compared the experimental results with the theoretically calculated values. In case of the traditional slingshot, the results differed a lot. I think it was mainly due to the fact that it was not possible to attach the traditional slingshot tightly to the robot arm. Another reason is for instance the fact that even a small uncertainty in one of the parameters (e.g. the weight of the projectile) influences the theoretical calculation to a high degree. The results concerning the flat slingshot were significantly closer to the theoretical values than in the previous case, yet still not perfect. In order to improve the results, the way the shooting angle is adjusted would have to change.

Despite the issues stated above, I was able to repeatedly hit the target hanging on the wall (see Figure 55). The success rate was around 50% for the traditional slingshot and above 90% for the flat slingshot. I made a video called *TraditionalSlingshot.mpg* featuring the traditional slingshot that shows the whole shooting procedure. It can be found on the attached CD.

Further work could involve connecting the shooting procedure with an automatic target recognition using for instance the Kinect sensor. The parameters of the shooting such as the shooting angle and the force with which the projectile is shot would then be computed automatically so that the projectile hits the target. In order to achieve such functionality, the slingshot would have to be redesigned further. For instance, the elastic string should be made of a different material so that the force changes gradually with its stretch and not sharply as it is the case of the present elastic string.

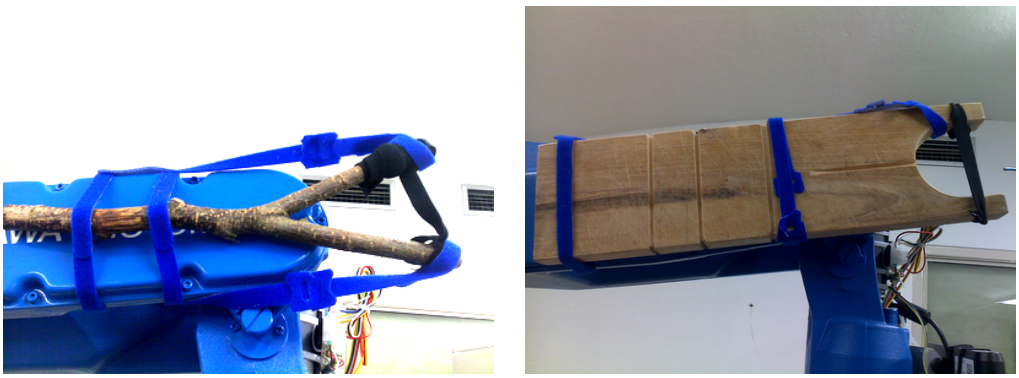


Figure 54: Left: The traditional slingshot. Right: The flat slingshot.

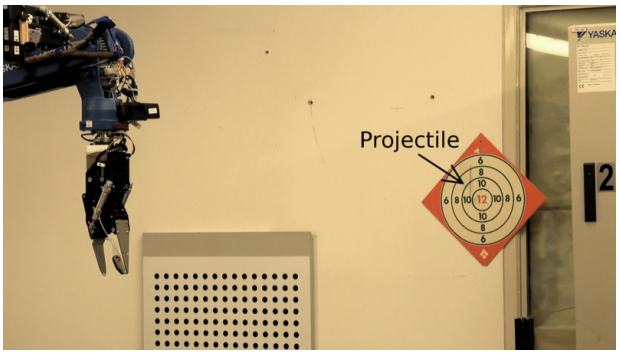


Figure 55: The projectile hits the target.

8.2 Knot-tying

I successfully implemented a fully automated knot-tying procedure. I tested the proposed knot-tying sequence on two persons (see Figure 56); one that sees normally (subject A) and one that is blind (subject B). I made conclusions for the robotic knot-tying based on the observed behavior of the two subjects. I especially focused on the blind subject, because his movements resemble in a way the movements of the *CloPeMa* robot more closely than the movements of the normally seeing subject.

I implemented a procedure that finds the rope end in an image using the connected components algorithm (see Algorithm 1). The rope end is found in the depth image, which makes the whole procedure invariant to the color of the rope. However, the rope segmentation from the depth image is prone to errors if the rope is twisted and shiny (see Section 4.2.1. Two example images from a successful knot-tying are shown in Figure 57.

Moreover, I used the feedback from the force sensor to determine that the knot has been already tightened. In that case, further stretching of the rope is stopped.

