Arts Sciences et Technologies et Métiers

2023

MASTER THESIS

Development and Control of a Robotic Arm for Precision Makeup Application on the Human Face

SU Sichen

L'ORÉAL Recherche & Innovation

BICHON, Jean-Christophe tuteur industriel

Supervisor BICHON Jean-Christophe, BA Sileye, LI Tao

BEAREE Richard tuteur pédagogique

NOTICE BIBLIOGRAPHIQUE

YEARS: 2022-2023 **N°:** LI-2022/12/11556

DOCUMENT TYPE: Internship Report, Master thesis

ATTACHED CAMPUS : Arts et Métiers Lille

AUTEURS: SU Sichen

TITLE: Development and Control of a Robotic Arm for Precision Makeup Application on the Human Face

SUPERVISOR: Prof. BEAREE Richard

PARTNER COMPANY: L'Oréal Recherche & Innovation

NUMBER OF PAGES: 53

NUMBER OF BIBLIOGRAPHIC REFERENCES: 9

SUMMARY: The project aims to develop a robotic assistant capable of applying makeup on the human face. The objective of this internship is to establish a framework for the control of the robot, using cosmetic products to perform makeup on different parts of the human face. In this project, we rely on the Robot Operating System (ROS) to develop this framework. This report will specifically present the following work: •Definition and architecture of the MURA project (Make-Up Robot Arm). •Construction of a simulation environment within the ROS framework. •Development of control tools for a robot, camera calibration, and hand-eye calibration. •Computer Vision part, using MediaPipe to perform face detection and extract characteristic points. In addition, based on MediaPipe, we have also achieved virtual makeup. •Control of the robotic system, using forward and inverse kinematics to deploy servomotors in real-time and plan a more efficient trajectory. •Use of reinforcement learning to improve the performance of the robotic assistant in makeup application.

KEYWORDS: Robotic assistant, Collaborative robot, Real-time robot control, Computer vision, Force control, Machine Learning, ROS.

Catalog

1. INTRODUCTION

1.1 Company Presentation

L'Oréal Research & Innovation is the research and development department of L'Oréal, L'Oréal is the world's largest cosmetics company. L'Oréal Research & Innovation plays a key role in the creation and development of new beauty and hair care products. Their main goal is to continually innovate to meet the needs and expectations of consumers worldwide.

L'Oréal Research & Innovation has more than 4,000 researchers, including chemists, biologists, computer vision experts, and artificial intelligence specialists. Their mission is to develop new products for L'Oréal. They are consistently at the forefront of technology, exploring fields such as augmented beauty using optical techniques, computer vision, and machine learning. These technological advances facilitate the rapid development of innovative products in the beauty industry. The laboratories are equipped with cutting-edge tools and employ sophisticated techniques to explore new areas such as genomics, cell biology, laboratory automation, computational modeling, and bio-imaging.

In summary, L'Oréal is a major player in the cosmetics industry, discovering new technologies, ingredients, and formulas to provide high-quality beauty products to consumers worldwide.

1.2 Internship Context

During my internship, I worked on a robotic project called MURA, which stands for "Make-up Robotic Arm." This project was the result of a collaboration between the following teams:

- The AI team
  Sileye BA, senior machine learning scientist, responsible for artificial intelligence section.
  Sichen SU, intern in AI and robotics, responsible for robot control.
- The computer vision team
  Tao LI, senior computer vision engineer, responsible for computer vision direction.
  Vaibhav ELANGOVAN, engineer in robotics and computer vision.
- The mechatronics team.
  Jean-Christophe BICHON, my professional tutor and also the project manager of MURA.
  Florian Dupuis, an R&D engineer in mechatronics, responsible for project management.

The MURA project aimed to develop a platform with a robotic arm capable of applying makeup to the human face. In this platform, we wanted the robot to perform the following tasks:

- **Human face detection and identification of areas of interest.** The system used images to generate a mesh of the human face, providing geometric information such as pixel position and head orientation in the coordinate system.
- **Robot trajectory planning**. Generating a trajectory based on the customer's face mesh, allowing the application of different cosmetic products.
- **Robot compliance control.** Controlling the force and torque exerted by the robot to avoid injuring humans; the system must precisely follow a predefined trajectory and perform force control.

1.3 Problem Statement

Robots have tremendous potential to assist people in daily life. In our project, we aim to develop a robotic system capable of automatically applying makeup, particularly for disabled people, and on the other hand, we want to perform a task to test makeup products on a model head.

This project presented us with both exciting and complex challenges, such as robot motion planning, robot dynamics control, and physical interaction, among others. To test different algorithms and various modules, the first task we need to address is the construction of a simulation environment. It is in this environment that we can develop and test algorithms.

In this environment, it must contain basic elements: a manipulator robot and a human model with appropriate physical properties. In addition, we need to add sensors such as RGBD cameras, force sensors to detect data. In this environment, we also need to perform makeup on the human model to evaluate the quality of the makeup.

Then, once the simulation environment is built, we need to develop a computer vision module. This module uses an RGBD camera to acquire 3D information. We also need to develop an algorithm to detect the human face and provide Mesh information to determine the position of key points. Additionally, this module must calculate head orientation in real time. Finally, this module must provide me with a UV map so that I can apply makeup to the area of interest.

Another challenge is the robot control module. This module must rely on inverse and forward kinematics. In this module, we can perform collision definition, robot motion model definition. It can accurately calculate robot configurations to perform a movement.

The final challenge concerns the establishment of a framework for reinforcement learning. Our goal is to achieve more precise control, and reinforcement learning represents an excellent way to achieve safer and more precise robot control. We want to develop a framework that allows the robot to execute specific actions, and based on the actions performed, rewards will be assigned. Additionally, this framework will also be able to effectively evaluate the robot's control policy.

1.4 Report Structure

In this thesis, we will present the evolution of the MURA project. Firstly, we will detail the MURA pipeline. Then, we will discuss the construction of the simulation environment before addressing the development of the control tool and calibration processes. Subsequently, we will talk about the computer vision module and explore the various functions we can perform using information from the camera. Additionally, we will explain how to generate a more efficient trajectory. Based on our initial system, we will then show how to achieve more optimized and precise control. Finally, we will conclude on the MURA project.

2. MURA Project Pipeline

The MURA project aims to develop a robotic system, assisted by computer vision and artificial intelligence, to achieve automatic makeup application. The goal is to test new products on a model of a human head, and ultimately, to deploy this system on real people, allowing physical interaction between the human and the robot.

To successfully carry out this mission, we have established a pipeline to guide our project. This pipeline involves two essential preparation tasks: the calibration of the camera and that of the robot. The calibration parameters are crucial to ensure the accuracy of the robotic system. Once all calibrations are performed, our goal is to set up a control of the robot allowing it to reach a specific position to identify the areas of interest. In this regard, we rely on computer vision to detect the lips and perform virtual makeup on them. When the client or operator chooses a lipstick color, the robot follows a previously generated trajectory to accomplish the makeup application.

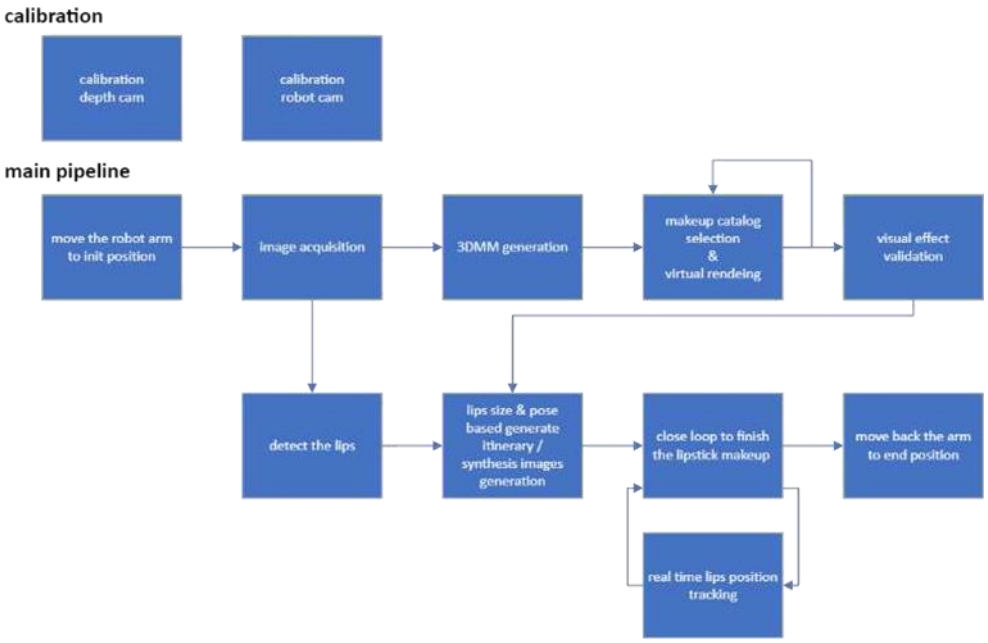


Figure 1 MURA project Pipeline

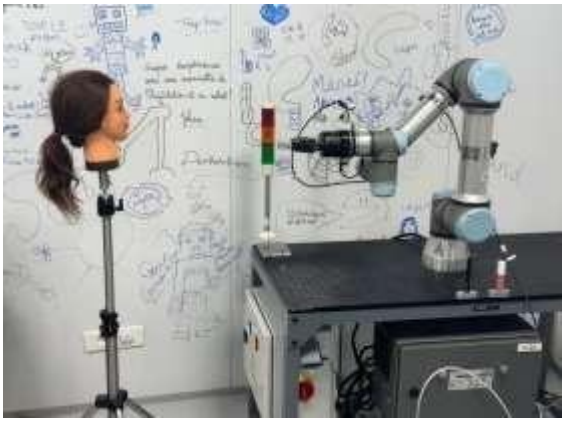


Figure 2 mura system

In our system, we have integrated a UR5e robot, a RealSense D415 camera, a Robotiq gripper, as well as various makeup products. Our system is divided into three modules. If we were to test all these modules simultaneously on a single robot, it could lead to waste. Therefore, we decided to establish a simulation environment to test all the algorithms associated with our project.

This simulation environment allows us to validate the functionality of each module individually before integrating them into the real robotic system. It also provides a safe and controlled

setting to experiment and optimize our algorithms, ensuring they work as expected and reducing the risk of potential errors or accidents when applied to the actual robot.

3. Simulation Environment Construction

3.1 Context and Issues

Context

Technological development in robotics in recent years has made robots an integral component of various industries and applications. To ensure the success of the deployment and operation of robotic systems, it is crucial to develop and test them in realistic and dynamic environments. However, constantly testing a robotic system in a real environment during the development process can lead to increased development costs and risks of damage to the robot. To avoid these issues, creating a suitable simulation environment for robotic systems is an essential component of the development process.

Simulation environments play a crucial role in the development and evaluation of robotic systems. They offer a cost-effective and safe way to prototype, test, and validate the behavior of robots before their deployment in real scenarios. Simulations enable researchers, engineers, and developers to design and optimize algorithms to model robot dynamics and assess their performance in a controlled and reproducible environment.

