

|  |  |
| --- | --- |
| Touch-driven servoing of a robotic arm based on vision-based tactile sensors | |
| 2nd year Internship report  Presented by | |
| Antoine Lucazeau | |
| Une image contenant Police, logo, Graphique, symbole  Description générée automatiquement | Company:  Centro Singular de Investigación en Tecnoloxías Intelixentes da USC (CiTiUS), Spain  Company supervisor:  Juan Antonio CORRALES RAMON  SIGMA Clermont supervisor:  Nicolas Bouton  Date : Septembre 2023 |
| Référence du fichier : Lucazeau\_ Antoine\_RapportStage2A.pdf | |

**SIGMA Clermont**   
Campus de Clermont-Ferrand – CS 20265 – 63178 Aubière– France  
Tél. +33 (0)4 73 28 80 00 – Fax +33 (0)4 73 28 81 00 –https://www.sigma-clermont.fr/

**FICHE D’IDENTIFICATION DU DOCUMENT**

|  |  |  |  |
| --- | --- | --- | --- |
| **Rapport de Stage 2ème Année** | | | |
| **TITRE DU DOCUMENT :** | | Contrôle d’un bras robotique basé sur des capteurs tactiles visuels | |
|  | | *Touch-driven servoing of a robotic arm based on vision-based tactile sensors* | |
| **AUTEUR :** | | Antoine Lucazeau – IMR - MMS | |
| **Date du document** | **Nbre de pages** | | **Référence du document** |
| 05/09/2023 | 41 | | NomEleve\_ PrenomEleve\_RapportStage1A.pdf |
| **ABSTRACT:**  This paper presents the work done in a 15-week long internship in a robotics research laboratory. In this paper is described the path followed to achieve the two main goals of the internship. | | | |
| **Keywords:** ROS, MoveIt, Python, C++, Genetic Algorithm, Spain | | | |
| **RESUME :**  Ce document présente le travail accomplit lors d’un stage de 15 semaines dans un laboratoire de robotique. Dans ce document sera décrit le chemin poursuivit pour accomplir les deux objectifs principaux du stage. | | | |
| **Mots clés :** ROS, MoveIt, Python, C++, Algorithme Génétique, Espagne | | | |
| **Visa de l’entreprise :**  Ce rapport de stage a été visé par (Nom, Prénom, Qualité) : ……………………………………..  Date :  Signature :  Je demande que ce rapport soit confidentiel (non diffusé sur l’intranet de l’école) :  OUI  NON | | | |

**SIGMA Clermont**   
Campus de Clermont-Ferrand – CS 20265 – 63178 Aubière– France  
Tél. +33 (0)4 73 28 80 00 – Fax +33 (0)4 73 28 81 00 –http://www.sigma-clermont.fr/

SUMMARY

[SUMMARY 5](#_Toc145945687)

[Acknowledgment 7](#_Toc145945688)

[CHAPTER 1. Introduction 8](#_Toc145945689)

[1.1. Presentation of the Internship 8](#_Toc145945690)

[1.1.1. Presentation of the laboratory 8](#_Toc145945691)

[1.1.2. The experience of working in Santiago de Compostela 9](#_Toc145945692)

[1.1.3. The experience of working in the laboratory 9](#_Toc145945693)

[1.1.4. The subject of the internship 10](#_Toc145945694)

[CHAPTER 2. Panda Control 13](#_Toc145945695)

[2.1. ROS 13](#_Toc145945696)

[2.1.1. ROS’s Structure 13](#_Toc145945697)

[2.1.2. Programming with ROS 14](#_Toc145945698)

[2.2. MoveIt 14](#_Toc145945699)

[2.2.1. MoveIt’s Structure 15](#_Toc145945700)

[2.2.2. Programming with MoveIt 15](#_Toc145945701)

[2.3. The Franka Control interface 16](#_Toc145945702)

[2.3.1. The LibFranka structure 16](#_Toc145945703)

[2.3.2. Programming for the LibFranka 16](#_Toc145945704)

[2.4. The Point Cloud Library 17](#_Toc145945705)

[2.4.1. The PCL components 17](#_Toc145945706)

[2.4.2. Programming using the PCL 17](#_Toc145945707)

[2.5. My Program 17](#_Toc145945708)

[2.5.1. Defining the goal of the program 17](#_Toc145945709)

[2.5.2. My package’s structure 17](#_Toc145945710)

[2.5.3. The Euclidean cluster node 18](#_Toc145945711)

[2.5.4. The GelSight server 18](#_Toc145945712)

[2.6. The Automatic picker 19](#_Toc145945713)

[CHAPTER 3. Design of the Hand 20](#_Toc145945714)

[3.1. Design of a joint 20](#_Toc145945715)

[3.2. Design of a phalange 21](#_Toc145945716)

[3.3. Design of a finger 22](#_Toc145945717)

[3.4. Design of a Hand 22](#_Toc145945718)

[3.4.1. The simulation 23](#_Toc145945719)

[3.4.2. The genetic algorithm 25](#_Toc145945720)

[3.4.3. Results 27](#_Toc145945721)

[CHAPTER 4. Conclusion 29](#_Toc145945722)

[Bibliography 30](#_Toc145945723)

Acknowledgment

I would like to express my sincere thanks to all the people who have helped me during the internship and has made this internship possible.

First, I would like to thank Mr. Juan Antonio CORRALES RAMON, the main supervisor of this Internship. Mr. CORRALES has given me advice throughout the entirety of the internship and has brought enthusiasm to our collaboration together. He also was the one who’ve chosen the subject of the internship and kept it interesting during my stay at the CiTIUS.

Then I would like to thank all of the CiTIUS staff for helping me during the internship. Everyone was very helpful and welcoming there and I’ve made a couple of friends along the way.

Finally, I would like to thank the MMS pole for their classes and teachings giving me the skills needed for this internship. This includes Mr. Youcef MEZOUAR who has put Mr. Corrales and I into contact and Mr. Bouton for his supervision.

Without all of you, this internship couldn’t have existed. It was a pleasure working with you.

# Introduction

## Presentation of the Internship

This internship was conducted in a research laboratory called CiTIUS.