Further work could focus on making the rope end detection better. First, the code involving image processing could be made faster by for instance rewriting the Python code

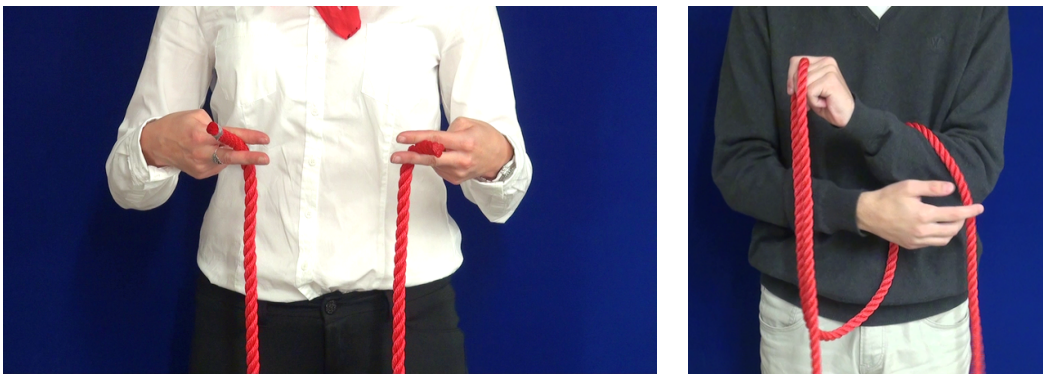


Figure 56: Left: Subject A holds the rope with 2 fingers. Right: Subject B caught the rope end.

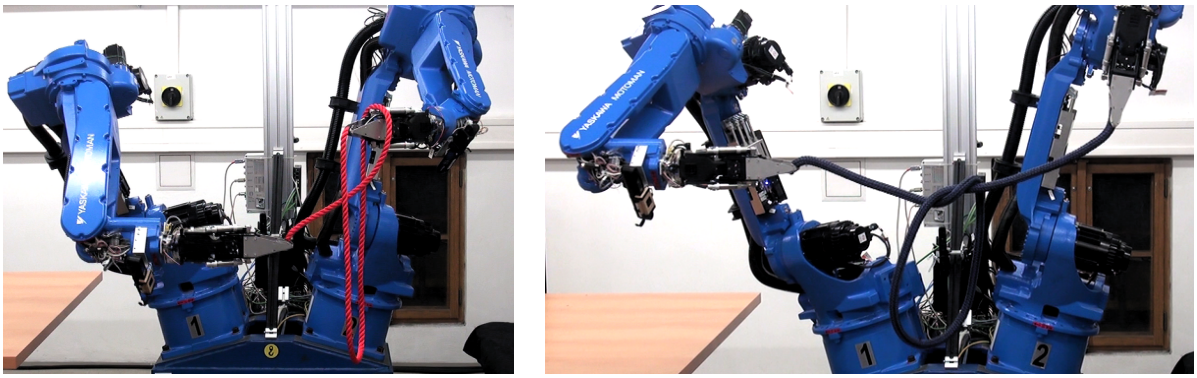


Figure 57: Left: Tightening the knot on the rope R1, Right: The knot made on the rope R4.

into C++. Second, the information about the color of the rope could be combined with the depth information to improve the rope segmentation procedure. In addition, further work could focus on distinguishing the case when the rope end was successfully caught from the case when it was not. The finding of the rope end and the catching of it could be then repeated if necessary.

8.3 Ribbon manipulation

Based on the observation of a human rhythmic gymnast, I developed a procedure that allows the *CloPeMa* robot to catch a swinging pole hanging on a ribbon. The task involves mathematical modeling of the swinging pole and ribbon and exploring the capabilities of the gripper such as its maximum closing speed. The light and proximity sensors placed in the robot gripper were used to detect the presence of the pole and to determine whether the pole was caught well or not. Figure 58, left side shows the human rhythmic gymnast that caught a swinging pole and Figure 58, right side shows the *CloPeMa* robot performing

the same.