In the field of robotic simulation, several popular software tools are available, such as CoppeliaSim, Gazebo, Pybullet, and MUJOCO. For our project, we have chosen Gazebo, which is a powerful and widely-used open-source simulation environment for research and development in robotics. Gazebo offers a physics engine, sensor simulations, and realistic rendering capabilities. It enables the simulation of complex robotic systems, including multiple robots and dynamic environments. Additionally, Gazebo integrates with ROS (Robot Operating System), facilitating communication between simulated robots and other ROS-compatible components. Gazebo is also highly extensible, thanks to a large community that contributes to the development of plugins and models for various robots and sensors.

Issues

The issues involved in creating a robotic simulation environment for the deployment of our Automated Robotic Make-Up (ARM) project encompass several essential questions that need to be resolved:

- Creation and Display of a Robot Model
  This involves defining and creating a digital representation of the robot that is accurate enough to reproduce its movements and interactions in the simulation environment. This implies modeling each part of the robot, including its joints and actuation mechanisms. The simulation interface must also be able to visually display the robot model, allowing intuitive interaction with users.
- Construction of a Simulation Environment:
  A realistic simulation environment must be created to test the robot. This must include all the elements with which the robot will interact, such as the human face on which the makeup will be applied, makeup accessories, and any other elements in the environment that could affect the robot's performance. The environment must also be dynamic,

allowing for the simulation of different lighting conditions, facial movements, and various skin types.

- Implementation of Interaction Between the Robot Model and the Simulation Environment: It is crucial to program the robot model so that it can interact realistically with the simulation environment. This means that it must be capable of perceiving the environment, performing actions, and receiving feedback on these actions. For example, the robot must be able to detect the position and shape of the face, apply makeup with the correct pressure and precision, and receive visual and tactile feedback on the result. The simulation must also be capable of reproducing the effects of these actions on the environment, for example, how the makeup is applied to the face.

These challenges must be addressed while considering performance and resource constraints, ensuring that the simulation environment is sufficiently flexible and scalable to allow for future improvements and adaptations.

3.2 Methods Used to Address the Issues

Context

In the context of our automated makeup robot project, we have based our work on ROS, the Robot Operating System. ROS is a flexible framework designed to facilitate the development of software for robots. It provides a set of libraries and software tools that aid in building robotic applications. The platform allows for efficient code sharing and reuse, which simplifies the work for developers. It also includes a variety of software libraries for various robotic functionalities ranging from trajectory planning and perception to control. Additionally, ROS is compatible with a wide range of robotic systems and sensors.
For this project, we are using three main software tools: Gazebo, Rviz, and MoveIt.

- Gazebo is a 3D dynamic simulator. It is designed to display robot models and create accurate and efficient simulation environments. These environments simulate robots in complex indoor and outdoor settings. Like game engines that provide high-fidelity visual simulations, Gazebo offers high-fidelity physical simulations, a comprehensive set of sensor models, and user- and programmer-friendly interaction.

Figure 3 GAZEBO

- Rviz is ROS Visualization Tool primarily aims to visualize ROS messages in three dimensions, providing a visual representation of data. For example, it can display robot models, show the distance between a sensor and an obstacle, or visualize point cloud data from 3D distance sensors like RealSense, Kinect, or Xtion, as well as image values from cameras.

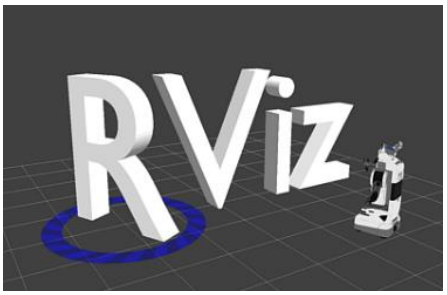


Figure 4 Ros Visualisation Tool (Rviz)

- MoveIt is open-source software designed for motion planning and manipulation tasks in robotics. It offers a comprehensive set of tools, libraries, and APIs for manipulating a robot's arm, trajectory planning, collision control, and trajectory execution. MoveIt is widely used in the robotics community to simplify the development and deployment of robotic systems. It is designed to work with a variety of robotic platforms, supporting industrial and research robots, controlling different types of robotic arms, including serial and parallel manipulators, mobile robots, and even humanoid robots.

Figure 5 Moveit Motion Planning application

Modelling of Robot and End Effector

- Modelling of Robot model

  In the context of the MURA project, we used a UR5e robot and a Robotiq 2F-85 gripper. Consequently, based on these two types of equipment, we created URDF files to describe the properties of the robot and the gripper.

  URDF (Unified Robot Description Format) is a format based on the XML language that allows for the description of a robot's structural properties. It enables the definition of the robot's links and joints, adding physical properties, and integrating various plugins for sensors.

Figure 6 Robotiq Gripper 2F-85

Figure 7 UR5e collaborative robot

The URDF allows for the description of a robot in terms of elements such as:

- Links: These are the rigid parts of the robot. For example, in a robotic arm, each segment of the arm would be a link.
- Joints: These are the movable connections that link the links together. Joints can have different types of movement (for example, rotating or sliding).
- Sensors: Information about the robot's sensors can also be included in the URDF. For example, a robot could have vision or force sensors.
- Physical properties: It is also possible to include information about the physical properties of the links, such as their mass and center of gravity.

Indeed, using URDF saved us from having to take manual measurements. Therefore, our main tasks were as follows:

- Build of a Common Xacro File

Xacro (XML Macros) is an XML preprocessing language for URDF files that allows for the creation of more readable and maintainable robot models by defining and using macros that encapsulate URDF patterns. This process facilitates the management of URDF files, especially for large robot models.

In our project, we had two URDF files, one for the UR5e (ur5e.xacro) and the other for the Robotiq 2F-85 (robotiq_2F_85.xacro). We created a new xacro file to group them together, and then imported these two files into this new file.

To simulate the physical connection between these two objects, we also defined a "coupler" in our xacro file. We listed the physical properties of this coupler, including the characteristics of the links and joints. By defining this link, we were able to effectively connect the gripper to the robot in our simulation.

This means that in our xacro file, we defined how the Robotiq 2F-85 is physically connected to the UR5e, specifying where and how they are attached. By grouping these two components in a single xacro file, we created a comprehensive representation of our robotic system that can be easily used and modified in our simulation.

- Adding Gazebo Plugins

Once the description of the robot's physical properties was complete, we needed to add plugins as well as define collision and inertia properties. Indeed, as Gazebo is a simulation environment, collision detection is a necessary feature. Defining collision properties provides the basis for collision detection in the simulation.

In addition, adding the "inertial" tag defines the inertia matrix of a rigid part of the robot, which is used in certain simulations related to mechanics. This allows us to more accurately simulate the dynamic behavior of the robot.

We added a control plugin for the Robotiq Gripper 2F-85. This plugin allows us to control the gripper through our software system or simulation environment. In our configuration, we defined two links of the gripper ("gripper_link") and a link representing the palm ("palm_link"). These links specify the components of the gripper in the Gazebo simulator.

Finally, we added a force-torque sensor to the robot's wrist (at joint 3). This sensor allows us to measure the forces and torques applied to the robot's wrist, which is essential for performing delicate manipulation tasks, such as applying makeup. This also ensures that our robot does not apply excessive force during interaction with the user, thus ensuring their safety.

- Launch Files Building

Once all the robot's properties are correctly defined and the necessary plugins are added, we need to create launch files. These are XML files that allow us to start ROS nodes in an organized and coordinated manner.

In our project, we created a launch file to start the Gazebo simulation node with our UR5e robot and the Robotiq gripper. This launch file loads the robot's URDF model into the ROS server parameter and launches the Gazebo simulation node. It also launches the robot's control node so that we can send commands to the robot from our ROS code.

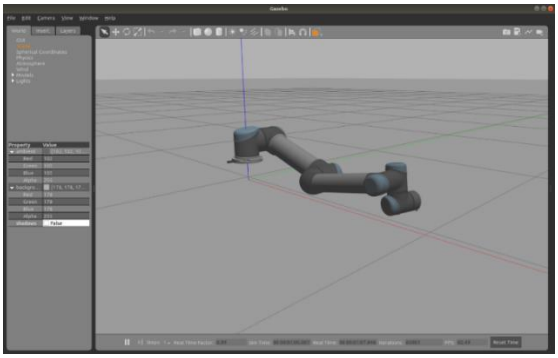


Figure 8 UR5e simulation model

Construction of Model of the RealSense Camera D415

In the context of our project, the computer vision module is of crucial importance. Therefore, integrating at least one camera into our simulation environment is essential. We opted for the RealSense D415 camera model, capable of providing color (RGB) and depth images.

To integrate the RealSense with ROS, it was necessary to install certain dependencies. Following that, we designed a xacro file for our camera module. We used the Gazebo RGBD camera plugin, inspired by the Kinect camera model, adjusting only the camera's performance to align with that of the RealSense D415. Additionally, we incorporated Gaussian noise into our images to simulate realistic conditions and increase the robustness of our system.

After defining the camera's properties, we integrated it into our main Xacro file by creating joints to connect it to the robot. This integration provides us with a camera perspective that reflects what the robot perceives in the simulation, which is crucial for the success of our automatic makeup application tasks.

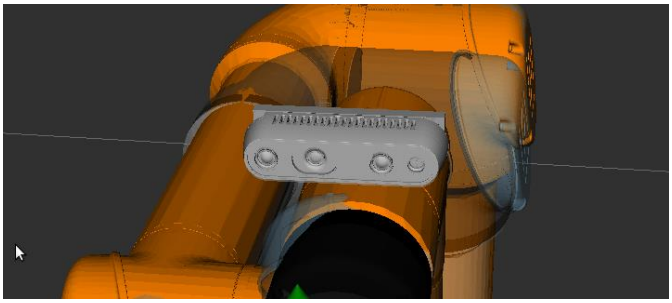


Figure 9 RealSense Model D415

Adding Force Sensors

According to the project's requirements, we aim to implement force control; therefore, a force-torque sensor is necessary for our simulation environment. In our case, we have added a force sensor at wrist 3. It can provide us with force and torque output in the x, y, z directions.

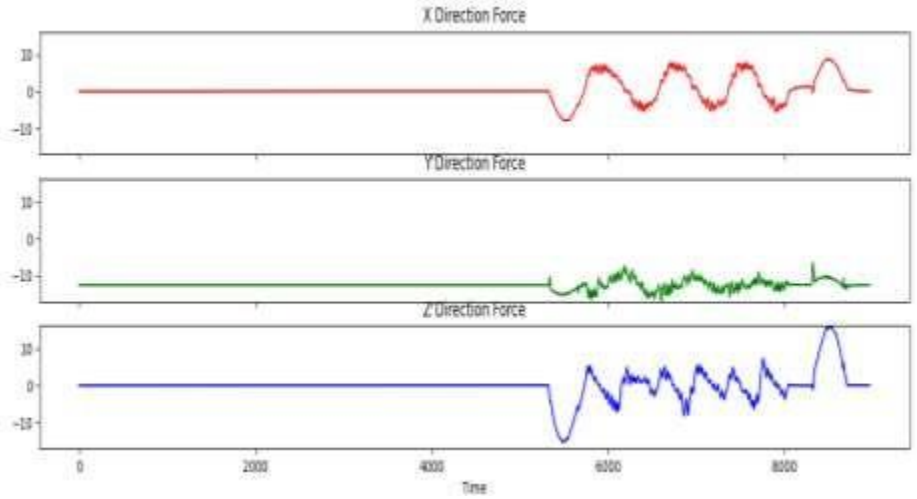


Figure 10 Force capteur

Construction of Gazebo World

When we finished building the robot model, integrating the RGBD camera, and the force-torque sensors, we have a complete robotic system. It is then time to start constructing a workspace to set up our robot. Based on our real workspace, we created a "world" file to describe our working environment.