### Presentation of the laboratory

The CiTIUS (Centro de Investigación en Tecnoloxías Intelixentes da Universidade de Santiago de Com-

postela) was established in 2010 as a research center within Campus Vida in Santiago de Compostela, specializing in the field of Information Technology (IT). This center is led by a team of 33 members, all of whom possess extensive research experience in collaborative R&D projects. They oversee a workforce of 100 individuals, comprising 30 senior researchers, 14 postdoctoral scholars, and numerous doctoral students, all united by a common goal: the pursuit of scientific excellence to benefit society.

Despite it being relatively new, the CiTIUS has emerged as one of the most prominent laboratories not only in Galicia but also in Spain as a whole. In December 2019, the CiTIUS received recognition as one of the eight Research Centers in Galicia as part of the Funding Program aimed at accrediting, structuring, and enhancing research centers within the Galician University System.

The CiTIUS is engaged in making its discoveries public. The CiTIUS’s goal is to improve society through Intelligent Technologies. I’ve noticed that this is accomplished through its many collaborations. My research involved reading multiple papers and the laboratory has also held multiple public meetings. For example, a Galician television channel has filmed some of my work during my stay.

The CiTIUS is concerned with sustainable development. As the CiTIUS is part of the Universidade de Santiago do Compostella (USC), it is involved in the campus’s sustainable development plan. It also has multiple research fields in societal issues, making artificial intelligence by and for people.

Below is the CiTiUS’s organizational chart:

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

Figure : CiTiUS’s organizational chart

### The experience of working in Santiago de Compostela

Working abroad has brought many different aspects compared to working in France.

* First there was the language barrier. At the beginning of the internship, I did not know how to speak Spanish. This was made worse by the fact that people also speak Galician there. I had a lot of trouble communicating with people there. However, most people in the laboratory spoke English and I have managed to learn a bit of Spanish by the end of the internship.
* Because France and Spain are neighbors, the cultural difference is not really noticeable, it was slightly noticeable from time to time, but I didn’t ever feel alienated.
* The cost of life in Galicia is also cheaper than in France, however, in Santiago de Compostela, it is quite difficult to find a place to live in. Santiago de Compostella is quite a small city, but it welcomes many visitors for the pilgrimage and many students. This has made it really difficult to find a place to rent for the period of my internship.
* Finally, living in Santiago de Compostela was really pleasant because of the cool weather and the nice vistas surrounding the city.

### The experience of working in the laboratory

Working as an intern in a laboratory is an enriching experience:

* the hands-on nature of the work is interesting. You get to apply the knowledge you've gained in the classroom to real-world experiments and projects. You're constantly using your problem-solving skills.
* In the laboratory, you are surrounded by researchers which allows you to share your findings and help you in their fields. This develops a sense of collaboration and teamwork. It's also an opportunity to learn from experts who are eager to share their experience with you. This also means that in some cases, you will have to create documents for other researchers in order to use your work later.
* However, it's important to note that working in a laboratory also comes with its fair share of challenges. It means working in fields that are new and are not always well documented. For me it meant a lot of debugging and researching time.

In the end, the experience of working in a laboratory as an intern is an excellent one. Working for discovery is interesting and by the end of your experience, you’ve got work you can call your own.

### The subject of the internship

The subject of my internship can be divided in two main parts.

#### Control of the panda

The first part of my internship was to control the Franka Panda robotic arm. The Panda Franka robotic arm, often referred to simply as Panda, is an innovative and highly versatile robotic manipulator designed for a wide range of applications. Developed by Franka Emika, a German robotics company, Panda is renowned for its advanced capabilities and adaptability. Here is a picture of the robot:

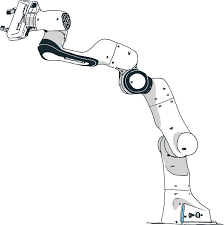


Figure 2: The Panda arm

The panda is made for many different purposes. It is also used for many tutorials and for research in general because of its adaptability.

I was tasked to develop programs to control it by using the visual based sensor called GelSight Mini in order to touch and grasp objects.

The GelSight Mini is a visual based sensor. It works by having a silicon membrane which can be deformed and by having a camera inside the sensor to monitor the membrane’s deformation. Here is a picture of the sensor:

Une image contenant conteneur, boîte

Description générée automatiquement avec une confiance moyenne

Figure 3: Picture of the GelSight mini

Visual-based sensors offers multiple advantages compared to the classic capacitive sensors.

* First, they offer a lot more information than the capacitive ones. With the entire image can be computed the forces and torques in the membrane instead of the point measured by one capacitive sensor.
* Also, with the picture of the membrane, the sensor can see what object it is picking up. The membrane allows to capture a lot of detail of the object even as small as fingerprints.
* The membrane can also be deformed which allows for better grasp of objects as the sensor adapts to the object in contact.

One disadvantage of the visuotactile sensors is that they do not offer a direct measurement of the deformation or the forces on the membrane. However, those measurements can be gotten indirectly from the images it outputs. Here a picture of strain measurements made by a GelSight paper [2]:

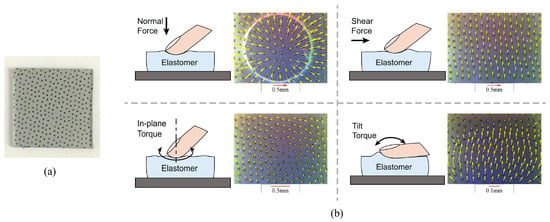


Figure 4: Computing strain in the membrane

Visual-based sensors have existed for a bit less than two decades, but their application had remained restricted. Until recently, they weren’t really used for robotics. The GelSight mini is a sensor which aims to make GelSight sensors smaller in order to be used for applications such as robotics. This sensor only came out to the public last year.

Additionally, I was tasked to make an algorithm to detect objects automatically in the Panda’s workspace in order to later use the GelSight sensors on it.

#### Making a Hand for the panda

My second task was to make a hand for the panda to better hold and touch with than the default panda gripper. The default panda gripper does not allow for a good exploration of objects with touch and can only grasp a couple of objects.