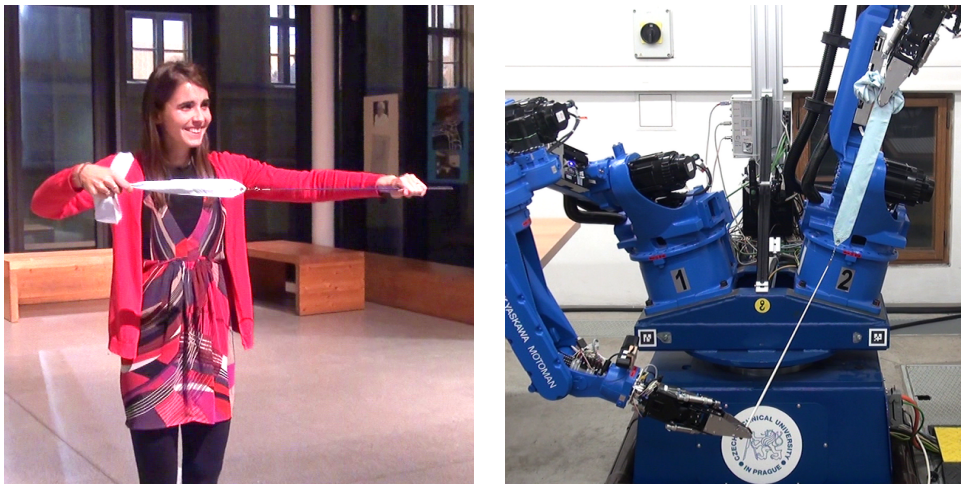


Figure 58: Left: The human rhythmic gymnast catching a swinging pole. Right: The robotic gymnast catching a swinging pole.

I also investigated the possibilities of performing fast motions with the *CloPeMa* robot. The aim was to mimic a human rhythmic gymnast performing fast motions with the ribbon (see Figure 59, left side) thus forming shapes that are nice to watch. Figure 59, right side shows the robotic counterpart doing the same.

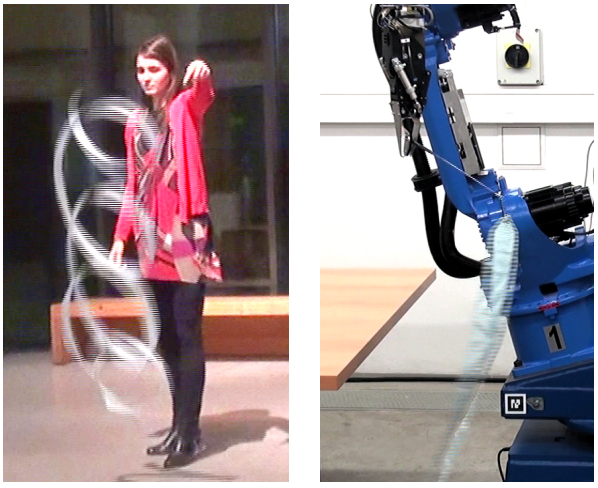


Figure 59: Left: The human rhythmic gymnast performing fast motions with the ribbon. Right: Robotic gymnast performing fast motions with the ribbon.

However, the fact that the maximum speed of the robot is limited to 20% when it is operated through ROS does not allow the ribbon to be swung fast enough to form a visually nice shape. The results obtained when operating the robot through a teach pendant (thus

reaching the maximum robot speed) were better. Yet operating the robot at its maximum speed in an unskilled manner for longer periods of time might cause damage.

8.4 Regrasping a rope

I successfully designed a scenario in which the *CloPeMa* robot regrasps a rope from one gripper to the other and back. The procedure is fully automated. The proximity sensor is used to detect the presence of the rope in the gripper. To make the rope detection process more robust so that it can handle ropes of different thickness, the gripper closes when a peak in the output of the proximity sensor passes (I do not use a simple thresholding).

The regrasping procedure was recorded in the video that is called *RegraspRope.mpg* and can be found on the attached CD. I have also taken a few screen shots from the procedure to illustrate it better. They are shown in Figure 60.

Further work could improve the release of the rope. The present solution does not allow regrasping of the rope many times (see the difference between 1 and 6 in Figure 60), since it is not sure on which side of the gripper the rope falls down. Therefore, it is not possible to ensure that the rope is unwrapped correctly. This issue can be solved by avoiding the release of the rope directly above the lower gripper. The rope should be released a bit to the side, so that it falls to a predictable side of the lower gripper.

8.5 Documentation of the developed code

I made a documentation of the code that I wrote. The full version in HTML (see screenshot in Figure 61) and pdf can be found on the attached CD.

A part of the developed code are the so called managers. These are classes that simplify the work with the *CloPeMa* robot. *MoveManager* and *GripperManager* provide the functionality connected with the movement of the robot and its grippers. *ForceManager*, *CameraManager* and *ProximityManager* encapsulate methods connected with the force/-torque sensor, Kinect sensor and the light and proximity sensors in the grippers. These classes could be the entry point for somebody new who wants to start working with the robot.