In this space, we have a table on which to place the robot, as well as a small cylinder to represent a lipstick. Moreover, we wrote an SDF (Simulation Description Format) file to add a 3D model of a human being. This human model has its own 3D mesh and skeleton, as well as a texture to depict its face.

With these elements, we can create a realistic simulation environment for our automatic makeup robot. This allows us to test our robot in a variety of situations, ensuring that it operates correctly before deploying it in a real environment. This is a crucial part of our robot's development process, as it allows us to address any potential issues before actual deployment.

In addition to the other elements of the simulation environment, we have also added a chessboard. It is often used in computer vision for camera calibration. The calibration board is typically printed with a specific pattern, often a chessboard grid, which allows for precise camera alignment and internal parameter adjustment.

Camera calibration is a crucial step in preparing our computer vision system, as it helps to correct image distortions caused by the camera lens and provides accurate information about the position and orientation of objects in the image. Once the camera is calibrated, we can use this information to obtain an accurate 3D image of the world, which our robot can use to effectively interact with its environment.

By adding this calibration box to our simulation, we can verify and adjust the camera's calibration parameters before deploying it in the real world. This helps ensure that our computer vision system will operate accurately and efficiently in reality.

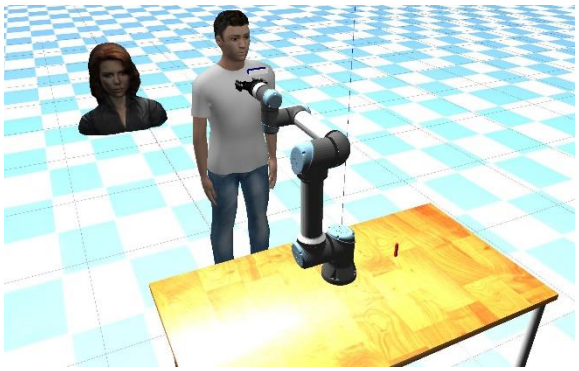


Figure 11 MURA projet Robot Simulation Workspace

Creating a MoveIt Configuration for UR5 and a Gripper

The integration of MoveIt! into our project has added the capability for complex motion planning to our robot. MoveIt! is a powerful software that provides a high-level interface for motion planning, object manipulation, 3D perception, kinematics, control, and navigation. It is capable of handling both inverse and forward kinematics, collision detection, and many other important aspects of robotic motion control.

In the context of our project, we first imported the URDF that we created for our robot into MoveIt. This allowed MoveIt! to generate a configuration for our specific robot. With this configuration, we were able to check the collision state between the different parts of our robot, which is crucial to prevent potential accidents.

Next, we defined planning groups for our robot, ranging from the base link ("base_link") to the end-effector link ("ee_link"). These planning groups enable MoveIt! to plan and control the movements of different parts of our robot.We also added a group for the gripper, allowing us to precisely control its movements.

After that, we defined the initial position of our robot as the starting position for all our motion planning. In addition, we added definitions for the open and closed states of our gripper.

To complete the setup, we defined the passive joints for our gripper and added the necessary joint controllers. Finally, we generated the configuration package for our robot, allowing us to implement precise and complex control of our robot through MoveIt!

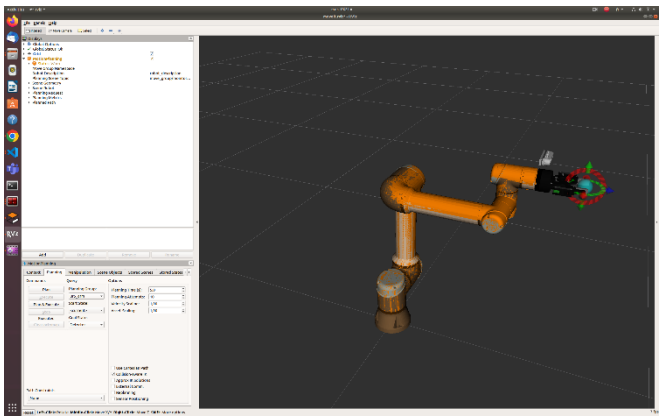


Figure 12 Moveit Package robot configuration

3.3 Obtained Results

After completing the construction of the simulation environment, we established a basic setup. In this environment, a UR5E robot equipped with a Robotiq gripper is mounted on a table. On this same table, a lipstick model is placed next to the robot. Additionally, our robot is equipped with two cameras: one is integrated into the gripper, while the other is mounted on the robot's head. Regarding our human models, we have two human models, both meticulously designed with proper collision properties.

This setup provides a comprehensive environment for testing and developing various functionalities of our robot, particularly in the context of automated makeup application. The integration of cameras allows for real-time monitoring and interaction with the environment, ensuring that the robot can accurately perceive and respond to its surroundings. The human models with collision properties enable safe interaction, ensuring that the robot can operate near a human face without causing harm.

4. Robot Control

4.1 Context and Problem Statement

Contexte

Following the creation of our simulation environment, the next steps involve controlling our robot to perform several simple movements. For example, performing inverse kinematic control, forward kinematic control, as well as gripping and placing actions (picking and placing). Additionally, we planned various trajectories suitable for the UR5 robot, including linear and curved trajectories, executed at a constant speed. To achieve this, we had to define different motion planning methods, such as "move_j", "move_p", and "move_l", in order to meet the specific needs of our UR5 robot.

Problem

After generating the configuration for our robot model, we were able to obtain the planning groups using MoveIt. The Rviz tool allowed us to direct the robot to reach the predefined poses. Although we had the option to define goals through the Rviz interface, this method did not allow us to obtain sufficiently precise control. Consequently, the challenges we faced include the following issues:

- How to use the MoveIt API to perform robot kinematic control, i.e., how to pilot the robot using Python or C++ code, to achieve control via forward kinematics and inverse kinematics.
- How to perform gripper control to open and close the end effector.
- How to perform collision avoidance, i.e., how to add collision objects in MoveIt.
- How to perform motion planning, i.e., how to develop a trajectory planning. When we pilot a robot, we want our robot to move along a predefined trajectory with constant speed and constant acceleration.

4.2 Methods Used to Address the Problem Statement

Context

1. **Robot Kinematics**

In the field of robotic control, we have two main types of control methods: one is called forward kinematics, and the other is known as inverse kinematics. In our project, we used both types of methods.

- **Forward Kinematics**

Forward kinematics is a process that determines the position and orientation of the end effector (the part of the robot that directly interacts with the environment, such as a gripper or "hand") based on the values of its joints.

In simpler terms, given the angles of each joint of the robot, forward kinematics allows us to calculate the exact position and orientation of the end effector in the workspace. So, it is a function that takes the angles of the joints as input and gives the position of the end effector as output.

$$Input = [\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6]$$

Equation 1 Forward Kinematic Input

- **Inverse Kinematics**

Inverse kinematics, as the name suggests, is the reverse process of forward kinematics. Instead of calculating the position of the end effector from the joint angles, inverse kinematics calculates the required joint angle values for the end effector to reach a specific position and orientation.

This task is generally more complex than forward kinematics because there can be multiple possible solutions (multiple joint configurations that allow reaching the same end effector position), or no solution at all if the target position is out of the robot's reach.

$$Input = [x_{end}, y_{end}, z_{end}, R_{end}, P_{end}, Y_{end}]$$

Equation 2 Inverse Kinematic Input

These two types of kinematics are essential in programming and controlling robots to perform precise and complex tasks.

2. Robot Trajectory Planning

Indeed, once we are capable of performing inverse or direct kinematic control, we must define the different types of movements for our robot. In practical terms, and more particularly for UR robots, there are four main types of motion planning: Joint Motion (DéplacementJ), Linear Motion (DéplacementL), Path Motion (DéplacementP), and Circular Motion (DéplacementC). In our case, we will focus only on the first three.

- **Joint Motion (Move_J)**

Joint Motion, or Move_J, is a type of movement where each joint of the robot moves independently of the others to reach the target position. This means that the joints can reach their final position at different times. This mode of movement is often used when the precise trajectory of the end effector is not important, but only its final position and orientation.

- **Linear Motion (Move_L)**

Linear Motion, or Move_L, is a type of movement where the robot's end effector moves along a straight line between its initial and final positions. To achieve this, all the robot's joints must move simultaneously and in a coordinated manner. This mode of movement is often used

when the trajectory of the end effector is important, such as when handling delicate objects or assembling parts.

- **Path Motion (Move_P)**

Path Motion, or Moth_P, is a command that moves the robot's tool linearly at a constant speed. The trajectory is smoothed by circular arcs at each corner, ensuring constant speed throughout the movement. This is particularly useful for process operations such as gluing or dispensing where constant speed and smooth motion are required to ensure work quality.

Robot Kinematics and Robot Motion Planning

- **Robotique Kinematics**

To perform robotic kinematic control, we relied on MoveIt. MoveIt is a software library used for motion planning and robot control, offering APIs in C++ and Python. Using MoveIt, we can easily implement a robot with both forward and inverse kinematics.

Inverse Kinematics: For inverse kinematics, we utilized MoveIt's predefined configurations to command the robot. We used the ROS Pose message format to define the target information in 3D. In this context, the input format is the position (x, y, z) and the quaternion (x, y, z, w). After setting the target, we fed this pose to MoveIt's "set_pose_target" function and used the "plan()" command to perform inverse kinematics.

Forward Kinematics: As for forward kinematics, the procedure is quite straightforward. We simply define the desired joint angles and use the "set_joint_value_target" function. Then, using the "get_current_pose" command, the robot will reach the target position.

MoveIt provides a simplified API for robotic kinematics, facilitating easy control.

- Robot motion planning

In the realm of robotic motion planning, we aim to implement motion methods that stand out from those used by conventional robots. To this end, we chose the RRT (Rapidly-exploring Random Tree) algorithm within MoveIt, capable of generating both safe and efficient trajectories for the robot. Additionally, MoveIt allows us to directly define the robot's acceleration and speed. More specifically, in our case, we defined the functions move_J, move_P, and move_L for the MURA project. Moreover, we also set collision boundaries to ensure increased safety in the robot's movements.

Move_J: For Move_J, we relied on forward kinematics. We simply provide the angle values for each joint, and the robot moves to the target position at a constant speed and acceleration.