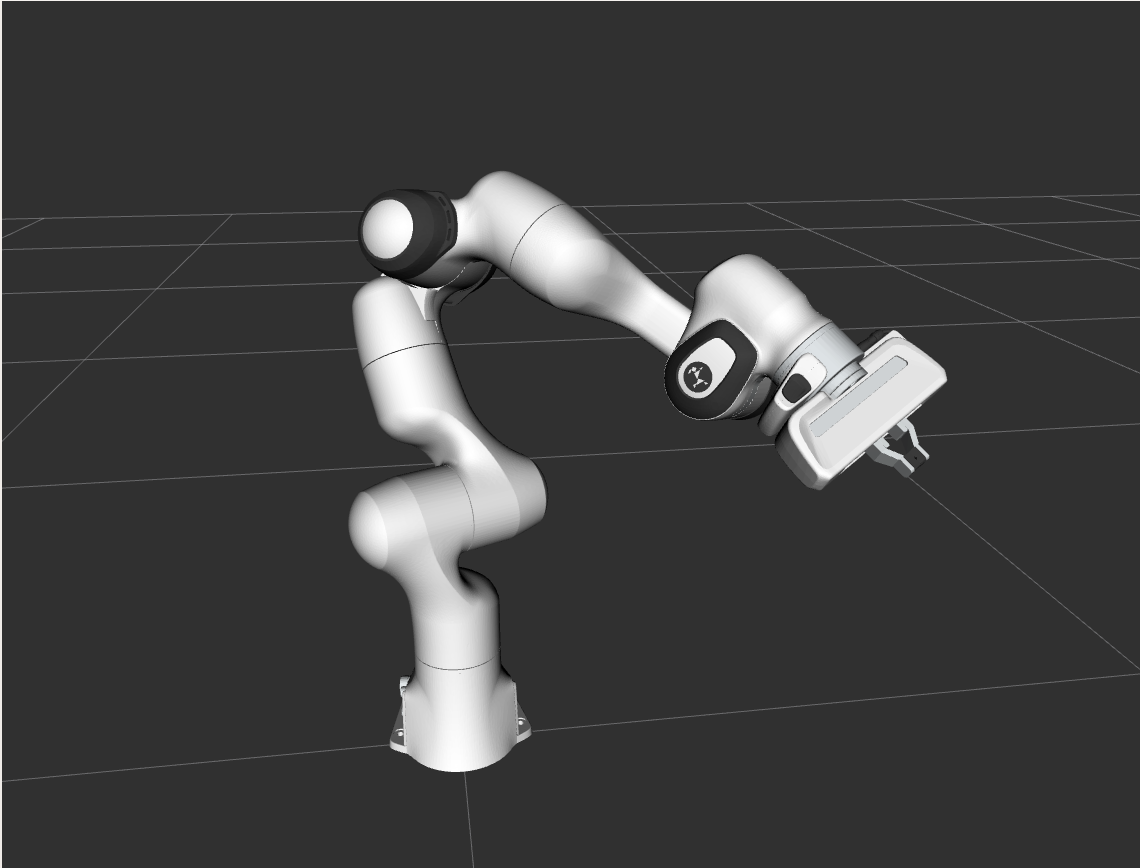


Figure 5: Picture of the default panda gripper

At first, I’ve designed prototypes of a robotic hand which could incorporate the GelSight mini sensors. After a meeting with my supervisor, we’ve decided that we would be using a genetic algorithm to design an optimized hand to touch and get information from those sensors. My aim was to implement an algorithm to optimize the geometry of the hand using genetic algorithms.

# Panda Control

The task I had was to control the panda so that it can automatically grab an object of any geometry in any place of the workspace.

To control the Panda Franka robot, I had to use many different tools. In this chapter, I will first describe the tools used and then I will explain the code I’ve written to accomplish the task.

## ROS

Une image contenant dessin humoristique, Graphique, affiche, dessin

Description générée automatiquementThe first program I had to use was ROS. ROS (Robot Operating System) is a node-based programing environment designed for robots and machines. ROS’s purpose is to allow communication between any number of sensors and actuators with any programming language.

Today, ROS is the industry standard. ROS makes it easier for engineers and programmers to build and control robots because it provides a common set of tools and rules. Its modular workflow is especially appreciated.

The version of ROS I was using is ROS noetic. This version is considered to be the recommended version for ROS 1. It has many advantages as it is the most stable and this is the version favored by the community. The latter means that it has more libraries and more support overall.

Figure : ROS Noetic logo

ROS also comes with software like Rviz or Gazebo, one allowing to visualize data and robots in a 3d environment and the other allowing to simulate physics in a robot in a 3d environment. Those programs are really useful in robotic applications. I will only be using Rviz as it is the program used by the main library I’ll be using and because I don’t need to simulate physics.

### ROS’s Structure

As I said before, ROS is a node-based programing environment. This means it is highly modular. I will explain the main architecture of ROS.

Master

Figure 7: ROS’s Architecture

**Publish**

**Subscribe**

Topic

Package

Node

Node

Node

As shown above, ROS’s main function is called the master. This is the program which handles every communication inside of ROS. It also registers different programs called nodes. Each node are individual programs written in any programing language that you can write. A group of nodes is called a package. The package handles the compiling and is a way to store and share multiple nodes. For nodes to communicate with each others, they have to use a topic registered by the master. Any node can send a pre-defined message to the topic, this is called publishing. On the other hand, any node can receive a message from the topic, this is called subscribing.

This is the basic structure any ROS program will have. It is important to note however that there exist other ways for two nodes to communicate with each others.

* The first is the service, client structure. One node is a service which can receive an input message and which will output another message to the client which called it.
* The second is an action server. While it is running, any node can send a goal to the server which will run the server code. While the server code is running nodes can get a feedback from it. When the server is done executing, it will send a result. This structure is especially handy for robotic purposes as the feedback is needed most of the time.

### Programming with ROS

In general, programming with ROS is straight forward as long as you stay organized. It requires a bit of experience and a bit of knowledge of its structure. Additionally, it requires some knowledge in xml in order to write some files like launch files. Its community is pretty active. It requires a bit of experience to really be productive with but I had already the basis thanks to my classes in Sigma Clermont. This experience allowed me to gain confidence and efficiency in using it.