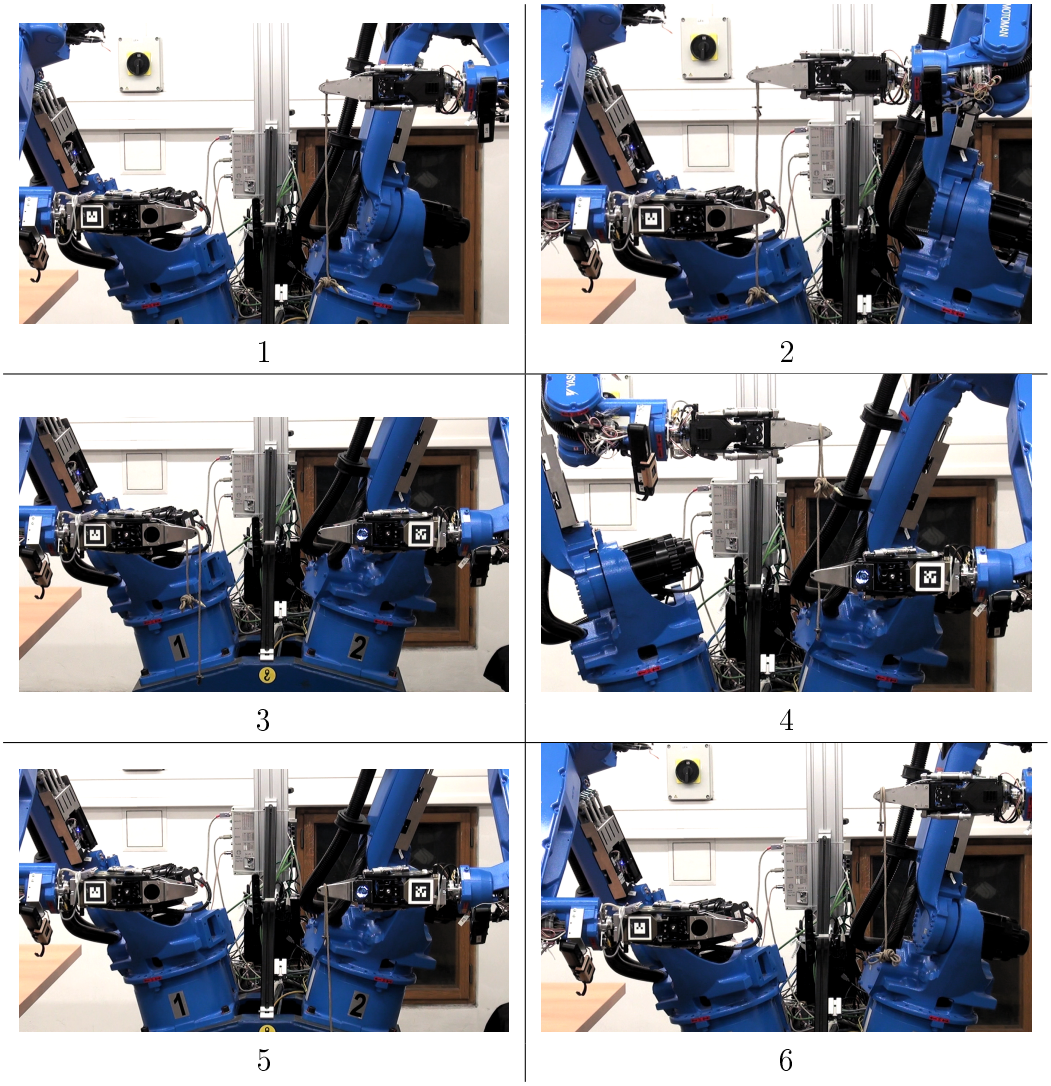


Figure 60: 1: The initial pose. 2: The rope caught with the right gripper. 3: The rope released from the left gripper. 4: The rope caught with the left gripper. 5: The rope released from the right gripper. 6: Back to the initial pose.

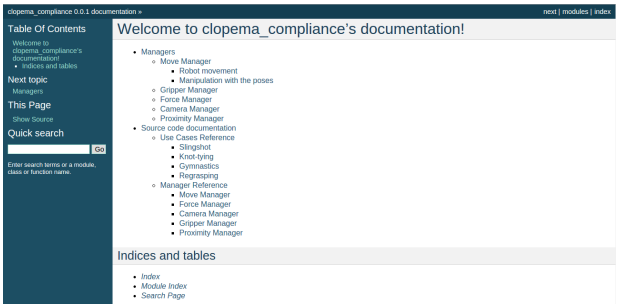


Figure 61: HTML documentation of the source code.

References

[1] D. Bourne, "My boss the robot," *Scientific American*, May 2013.

[2] T. Lejsek and V. Hlavac, "Experiments with the compliant motion control using the clopema test-bed," tech. rep., CTU, 09 2014.

[3] S. J. Buckley, *Planning and teaching compliant motion strategies*. PhD thesis, Massachusetts Institute of Technology, 1987.