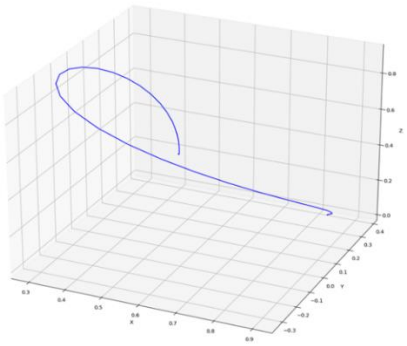


Figure 13 Move_J Trajectoire

Move_P: As for Move_P, it is a position-based movement using inverse kinematics. We simply provide the target position information, and the robot follows a safe and efficient trajectory to these points at a constant speed.

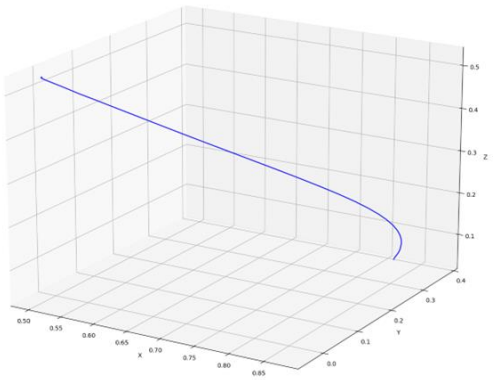


Figure 14 Move_P Trajectory for take a lipstick

Move_L : Move_L is also a position-based movement using inverse kinematics. In this command, we can provide not just a target point, but also multiple waypoints. The system then performs several iterations to generate a linear trajectory at a constant speed and acceleration.

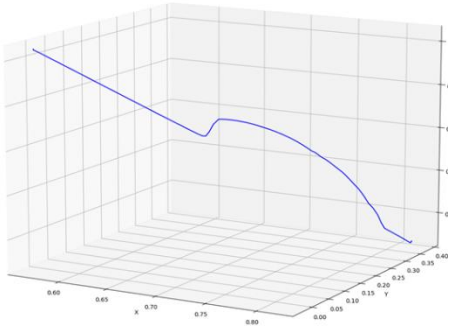


Figure 15 Move_L Trajectory for take a lipstick

Collision: When conducting motion planning, especially for move_P, we discovered some dangerous and undesirable motions. Also, to simulate our real environment in our simulation, we added collision scenes. At the base of the robot, we added a 0.8×1.2 scene to simulate the table, a scene of the same size to simulate the wall, and a scene at the top of the robot to simulate the ceiling**.**

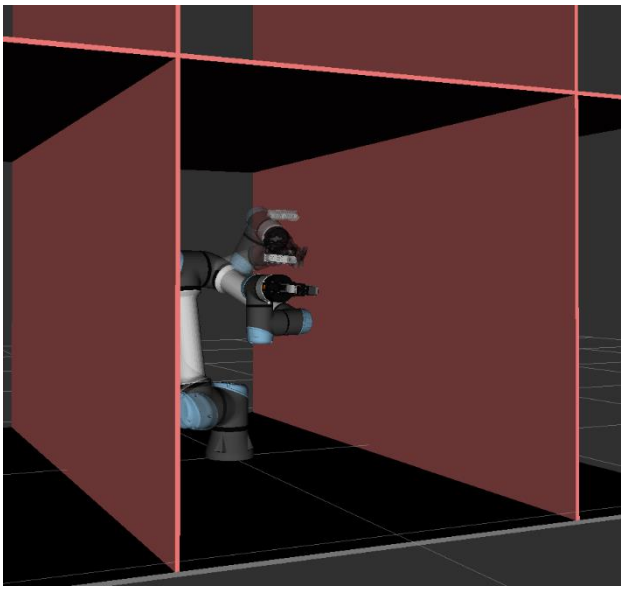


Figure 16 collision scene

4.3 Obtained Results

In this part, we implemented the robot control framework. We achieved the following results:

- We are capable of performing both inverse and forward kinematics.
- We can carry out motion planning that generates either a linear or curved trajectory with constant speed and constant acceleration.
- We have successfully avoided undesirable movements while the robot performs its tasks.
- We have implemented real-time control. When the computer vision module detects the targets, and these targets are accessible, our robot is capable of executing the corresponding task.

5 Camera calibration and Computer vision

5.1 Context and Problem Statement

Context

In the context of the Mura Project, we have integrated cameras to establish a servo-vision system. This setup necessitates meticulous camera calibration and eye-in-hand calibration to ensure that the 3D data we acquire is both accurate and precise.

With these calibrations complete, our robot is now equipped with a camera calibrated, enabling it to capture 3d visual information and identify specific targets through the lens of the camera. The next stage in our project will be dedicated to implementing facial recognition technology. This will allow us to extract valuable facial data, which is a critical component of our makeup application initiative. Our goal is to develop a comprehensive system capable of tracking facial movements, analyzing facial features, and generating appropriate robot trajectories. By applying inverse kinematics, we will guide the robot through the necessary waypoints, ensuring precise and accurate makeup application.

Problem

We are currently facing several challenges:

1. Perform a camera calibration : This step is crucial to ensure that the images captured by the camera precisely correspond to the real environment. It involves determining the internal parameters (such as focal length) and external parameters (such as position and orientation) of the camera.
2. Perform a hand-eye calibration of the robot and the camera: This step aims to determine the spatial relationship between the camera and the robot's gripper (or "hand"). This allows the system to know exactly where the object grasped by the gripper is in relation to the camera, and vice versa. This calibration is essential for allowing the robot to interact accurately with its environment based on the visual information collected.
3. How to achieve precise human face detection and extract the information that interests us?
4. How to accurately determine the head pose?
5. How to perform a transformation from 2D to 3D, and switch from one coordinate system to another?
6. How to control the robot so that it executes the trajectory we have defined?
7. Once the trajectory is complete, how to apply color to the part of the face that we wish to apply makeup to?

5.2 Methods Used to Address the Issue

Context

1. Camera calibration and hand-eye calibration

In order to tackle the challenges associated with precise face detection, head pose estimation, and robot control for makeup application, we have employed a combination of camera calibration and hand-eye calibration techniques. These methods are crucial for ensuring accurate mapping between the robot's coordinate system and the camera's coordinate system, ultimately leading to more precise and reliable results. Below, we delve into each of these methods in detail:

Camera Calibration: This process is essential for determining the intrinsic and extrinsic parameters of the camera, which allows us to understand how the camera maps 3D points in the world to 2D points in the image. Intrinsic parameters include the focal length, principal point, and lens distortion coefficients, while extrinsic parameters describe the camera's orientation and position in space. We have utilized standard calibration methods, employing a calibration pattern (a checkerboard) and capturing multiple images of the pattern at different orientations and distances. By analyzing these images, we have been able to estimate the camera parameters, allowing for the correction of lens distortion and the computation of the transformation between the camera's coordinate system and the world coordinate system.

Hand-Eye Calibration: This method is used to find the transformation between the robot end-effector and the camera. Precise knowledge of this relationship is crucial for ensuring that the robot accurately follows the defined trajectory relative to the face and applies makeup to the correct locations. To perform hand-eye calibration, we have collected a series of robot poses and corresponding camera images, using these to compute the transformation matrix that relates the two coordinate systems, We used the AX=XB algorithm, may be employed for this purpose, ensuring robust and accurate calibration results.

2. Computer Vision

Face detection is a key area in computer vision. Although numerous models have been developed to effectively perform this detection, our project requires more. We aim not only to

detect faces but also to extract characteristic points, or "landmarks," on the face. Therefore, we have decided to use MediaPipe, a deep learning-based model developed by Google.

Media Pipe provides us with valuable information about faces and their features. It is capable of providing us with 468 unique features, including details like the contours of the eyes, nose, and mouth. By exploiting these characteristic points and the relationships between them, we have acquired a structured representation of the face, or "mesh." This mesh allows us to perform virtual makeup in our simulation environment.

Moreover, when we combine Media Pipe with an RGBD camera, we are able to extract 3D information from the characteristic points. This information allows us to determine a trajectory for our robot, thus increasing the accuracy and efficiency of our automated makeup system.

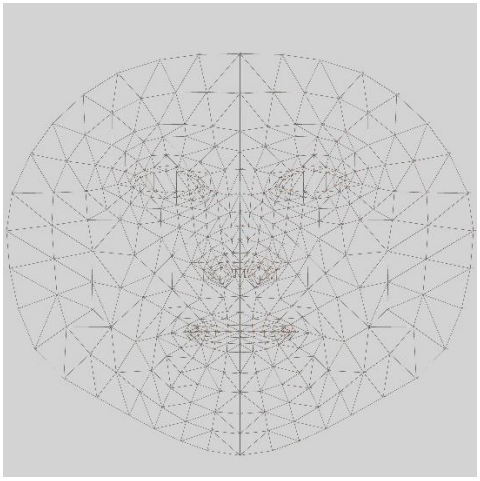


Figure 17 Mediapipe mesh with 468 key points

Camera Calibration and Hand-Eye Calibration

For camera calibration and "eye-in-hand" calibration in our project, we combined the two to acquire both the properties of the camera and information on the relationship between the camera and the gripper. We used a calibration tool called ArUco. In our simulation, we created an ArUco board as a reference coordinate system.

The calibration process unfolds in several steps:

1. Localization of the ArUco Board: The image captured by the camera is used to locate the ArUco board. By detecting the ArUco markers, we can determine its position and orientation in space.

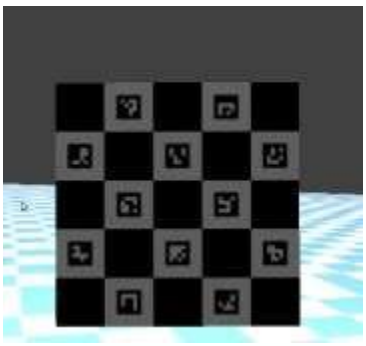


Figure 18 ArUco Board

2. Feature Extraction: The detected ArUco markers allow us to extract the necessary features for calibration, such as the positions of the board corners and the marker IDs.

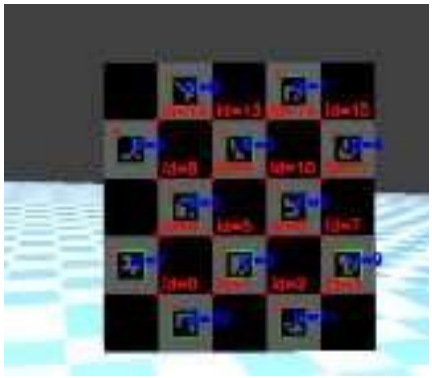


Figure 19 Extraction of key point

3. Image Acquisition from Different Gripper Positions: To achieve a hand-eye calibration, which is crucial for precise robot control, we altered the position of the gripper in order to obtain images from various angles. This process was repeated 10 times to ensure accurate calibration and to gather a sufficient amount of visual data. These data allow us to refine the spatial relationship between the camera (the eye) and the gripper (the hand), thereby facilitating more precise robot control.

Figure 20 Different pose

4. Calculation of Calibration Parameters: Using the extracted features, we can calculate the camera's calibration parameters, such as the intrinsic matrix (focal length, optical center) and radial distortion.
5. Eye-in-Hand Calibration: Once the camera calibration parameters are obtained, we proceed to eye-in-hand calibration. This involves determining the transformation between the camera and the gripper. We use the precise gripper movements made 10 times, as well as the camera pose readings, to calculate this transformation.
6. Calibration Validation: To verify the accuracy of the calibration, we use some points measured by myself. We take measurements with the gripper in known positions and compare the calculated coordinates to the actual coordinates.