## MoveIt

MoveIt is a software library that is made for ROS. It serves as a critical tool for programming and controlling robotic systems.

At its core, MoveIt specializes in motion planning and execution for robots. It helps robots determine how to move their various parts, such as arms or wheels, from one position to another while taking into account factors like physical limitations, obstacle avoidance, and joint constraints.

MoveIt provides functionalities like trajectory panning, forward and inverse kinematics. It also handles obstacle mapping of 3d environments and collision checking. This makes it easier for developers and engineers to control robotic systems and for example, allows them to not program inverse kinematics each time they need it.

MoveIt is a ROS package, it can be used in python or C++ nodes.

### MoveIt’s Structure

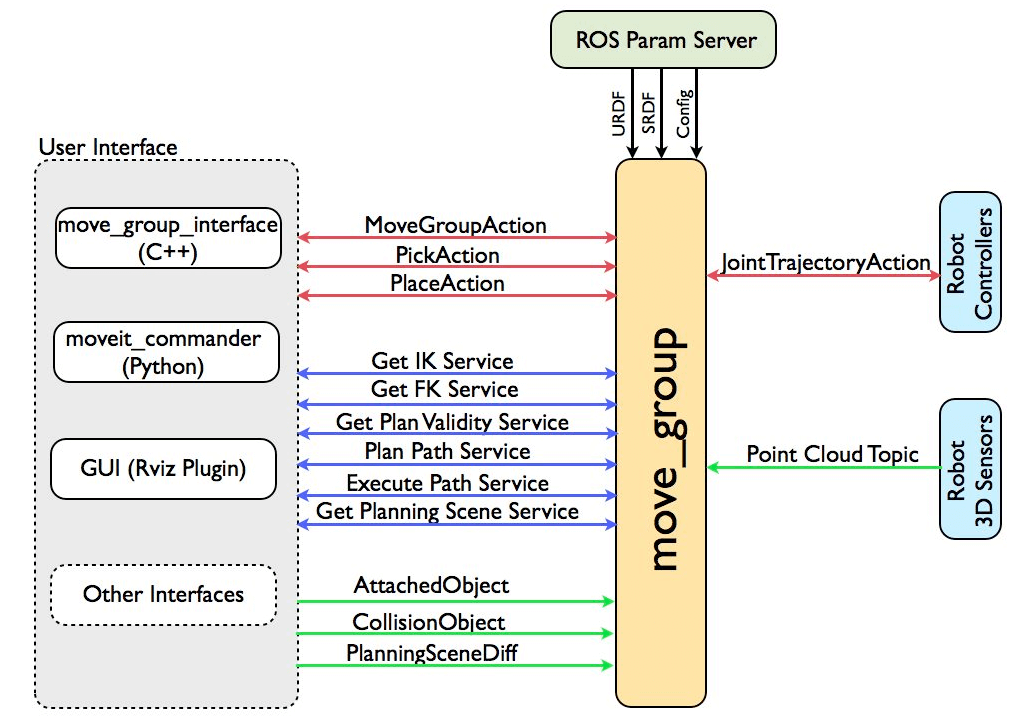


Figure 8: MoveIt's architecture

MoveIt is structured around its main node: the move group node. It handles everything MoveIt-related, every computation and communication. In the picture above its many functions are depicted.

* The user interface is what can be used in nodes to communicate to the move group and to control it. The user interface differs slightly from python to C++ but they remain similar. MoveIt also provides control through Rviz.
* The ROS param server handles all the parameters that are needed for the move\_group. It stores the URDF and SRDF files which are files needed to describe the robot used. This includes the part’s collisions and visuals.
* To control real robots, MoveIt’s move group also communicates with the robot’s given controller. In my case, this was the Franka controller from the library LibFranka.
* Finally, the move group also receives point cloud information from a 3D sensor and computes the Octomap.

The Octomap or occupancy map is a 3D map composed of voxels, cubes indicating that the point cloud is occupied in the region of the cube. All of those cubes are then used for collision detection by the move group node. Before using the point cloud, it is also filtered in order to remove the robot’s mesh from it. With the final Octomap, the move group can detect if the robot collides with its environment.

### Programming with MoveIt

MoveIt provides a tutorial package. This is handy to understand how to use it. I’ve spent the first week trying to understand and trying to use those tutorials.

Programming with MoveIt is pretty difficult. MoveIt is not really well documented and the version I’ve been using had multiple issues inside of it. For example, some specifications of the robot were off by up to 5 degrees and one of the tutorial had one main line of code commented out. This led to the termination of the program and an error message.

When learning to use MoveIt, the most important part is learning about the messages the library will be using and how to use them.

## The Franka Control interface

The Franka Control Interface (FCI) is a powerful framework designed to facilitate seamless communication and control of Franka Emika robotic arms. It is constituted of the ROS library LibFranka which allows for communication between ROS and the FCI.

### The LibFranka structure

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

Figure 9: Architecture of the LibFranka

LibFranka handles communication of external programs with the control computer which will control the real robot. It requires a real-time kernel of 1 KHz.

Here are the key components of LibFranka's structure:

* API Abstraction Layer: Provides a high-level interface for controlling the robot.
* Real-time Control Modules: Modules dedicated to enabling real-time control of the robot's joints and end-effector. This allows for precise and responsive movements.
* Collision Avoidance Tools: Tools and functions to implement collision avoidance strategies, ensuring safe robot operation in dynamic environments.
* Force and Torque Sensing Integration: Capabilities to access and utilize the force and torque sensors embedded in the Franka robot, facilitating tasks requiring force feedback.
* Trajectory Generation and Execution: Functions for planning and executing smooth and optimized trajectories, crucial for tasks like pick-and-place operations and assembly.
* Utility Functions: Additional functions and tools that assist in various aspects of robot control, such as kinematic computations, transformations, and error handling.
* Safety Features: Incorporates safety mechanisms to monitor robot behavior and implement emergency stop procedures in case of abnormal conditions.
* Error Handling and Reporting: Mechanisms to handle and report errors or exceptional situations that may arise during robot operation.