[4] "Manually guiding an industrial robot arm." http://www.weiss-robotics.de/en/force-capture/6-axes-force-torque-sensors/force-torque-sensor-kms-40.html.

[5] T. Kröger, D. Kubus, and F. Wahl, "6d force and acceleration sensor fusion for compliant manipulation control," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2626–2631, Oct 2006.

[6] J. Kubeš, "Využití šestiosého silového senzoru pro řízení robotického manipulátoru," Master's thesis, Czech Technical University in Prague, 2014.

[7] "Compliant motion of the robot hand – first experiment." https://www.youtube.com/watch?v=a73p5WDjLl0, 2014.

[8] A. A. Syed, X.-g. Duan, C. Gao, X. Wang, and Q. Huang, "Maxillofacial surgery using virtual force feedback," in *Robotics and Biomimetics (ROBIO), 2011 IEEE International Conference on*, pp. 419–424, IEEE, 2011.

[9] J. Stria, D. Prusa, V. Hlavác, L. Wagner, V. Petrik, P. Krsek, and V. Smutny, "Garment perception and its folding using a dual-arm robot," in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pp. 61–67, IEEE, 2014.

[10] M. Quigley, A. Asbeck, and A. Ng, "A low-cost compliant 7-dof robotic manipulator," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 6051–6058, IEEE, 2011.

[11] "Sling robot - modeling, simulations and experiments." http://www.youtube.com/watch?v=WT2Ih0d7K24, 2011.

[12] "Vex hurler trebuchet." http://www.youtube.com/watch?v=iLFhnW71pjk, 2008.

[13] "Robotic ball shooter." http://www.youtube.com/watch?v=vgASkE04Xlo, 2013.

[14] J. Schulman, J. Ho, C. Lee, and P. Abbeel, "Generalization in robotic manipulation through the use of non-rigid registration," *Submitted. Manuscript at http://rll. berkeley. edu/isrr2013lfd*, 2013.

[15] "Robot tying knots and folding towels." http://www.youtube.com/watch?v=IBY4t8XxH7E, 2013.

[16] M. Bell and D. Balkcom, "Knot tying with single piece fixtures," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pp. 379–384, IEEE, 2008.

[17] H. Mayer, F. Gomez, D. Wierstra, I. Nagy, A. Knoll, and J. Schmidhuber, "A system for robotic heart surgery that learns to tie knots using recurrent neural networks," *Advanced Robotics*, vol. 22, no. 13-14, pp. 1521–1537, 2008.

[18] B. Bauml, T. Wimbock, and G. Hirzinger, "Kinematically optimal catching a flying ball with a hand-arm-system," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pp. 2592–2599, IEEE, 2010.

[19] "Rollin' justin robot catches balls tossed in its direction." https://www.youtube.com/watch?v=R6pPwP3s7s4, 2011.

[20] "Abb robotics - fanta can challenge - level ii - superior motion control." https://www.youtube.com/watch?v=SOESSCXGhFo, 2009.

[21] M. Kazemi, J.-S. Valois, J. A. Bagnell, and N. Pollard, "Robust object grasping using force compliant motion primitives," 2012.

[22] A. Jain and C. C. Kemp, "El-e: an assistive mobile manipulator that autonomously fetches objects from flat surfaces," *Autonomous Robots*, vol. 28, no. 1, pp. 45–64, 2010.

[23] V. Salansky, "Garment presence recognition," research report, Center for Machine Perception, K13133 FEE Czech Technical University, Prague, Czech Republic, September 2013.

[24] L. Wagner, "Notes on clopema robot speed," research report, Center for Machine Perception, K13133 FEE Czech Technical University, Prague, Czech Republic, April 2014.

[25] "Python software foundation." https://www.python.org/.

[26] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, 2009.

[27] I. A. Sucan and S. Chitta, "Moveit!." Online: `http://moveit.ros.org`.

CD content

ThesisTadeasLejsek.pdf	The PDF version of this document.
src/	LaTeXversion of this document.
clopema_ compliance/	ROS Package with the library source codes.
doc/	HTML and pdf documentation to the source code.
video/	Selected videos from the experiments.

CHAPTER

TWO

SOURCE CODE DOCUMENTATION

2.1 Use Cases Reference

2.1.1 Slingshot

class shooter2.**Shooter**

Shoots a projectile with a slingshot.

Attributes: alpha (float): Shooting angle.

thresh_load (float): Force threshold for detecting that the elastic string was touched.

thresh_fire0 (float): Force threshold used to detect that the elastic string starts to be stretched.

thresh_fire1 (float): If this force threshold is exceeded, the slingshot is ready to fire (the elastic string stops to be stretched).

slow_speed (float): How many times the speed of the robot should be slowed down during loading and firing.