MediaPipe Configuration to Adapt to ROS

In our project, we rely on ROS Melodic, which is built on Python 2.7. However, MediaPipe, the tool we are using, requires a Python 3 environment. Therefore, our first challenge was to resolve the configuration between MediaPipe and ROS Melodic.

We used Conda to set up our MediaPipe environment. The first problem in this environment was how to send images from Gazebo to MediaPipe. In ROS, we can use topics to allow communication between different parts, but this relies on various ROS message formats.

Initially, we used this method, but we encountered a serious conflict between Python 2.7 and Python 3 because of cv_bridge.

As a result, we decided to bypass this problem and designed two methods to transmit images to MediaPipe. One of them is based on the server-client communication structure of ROS. We store the images and publish their address. When MediaPipe needs a new image, it sends a request to the ROS master, which forwards this request to the server topic. The server topic then sends the image address to MediaPipe, allowing it to acquire an image.

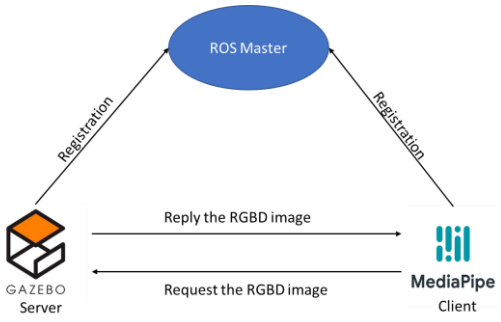


Figure 21 ROS server-client communication

The second solution we developed involves creating a shared memory space in a terminal. We sized this memory according to the size of the image, allowing it to temporarily store up to 10 images. Based on our tests, we discovered that there is a 0.002s delay receiving an image. We subscribed to the image topic and acquired the image in the form of a cv2 object. Afterwards, we transmitted this image to the shared memory. Thus, MediaPipe can access this memory to retrieve the images. This solution provides an efficient means to resolve compatibility issues between different versions of Python while ensuring seamless communication between ROS and MediaPipe.

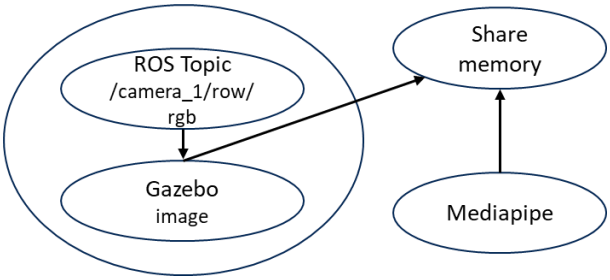


Figure 22 Share memory for image acquisition

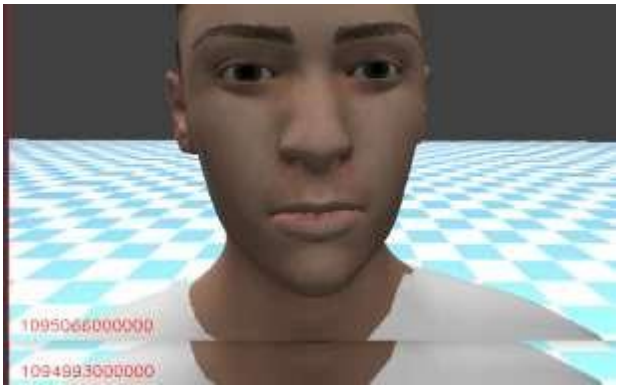


Figure 23 Delay time comparison

The two methods we have implemented both effectively transmit images, yet they come with distinct advantages and disadvantages.

The client-server method requires a substantial amount of disk storage to save the images. Its advantage lies in the rapid transmission of images, resulting in low latency. This can be crucial for applications that demand real-time responsiveness and is well-suited for actual system deployment.

On the other hand, the use of shared memory incurs a greater delay compared to the client-server method, but it offers a significant advantage: there is no need to delete the stored images each time the simulation environment is launched. This can simplify resource management and enhance the efficiency of the work environment.

Therefore, it is crucial to weigh these considerations when selecting the most suitable method for a specific project, balancing performance needs, storage resources, and ease of management.

Figure 24 RGB and depth image acquisition

Acquiring 3D Facial Information in Camera Coordinates.

Once we have resolved the issue of image transmission, we are able to obtain RGB images as well as depth images. For the MediaPipe model, it utilizes RGB images to detect the face and predict landmark points. It can provide us with the facial contour and the proportional values of each point (Px, Py, Pz). By multiplying these values by the image size (width, height), we obtain the pixel positions of the key points.

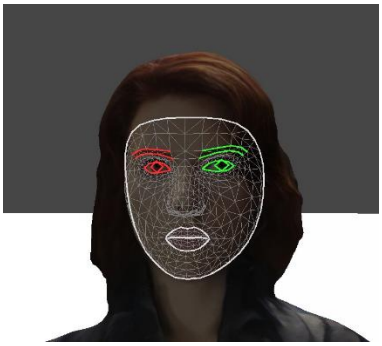


Figure 25 output of mediapipe

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} P_x \\ P_y \end{bmatrix} \begin{bmatrix} W_{image} \\ H_{image} \end{bmatrix}$$

Equation 2 Pixel coordinates acquisition

When we have the pixel coordinates, we can use the camera properties, represented by the calibration matrix K, to compute a transformation from pixel coordinates to camera coordinates.

However, before proceeding with this calculation, it is important to perform a distortion correction step to mitigate the undesirable effects of optical distortion.

To correct these distortions, we use the distortion coefficients associated with the camera D. These coefficients are typically determined during a camera calibration step, where images of a known calibration pattern are used to estimate the distortion parameters. Once we have these coefficients, we can apply mathematical formulas to correct the distortion in the images.

$$Pixel_{undistorsion} = f(K, D, Pixel_{distortion})$$

Equation 3 undistorsion of pixels function

Once the pixels have been corrected for distortion, we can proceed to calculate the transformation from 2D to 3D. To do this, we use the calibration matrix K, which contains the intrinsic parameters of the camera such as the focal length, the coordinates of the principal point, and the pixel scales. Using this matrix, and the depth information we have obtained from the depth image, we can project the pixel coordinates in the image onto a 3D plane.

$$\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = [K]^{-1} \begin{bmatrix} U \\ V \\ 1 \end{bmatrix} * Depth$$

Equation 4 Transformation from 2D to 3D

When we acquire 3D information, we can perform the following tasks:

1. Select the information that is relevant to our project, such as the 3D data of the lips.
2. Transform the camera coordinates to the robot base frame.
3. Generate a trajectory for the robot.
4. Evaluate the orientation of the head.
5. Perform virtual makeup on a human model.

Lip Information Acquisition

After computing the 3D information, we obtained 468 points in 3D. To select the information concerning the lips, we used the indices provided by MediaPipe $Indexi$, selecting 60 key points (xi, yi, zi), and the connection relationships to group the mesh ($indexi$, $indexj$, $indexk$). These pieces of information are the foundation for our project; they can be used in trajectory planning and the colorization of human models.

Figure 26 lips part key points

Transforming Camera Coordinates to Robot Base Framer

When we have acquired information in the camera's coordinates, the next step is to transfer these camera coordinates to the robot base frame. In this case, we use the data from the eye-

to-hand calibration, namely the translation T and rotation R.This allows us to obtain the transformation matrix of the gripper relative to the camera:

$$T_{grippeur}^{camera} = [R\ |T]$$

To obtain the head pose in the robot base coordinates, we use the following transformation:

$$T_{base}^{tête} = T_{base}^{grippeur} \times T_{grippeur}^{camera} \times T_{camera}^{tête}$$

Équation 5 transformation equation from head to robot base

In our cas, we record the gripper's pose using the Ros topic to compute $T_{base}^{grippeur}$ and we also computed the head pose. Once we have acquired these positons, we can compute the head orientation, which allows us to compute the transformation matrix $T_{camera}^{tête}$.

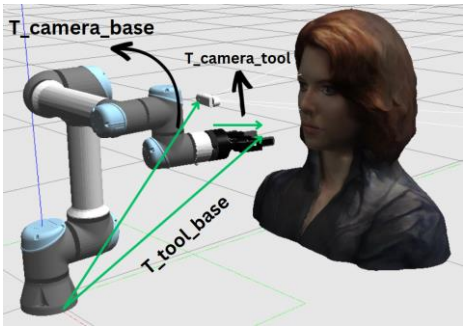


Figure 27 Coordinate transformation from camera to base robot

Head Pose Estimation

In computer vision, the pose of an object refers to its orientation and position relative to a camera. In our project, the head pose estimation issue we are facing is known as the Perspective-n-Point problem. In this problem, when we have a calibrated camera, as well as the 3D positions of n points and their corresponding 2D projections, our goal is to determine the pose of this object in the camera's coordinates.

As a result, when we want to calculate the 3D pose of the object, we need the following information:

- 2D pixels of key points : We have chosen 6 points $[U_{ref}, V_{ref}]^T$ to evaluate the pose, MediaPipe gives us many options, and we will use the tip of the nose, the chin, the left corner of the left eye, the right corner of the right eye, the left corner of the mouth, and the right corner of the mouth.
- 3D position of the same points: Here, we have selected 3D locations of some points in an arbitrary reference frame $[x_{ref}, y_{ref}, z_{ref}]$.

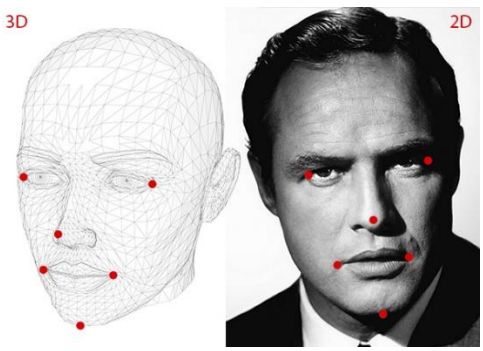


Figure 28 Head reference points

To estimate the head pose, we used three coordinates: world coordinates, camera coordinates, and image coordinates. The pose relations represent the transformation between world coordinates and camera coordinates. When we know the 2D information of a pixel, we can use this information to calculate the 3D information in the camera coordinates. Thus, the estimation problem can be simplified by the following equation:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = [R \mid t] \begin{bmatrix} x_{ref} \\ y_{ref} \\ w_{ref} \\ 1 \end{bmatrix}$$

Equation 6 Calculation equation for head pose

In this equation, $[X, Y, Z]^T$ represents the 3D coordinates in the camera corrdiante systems, $[x_{ref}, y_{ref}, z_{ref}]^T$ represents the 3D coordinates in the fashion coordinate system. R and T represent the head pose. That is why we must choose several relations between pixel points and 3D points to calculate this transformation relationship.