### Programming for the LibFranka

Making programs to use the LibFranka is pretty straightforward. The library uses action servers. It allows for a variety of actions which are both safe and fast. However, the direct use of the library is not very well documented.

## The Point Cloud Library

The Point Cloud Library (PCL) is an open-source framework designed for processing and manipulating 3D point cloud data. It provides a wide array of algorithms and tools for tasks such as filtering, segmentation, feature extraction, registration, and visualization of point cloud data. PCL is widely used in fields like robotics, computer vision, and 3D perception for applications ranging from object recognition to environmental mapping and localization. Its set of functionalities makes it an important library for working with point cloud data in various contexts.

### The PCL components

Here are the main functions of the PCL I’ve been using in order:

* Passthrough filter: This filter is simple. It selects the points from the cloud which are in a selected zone.
* Voxelize filter: This filter is a filter applied to most point clouds. It reduces the number of points in the cloud and equalizes the distribution of points throughout the cloud. This reduces the computation time highly while trying to keep as much important details as possible.
* Euclidean filter: This filter detects “blobs” of points. In order to differentiate objects from a point cloud, the filter uses the point’s proximity to other groups of points. It returns points with indexes for each group.

### Programming using the PCL

The PCL provides multiple tutorials for different use cases. Using the library is as straightforward and simple as it gets.

## My Program

In this part of the report, I’ll describe the program I’ve built. I’ll first clarify the aim of the program, then I’ll explain how the package is structured and finally, I’ll explain how the nodes work.

### Defining the goal of the program

The goal of the program is to pick an object with the GelSight mini. At first, I’ve built a program to work with the regular Panda gripper. I’ve used the MoveIt Grasp pipeline to do so. Then I had to use a 3D camera to grasp an object in any place in the workspace of the Panda. This implied making another node and using the PCL. Finally, I’ve added another node to implement the GelSight mini. By then end of my task, my program could automatically detect where an object is located, move to grab it, close the gripper progressively to grab the object softly and finally, lift it up and place it to a predefined location.

### My package’s structure

Down is the structure of the package I’ve written described:

GellSight

Figure : The Package's Structure

3D Camera

MoveIt commander

Automatic picker

GelSight server

Euclidean cluster

Package

The main node is the automatic picker. It handles the commands to the robot and receives all the data from the sensors. The Euclidean cluster node subscribes to the 3D point cloud, filters the point cloud and publishes the position of the grasp for the object. The GelSight server receives the frames of the camera. It publishes for each frame if a contact is detected and also has an action server in order to recalibrate itself.

Additionally, the package has launch files which are files designed to launch sets of nodes at once with given parameters. I’ve created two: one to launch controls, sensors and Rviz and a second to launch the automatic picker node and the two sensor nodes.

### The Euclidean cluster node

The Euclidean cluster node uses the point cloud and applies a passthrough filter, then a Voxelize filter and finally, a Euclidean cluster. After having applied those filters, with the biggest group of points, the code computes the center of the group and combines it with the maximum height of the group of points to output the grasp position.

It’s important to note that the position given will be in the camera’s frame. Before using the grasp position, it will have to be translated into the robot’s frame.

### The GelSight server

This nodes handles the GelSight camera with OpenCV. The camera has to be opened and filming the whole time. I experimented with only taking in frames when I needed to, but the node was too slow because of that.

The code of this node consists of one class. Inside the class is a loop which publishes information about the frames. The information sent is the difference between the current frame and a frame at rest. Additionally, the class has a calibration function which captures the frame at rest and computes the average difference caused by noise in the camera. This function is called whenever the action server receives a goal.

It' s important to note that a median filter is applied to each frames in order to reduce the camera noise.

## The Automatic picker

Finally, I will describe each steps the automatic picker node takes.

* First the robot has to get into its default state. This means each joint of the robot are set to a default value. Also, the gripper gets a “homing” action. Homing means the gripper will close and open to calibrate it and check the range of movement the gripper fingers can have. To send this homing action, I only had to send a goal to the Franka gripper controller directly.
* Then the robot waits to receive the grasp position from the Euclidean cluster node and creates a Grasp message. Grasps messages usually include a post grasp position which describes how after having grasped the object, the arm should lift it up. Because I couldn’t control how the gripper grasped using this pipeline, I’ve only used it to get the gripper in position to grasp without actually closing the fingers.
* After the GelSight’s calibration is done and when the fingers are in front of the object, they will close progressively until a deformation is detected in the GelSight. To do so, I close the fingers step by step. Each step, the program checks if a deformation is detected in the membrane in which case, it will stop closing. To implement this, I couldn’t continuously close the fingers and stop when the GelSight detects contact because stopping the fingers in their course could take more than a minute sometimes. This was too unpredictable to be used. Also, I have not used MoveIt to send the commands in order to close the fingers progressively as it was faster to send goals directly the Franka Gripper Controller and when iterating through many different positions, the difference in speed is felt quite heavily.
* When the object is grasped, the program uses the Place object pipeline to place the object to a given position.
* Finally, the robot gets back into the default state.

To conclude, this program can make the Panda pick any objects in any position in the workspace. The objects can be of multiple shape, but it can’t be too wide for the Panda to pick it up. Because of the limited vision of the 3d camera, the gripper is limited to grasping in the same plane of the camera. This would be fixed in further research allowing the Panda robot to “see” through touch giving it more different ways to grasp objects.

The program can sometimes fail, however. This is due to the inverse kinematics of MoveIt which can give results that are too close to joint limits and not optimal.

# Design of the Hand

As said before, I needed to design another hand to use the GelSight mini sensors and to allow the hand to better touch any object.

Une image contenant cercle, Graphique, graphisme, capture d’écran

Description générée automatiquement

For the entirety of the design of the Hand, I’ve used blender. Blender is a versatile, open-source 3D computer graphics software program. It encompasses a lot of tools for modeling, animation, rendering, compositing, and video editing. It notably has a node-based way of generating geometry and a python script combability, two tools which will be useful for our application. My supervisor and I decided to use blender after I’ve been using it to prototype the fingers.