AdjustAlpha(*alpha*)

Adjusts the shooting angle *alpha*.

Args: alpha (float): shooting angle in degrees <0, 40>. *alpha* = 0 means horizontal shooting. *alpha* > 0 means shooting upwards.

Fire()

Starts stretching the elastic string.

The position when a certain force (given by *thresh_fire0*) is exerted on the string is noted. When the measured force exceeds *thresh_fire1*, the movement is stopped. The *dx* is computed.

InitPosition()

Go to initial position.

InsertBullet()

Place the projectile to the opened gripper. When you exert a force on the gripper, it closes automatically.

Load()

Loads the projectile.

The projectile is moved towards the elastic string. When the string is touched, the movement of the robotic arm is stopped.

ShootMultiple()

Shoots a projectile multiple times.

After every shot the user can decide, whether he or she wants to fire a next one. Type *y* for the next shot, *n* to stop.

ShootOne()
Shoots a projectile once.

ShootingPosition()
Go to the shooting position.

2.1.2 Knot-tying

class tyer2.**Tyer**
Ties an overhand knot on a given 2m rope.

Attributes: width (int): A width of a gap that can be stepped over during finding connected components [pixels].

show (int): 0 for no images, 1 to show rope end, 2 to show rope end and all images from the segmentation process.

CatchX (float), CatchY (float), CatchZ (float): x, y and z offset for rope end catching [m].

catch(*catchX=None*, *catchY=None*, *catchZ=None*)
Catches the rope end and informs the uses whether the rope end was actually caught.

get_image()
Turn 'r2_arm', move the 'r1_arm' so that the rope end gets in sight of xtion1 and finds the rope end.

init()
Move to initial position: both grippers facing each other.

insert_rope()
Insert the rope to both grippers.

tie_a_knot()
The whole knot-tying procedure.

tighten()
Moves both arms to the tightening pose, helps the rope slide down the 'r2_arm' and tightens the knot by stretching both arms.

wrap()
Wraps the rope around the 'r2_arm'.

2.1.3 Gymnastics

class pendulum2.**Pendulum**(*catch_angle*, *y1_diff*)
Pendulum - ribbon catching.

Args:

y1_diff (float): Adjust the position of the 'r1_arm' gripper in x-z plane [m].

catch(*T*)
Swing the pole hanging on a ribbon and catch it.

Args: T (float): Swing period [s].

dance(*num*, *T*)
Swings the robot arm in joint *r1_joint_t* there and back.

Args: T (float): Period of the swings [s]. num (int): Number of swings.

dancing_pose()
Moves the 'r1_arm' to the dancing pose.

sequence()
Executes the whole sequence: swings the pole, catches it with the other arm and performs a few fast movements with it.

swing(*T*, *num*, *async=False*)
Swings the arm in *r2_joint_r* between trajectory points *rr_0* and *rr_2*.

Args: T (float): Swing period [s]. num (int): Number of swings. async (boolean, optional): *True* for asynchronous execution of the motion.

2.1.4 Regrasping

class regrasper.**Regrasper**(*h=0.3*, *dy=0.03*, *peak_thresh=20*)
Regrasps a hanging piece of rope or string.

Args: h (float, optional): distance between the two grippers [m].

dy (float, optional): adjusts the position of the gripper used for catching [m].

peak_thresh (int, optional): When this threshold is exceeded, a peak in the output of the sensor starts to be recorded.

grasp_left()
Moves the rope towards the 'r1_arm', stops the motion of 'r2_arm' upon the detection of the presence of the rope and finally closes the gripper of 'r1_arm'.

grasp_right()
Moves the rope towards the 'r1_arm', stops the motion of 'r2_arm' upon the detection of the presence of the rope and finally closes the gripper of 'r1_arm'.

init()
Go to initial position and open both grippers

regrasp()
Regrasp the rope from 'r1_arm' to 'r2_arm' and back.

switch_arms_left()
Switches the poses of both arms.

switch_arms_right()
Switches the poses of both arms.

2.2 Manager Reference

2.2.1 Move Manager

class move_manager.**MoveManager**(*init*, *frame='base_link'*, *eef_l='r1_ee'*, *eef_r='r2_ee'*)
Move manager is used to control the movements of the arms of the *CloPeMa* robot.

Args: init (int): If set to 1, a ros node 'move_manager' is initialized.

frame (string, optional): Sets the default world coordinate system. Used as a ClopemaRobotCommander reference frame.

eef_l (string, optional): End effector link for the left arm ('r1_arm').

eef_r (string, optional): End effector link for the right arm ('r2_arm').