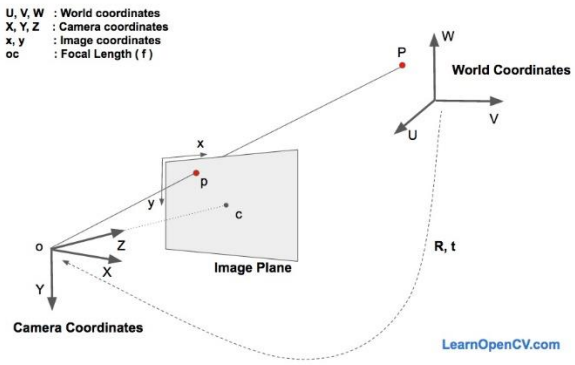


Figure 29 Head-to-camera cord transformations

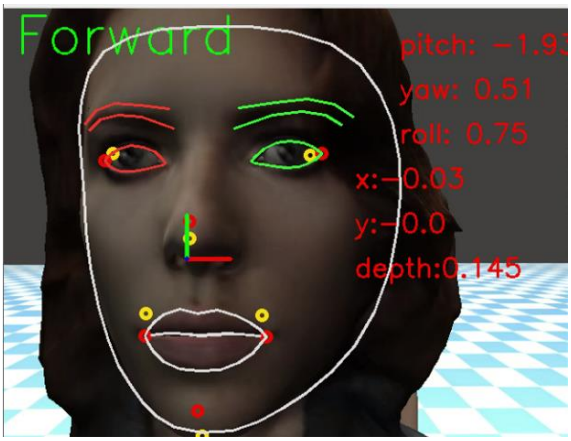


Figure 30 head pose estimate

Generalizing a Trajectory for a Robo

Once we have obtained the head pose and the key point poses using equation 7, we can calculate the feature points' poses in the robot base coordinate system.

In our case, we have chosen 60 points. The next step is to solve the issue of creating a lip outline. The robot's trajectory is not a linear trajectory but a curve. To obtain a trajectory that fits the lips well, we have decided to add waypoints to break the curve into segments.

In our approach, we use the base 60 points and iterate to obtain 120 points. This allows us to create a more detailed and precise trajectory for lip movements.

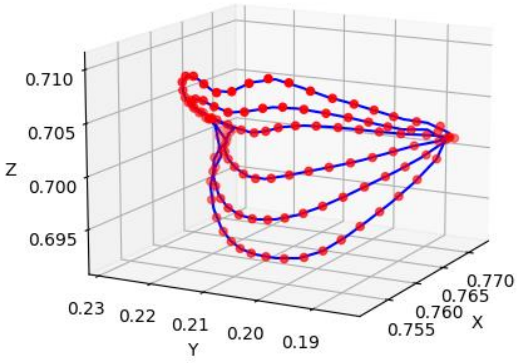


Figure 31 trajectory of lips part

5.3 Results

In the Computer vision section, we obtained the following results..

1. We have acquired the camera properties, the intrinsic matrices, and the distortion matrix.
2. We have obtained an extrinsic matrix, which contains the translation relationship and orientation between the camera and the gripper.
3. We have properly evaluated the head pose, based on Mediapipe and RGBD image, we have obtained the characteristic points for the areas of interest.
4. Based on Mediapipe, we have achieved facial tracking, we have also evaluated the head pose and we have acquired the 3D information of the characteristic point that we need.
5. We have performed a coordinate transformation, based on the properties of the hand-eye calibration and the initial pose of the robot, we have calculated the distance between the robot base and the target. We can control the robot to reach the targets that we have collected.
6. Based on the characteristic points from Mediapipe, we have defined a trajectory for the lip area; here, we want the robot to traverse all the points, that is, when the robot crosses these points, the robot has completed a makeup application on the face.

6 Make-Up virtual and Make-up in Gazebo

6.1 Context and Problem Statement

Context

In our endeavor, we require an algorithm that can realistically depict the application of makeup on a human face, providing a visual preview of the expected outcome prior to the initiation of the robotic process. Following this, it is essential to adapt and apply this visualization to a human model within the Gazebo simulator, mirroring the actions as the robot commences its makeup application task. This not only facilitates a priori validation of the intended makeup look but also enables real-time simulation to monitor the robot's performance and refine its actions through subsequent iterations.

Problem

We are currently facing several challenges:

1. How to implement augmented reality (AR) makeup on a human face.
2. How to apply virtual makeup to a face on a Gazebo human model.
3. How to implement real-time texture updates on a Gazebo model

6.2 Methods Used to Address the Issue

Context

In our project, we used Mediapipe for face and key point detection. Utilizing these detected key points, we delineated the makeup application zones on the image, such as the eyebrows and lips.

For the real-time makeup application on a Gazebo simulation model, it's crucial to establish a correspondence between the key points identified by Mediapipe on the human face and the texture key points on the Gazebo model. Once this relationship is ascertained, we can execute virtual makeup within the simulation environment. To achieve this, we plan to develop a plugin that enables real-time texture updates in response to the robot's actions during the simulation.

Colorisation de la model humain.

Lorsque notre robot a terminé son mouvement, nous souhaitons effectuer une coloration sur notre modèle, c'est-à-dire changer la texture de notre modèle. Pour réaliser cette tâche, nous devons effectuer les étapes suivantes :

- Comparer les maillages (mesh) de Mediapipe et du modèle dans Gazebo. Nous effectuons une projection du maillage de Mediapipe sur le maillage du modèle dans Gazebo, puis nous établissons une relation entre la texture de Gazebo et le maillage de Mediapipe. Cela nous permet de comprendre comment appliquer la texture du modèle sur les parties correspondantes du maillage de Mediapipe
- Comparer les trajectoires réelles avec la trajectoire prédéfinie pour sélectionner la zone à colorier. Nous comparons les trajectoires suivies par le robot avec la trajectoire prédéfinie pour déterminer les parties du modèle qui nécessitent une coloration.

Make-up effects demonstration

In our section, we focused the lips zone makeup, based on the mediapipe, we have chosen 40 keypoints that follow the contours of the lips, forming the closed curve that defines our coloring area. Then, we have chosen the makeup color that we what.

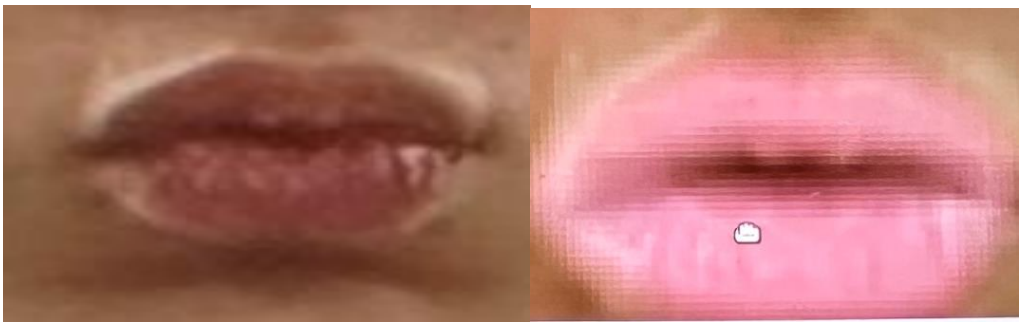


Figure 32 virtual makeup image

Mesh projection

When we compared the meshes of Mediapipe with those of the Gazebo model, we faced an initial challenge: extracting the information contained in the .dae file. This file includes information that we need, such as UV mappings, 3D vertex data, and vertex group indices for each mesh, as well as pixel index groups for each triangle.

To enhance the readability of the 3D file, we turned to MeshLab software to convert the .dae file into an .obj file. This allowed us to better understand the structure of the 3D mesh document and to easily extract the mesh-related information.

After we extracted the mesh information from the .obj file, we began to collect the specific 3D information related to the lip area. We selected four key points as boundary values and projected these 3D points onto the y,z plane. Based on these four points, we defined an oval shape and looked only for points within this area. Those points constituted our initial selection.

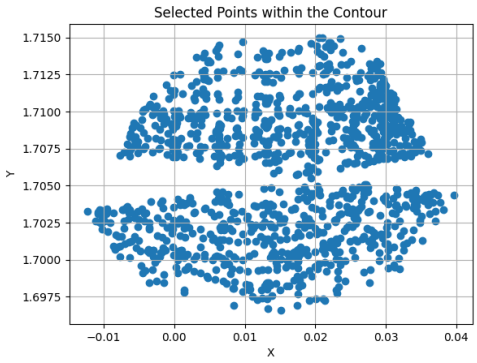


Figure 33 2D projection for the lip section

Then, we used the depth information to eliminate superfluous points, to reduce the sphere of analysis.

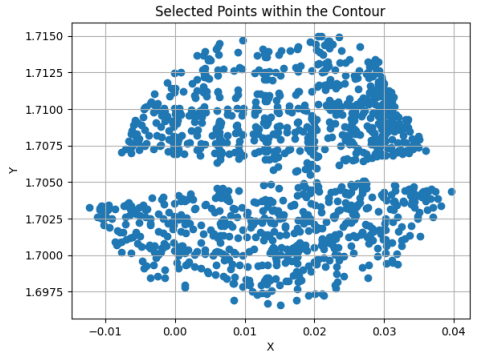


Figure 34 Second selection by lip depth

After these two preselection steps, we normalized the 3D points from Mediapipe and those of the Gazebo mesh. We then compared these normalized values to select 60 specific points among the 3D points of the Gazebo mesh.

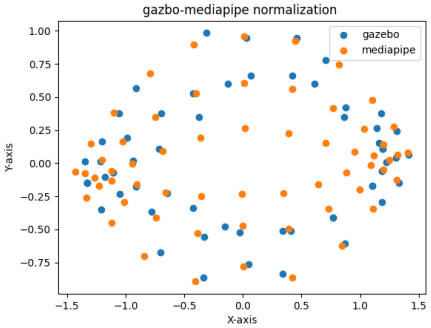


Figure 35 Projection of mediapipe mesh and gazebo mesh

Once the projection of Mediapipe's 3D points onto Gazebo's 3D points was completed, we relied on UV mapping to select the pixel coordinates of the 60 points from Gazebo. Here, by using the Mediapipe mesh, we grouped the 60 points to form triangles. We were able to perform coloring using these triangles.

Comparing actual trajectories with the predefined trajectory

Like the projection of Mediapipe and Gazebo, here, we have chosen 60 points in the trajectory of lips part that closest to the target points and compared them to assess the performance of this trajectory, or to put it another way, the quality of the makeup application.

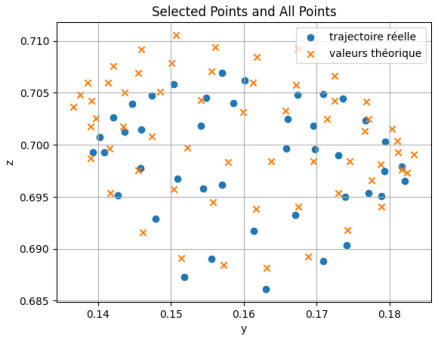


Figure 36 theoretical and actual points

After we obtained the trajectory points, we grouped them using the triangular relations from MediaPipe. This way, we were able to form groups as follows.

$$\begin{bmatrix} x_i' & y_i' & z_i' \\ x_j' & y_j' & z_j' \\ x_k' & y_k' & z_k' \end{bmatrix} \text{ et } \begin{bmatrix} x_i & y_i & z_i \\ x_j & y_j & z_j \\ x_k & y_k & z_k \end{bmatrix}$$

Here x_j' represents the real value, x_j representes a theoretical value. Then, we calculate the values for each triangle and compare the theoretical and actual averages. if $\Delta = |\bar{m} - \bar{m}'| < 0.002$, we can determine that the actual triangle is close to the theoretical triangle, which allows us to proceed with the coloring. Next, we use the triangle index group to obtain the pixel coordinates and then execute coloring on the Gazebo texture.