In this chapter, I will describe the tools and the techniques I’ve used to design the hand for the Panda robot. To do so, I’ll first elaborate on the design of the joints, the phalanges, then move on to the fingers individually and finally on the hand.

## Design of a joint

The first step of designing the hand was to design a phalange, to do so I first had to make prototypes of the joints the finger will be using. The joints would have to be 3D-printable which added a bit of constraints to their actual designs, this also meant that the designs had to be kept simple. Here is a picture of the joint parts isolated:

Une image contenant salle de bain, toilettes, conception, noir et blanc

Description générée automatiquement

Figure 11: Picture of the joint

One joint is composed of two parts, the inner and outer. The outer part is really simple, it’s a portion of a ring. I’ve chosen it because it made the assembly of the phalanges easy. To assemble a joint, the outer part can be clipped onto the inner part. It’s important to note that because the assembly required the bending of the part, the printing parameters had to match. The inner part is a simple cylinder attached to a link for the rest of the phalange. Its radius is reduced where the joint’s cylinder contact is. The angle of the those tapered parts is chosen to reduce overhangs in order to print without supports. After some tuning, the joints were working great. The touching fingers were not going to be under a lot of force so the robustness of the fingers wasn’t a priority in my prototypes.

## Design of a phalange

After having designed the joints, I had to design a phalange. This design is pretty straightforward because it had to be kept simple. Also, as the phalange had some geometric parameters that had to be left open for optimization later, I had to use blender’s geometry nodes to generate their geometry. This is equivalent to coding the geometry of the finger. Here is a picture of the phalange.

Une image contenant conception

Description générée automatiquementUne image contenant conception, serviette en papier

Description générée automatiquement avec une confiance moyenne

Figure : Picture of a phalange

Each phalange is composed of an inner and outer joint part and of two ridges on each sides. Each ridge also has one loop. The phalanges have those ridges and loops to allow a tendon to fit in and be attached. For my prototypes, I’ve decided to use tendons to move the fingers because it was the simplest solution.

Une image contenant capture d’écran, Rectangle, conception, art

Description générée automatiquementUne image contenant Graphique, conception, symbole, Police

Description générée automatiquementUne image contenant conception, capture d’écran, Graphique

Description générée automatiquementUne image contenant capture d’écran, Rectangle, conception

Description générée automatiquementHere are the steps the geometry nodes use to generate the phalanges:

Figure 13: Process of generating a phalange.

1. A profile is extruded. In the picture the profile is a square to keep it simple and printable. The dimensions of the profile are set to fit the joint’s dimensions. The length of the extrusion is the length parameter of a phalange
2. A difference boolean is applied to the part to cut the part at the right angle. This is done so that the finger will be allowed to bend to the wanted angle.
3. Other difference booleans are applied to make place for the joints and for the ridges.
4. Finally, The joints parts are fused to the part and the phalange is finished.

## Design of a finger

Une image contenant jouet, plastique, intérieur

Description générée automatiquementThe fingers are made of 3 phalanges. One tendon would curl the finger by applying torque on its joints. This makes it so that pressure is applied on the sensor at the end of the finger and that the fingers will adapt to the geometry. The tendon would also be replaceable by a spring and a rope or string. In future applications, the fingers could also be motorized using a motor to wind up or down the string or even a material which’s deformation can be controlled. The tendon approach is inspired by how human’s hands are made. Here is a picture of a prototype of a finger:

Figure 14: Picture of a protoype of a finger

In this image, the last phalange is made to be able to hold the sensor. A rope is attached in one of the ridges. This prototype allowed me to test how the finger would behave when used. I’ve been able to observe that when un-curled, the joints rotate one after the other. The first joint to rotate is the first one of the finger (the one between the green and the blue phalange) later followed by the other joint when the first one can no longer rotate. This rule is important as it makes the trajectory of the finger when un-curling or curling up unique.

## Design of a Hand

The hand I have designed has two fingers for touch and optionally another finger to grasp. The touching fingers will be the one to be optimized. The reason why there’s only two touching fingers is because in the laboratory, there only were two GelSight mini at our disposal.

As mentioned above, the design of the hand was made using a genetic algorithm. This idea came up after a reunion with my supervisor where we’ve discussed about a paper he supervised written by him and his colleague from Sigma Clermont **[1].**

### The simulation

To optimize the geometry of the hand, I had to realize a simulation of the physics of the fingers. The aim of my simulation is to get the equilibrium state when the finger is applied a force and is also touching an object. Here’s how my simulation works:

Une image contenant croquis, noir et blanc

Description générée automatiquementUne image contenant croquis, conception, noir et blanc

Description générée automatiquement avec une confiance moyenne

Figure 15: Illustration of a finger touching an object

1. Discretize the different position of the sensor’s skin. This is represented by the many planes the skin will occupy.
2. Find in what position, the skin touches the object.
3. For each planes after that first touch position, see if the force applied to the skin by the object matches the force applied to the finger.
4. Repeat until equilibrium is found.

It’s important to note that even though the simulation aims to find the equilibrium of the forces, in certain configurations, one of the forces will always be higher or lower than the other because the finger is bent at its maximum or at its minimum.

After having completed all of these steps, the simulation returns the position of the finger and also the exact force applied to the sensors.

Also, the simulation needs to check for collisions. Collisions between two fingers and collisions between the fingers and the floor. Because the simulation was already computationally expensive, I couldn’t have a lot of precision in the collision checking. The fastest way was to simulate each of the phalanges’ collisions by spheres.

I will now proceed to explain more in detail how each steps works and the decisions that I’ve made which led up to it.

#### Discretization of the end effector’s position

To be able to run a simple simulation, I first had to discretize the possible position the end effector’s (EEF) position. It is controlled by the two angles of the two joints.

Because simulating each parts would be too computationally expensive, I’ve chosen to use the observation I’ve made earlier concerning the law on how the different joints reacts when forces are applied to the finger. This means that the trajectory the EEF will follow is unique, simplifying the problem.

Because this trajectory is discretized, this means that I can chose how many plane there is. This is a parameter of the simulation which affects the most performance and precision.