Attributes: crc (ClopemaRobotCommander): ClopemaRobotCommander for both arms ('arms').

CloseGripper(*armName*)
Fully closes a gripper.

Args: armName (string): 'left' for r1_arm or 'right' for r2_arm

static CreatePose(*x*, *y*, *z*, *rx*, *ry*, *rz*)
Creates a pose in the 'base_link' coordinate system.

Args: x (float): x-coordinate [m]

y (float): y-coordinate [m]

z (float): z-coordinate [m]

rx (float): rotation around x-axis [degrees]

ry (float): rotation around y-axis [degrees]

rz (float): rotation around z-axis [degrees]

Returns: ps (PoseStamped)

GetCurrentPose(*link_name*)
Get the current pose of a specified link *link_name*.

Arsg: link_name (str): Name of the robot link e.g. 'r1_ee'.

Returns: (PoseStamped): Pose of the requested link

GetRobotSpeed()
Get the current speed of the robot.

Returns: speed (float): Current robot speed. <0, 0.2>

static GetXYDistanceBetweenFrames(*frame1='xtion1_rgb_optical_frame'*, *frame2='r2_ee'*)
Compute the distance in x-y plane (top view) between two frames.

Args: frame1 (str): First frame. frame2 (str): Second frame.

Returns: dist (float): distance in x-y plane between *frame1* and *frame2*

Home()
Move both arms to their home positions.

static LeftHome()
Move the 'r1_arm' to its home position.

Move(*ps*, *armName*, *params={'async': False*, *'step': 0.01*, *'jump_thresh': 1.2}*)
Cartesian move of one arm.

Args: ps (PoseStamped): Target pose.

armName (string): 'left' for r1_arm or 'right' for r2_arm

params (optional): step (float): Distance between the generated trajectory points.

jump_thresh: Max allowed jump.

async (Boolean): If set to *True*, the trajectory is executed asynchronously.

MoveBoth(*psl*, *psr*, *params={'async': False*, *'step': 0.01*, *'jump_thresh': 1.2}*)
Cartesian move of both arms.

Args: psl (PoseStamped): Target pose of the left arm.

psr (PoseStamped): Target pose of the right arm.

params (optional): step (float): Distance between the generated trajectory points.

jump_thresh: Max allowed jump.

async (Boolean): If set to *True*, the trajectory is executed asynchronously.

OpenGripper(*armName*)
Fully opens a gripper.

Args: armName (string): 'left' for r1_arm or 'right' for r2_arm

static RightHome()
Move the 'r2_arm' to its home position.

static RotatePose(*poseStamped*, *angle*, *axis*, *point*)
Rotates a given pose *poseStamped* around axis *axis* that goes through a point *point* around the angle *angle*.

Args: poseStamped (PoseStamped): Initial pose.

angle (float): Angle of rotation [rad].

axis (numpy float array 1x3): Axis of rotation.

point (numpy float array 1x4): Point lying on axis in homogenous coordinates.

Returns: ps (PoseStamped): Rotated pose.

SetRobotSpeed(*speed*)
Sets robot speed.

Args: speed (float): Robot speed. <0, 0.2> 0.2 is the maximum when operating the robot through ROS.

StopExecution()
Stops the execution of an asynchronous movement of the robotic arms.

static TransformPoint(*point*, *frameFrom*, *frameTo*)
Transforms a point *point* from one coordinate system to another.

Args: point (Point): Point to be transformed.

frameFrom (str): Name of the current frame.

frameTo (str): Name of the target frame.

Returns: (point): Point in the new coordinate system.

TurnL2T(*angle*, *time=1.0*)
Turn r1_joint_t to specified angle (degree)

TurnR2R(*angle*, *time=1.0*)
Turn r2_joint_r to specified angle (degree)

2.2.2 Force Manager

class force_manager.**ForceManager**(*moveManager*)
Force Manager provides the functionality connected with the force/torque sensor.

Args: moveManager (MoveManager): Move manager.

Attributes: force_left (str): Name of the topic connected with the force/torque sensor placed in 'r1_arm'.

force_right (str): Name of the topic connected with the force/torque sensor placed in 'r2_arm'.

CloseOnForce (*armName*)
Opens the gripper and closes it when the force exerted on it exceeds the *CLOSING_THRESH* threshold [Newton].

Args: armName (string): 'left' for r1_arm or 'right' for r2_arm.

GetForce (*armName*)
Gets the force output from the force/torque sensor.