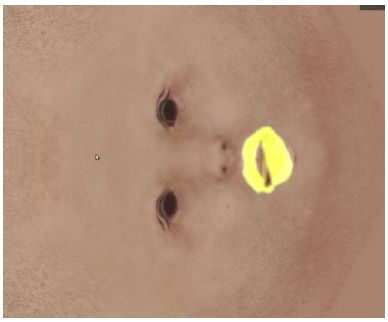


Figure 37 Gazebo texture coloring model

Afterward, when we refresh the Gazebo simulation, we can discover that the model has been colored by us.

Figure 38 3D virtual make-up in simulation environments

6.3 Results

We implemented virtual makeup and makeup in simulation environment using MediaPipe.

1. we tested the feasibility of makeup with an RGB imge by relying on the MediaPipe mesh to perform lip coloring.
2. we found the projection of this mesh between MediaPipe and Gazebo. Using the mesh data, we utilized the Gazebo texture to apply makeup to the model.it means, we implemented the real-time update makeup in Gazebo.

7 Robot Motion planning

7.1 Context and Problem Statement

Context

After finishing the computer vision part, we began the motion planning phase. In our control box defined, we have various motion controls such as move_J, move_P, and move_L. We need to utilize these motion tools to optimize our trajectories and compare them to select a group of motions that are safer and more efficient.

Problem

Les problèmes que nous devons résoudre sont les questions suivantes :

1. How can we enhance the quality of the makeup while reducing the time required?
2. How can we avoid robot singularity point when robot move and increase safety during interactions?

7.2 Methods Used to Address the Issues

Contexte

At the beginning of the project, we used inverse kinematics to control the robot. We would define a point and control the robot to reach it using an RRTs solver, which gave us movements that were less energy consuming and faster. However, as the project progressed, we found that it took a long time to track a trajectory and when the robot passed the singularity there were excessively large movements.

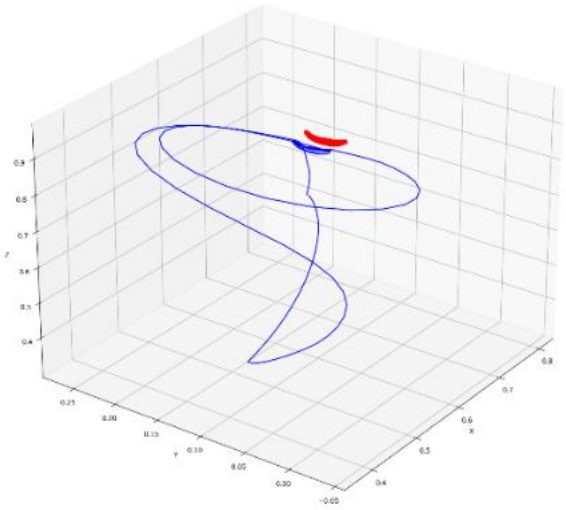


Figure 1 Move_P trajectory calculated by an RRTs

Moreover, we found that using Move_P to control the robot was not an ideal approach, as executing a makeup movement involves not only kinematic but also dynamic issues. We need to consider the velocity in both x and y directions. For Move_P, it has a significant flaw in that it reaches points one by one. Throughout the process, there are phases of acceleration and deceleration, going from zero to zero. This results in slight pauses as it passes all the waypoints.

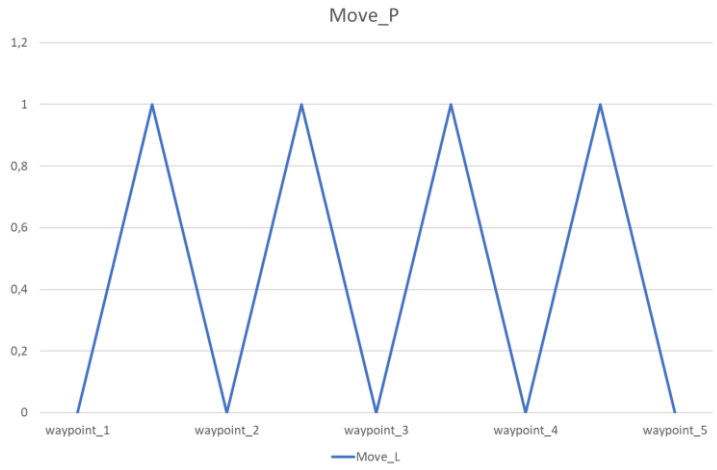


Figure 2 Move_P velocity variation for all waypoints

Therefore, we decided to use the Move_L tool. With this tool, we can define different numbers of waypoints and set their resolution. It can calculate in one go the robot's configuration groups to cross all waypoints at a constant speed.

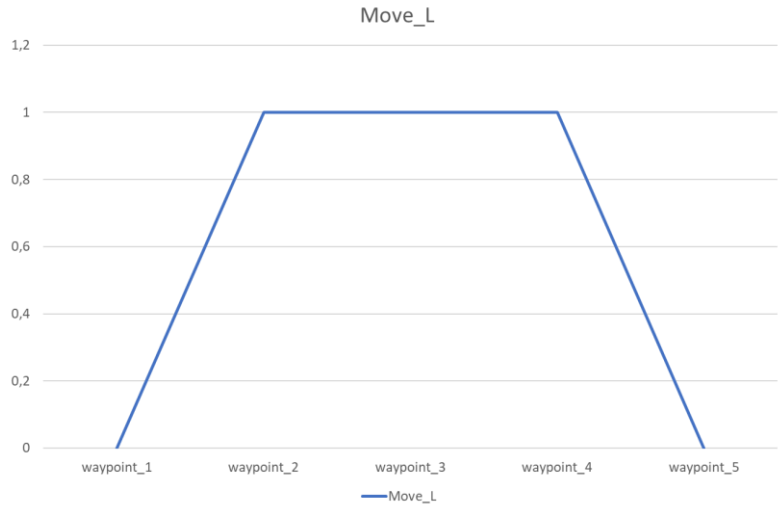


Figure 3 Velocity variation of move_L

Trajectory Optimization

In our project, we divided the robot's trajectory into two parts: one for picking up the lipstick, and the other for applying the makeup.

Let's start with the trajectory for picking up the lipstick. In our case, we have defined the lipstick's position to the left of the robot at [0,-0.3,0] in the robot's base coordinate system. To accomplish this task, we have defined five waypoints: the default position p_0, the initial point p_1, the preparation point p_2, the picking point p_3, and the detection point p_4. We compared Move_P and Move_L methods and found that the calculation of Move_P is not always stable. Sometimes, it leads to an unstable trajectory, and the robot is unable to pick up the lipstick correctly. Therefore, we decided to use Move_L, which is more stable and allows the robot to follow the trajectory we have set correctly.

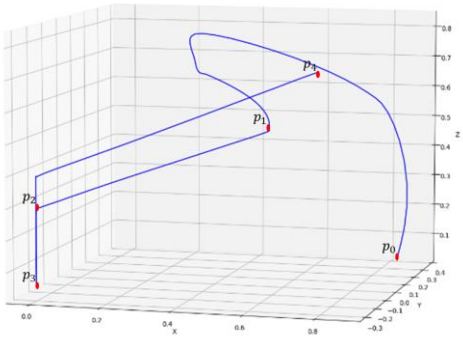


Figure 42 Move_L and Move_J trajectory for picking lipstick

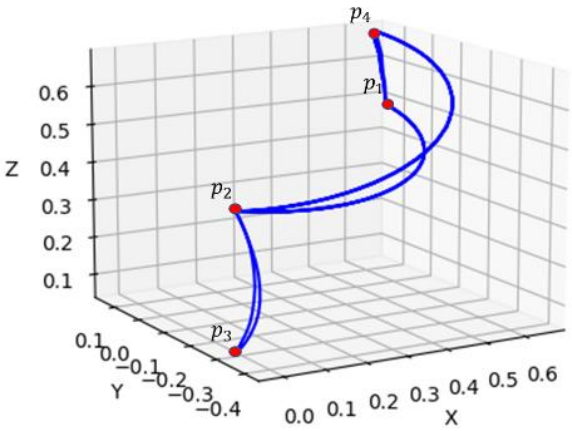


Figure 43 Move_P trajectory for picking lipstick

In the context of makeup trajectory, we have made a comparison between Move_P and Move_L. We compared the time spent from receiving the command to completing all tasks. We also compared the trajectory for each set of waypoints.

We relied on ROS (Robot Operating System) and used topic communication to publish and subscribe to information related to commands. We broadcast the predefined trajectory to our robot. When the robot receives a command, it follows this command and executes the actions we have defined.

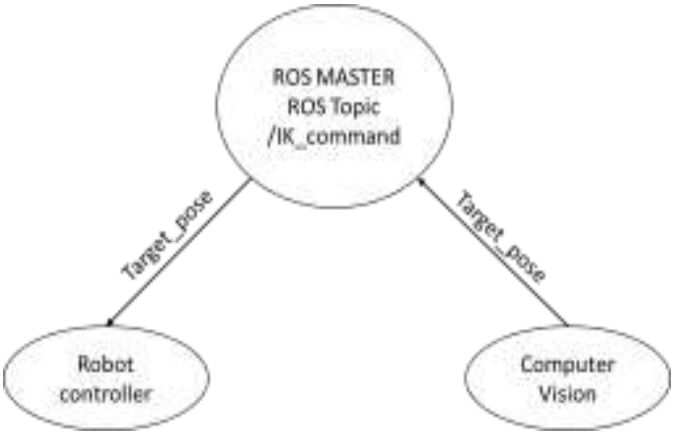
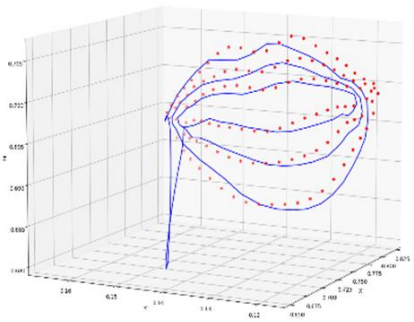
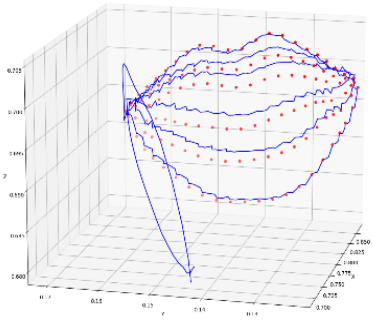
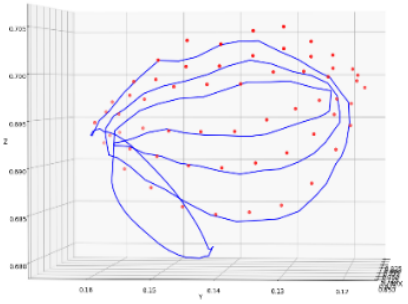
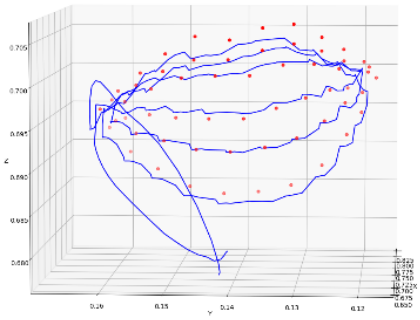


Figure 44 robot structure servo control in real time

After the robot has completed the task, we counted the usage time and the trajectory of the robot. In our system, when the robot performs the Move_P, it carries out an action as soon as it receives a command. If the robot performs the Move_L, it receives all the commands, then carries out the calculations before executing the actions. Therefore, when executing a Move_L, the robot spends time collecting information and performing calculations. For Move_P, it doesn't do all the calculations at once but carries them out one by one. This makes the Move_P slower than Move_L.