#### Find the position of the touch

Now, the finger needs to touch the object in order to get a counter force to the one applied to the finger. To do so, instead of computing the force received in the skin for each plane, I use a ray cast tool from blender from the object’s vertices to all of the planes to sea which planes are inside the mesh and which is the first plane inside the mesh. This returns the first plane to intersect with the mesh.

It’s important to note that the approach of ray casting from vertices of the mesh requires a mesh of sufficient definition.

#### Find equilibrium in the forces

In order to compute the force applied to the finger by the object, I need have a physical model of the silica of the membrane. The aim of this model is to compute the strain given the deformation of the skin.

To do so the skin is divided in many planes. Then, the stress in every plane is computed which will then be used to compute the strain of each planes using a Neo-Hookean for soft materials model. With the strain of each plane, I can get the sum of forces applied to the silica membrane. The idea of using Neo-Hookean physics and the parameters of the model came from a GelSight paper: [[2]](#RéférenceDocument)

The Neo-Hookean model is a widely used hyperelastic material model for simulating the behavior of soft, rubber-like materials. The model is particularly applicable to materials that exhibit large deformations while remaining elastic (meaning they return to their original shape after deformation).

Here are the basic Assumptions needed:

* The material is considered isotropic, meaning its mechanical properties are the same in all directions.
* The material is hyperplastic, which implies that its behavior is described using a strain energy density function (SEDF) that only depends on the deformation.

Here is how the Neo-Hookean model works:

* First the plane is projected on to the geometry which intersects with the skin. This is done so by the shrink wrap modifier provided by blender.
* In the resulting geometry, each vertices are only deformed on the z access making the computation of the stress much less complicated and making the matrix expression of the stress simple.
* Then the strain is computed using the following equation given by the Neo-Hookean model:

With the strain matrix, the stress matrix and the identity matrix.

and are both parameters of Neo-Hookean materials.

Because of the geometry of the problem,

This gives us:

This allowed me to get the force applied from the object to the EEF with any object but unfortunately, I was not able to test this model with real life experiences because we didn’t have the right sensors for it. This model was still useful however because it gave at the very least an approximation of the strain inside of the skin for the genetic algorithm.

### The genetic algorithm

To optimize the geometry of the hand, I’ve used a genetic algorithm. Genetic algorithms are machine learning algorithms inspired by how evolution works in nature.

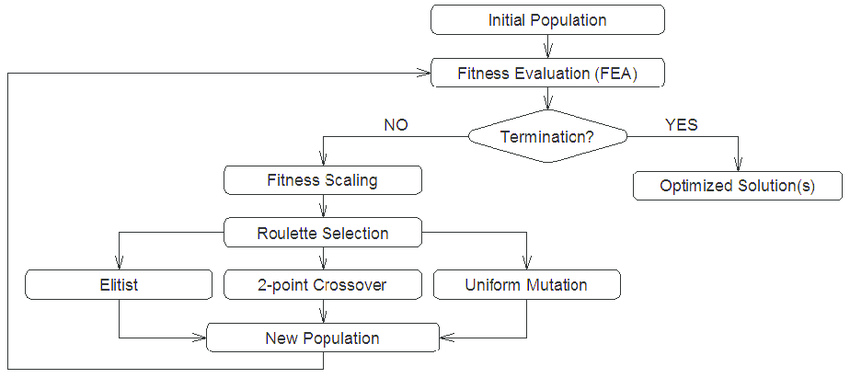


Figure 16: Genetic algorithm scheme by Research Gate

Here is how they work:

* First a population of random individuals is created.
* The fitness of each is evaluated.
* If the algorithm is not done, apply selection to the population.
* Make crossovers of individuals. This is comparable to two individuals having a child.
* Additionally apply Elitism and Mutation. Elitism is when the most fit individuals can be kept for the next generation where the others will not. Mutation is when random small changes are applied in each individuals of the next generation.

Genetic algorithms are particularly useful in problems where the solution space is large, complex, and poorly understood. They have been applied in a wide range of domains, including robotics.

Even though those algorithms can be powerful, they are not always the most efficient or suitable approach for every problem. The choice of optimization algorithm depends on the specific characteristics of the problem at hand.



To implement the algorithm in Python, I’ve used Pymoo [[3]](#RéférenceDocument). Pymoo is a Python library designed for multi-objective optimization. It provides a collection of genetic algorithms and other optimization techniques to handle problems. The name "Pymoo" stands for "Python Multi-Objective Optimization."

#### Definition

In this part of the report, I will define the optimization problem the algorithm will handle.

An individual consists of a hand made of two finger. Each individual is defined by 14 different parameters: 2 for the lengths of each fingers, 3 for the position in Cartesian space of each finger and 3 for the Euler angles of each fingers. This is a lot of parameters. I’ve simplified the problem as well as I could, but the algorithm will take more time to find correct solutions because of that.

The goal of the algorithm is to maximize two objectives, one for each finger. The way the finger’s fitness is evaluated is by how much information the sensor is capturing. This is made by computing the volume of the part that is inside the sensor.

A genetic algorithm can also be constrained. I’ve constrained the problem with 4 inequalities and 3 equalities. I differentiate between inequalities and equalities because the latter are a lot more complicated to handle for genetic algorithms in general. There are 2 inequalities for each finger, each for the lower and upper bound of force allowed on the sensor in order to touch. The remaining 3 equalities are for collision detection, one between the two fingers and one for each finger for itself and the floor.

This is the optimization problem the algorithm has solved. It’s far from an easy problem to compute. This explains the many simplifications I had to make for computational cost.

#### Training data

To train the finger to touch, it was evaluated on objects from the Yale-CMU-Berkeley dataset for robotic manipulation research [[4]](#RéférenceDocument). This database is composed of many different day-to-day objects such as apples and tuna-cans.

To import them into blender, I first had to change their geometry as their geometry were not good. Their geometry had holes and noise which I had to fix.