Args: armName (string): 'left' for r1_arm or 'right' for r2_arm.

Returns: (point): x, y and z coordinate with force [N].

OpenOnForce (*armName*)
Opens the gripper when the force exerted on it exceeds the *CLOSING_THRESH* threshold [Newton].

Args: armName (string): 'left' for r1_arm or 'right' for r2_arm.

WaitUntilForceX (*armName*, *thresh*)
Waits until the x-coordinate of the force exerted on the robot arm exceed the threshold *thresh*.

Args: armName (string): 'left' for r1_arm or 'right' for r2_arm.

thresh (float): Force treshold [N].

WaitUntilForceZ (*armName*, *thresh*, *max_it=1000*)
Waits until the z-coordinate of the force exerted on the robot arm exceed the threshold *thresh*.

Args: armName (string): 'left' for r1_arm or 'right' for r2_arm.

thresh (float): Force treshold [N].

max_it (int, optional): Maximum amount of iterations. After this number is exceeded, the waiting stops.

2.2.3 Camera Manager

class camera_manager.**CameraManager** (*init=1*)
Camera manager provides a functionality connected with the xtion sensor.

static GetDepthImage (*numpyPointCloud*)
Gets a depth image from the PointCloud.

static GetPointCloud (*topicName='/xtion1/depth_registered/points'*)
Get a PointCloud message from the xtion sensor.

Args: topicName (str): Name of the topic for the corresponding xtion sensor (*xtion1* or *xtion2*).

Returns: numpyPointCloud (numpy array): Point cloud. header (str): Name of the camera coordinate system.

static GetRgbImage (*numpyPointCloud*)
Gets a RGB image from the PointCloud.

static GetXYZ (*numpyPointCloud*)
Transforms the PointCloud obtained from the xtion sensor into numpy array.

Returns: xyz (numpy array): m x n x 6 (x, y and z-coordinate, r, g, b)

static find_component2 (*mask*, *width*)
Algorithm that finds connected component in the given image.

Args: mask (numpy array): 1 is foreground, 0 is background. width (int): Algorithm will step over a gap < *width* [pixels].

Returns: mask_ref (numpy array): 1 is foreground of the newly found connected components, the rest is 0 (background).

static find_rope_end(*numpyPointCloud*, *rope_dist*, *width*, *show=0*)
Finds the coordinates of the rope end.

Args: numpyPointCloud (numpy array): Point cloud.

rope_dist (float): Estimated distance between the rope and the camera coordinate system.

width (int): Parameter of the algorithm used to find connected components.

show (int, optional): 0 for no images, 1 to show rope end image, 2 to show all images (rope end, segmentation).

Returns: Xc (point): Rope end in the camera coordinate system.

2.2.4 Gripper Manager

class gripper_manager.**GripperManager**(*armName*)
Gripper Manager

Provides a fine control of the grippers. One instance per gripper is needed.

Args: armName(str): 'left' or 'right'

static GetGripperFrequency(*armName*)
Gets the gripper opening/closing speed.

Args: armName (string): 'left' for r1_arm or 'right' for r2_arm

Returns: Gripper frequency (int): <0, 25000>, 0 is min, 25000 is max.

MoveAbsolutePercentage(*perc*)
Opens/Closes the gripper to a specified degree

Args: perc: From 0 (full open) to 100 (full close)

static SetGripperFrequency(*armName*, *freq*)
Sets the gripper opening/closing speed.

Args: armName (string): 'left' for r1_arm or 'right' for r2_arm

freq (int): Frequency of the gripper. <0, 25000>; 0 is minimum, 25000 is maximum.

2.2.5 Proximity Manager

class proximity_manager.**ProximityManager**(*moveManager*)
Proximity manager is used to provide functionality connected with the light and proximity sensors that are placed in the grippers.

CloseOnProximity(*arm_name*, *thresh=70*)
The gripper closes when the output of the light sensor in exceeds the threshold *thresh*.

Args: arm_name (str): 'left' for r1_arm or 'right' for r2_arm.

thresh (float): Threshold for the output of the light sensor.

GetProximity()
Get the data from the tactile, light and proximity sensors from both grippers.

Returns: sensor_responses (array)

WaitUntilProximityPeak(*thresh*, *ind*, *max_it*)

Measures the output of the particular sensor in the gripper and waits until a peak passes.

Args: thresh (int): If the sensor output exceeds *thresh*, the beginning of the peak was detected.

ind (int): Index to the *sensor_responses* array. 35 for 'r1_arm' proximity, 34 for 'r2_arm' proximity.

max_it (int): Maximum amount of iterations. After this number is exceeded, the waiting stops.