Table 1 trajectory comparison for different waipoints and different moves

	Move_L	Move_P
60 waypoints	23s	57s

120 waypoints	53s	247s
120 waypoints Trajectory		
60 waypoints Trajectory		

Following the tests conducted, we observed that an increase in the number of waypoints leads to an increase in the time required for calculating the robot's configuration. This is especially true when using the Move_L method, which, although it saves time compared to Move_P for the same number of waypoints, offers less precision.

Indeed, we have noted that Move_P offers higher precision than Move_L, being able to pass through more key points for an identical number of waypoints. However, it is important to emphasize that this increased precision comes with a longer duration and may present more risks due to the accelerations and decelerations necessary during the actions.

In conclusion, if Move_L is faster than Move_P, it is Move_P that is the most precise of the two. Nonetheless, the slowness and the higher level of risk associated with Move_P must be taken into account, especially because our solver can sometimes propose a non-optimal configuration of the robot, generating significant movements during the execution of a task.

7.3 Résultats obtenus

Once the trajectory planning is complete, we are able to perform a makeup application on our model in a static scenario. However, applying makeup on an actual human being still represents a major challenge. The first challenge is the management of force. If we cannot control the force, real-word implementation becomes unviable.

8 Renforcement Learning Framework

8.1 Context Problem Statement

Context

In the seventh part, our robot can perform a trajectory with constant velocity, the next step, we want to control the robot's output a constant and safe force. In other words, we need to maintain a constant force in the direction normal to the trajectory. To solve this dynamic challenge, we have two solutions: one model-based and the other data-driven.

About the model-based approach, we assume that the robot's joints function like a second-order mass-spring-damper system. We need to define a dynamic function to represent the state of the robot, and an output function to represent the robot's output. This is a relatively complex process, particularly in constructing the dynamic model.

Then data-driven approach, the robot can simply rely on a policy to perform an action that is more suited to the current state. This method can be more flexible and adaptable, but it requires a large amount of data for learning.

Problem

The problems we need to solve are the following questions :

1. Develop a Reinforcement Learning framework to perform precise actions, receive relevant observations, and acquire a reward.
2. Select a training strategy to train a policy.

8.2 Methods Used to Address the Issues

Construction the RL Framework

In our project, we built our reinforcement learning framework based on OpenAI Gym. We integrated components such as Gazebo and Mediapipe and added various topics to our framework.

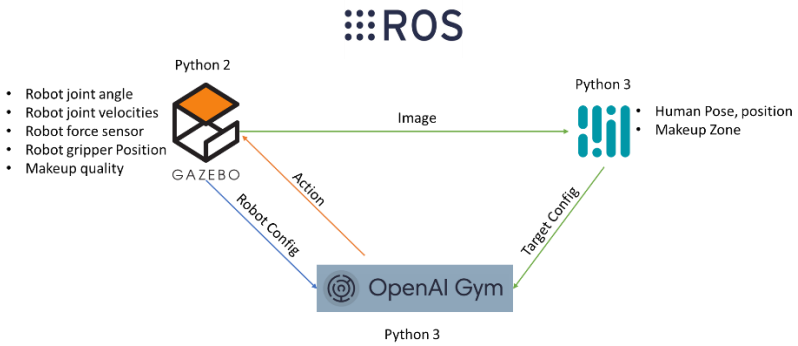


Figure 45 MURA Project RL Framework

Moreover, while building the reinforcement learning framework, we encountered a significant information bottleneck and unstable connections when using the rostopic to transmit information. Consequently, we switched to an alternative communication method. We established a client-server communication system to realize control through inverse

kinematics. Here, we sent the robot control message, and the server provided the gripper's position.

In our framework, we have an action space based on inverse kinematics, which allows for executing actions such as moving forward, backward, and turning, based on the agent's decision. We established an initial position$[x, y, z, x_q, y_q, z_q, w_q]$ and a value δ =0.001 which allows the robot to execute actions, as demonstrated by the following function:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} \pm \delta = robot\ action$$

Here, we do not wish to change the gripper's orientation, in order to simplify the learning process. We only want our robot to reach the waypoints and find a trajectory that adapts well to the shape of the lips.

Next, for an observation space, we want to acquire the robot's pose, the information from the force sensor, the quality of the makeup, and the characteristic points of the lips.

$$\begin{bmatrix} p^{6\times n} \\ F^{1\times n} \\ p_{target}^{6\times n} \end{bmatrix}$$

For the reward component, here we have defined bonuses for our action.

- Force reward

$$\Delta_1 = F_{output} - F_{target}$$

  Here, we can define a desired value for force. we can read the actual forces in the observation space and minus the target value to obtain a value Δ_1. In this case, we have a force threshold ε=1. When we get a value $|\Delta_1| > \varepsilon$. we do not receive a reward but if the value is less than ε ,we can have a reward.

$$r_1 = \begin{cases} r_f, & |\Delta_1| \le \varepsilon \\ 0, & |\Delta_1| > \varepsilon \end{cases}$$

- Key points Reward

  In our case, we have 60 targets, here we make a comparison between the theoretical point and the real points, by calculating the gap between the gripper's position and that of the target, if the robot approaches the targets, it will receive a rewa.

$$\Delta_3 = P_{\text{gripper}} - \mathrm{P}_{\text{target}}$$

$$r_2 = \begin{cases} r_p, & |\Delta_2| \le \varepsilon \\ 0, & |\Delta_2| > \varepsilon \end{cases}$$

- Makup Quality Reward

  Here, the reward for makeup quality represents a significant bonus. We have collected the positions of the gripper, then, after training, we compared them to the target points. If we have crossed all the points, we will receive a large bonus proportional to the number of points crossed by the robot.

$$r_3 = N_{points}$$

  Then, every time the robot executes an action, it is possible to acquire a reward. When the robot finishes the actions, or it has crossed all the points, it will have a large reward.

$$r_{n\_action} = \left(\sum r_1 + r_2 + r_3\right)$$

8.3 Résultats obtenus

As for the part about reinforcement learning, I have not yet obtained results that can be deployed on a real robot. Sometimes we encounter information blockages between different topics, which interrupts the training of the model. While writing this report, we are also working to solve these problems, such as overcoming the limitations of Gym and avoiding complex processes related to the request for a ROS topic. We are trying to use a server-client communication, where the robot, when executing an action, can send the gripper's position once the action is completed. This method of communication seems to fit well with the needs of the reinforcement learning framework.

9 Synthèse, conclusion et perspectives

The "Make-up for Robot Arm" project aims to achieve automated makeup application using a robotic arm. This project seeks to create a physical interaction between humans and robotics, presenting promising potential for the future. Within this project, we have developed a synchronized simulation environment with our real environment, using the ROS (Robot Operating System) framework. Using this environment, we have deployed a computer vision module as well as a robot control module. This has allowed us to effectively detect a face and extract its feature points. Then, thanks to the control module, we were able to calculate the configurations of the robot's joints necessary to perform movements along a predefined trajectory. During this internship, we not only carried out simulations but also tests and deployments of the "media pipe" and "robot control" modules on our real robot. These tests have allowed us to verify the proper functioning of our algorithms.

Overall, our project has achieved its set objectives and demonstrated the feasibility of automated makeup application by a robotic arm. However, there are still opportunities for improvement and prospects to explore. Here are some directions for future developments:

1. Addition of a force control module: In our project, we have not yet implemented force control. However, in the interaction between humans and robots, it is essential to consider the force to protect the human and improve the quality of the makeup application.

2. Improving precision: Refine the face detection algorithms and feature point extraction for even more precise and reliable results.

3. Improving the reinforcement learning module: During my internship, I didn't have much time to test different RL models, such as the Actor-Critic model. We compared the value-based model, the Policy model doesn't need a large space to memorize actions. It is more suited for robot control especially for continuous actions. Moreover, I think that a hybrid model is also a good direction for the RL model.

4. For the computer vision part, I think we need to consider how to make a prediction or calculate the orientation of each feature point. In our case, we used the orientation of the head as the gripper's orientation, but ideally, the gripper's orientation should be normal to each point.

In conclusion, the "Make-up for Robot Arm" project has been an enriching experience that led to the realization of automated makeup application by a robotic arm. The results obtained pave the way for new possibilities in the field of human-robot interaction in the beauty sector. The prospects for improvement and future development are promising, and we look forward to seeing how this project will evolve and contribute to the future of automated makeup application.

10 Bibliographie

[1] ROS Tutorial: Control the UR5 robot with ros_control – How to tune a PID Controller https://roboticscasual.com/ros-tutorial-control-the-ur5-robot- with-ros_control-tuning-a-pid-controller/

[2] Camera calibration https://fr.mathworks.com/help/vision/ug/camera- calibration.html

[3] Head Pose Estimation using Python https://towardsdatascience.com/head- pose-estimation-using-python-d165d3541600

[4] Ros wiki Documentation http://wiki.ros.org/Documentation*

[5] Gazebo Tutorials https://classic.gazebosim.org/tutorials

[6] Lugaresi, C., Tang, J., Nash, H., McClanahan, C., Uboweja, E., Hays, M.,

Zhang, F., Chang, C-L., Yong, M.G., Lee, J., Chang, W-T., Hua, W., Georg, M., & Grundmann, M. (2019). MediaPipe: A Framework for Building Perception Pipelines. arXiv preprint arXiv:1906.08172.

[7] Liu, W., Niu, H., Pan, W., Herrmann, G., & Carrasco, J. (2023). Sim-and- Real Reinforcement Learning for Manipulation: A Consensus-based Approach. arXiv preprint arXiv:2302.13423.

[8] Franceschetti, Andrea & Tosello, Elisa & Castaman, Nicola & Ghidoni, Stefano. (2021). Robotic Arm Control and Task Training through Deep Reinforcement Learning.

[9] Liu, Wenxing & Niu, Hanlin & Mahyuddin, Muhammad Nasiruddin & Herrmann, Guido & Carrasco, Joaquin. (2021). A Model-free Deep Reinforcement Learning Approach for Robotic Manipulators Path Planning.

[10] Singh, A., Yang, L., Hartikainen, K., Finn, C., & Levine, S. (2019). End-to- End Robotic Reinforcement Learning without Reward Engineering. ArXiv:1904.07854 [Cs.LG].