For the training process. In order to reduce overfitting, every fitness evaluations would be done on a different object. This made it so that the genetic algorithm didn’t have time to settle on a solution that’s only good for one object in particular as described in this paper from the Journal of physics [[5].](#RéférenceDocument)

#### Parameters

For my application, I’ve used the Non-Selective Genetic Algorithm (NSGA II). This choice has been made because it is the standard of multi-objective genetic algorithms.

Genetics algorithms (GAs) are hugely dependant on their parameters. It affects algorithm's behavior, convergence speed, and overall effectiveness in finding optimal solutions. Here are some of the key parameters in a GA and their importance:

* Population Size: The number of individuals in each generation. A larger population size can lead to a more diverse set of solutions and increase search space. However, it also increases computational cost.
* Crossover Rate: This parameter determines the likelihood of two individuals undergoing crossover. A higher crossover rate makes for more exploration of the solution space.
* Mutation Rate: The probability of a mutation occurring in a given gene. Mutation introduces random changes in the genetic information. This reduces the risk of finding a local optima. It’s important to note that there exist multiple mutation algorithms which all behave differently.
* Selection Mechanism: The method used to choose individuals for reproduction. The choice of selection mechanism impacts the diversity of the population and the convergence speed of the algorithm.
* Termination Condition: This specifies the stopping criterion for the algorithm.

In order to choose those parameter’s, I’ve manually fine-tuned them. I’ve found that a population size of 100 is generally needed to keep a bit of diversity. I’ve chosen to use polynomial mutation and a termination when achieving the 600th generation. The algorithm always converges before the 600th generation.

### Results

In this part I’m going to discuss the results the genetic algorithm returned while also elaborating on the process of running the algorithm.

At first, the algorithm wasn’t returning correct results. The code had holes the algorithm could exploit to get around the constraints I had established. For example, there was a way the skin could be enormously deformed without returning incoherent forces. This was due to the fact that the skin could completely detach from the rest of the finger without any opposing forces. After having fixed this, tuned the constraints of the algorithm and restrained the search space, the algorithm returned promising results.

Une image contenant croquis

Description générée automatiquement

Figure 17: First results of the algorithm

Those results were optimized but there was on issue. This configuration of two fingers cannot be printed. The two fingers overlap and one is over the other. If I needed to design a hand with two fingers in this pose, the 3D printing process would be slow, cost a lot of filament and require a lot of supports. Also, because the two fingers overlap, there’s another issue because in this case the two fingers do not collide but for different geometries, they might.

To fix that, I have found two possibilities. I could prevent the two fingers from overlapping by adding other constraints to the algorithm or I could restrain the algorithm’s search space further until overlapping fingers are no longer possible. I have chosen the latter and added some bias to the algorithm.

To add this bias, I took the optimized but unprintable configuration and moved the fingers as little as possible to stop them from overlapping. I then reran the algorithm with multiple of those as the starting population and got this result:

Une image contenant Police, symbole, blanc, typographie

Description générée automatiquementUne image contenant croquis

Description générée automatiquement

Figure 18: Final results of the algorithm

Those result were far better for 3D printing so I’ve designed the hand to link both fingers together and I added a support to fix it the Panda gripper. Here is how it looked in real life.

Figure : The 3D printed hand

Une image contenant texte, roue, boisson, intérieur

Description générée automatiquementUne image contenant câble, machine, Électroménager, intérieur

Description générée automatiquement

I had 3d printed the two fingers, the support and the hand, attached elastic bands to the fingers so they would curl up as expected and attached it to the robot with the support. Note that the Panda arm has to approach differently than with the default gripper. This was the last thing I’ve accomplished during my internship.

# Conclusion

In conclusion, my internship abroad in the robotics laboratory has been an invaluable experience. I’ve not only gained a lot of technical knowledge in robotics, but I’ve also expanded my view of innovative technologies and of collaborative work environments.

This opportunity has reinforced my passion for robotics and provided me with solid experience which will with no doubt be of use to me later. In one hand I’ve gained in skills with programming for robotics and in the other I’ve learned to work in research. Notably, I’ve reinforced my knowledge in C++, ROS, OpenCV and python.

To summarize, by the end of the Internship, I’ve delivered the following to the CiTIUS:

* A MoveIt package which allows the robot to detect, grasp, and place any object in its workspace. In this package there are also tools to calibrate the 3d camera and a text file to guide the next colleagues who are going to work with it.
* A blender file containing the code to generate the entire robot hand and to run the genetic algorithm in order to optimize the hands geometry.
* Multiple 3d printed objects such as the final printed Hand and supports to hold cameras.

I am grateful to have been able to work under these conditions and to have been able to apply both the knowledge taught in Sigma Clermont and the knowledge I’ve gained during my hobbies. I also hope that my work will be useful to my colleagues in CiTIUS.

Bibliography

[[1](#RéférenceDocument)] **A. Pagoli, F Chapelle, J Corrales, Y Mezouar, Y Lapusta.** *Design and Optimization of a Dextrous Robotic Finger: Incorporating a Sliding, Rotating, and Soft-Bending Mechanism While Maximizing Dexterity and Minimizing Dimensions.* Institute of Electrical and Electronics Engineers (IEEE), 2020. <https://ieeexplore.ieee.org/abstract/document/9217429>

[2] **W. Yuan, S. Dong, E. Anderson** *GelSight: High-Resolution Robot Tactile Sensors for Estimating Geometry and Force.* Computer Science and Artificial Intelligence Laboratory (CSAIL), <https://www.mdpi.com/1424-8220/17/12/2762>

[3] **J. Blank, K. Deb** *Pymoo: Multi-Objective Optimization in Python.* Computer Science and Artificial Institute of Electrical and Electronics Engineers (IEEE), <https://www.researchgate.net/publication/340976217_Pymoo_Multi-Objective_Optimization_in_Python>

[4] **B. Calli, A. Singh, J Bruce, A Walsman** *Yale-CMU-Berkeley dataset for robotic manipulation research.* Yale University, https://www.researchgate.net/publication/315926867\_Yale-CMU-Berkeley\_dataset\_for\_robotic\_manipulation\_research

[5] **X. Ying** *An Overview of Overfitting and its Solutions.* Journal of Physics, https://iopscience.iop.org/article/10.1088/1742-6596/1168/2/022022